ABSTRACT

Title of Dissertation: COORDINATION AND LEARNING ALGORITHMS FOR MULTI-ROBOT INFORMATION GATHERING

Guangyao Shi
Doctor of Philosophy, 2023

Dissertation Directed by: Professor Pratap Tokekar
Department of Computer Science

With the rapid improvement in perception and planning technology, robots are being increasingly used as smart, adaptive sensors to gather information in applications such as environment monitoring, infrastructure inspection, and security and surveillance. To fully exploit the potential offered by robotic sensing, we need efficient and reliable decision-making techniques to decide when, where, and how to gather information. Such decision-making techniques need to account for the uncertainty and partial knowledge inherent in the working environment. The goal of this dissertation is to design algorithms to enable a multi-robot team to collectively and efficiently gather information on spatiotemporal fields without full knowledge of the environment. Our contributions span the full spectrum of the knowledge of the environmental conditions: from one extreme where the environmental model is fully known to the other extreme where the environmental model is unknown but can be learned from empirical data. We present several efficient (i.e., polynomial time) and effective (i.e., optimal or bounded approximation guarantees) algorithms for multi-robot information gathering.

In the first part of the dissertation, we study coordination algorithms when the environmental model is fully or partially known. Specifically, for the case where the environmental model is fully known, we consider the challenge imposed by the connectivity requirement of the team. We present an algorithm for connectivity-constrained submodular maximization for information gathering that requires intermittent communication among the robotic team. For the case where the environment is partially known, and uncertainty exists, we seek to make the multi-robot team robust to the possible failures caused by the uncertainty. When the uncertainty is upper-bounded, we present a constant-factor approximation algorithm for robust multiple path submodular orienteering. When the uncertainty is stochastic, and the distribution is known, we introduce two risk-sensitive coordination problems for aerial-ground long-term information gathering.

In the second part of the dissertation, we study the case where the environmental model is initially unknown and needs to be learned from the data. Classically, such a learning process is independently conducted without considering the downstream task. By contrast, we present a framework that incorporates the downstream decision-making problem into the learning process. Such integration will help reduce the misalignment between the prediction model and the downstream task. The misalignment refers to a predictor that despite achieving high predictive accuracy in the learning phase may not necessarily result in good decisions in the downstream task. The general methodology to achieve such integration is to make the combinatorial optimization differentiable, which then can be treated as a differentiable module in the learning process. In addition to algorithm design, we present empirical results for applications such as active target tracking, ocean monitoring, and persistent monitoring.

COORDINATION AND LEARNING ALGORITHMS FOR MULTI-ROBOT INFORMATION GATHERING

by

Guangyao Shi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Advisory Committee:
Professor Pratap Tokekar, Chair/Advisor
Professor Nikhil Chopra, Dean's Representative
Professor Dinesh Manocha
Professor Kaiqing Zhang
Professor Ryan K. Williams

To my family.

“ Life is a marathon, not a sprint. ”

Acknowledgments

The six-year-long PhD life is like Odysseus' Journey back home. It is full of challenging obstacles and calls for faith (state estimation of where I am now and where I will be next) and perseverance (planning for what to do next) to embark on my Ithaca. I owe my deepest gratitude to all the people who have been with me on this journey.

First and foremost I would like to thank my advisor, Prof. Pratap Tokekar, for giving me an invaluable opportunity to work on this interesting and challenging research topic. I was extremely fortunate to join the RAAS lab right one month before the pandemic lockdown. Dr. Tokekar has always made himself available for help and advice and constantly encourages us in research. His dedication to research and willingness to assist students make him a great advisor and collaborator. I am deeply grateful for the freedom of exploration and exploitation that he gave me so that I can pursue a broader range of research without being lost in the forest.

Many thanks to my committee members – Dr. Ryan K. Williams, Dr. Dinesh Manocha, Dr. Kaiqing Zhang, and Dr. Nikhil Chopra – who graciously accommodated my back-and-forth emails to arrange the defense date. I thank Dr. Dinesh Manocha for the first-year summer rotation opportunities and his introduction to several UMD research mates including Shuangqi Luo, Senthil Hariharan Arul, and Liangliang Zhao.

Many thanks to my co-authors and collaborators – Lifeng Zhou, Nare Karapetyan, Ahmad Bilal Asghar, Xiaotian Xu, Yancy Diaz-Mercado, Chak Lam Shek, Rui Liu, Md Ishat-E-Rabban,

and James Humann. My special thanks also go to Dr. Lifeng Zhou who helped me publish the very first paper of my PhD life. I am looking forward to continuing to collaborate with you in the future.

I would also like to thank many of my friends and labmates – Tao Hu, Vishnu Sharma, Ming Yi, Prateek Verma, Griffin Bonner, Peihong Yu, Meixuan Li, Xingan Jiang. Thank you for bearing with me and supporting me. Finally, I would like to express my gratitude to my family: *"You raise me up so I can stand on mountains; You raise me up to walk on stormy seas; I am strong when I am on your shoulders"*.

Table of Contents

Dedication ii

Acknowledgements iii

Table of Contents v

List of Tables viii

List of Figures ix

Chapter 1: Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Overview of the Dissertation . . . 4

Chapter 2: Multi-Robot Information Gathering with Full Knowledge of the environmental model 9
- 2.1 Overview . . . 9
- 2.2 Preliminaries . . . 14
- 2.3 Problem Formulation . . . 15
- 2.4 Heuristic Algorithm . . . 17
- 2.5 Case Study: Active Target Tracking . . . 20
- 2.6 Conclusion . . . 25

Chapter 3: Multi-Robot Information Gathering with Partial Knowledge of the environmental model: Non-Bayesian Case 27
- 3.1 Overview . . . 27
- 3.2 Preliminaries . . . 32
- 3.3 Problem Formulation—Offline RMOP . . . 34
- 3.4 Algorithm for Offline RMOP . . . 35
- 3.5 Problem Formulation—Online RMOP . . . 40
- 3.6 MCTS for Online RMOP . . . 44
- 3.7 Performance Analysis . . . 48
  - 3.7.1 Sequential Greedy Assignment . . . 48
  - 3.7.2 Analysis for Algorithm 3 . . . 49
  - 3.7.3 Running Time . . . 50
- 3.8 Case Study for Offline RMOP: Ocean Monitoring . . . 52
  - 3.8.1 Simulation Setup . . . 52

3.8.2 Results . . . 54
3.9 Case Study for Online RMOP: Cave Exploration . . . 59
3.9.1 Simulation Setup . . . 59
3.9.2 Results . . . 61
3.10 Conclusion . . . 63

Chapter 4: Multi-Robot Information Gathering with Partial Knowledge of the Environmental Model: Bayesian Case 65
4.1 Overview . . . 65
4.2 Risk-aware UAV-UGV Recharging Rendezvous: One UAV and One UGV . . . 65
4.2.1 Preliminaries . . . 69
4.2.2 Problem Formulation . . . 74
4.2.3 Solutions to CMDP . . . 77
4.2.4 Simulation . . . 78
4.3 Risk-aware UAV-UGV Recharging Rendezvous: Multiple UAVs and UGVs . . . 85
4.3.1 Problem Formulation . . . 85
4.3.2 Simulation . . . 91
4.4 Conclusion . . . 94

Chapter 5: Multi-Robot Information Gathering with Decision-oriented Learning of the Environmental Model 96
5.1 Overview . . . 96
5.2 Learning a Context-aware Objective for Information Gathering: Monotone Non-negative Submodular Objective Case . . . 97
5.2.1 Problem Formulation . . . 103
5.2.2 Case Study: Ground Vehicle Routing Problem . . . 104
5.2.3 Learning Algorithm . . . 106
5.2.4 Simulation . . . 108
5.3 Learning a Context-aware Objective for Information Gathering: (Non)-monotone and Possibly Non-positive Submodular Objective Case . . . 111
5.3.1 Problem Formulation . . . 113
5.3.2 Case Study: Intervention-Aware UGV Routing . . . 116
5.3.3 Learning Algorithm . . . 117
5.3.4 Simulation . . . 121
5.4 Conclusion . . . 126

Chapter 6: Conclusion and Future Work 127
6.1 Summary of Contributions . . . 127
6.2 Future Directions . . . 128
6.2.1 Decentralized and Anytime Decision-making Algorithms . . . 128
6.2.2 Human-robot Teaming . . . 129
6.2.3 Decision-oriented Learning for a Broader Class of Combinatorial Optimization . . . 130

Appendix A: Proofs for RMOP 131

A.1 Proof of Theorem 1 131
A.2 Proof of Theorem 2 136
A.3 Proof of Inequality in Equation (A.27) 137
A.4 Proof of Inequality in Equation (A.28) 139
A.5 Proof of Inequality in Equation (A.29) 142
A.6 Proof of Inequalities in Equation (A.31) 143

Bibliography 147

List of Tables

4.1 Coefficients for stochastic energy consumption model 79
4.2 Empirical evaluation of failure probability $\delta = 0.1$. 84
4.3 Statistical results for UAVs 93

5.1 Test results for learned models. 110
5.2 Submodular Function for Qualitative Example 123
5.3 Test Results of Objective Values 124

List of Figures

1.1 Multi-robot information gathering tasks. (a) Autonomous underwater vehicles collect ocean data (courtesy of Hollinger et al. [1]). (b) Autonomous aerial vehicles detect crops and weeds. (courtesy of Popović et al. [2]) (c) Aerial and ground robots for disaster relief (courtesy of Harbers et al. [3] and Endeavor Robotics). (d) A Mars rover and an multi-rotor vehicle for space exploration(courtesy of NASA). 2

1.2 (a) Multi-robot active target tracking with communication constraints. (b) Case study of monitoring a marine environment with aquatic robots when some of the robots may fail. (c) When the UAV and UGV are executing tasks, they need to decide when and where to rendezvous to replenish the battery of the UAV. 4

2.1 One motivating example of this paper: multi-robot active target tracking with communication constraints. The team should maximize the number of targets observed at each planning step and keep connected. Colored squares represent targets. Grey dots and lines represent an induced communication network. 10

2.2 One counter example to show that the sequential graph greedy strategy can be arbitrarily bad. If robot 1 is attracted to some local information rich area and firstly chooses an action to go downwards (dotted blue arrow), it may bias the whole team away from the global information rich area, which is on the top (other robots are forced to choose the blue arrows for connectivity). 16

2.3 One illustrative example of the proposed algorithm. (a) There are $N = 6$ robots in the team and each robot has five trajectories: go left, go right, stay, go up, and go down which corresponds to five dotted line in the work space. Current position of robots are denoted as black dots and the corresponding communication links are black lines. (b) The team greedily maximizes the submodular objective function which results in a disconnected graph i.e. robots 2, 3, 4, 6 are connected but robot 1 and 5 are disconnected from other robots. (c) Generate a complete weighted graph using greedy selections and then extract a MST from weighted graph K_6. Two blue edges are added. (d) Solve the deviation minimization problem to find the final positions for robots. The shaded circles centered at current positions are reachable sets for robots and the red graph is the network to be realized at next epoch. 19

2.4 One illustrative example to demonstrate the discretization of the reachable set. Endpoints are uniformly sampled w.r.t radius (step size $\frac{r}{3}$) and angles (step size $30°$) 22

2.5 Screenshots of the target tracking process. There are eight robots tracking moving targets. The bottom right small graph in each figure show the network topology of the team at the start of that epoch. Sub-figures (a)-(d) show how the topology evolves over time. 24
2.6 Average number of targets observed over each epoch. (a) There are 5 robots and 80 targets. (b) There are 8 robots and 140 targets. (c) There are 12 robots and 200 targets. The proposed algorithm, greedy algorithm without considering communication at each epoch, and SGG are denoted in blue, orange, and greed bars respectively. The black line above bars represent one standard deviation. . . 25

3.1 Case study of monitoring a marine environment with aquatic robots. The robots are tasked with finding informative paths to gather data. The darker the color of the path is, the more valuable the path since it gathers information from a more important region. We investigate the question of how the robots should plan their paths if we expect some of the robots to fail due to adversarial elements or natural causes. 28
3.2 One example to show that rational attackers will not always launch attacks at the first step. Four robots start from node a, and each has a budget of 4. There are $\alpha = 2$ attacks in the environment. If attackers launch all attacks at the first step, the robots can collect 35 rewards in total by planning path (a, b, c, d, e) and (a, b, c, f, g) while robots can collect at most 25 if the attackers choose to wait until they know how robots move after node c. 43
3.3 One iteration of the general MCTS approach [4]. 46
3.4 Case study of monitoring macroalgal blooms using $N = 6$ robots assuming $\alpha = 3$ failures. Colored dots indicate locations to be monitored along with their importance (i.e., rewards). Red crosses indicate worst-case attacks found using brute force. Paths returned by the proposed algorithm manage to cover one of the three important areas (lower left corner) while SGA loses all three. The background map is part of the Yellow Sea, where green tides have prevailed every summer since 2007. 52
3.5 Results for Algorithm 3 and the baseline algorithms. (a) Rewards after worst-case attack with increasing α and $N = 10$. (b) Rewards after random α attacks and $N = 10$. 54
3.6 Results for the mixed strategy and the baseline algorithms. (a) Rewards after worst-case attack with increasing α and $N = 10$. (b) Rewards after random α attacks and $N = 10$. 56
3.7 Rewards after worst-case attacks of increasing sizes, $|\mathcal{S}_{\mathcal{A}}| \leq \alpha$. Here the planner uses $N = 7$ and $\alpha = 4$. 56
3.8 Rewards after: (1) no attacks; (2) α out of N robots under worst-case attack; (3) α out of N robots under random attack. 57
3.9 Running time of Algorithm 3 and SGA 58

3.10 A tunnel map for a case study of information gathering. Colored dots with various sizes indicate locations to be visited along with their importance, i.e., rewards. (a) A tunnel map with 69 locations of interest. (b) The graph abstraction of the tunnel map. The tunnel map is from the Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge. 60
3.11 A case study of information gathering in a tunnel with $N = 4$ robots assuming $\alpha = 2$ failures/attacks. Colored dots indicate locations to be visited along with their importance (i.e., rewards). The red cross indicates the attacks found when robots and attackers play the two-player sequential game, and both use MCTS with adversaries. v_{s_i} represents the starting position of the robot i. Attackers launch the first attack when they observe that robots 0 and 3 reach node 34 and robot 2 reach node 25 at the fifth step and launch another attack later when robot 1 reaches node 25 at the seventh step. (a) robot 0 follows the path $52 \to 37 \to 2 \to 1 \to 40 \to 34 \to 31 \to 62 \to 12 \to 21 \to 10 \to 44 \to 49$. (b) robot 1 follows the path $52 \to 37 \to 2 \to 1 \to 51 \to 1 \to 40 \to 25$ and is attacked after two steps. (c) robot 2 follows the path $52 \to 37 \to 2 \to 1 \to 40 \to 25$ and is attacked after three steps. (d) robot 3 follows the path $52 \to 37 \to 2 \to 1 \to 40 \to 34 \to 31 \to 62 \to 12 \to 21 \to 10 \to 5 \to 68$. 61
3.12 Collected rewards for different strategies in four different maps: (1) M-MA: robots use one-player MCTS (M) and attackers use MCTS with adversaries (MA); (2) MA-MA: both robots and attackers use MCTS with adversaries; (3) M-R: robots use one-player MCTS (M) and attackers attacks randomly (R); (4) Ma-R: robots use MCTS with adversaries and attackers attacks randomly (R). 63
3.13 The effect of increasing the number of iterations in MCTS with $N = 4$ robots and $\alpha = 2$ failures/attacks. 63

4.1 An illustrative example of the rendezvous problem considered in this paper. When the UAV and UGV are executing tasks, they need to decide when and where to rendezvous to replenish the battery of the UAV while minimizing the travel time of the UAV and satisfying the risk constraint induced by stochastic energy consumption. When they need to rendezvous, they will deviate from their task and meet at a chosen rendezvous location. 67
4.2 First step in the rendezvous process. The UGV (blue triangle) needs to deviate from its task node to rendezvous with the UAV at the rendezvous point (pink star). The rendezvous paths are in dashed lines. 72
4.3 State transition graph in CMDP. 73
4.4 Comparison of analytical data used to derive the polynomial regression fit model of UAV power requirement at three weights and across 11 airspeeds. 81
4.5 A qualitative example illustrates how UAV and UGV rendezvous with each other under the policy π obtained by solving the CMDP. The risk tolerance is set to be $\delta = 0.1$ in this case study. (a) The input of the risk-aware rendezvous problem. (b) One sample route of UAV when it executes the policy π. 81

4.6 How different risk thresholds influence the rendezvous behaviors. UAV route time with the best range speed is 4354 s, and the route distance is 61.0 km. (a) UAVs are very risk-averse to failures with a risk threshold equal to 0.01. (b) UAV is less risk-averse to failures with a risk threshold equal to 0.2. (c) UAV is neutral to the failures with a risk threshold equal to 0.5. 82
4.7 Results comparisons for the CMDP and greedy policies with $\delta = 0.1$. The dashed black line represents the task duration if the UAV moves with the best range speed without considering battery limitation. 83
4.8 Risk tolerance vs task duration Pareto Curve. 84
4.9 Recharging detour example. The UAV 1 leaves its route at the node a_1^1 to rendezvous with UGV 1 at the node g_1^3. The recharging occurs when they reach the node g_1^4 and the UAV moves to its next task node a_1^2. Note that if $d = 1$, no other UAV can recharge on this UGV between g_1^3 and g_1^4. 88
4.10 An example to show how to construct a bipartite graph for one planning horizon from the UAV and UGV route segments. 89
4.11 A qualitative example to illustrate how UAV and UGV rendezvous with each other solving the RRRP. The risk tolerance is set to be $\rho = 0.1$ in this case study. Subscriptions s and t denote the start and the terminal of the recharging process. (a) The input of the RRRP problem includes the UAV and UGV tasks and the road network. (b) One sample tour of UAV 1 when it persistently monitors the route. (c) One sample history of SOC for UAV 1. 92
4.12 Quantitative results. (a) Comparisons of the RRRP scheduling and the greedy strategies with $\rho = 0.1$. 92

5.1 Decision-Oriented Learning framework. The training loss is defined after the downstream task. 98
5.2 An illustrative example for vehicle routing problems. 98
5.3 An illustrative example to show the misalignment between the prediction model that achieves high predictive accuracy and the one that results in good decisions. 100
5.4 Per epoch loss curve of DOL. 110
5.5 The motivating case study of Intervention-aware UGVs routing. 112
5.6 The proposed framework that incorporates the non-monotone submodular maximization into the learning process. 112
5.7 Computational graph of the proposed differentiable algorithm. (a) The structure of the algorithm. (b) The internal structure of the differentiable cost-scaled greedy operation. 118
5.8 Simulation results for the network (D-CSG). 123
5.9 A case study with three candidate routes and two UGVs. (a) Application scenario. (b) Ground truth data and the optimal decision boundary. (c) Learned linear models using MSE loss. (d) Learned linear models using the DOL framework. 124
5.10 Per epoch loss curve of DOL. 125

A.1 Venn diagram, where $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_1^*, \mathcal{S}_2^*$ are defined as follows: Per run of proposed algorithm for RMOP, $\mathcal{S}_1$ and $\mathcal{S}_2$ are intermediate results such that $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, and $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. Let $\mathcal{S}_{\mathcal{A}}^*(\mathcal{S})$ be the optimal removal from $\mathcal{S}$. Then $\mathcal{S}_1^*, \mathcal{S}_2^*$ are defined such that $\mathcal{S}_1^* = \mathcal{A}^*(\mathcal{S}) \cap \mathcal{S}_1$ and $\mathcal{S}_2^* = \mathcal{A}^*(\mathcal{S}) \cap \mathcal{S}_2$. By definition, $\mathcal{S}_1^* \cap \mathcal{S}_2^* = \emptyset$ and $\mathcal{S}_{\mathcal{A}}^*(\mathcal{S}) = \mathcal{S}_1^* \cup \mathcal{S}_2^*$. 138

Chapter 1: Introduction

1.1 Motivation

Research in science and engineering is experiencing a paradigm shift from physics-based modelling to data-driven learning. The first step in data-driven learning is to collect the data or gather information. Access to high-quality data can be a major enabler to high-quality learning. For example, in the computer vision community, the large-scale dataset ImageNet [5] released in 2009 was critical to the success and popularity of deep neural networks. There are other scientific and engineering disciplines where access to data can be a significant enabler for data-driven learning. In this dissertation, our focus is on developing algorithms for a robotic system that can autonomously collect data and gather information for scientific and engineering studies that involve monitoring spatio-temporal process. Typically, data collection in the physical world has been limited to manual data collection or passive sensors deployed in the field [6–10]. However, this can be limiting for data-driven learning. Instead, using actively controlled robots, we can collect high-resolution data on-demand and adaptively making data-driven learning more efficient. Recently, there has been work on using robots for information gathering in various high-impact applications such as

- Environmental monitoring: autonomous robots are used to measure physical quantities

such as the temperature, pollution concentration, and water quality. (Fig. 1.1a)

- Agriculture and terrain mapping: autonomous robots are used to detect weeds and pests, count the number of fruits, and spread chemicals. (Fig. 1.1b)

- Search, rescue, and surveillance: autonomous robots are used to search and/or track targets in hazardous environment. (Fig. 1.1c)

- Space exploration: autonomous robots are used to collect samples of rocks and soils to seek preserved signs of biosignatures in on the planet. (Fig. 1.1d)

Figure 1.1: Multi-robot information gathering tasks. (a) Autonomous underwater vehicles collect ocean data (courtesy of Hollinger et al. [1]). (b) Autonomous aerial vehicles detect crops and weeds. (courtesy of Popović et al. [2]) (c) Aerial and ground robots for disaster relief (courtesy of Harbers et al. [3] and Endeavor Robotics). (d) A Mars rover and an multi-rotor vehicle for space exploration(courtesy of NASA).

Due to the advance in mobility and sensing capabilities over the past decades, the cost effective robotic platforms are widely available for information gathering research (e.g., Boston Dynamics [11], DJI [12], Clearpath [13], Unitree [14]). It is becoming increasingly popular to use a team of robots for information gathering tasks. Within a team, the robots may differ in

their mobility (e.g., aerial, ground, or surface robots), or in sensing capabilities (e.g., LIDAR or cameras). Compared to a single robot, such a team can extend the range of executable tasks, enhance task performance, and naturally provide redundancy for the system. To fully exploit the potential of such multi-robot teams, we need to consider how to coordinate robots to gather information effectively given the available knowledge about the environment and the capability constraints in robots.

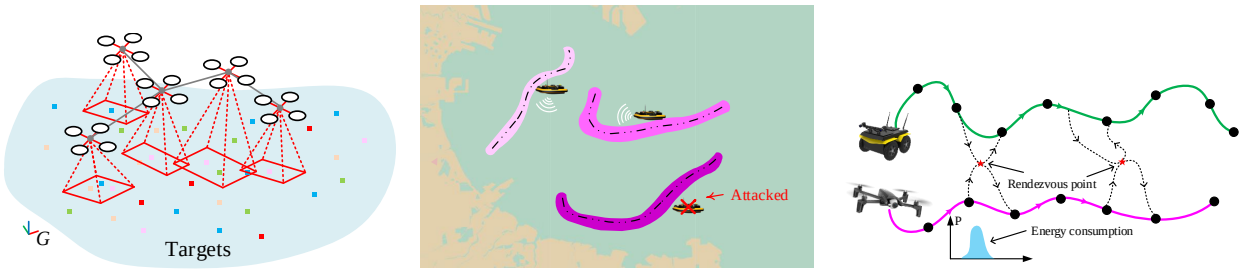
Multi-robot information gathering problems are challenging to solve mainly for the following two reasons. First, we have only limited knowledge about the environment in the planning phase. For example, in some applications, we have a parametric model of the environment based on the domain knowledge, but the exact value of the parameters of the model are not known in advance. The parameters may be set-valued, stochastic, or context-dependent.[1] We have to make decisions with incomplete knowledge. Second, the underlying combinatorial optimization may be NP-hard and we cannot find the optimal solution in polynomial time. On the one hand, the lack of knowledge of the environment will increase the size of the solution space. On the other hand, the constraints imposed by the capabilities of the robots, e.g., communication radius and battery capacity, will increase the complexity of the problem.

Therefore, the focus of this dissertation is to design algorithms to enable multi-robot teams to gather spatiotemporal information for diverse environments and tasks. Our research spans the whole spectrum of the knowledge of the environmental model: from one extreme where the environmental model is fully known to another extreme where the environmental model is unknown but can be learned from empirical data. When we have full or partial knowledge of the environ-

[1]Context refers to the side information of the environment obtained through either extra sensing modality or expert annotations.

mental model, given task requirements, we formulate coordination problems using tools such as submodular maximization and sequential decision-making to optimize the task performance with capability constraints of robots and aim at finding efficient (e.g., polynomial in time) and effective (e.g., optimal or bounded with guarantee) algorithms to solve the identified problems. When only empirical data is available for the environment, we study novel learning-decision-integrated algorithms to learn environmental models that lead to good downstream task decisions.

1.2 Overview of the Dissertation



(a) (b) (c)

Figure 1.2: (a) Multi-robot active target tracking with communication constraints. (b) Case study of monitoring a marine environment with aquatic robots when some of the robots may fail. (c) When the UAV and UGV are executing tasks, they need to decide when and where to rendezvous to replenish the battery of the UAV.

We start our research by considering the cases where the environmental model is fully known, and the robotic teams need to handle the constraints imposed by the connectivity when they gather information. Then, we consider the cases where the environmental model is partially known and uncertainty exists. When the uncertainty is upper-bounded, we introduce two novel robust combinatorial optimization-based formulations of information gathering problems (RMOP offline and RMOP online) and propose a constant-factor approximation algorithm for RMOP offline. These two cases are all rooted in the submodular maximization framework. For-

mally, for a set $\mathcal{V}$, a function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is submodular if and only if for any sets $\mathcal{A} \subseteq \mathcal{V}$ and $\mathcal{A}' \subseteq \mathcal{V}$, we have $f(\mathcal{A}) + f(\mathcal{A}') \geq f(\mathcal{A} \cup \mathcal{A}') + f(\mathcal{A} \cap \mathcal{A}')$ [15]. Such a submodular objective is ubiquitous in multi-robot information gathering. For example, the set $\mathcal{V}$ may refer to the set of motion primitives and $f(\mathcal{A})$ refers to the targets covered by the camera of robots.

When the uncertainty is stochastic with a known distribution, we utilize risk-aware sequential decision-making paradigms to coordinate a heterogeneous team. Moreover, when the environmental model is unknown, and only empirical data is available, we study the problem of how to learn an environmental model and propose a learning-decision-integrated framework. In addition to algorithm design, we present empirical results for applications such as active target tracking, ocean monitoring, and persistent monitoring as shown in Fig. 1.2. Details of the main contributions of this dissertation are given below.

In Chapter 2, we introduce a problem named Communication-aware Submodular Maximization (CSM) for a class of multi-robot information gathering problems that have submodular objectives and require *intermittent connectivity* between robots. Such a formulation is suitable for scenarios in which we have the full knowledge of the environment. In the proposed formulation, each robot needs to find one trajectory to be executed within the current planning epoch to maximize the submodular team objective. We also impose the constraint that the robots need to form a connected communication network at the end of the planning epoch. That is, the end positions of all the selected trajectories must form a connected network. We allow the robots to disconnect during the epoch temporarily. However, by ensuring connectivity at the end of the epoch, the robots will be able to exchange information gathered during the epoch and jointly plan for the next epoch. We propose a heuristic algorithm consisting of two stages, *topology generation* and *deviation minimization*, to solve CSM. Chapter 2 is based on our work [16] published in

2021 IEEE International Conference on Robotics and Automation (ICRA) and its journal version, which is in preparation for Autonomous Robots (AURO).

In Chapter 3, we consider the multi-robot information gathering problem in uncertain environments where some of the robots may fail during execution, and we have only *partial knowledge about the failures*: there are at most α failures. Once a robot fails, all the information collected as well as the robot will be lost. We focus on how to make the robot team robust to failures when they operate in such environments. We introduce the Robust Multiple-path Orienteering Problem (RMOP) where we seek worst-case guarantees against an adversary that is capable of attacking at most α robots. We consider two versions of this problem: RMOP offline and RMOP online. In the offline version, there is no communication or replanning when robots execute their plans, and our main contribution is a general approximation scheme with a bounded approximation guarantee that depends on α and the approximation factor for single robot orienteering. In particular, we show that the algorithm yields a (i) constant-factor approximation when the cost function is modular, (ii) $\log$ factor approximation when the cost function is submodular, and (iii) constant-factor approximation when the cost function is submodular but the robots are allowed to exceed their path budgets by a bounded amount. In the online version, RMOP is modeled as a two-player sequential game and solved adaptively in a receding horizon fashion based on Monte Carlo Tree Search (MCTS). Chapter 3 is based on our work published in 2020 Robotics: Science and Systems (RSS) [17] and its journal version published in IEEE Transaction on Robotics (TRO) [18].

In Chapter 4, we consider the multi-robot long-term information gathering problems in uncertain environments in which the Unmanned Aerial Vehicle (UAV) may be out of charge stochastically due to the disturbances in the environment, and we have only *partial knowledge*

about the disturbances: the probability distribution. If we consider only the worst-case disturbance, it will result in very conservative decisions. If we consider the average of the disturbance, the empirical failure (UAV is out of charge) rate may be high. Therefore, we need to make a tradeoff between the task performance and the failure rate. Specifically, the UAVs and Unmanned Ground Vehicle (UGV) need to finish the task cooperatively, and the UGVs need to recharge the UAVs as mobile rechargers periodically. Due to the disturbances (e.g., wind) in the environment, the energy consumption of a UAV is stochastic, and we need to take such disturbances into account in the planning phase. Generally, we are interested in such problems: *given the distribution of the stochastic disturbances in the environment, how to coordinate UAVs and UGVs to achieve high task performance and low failure rate (UAVs being out of charge)?* We study this type of problem for a team consisting of one UAV and one UGV in Section 4.2 and introduce a Chance Constrained Markov Decision Process (CCMDP) based formulation for risk-aware aerial-ground cooperative routing. We extend the formulation to multiple UAVs and UGVs in Section 4.3 based on a graph-matching model. Chapter 4 is based on our work published at the 2022 IEEE Conference on Decision and Control (CDC) [19] and the work published at the 2023 IEEE International Conference on Robotics and Automation (ICRA) [20].

In Chapter 5, we consider the multi-robot information gathering problem for the case where the environment *model is unknown and context-dependent*. Given empirical data, our focus is to learn context-aware environmental models for decision-making problems. Classically, such a learning process is independently conducted without considering the downstream problems. In contrast, we propose a decision-oriented framework that incorporates the downstream decision-making problem into the learning process. We show that such integration will help reduce the misalignment between the prediction model and the downstream task. The misalignment refers to

a predictor that despite achieving high predictive accuracy in the learning phase may not necessarily result in good decisions in the downstream task. The challenge to achieving such integration is to make the combinatorial optimization differentiable, which can then be treated as a differentiable module in the learning process. To make the combinatorial optimization differentiable, the general idea is to find continuous relaxations of the discrete problem. We study how to make two types of decision-making problems for the multi-robot gathering differentiable: monotone submodular maximization (Section 5.2), and non-monotone submodular maximization(Section 5.3). Chapter 5 is based on our work published at the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2023) [21] and the work submitted to the 2024 American Control Conference (ACC) [22].

Chapter 2: Multi-Robot Information Gathering with Full Knowledge of the environmental model

2.1 Overview

In this chapter, we consider the multi-robot information gathering problem when the environment is fully known. In such cases, the problem can be formulated as one classic combinatorial optimization problem with some application-specific constraints. Specifically, we consider the case where the objective to be maximized is a submodular function, and the robot team has to satisfy the intermittent connectivity constraint, which forces the team to form a connected network periodically. Such an intermittent connection is beneficial because it allows robots to exchange collected information and plan for the next epoch.

Many multi-robot cooperative tasks such as exploration and target tracking can be formulated as submodular maximization problems [23–27]. The objectives in such problems (e.g., mutual information, area explored, number of targets tracked) have diminishing returns property which is shown to be submodular. Intuitively, submodularity formalizes the notion that adding more robots to a larger multi-robot team cannot yield a larger marginal gain in the objective than adding the same robot to a smaller team. Despite the submodular maximization problem being NP-hard, a simple greedy approximation algorithm can achieve nearly optimal

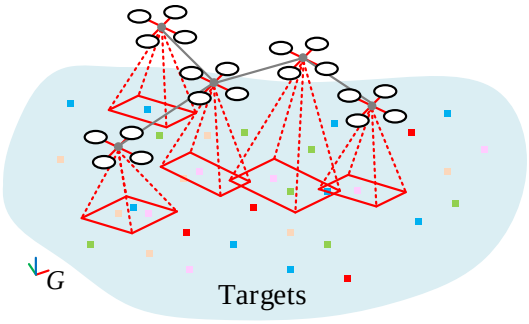


Figure 2.1: One motivating example of this paper: multi-robot active target tracking with communication constraints. The team should maximize the number of targets observed at each planning step and keep connected. Colored squares represent targets. Grey dots and lines represent an induced communication network.

performance [15, 28].

Communication plays a key role in successfully executing the greedy algorithm for a team of robots to choose actions. In the centralized case, robots may need to transmit their acquired information to either a leader of the team or a remote server; in the distributed case, robots may need to exchange local information with their neighbors to reach some global consensus [29, 30]. In either case, a team of robots needs to form a connected communication network to ensure information flow. However, it is possible that actions that maximize the objective may lead the network to become disconnected. In fact, we expect that to happen quite often given that many of these objectives relate to coverage where the robots may want to move away from each other to reduce overlaps. Therefore, there is a need to introduce a connectivity constraint during decision-making. However, in the existing literature related to submodular maximization, communication is usually neglected, which motivates us to think about whether we can jointly consider communication maintenance and submodular maximization in the decision-making process for the team.

To this end, we introduce a problem named Communication-aware Submodular Maximiza-

tion (CSM) for a class of multi-robot task planning. In the proposed formulation, each robot needs to find one trajectory to be executed within the current planning epoch to maximize the submodular team objective. We also impose the constraint that the robots form a connected communication network at the end of the planning epoch. That is the end positions of each of the trajectories that must form a connected network. We allow the robots to disconnect during the epoch temporarily. However, by ensuring connectivity at the end of the epoch, the robots will be able to exchange information gained during the epoch and jointly plan for the next epoch.

One motivating example is given in Fig. 2.1, in which a team of aerial robots with downward-facing cameras tracks targets on the ground. The objective is to maximize the number the targets observed within each epoch. On the one hand, robots need to move away from each other to reduce the overlap of the sensor footprints. On the other hand, they cannot move too far from each other while still ensuring a connected network at the end. A good solution for CSM is able to balance these two conflicting goals. In this paper, we propose a heuristic algorithm consisting of two stages, *topology generation* and *deviation minimization*, to solve CSM. The key idea of the proposed algorithm is that first, for each robot we discretize the problem by generating a set of candidate trajectories whose endpoints are within the reachable set of the robot and let robots choose trajectories greedily without considering communication constraints; then we make them minimally deviate from the endpoints corresponding to greedy selections to build connectivity. Specifically, in the *topology generation* stage, an edge-weighted graph is generated using robots' greedy selections, whose edge weights are defined over the distance between pairs of robots. Then a Minimum Spanning Tree (MST) of the edge-weighted graph is extracted as the communication graph for the next epoch. In the *deviation minimization* stage, a Quadratic Programming (QP) is formulated to find new positions within the reachable set that minimally deviate from

the greedy selections for robots to realize the MST in the workspace. We carry out extensive simulations to evaluate the performance of the proposed algorithm.

Submodular maximization and its variants have been widely used in multi-robot decision-making problems including coverage [31–33], target tracking [23, 34–36], exploration [24], and information gathering [17, 26, 28, 37, 38]. These studies all based on the fact that greedy algorithm and its variants can solve submodular maximization problem its variants efficiently with provable performance guarantee. However, communication is seldom considered in the existing works of robotic researchers, and most of them assume that a connected network is always there. Krause [38] uses the same assumption but associates some cost to edges of the network and jointly optimizes coverage and network costs. But their work is static in nature, and the decision is made only once instead of at each planning epoch without partition matroid constraint [37]. By contrast, we consider the scenario where robots are moving, and decisions need to be made at each planning epoch to maximize objective and guarantee a connected network. Gharesifard [39] considers the submodular maximization case where one decision-maker represented as one vertex in the graph can access only the decisions of its neighbor and analyzes the influence of graph topology. [40] considers a similar problem but mainly from the perspective of a system designer. However, the graph discussed in [39, 40] is more like a relational graph instead of a communication graph, and authors all assume that the actions or decisions of vertices will not change the graph properties, i.e., connectivity. Grimsman [25] considers submodular maximization under topology constraints for multi-robot task allocation problems. If the topology constraints can be described as matroid constraints, a greedy algorithm can have a constant approximation factor. However, connectivity constraint is in general not a matroid constraint for robot teams with the same communication radius, and authors in [25] make extra assumptions on task allocation

structures to make spanning tree constraint a matroid constraint, which makes it unsuitable for our problem.

Another line of related work is on connectivity maintenance based on algebraic graph theory [41–44]. In these studies, connectivity maintenance and tasks are usually separately considered. Even though connectivity can be formally guaranteed when the task is not considered, complex task behaviors of robots in practice can break the connectivity. Our work is different from these works in two aspects: first, our formulation is rooted in combinatorial optimization, i.e., submodular maximization rather than continuous optimization, and aims at solving discrete decision-making problems. Even though submodular set functions can be extended to continuous functions [45], the evaluation of the extended function involves exponentially many subsets of the ground set, which makes it unsuitable for robotic application. Second, we jointly consider team task and communication connectivity; therefore, task behaviors will not break connectivity.

This work is also closely related to submodular maximization over graphs [46–48]. [48] considers the problem of finding a rooted arborescence in a directed graph with a budget to maximize a submodular function while by contrast graph considered in this paper is undirected, and the budget is not a constraint. In [46], authors consider the problem of finding a connected subgraph to maximize the covered area, which is a special submodular set function, in the Euclidean plane and give one 2-approximation the algorithm. But their algorithm cannot be used to maximize general submodular functions that are interesting to multi-robot applications, for example, mutual information [24, 27] and the number of targets [23] because the performance of their algorithm relies on some additive properties of covered areas. [47] considers a similar problem as with [46] for general submodular functions. However, none of the algorithms proposed in [46–48] can deal with partition matroid constraints [24], which is one inherent nature in

multi-robot application since each robot has its unique set of choices. As a result, corresponding algorithms are not applicable to multi-robot applications.

2.2 Preliminaries

We first introduce some notations we will use in this section. Given a set $\mathcal{A}$, $2^{\mathcal{A}}$ denotes the power set of $\mathcal{A}$. Given another set $\mathcal{B}$, the set $\mathcal{A} \setminus \mathcal{B}$ denotes the set of elements in $\mathcal{A}$ but not in $\mathcal{B}$. Given a set $\mathcal{V}$, a set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$, and an element $v \in \mathcal{V}$, $f(v)$ is a shorthand that denotes $f(\{v\})$. We use $\Delta_f(r \mid \mathcal{A}) = f(\mathcal{A} \cup \{r\}) - f(r)$ to denote the marginal gain of adding r in $\mathcal{A}$. Whenever f is clear from the context, we will use shorthand $\Delta(r \mid \mathcal{A})$ for $\Delta_f(r \mid \mathcal{A})$. $[N]$ denotes the set $\{1, 2, \ldots, N\}$.

Definition 1 (Minimum Spanning Tree)**.** A minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

Definition 2 (Minimum Bottleneck Spanning Tree)**.** A minimum bottleneck spanning tree (MBST) in an undirected graph is a spanning tree with the most expensive edge as cheap as possible.

Suppose we have $N \geq 2$ robots in the environment, whose joint states at the start of the current epoch are denoted as $\mathbf{x} = \{x_1, x_2, \ldots, x_N\} \in \mathbb{R}^{dN}$, where d is the dimensionality and can be 2 or 3. Communication graph at the start of the current epoch $G = (V, E)$ is specified using proximity graph, i.e., a robot corresponds to a vertex $i \in V$ and $(i, j) \in E$ if $\|x_i - x_j\| \leq r_c$, where r_c is the communication radius.

Each robot is equipped with a sensor that gathers information based on the sensor footprint. For example, an aerial robot equipped with a downward-facing camera can sense the targets on

the ground as shown in Fig. 2.1. In this paper, specifically, our goal is to find a set of trajectories, one per robot, to ensure that we maximize the submodular team objective within each epoch and ensure a connected network at the end of each epoch.

Let $\mathcal{R}_i$ be the reachable set of robot i of the current epoch, i.e., a set of locations that robot i can reach at the end of the current epoch from the start position of the current epoch. In general, there is no closed-form expression for the reachable set. In this paper, we assume that the reachable set can be approximately represented as obstacle-free convex polyhedrons or ellipsoid using an iterative optimization method such as the one presented in [49]. Let $\mathcal{T}_i$ be the set of trajectories of robot i each of which is within the reachable set $\mathcal{R}_i$. It should be noted that there are infinitely many elements in $\mathcal{T}_i$. Let $\mathcal{P}$ be the operator that can extract the endpoints of trajectories.

2.3 Problem Formulation

With the above notations, the problem can be formulated as follows.

Problem 1 (Communication-aware Submodular Maximization)**.** *At current planning epoch, robots' positions induce a connected communication graph G_e, we want to preserve the connectivity in the next epoch, i.e., induced graph G_{e+1} after moving to new positions is connected, meanwhile maximizing a submodular objective function. Mathematically,*

$$\begin{aligned} &\max_{s_i \in \mathcal{T}_i, i \in [N]} f(\mathcal{S}) \\ &s.t.\ \ G_{e+1}(\mathcal{P}(\mathcal{S})) \text{ is connected,} \end{aligned} \tag{2.1}$$

where $\mathcal{S} = \{s_1, \ldots, s_N\}$ and $\mathcal{T}_i$ is the set of trajectories for robot i.

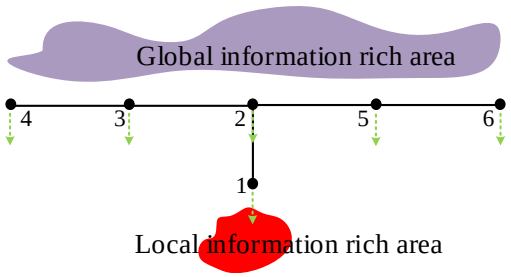


Figure 2.2: One counter example to show that the sequential graph greedy strategy can be arbitrarily bad. If robot 1 is attracted to some local information rich area and firstly chooses an action to go downwards (dotted blue arrow), it may bias the whole team away from the global information rich area, which is on the top (other robots are forced to choose the blue arrows for connectivity).

One way to solve Problem 1 is by first discretizing the reachable set $\mathcal{R}_i$ into a set of discrete locations and by using lower-level motion primitives such that each point in the discretized reachability set is associated with one dynamically feasible trajectory [50].

Even after discretization, Problem 1 is still hard because the graph topology can be quite diverse and there can be exponentially many candidates. However, it should be noted that with proper discretization there is always a solution to Problem 1: since communication graph G_e is connected and all robots have non-empty reachable sets, one apparently feasible solution is that all robots do the same movement, e.g., the team as a whole shifts to a new position and G_{e+1} has the same graph topology as that of G_e.

One naive way to solve Problem 1 is to use a sequential greedy strategy and consider connectivity during the construction process as shown in Algorithm 1, in which at each iteration one robot that can connect to the existing connected graph will make decisions and the connected graph will then be correspondingly expanded. However, such a strategy can be arbitrarily bad as shown in Fig. 2.2. In the Sequential Graph Greedy (SGG) strategy, robot 1 may choose the action to go down first since it has the largest marginal gain w.r.t. empty graph, but his action will bias the team from more valuable area on the top. Another problem with such a strategy is that if

robots have different reachable sets, SGG may end up with a disconnected graph. For example, in Fig. 2.2, if robot 1 has a large reachable set and move downwards to its best, other robots with a smaller reachable set may not be able to follow its pace to maintain the connectivity.

Algorithm 1: Sequential Graph Greedy

function SGG($f, \{\mathcal{T}_i\}_{i=1}^N, \mathcal{P}$)

Input :

- A monotone submodular function f
- Partitioned ground set $\{\mathcal{T}_i\}_{i=1}^N$
- Operator $\mathcal{P}$ that can extract the end points of trajectories.

Output: A subset Sol of the ground set

$Sol \leftarrow \emptyset$, $\mathcal{T} \leftarrow \bigcup_{i=1}^N \mathcal{T}_i$, $G \leftarrow (V = \emptyset, E = \emptyset)$

while $|Sol| < N$ **do**

  find the trajectory set $\mathcal{T}' \subseteq \mathcal{T}$ s.t. each element in $\mathcal{T}'$ can establish connections to the existing graph G at its endpoint

  # find the element with largest marginal gain and its group ID

  $a,\ i = \text{argmax}_{s \in \mathcal{T}'} \Delta f(s \mid Sol)$

  $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{T}_i$, $Sol \leftarrow Sol \cup \{s\}$

  G.add_node($\mathcal{P}(s)$), G.add_edge($\mathcal{P}(s)$)

end

return Sol

2.4 Heuristic Algorithm

We propose a two-stage heuristic algorithm to solve Problem 1. The key idea is to first let robots greedily select trajectories, which of course may break the connectivity, and then make the robots minimally deviate from greedy selections to establish connectivity.

Topology Generation: In this stage, robots will greedily select trajectories based on marginal gains without considering the communication constraint. It is equivalent to solving a submodular maximization problem with partition matroid using a greedy strategy as shown in Algorithm 2. These greedy selections will virtually drive them to positions $x_1^g, \ldots, x_N^g$. Then we generate a

Algorithm 2: Submodular Maximization with Partition Matroid

function Greedy($f, \{\mathcal{T}_i\}_{i=1}^{N}, \mathcal{P}$)

Input :

- A monotone submodular function f
- Partitioned ground set $\{\mathcal{T}_i\}_{i=1}^{K}$
- Operator $\mathcal{P}$ that can extract the end points of trajectories

Output: A subset Sol of the ground set

$Sol \leftarrow \emptyset$, $\mathcal{T} \leftarrow \bigcup_{i=1}^{N} \mathcal{T}_i$

while $|Sol| < N$ **do**

  # find the element with largest marginal gain and its ID

  $s,\ i = \text{argmax}_{s \in \mathcal{T}} \Delta(s \mid Sol)$

  $\mathcal{T} \leftarrow \mathcal{T} \setminus \mathcal{T}_i$

  $Sol \leftarrow Sol \cup \{s\}$

end

return Sol

complete graph K_N corresponding to these N robots. The edge weight between robots i and j is defined as $C(i,j) = \max(\frac{1}{2}(\|x_i - x_j\| - r_c), 0)$. If the distance between two robots is already within the communication radius, then the cost of that edge will be zero. If the distance is greater than r_c, then at least one of the two robots needs to move at least $\frac{1}{2}(\|x_i - x_j\| - r_c)$ to realize the edge.

After defining the K_N, we use MST algorithm [51] to extract a connected graph for the next epoch. The intuition for using MST to generate graph topology is: first, the costs of edges are defined over distances to be traveled to realize edges. Therefore, the graph returned by MST algorithm to some extent reflects the total deviation to achieve connectivity; second, as shown in [51], MST is also a MBST, which suggests that the graph returned by MST algorithm to some extent also reflects the minimum distance to be deviated to achieve connectivity.

Deviation Minimization: After generating the graph topology, in this stage we consider the problem of how to make robots minimally deviate from the greedy selections to establish

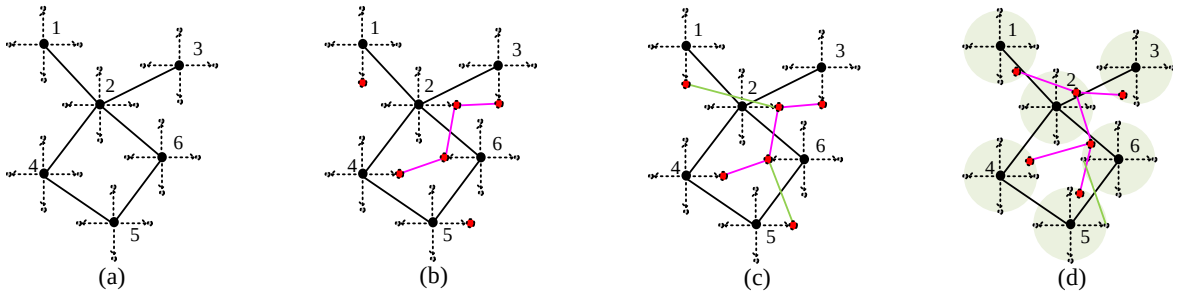


Figure 2.3: One illustrative example of the proposed algorithm. (a) There are $N = 6$ robots in the team and each robot has five trajectories: go left, go right, stay, go up, and go down which corresponds to five dotted line in the work space. Current position of robots are denoted as black dots and the corresponding communication links are black lines. (b) The team greedily maximizes the submodular objective function which results in a disconnected graph i.e. robots 2, 3, 4, 6 are connected but robot 1 and 5 are disconnected from other robots. (c) Generate a complete weighted graph using greedy selections and then extract a MST from weighted graph K_6. Two blue edges are added. (d) Solve the deviation minimization problem to find the final positions for robots. The shaded circles centered at current positions are reachable sets for robots and the red graph is the network to be realized at next epoch.

connectivity. The problem is described below.

Problem 2 (Deviation Minimization)**.** *Given current positions $x_1, \ldots, x_N$ of robots, positions $x_1^g, \ldots, x_N^g$ corresponding to greedy selection, reachable sets $\mathcal{R}_1, \ldots, \mathcal{R}_N$, communication radius r_c, and a connected graph G_{e+1} to be realized, the goal is find new locations $x_1^*, \ldots, x_N^*$ such that*

- *$x_1^*, \ldots, x_N^*$ can realize all edges in graph G_T, which is returned by MST algorithm;*
- *The deviations of robots from greedily selected positions $x_1^g, \ldots, x_N^g$ are minimized.*
- *$x_1^*, \ldots, x_n^*$ are within reachable set w.r.t their current positions.*
- *Robots will not collide in the new positions.*

Mathematically,

$$\begin{aligned} \min & \sum_{i=1}^{N} w_i \left\| x_i^* - x_i^g \right\| \\ & s.t. \left\| x_i^* - x_j^* \right\| \leq r_c, (i,j) \in G_T.edges \\ & x_i^* \in \mathcal{R}_i, i \in [N] \\ & \left\| x_i^* - x_j^* \right\| > r_s, \forall i \neq j, \end{aligned} \tag{2.2}$$

where $w_i \geq 0$ is parameter used to describe robot i's willingness to deviate from x_i^g and more details on choosing w_i are given in Sec. 2.5 and r_s is the safe radius for each robot.

It should be noted that the safety constraints are not convex but they can be well handled by available solves like Gurobi 9.0 [52].

One illustrative example is shown in Fig. 2.3 to demonstrate how our proposed algorithm works. In the formulation given above, the reachable set $\mathcal{R}_i$ can be an arbitrary set. We also have not defined w_i which can be chosen based on the application. In the following, we will describe one specific example on active target tracking in which we assume that the reachable set can be represented as a circle and present three ways to select w_i based on the number of targets that a robot can track during one epoch.

2.5 Case Study: Active Target Tracking

In this section, we present one case study on multi-robot active target tracking with communication constraints, in which each robot has a downward-facing camera and the team aims to maximize the number of targets that will be observed in each planning epoch. This case study is presented to demonstrate:

- Correctness: The proposed algorithm will return a solution that preserves connectivity. This will be validated by checking the network generated by the team after taking action.

- Performance: The proposed algorithm outperforms SGG strategy and has a competitive performance compared with a simple greedy strategy, which can be viewed as the empirical performance upper bound for the team at each planning epoch. We compare our algorithm with these two algorithms with respect to the number of targets tracked.

  Target Model: We assume each target has single integrator motion model,

$$p^j(t+1) = p^j(t) + v^j(t),$$

where $p^j(t)$ and $v^j(t)$ denote the position and the velocity of target $j = 1, \ldots, 50$. The robot obtains noisy measurements of the targets' positions and use a Kalman filter for estimation. The noise is set to Gaussian noise with zero mean and 0.5 standard deviation. Each target's velocity is initialized to be zero and is updated by using two consecutive measurements and the time interval of these two measures:

$$v^j(t') = \frac{\tilde{p}^j(t') - \tilde{p}^j(t)}{t' - t},$$

where $\tilde{p}^j(t')$ and $\tilde{p}^j(t)$ are two measures at time step t' and t with $t' > t$.

Robot Model: We assume that all aerial robots fly at fixed heights and the reachable set of each robot can be described as a circle centered at its current position with a radius 4 meters. Then the reachable set is discretized w.r.t. both the radius and the angle as shown in Fig. 2.4 to generate trajectory set. Each robot is equipped with a downward-facing camera and is able to observe ground targets inside the sensor footprints. We conduct 10 rounds of simulation and in

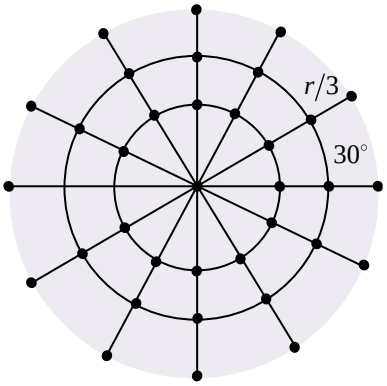


Figure 2.4: One illustrative example to demonstrate the discretization of the reachable set. Endpoints are uniformly sampled w.r.t radius (step size $\frac{r}{3}$) and angles (step size 30°)

.

each simulation there are 10 epochs. Communication radius is set to be 10 meters.

Weight Selection: in this paper we test two ways to set parameter w_i, which is about robot i's willingness to deviate from its endpoints, for Problem 2.

1. w_i is set to be the individual gain of each robot, i.e.,

$$w_i = f(s_i),$$

where s_i is the trajectory for robot i selected using greedy algorithm. We will refer this approach as *Weight1* in the fololwing. The intuition here is that if a robot itself alone can observe many targets, it is less willing to deviate from its selection.

2. w_i is set to be marginal gain of robot i's selected trajectory $s_i \in \mathcal{T}_i$, i.e.,

$$w_i = f(\mathcal{S}) - f(\mathcal{S} \setminus s_i),$$

where $\mathcal{S}$ is a set of trajectories selected using greedy strategy. We will refer this approach

as *Weight2* in the following. The intuition here is that the willingness of a robot to deviate from the greedy selection is reflected in its contribution to the whole team. The more it contributes to the team, i.e., large marginal gain, the less willing it is to deviate from its selection.

3. Another way to set w_i is to consider the variation of objective values around the point x_i^g. Specifically, given the position x_i^g of robot i corresponding to its greedy selection, we uniformly sample a set $\{x_1^i, \ldots, x_W^i \mid \left\|x_j^i - x_i^g\right\| \leq 1, j = 1, \ldots, W\}$ and define the weight for robot i as

$$w_i = f(x_i^g) - \min_{x \in \{x_1^i, \ldots, x_W^i\}} f(x).$$

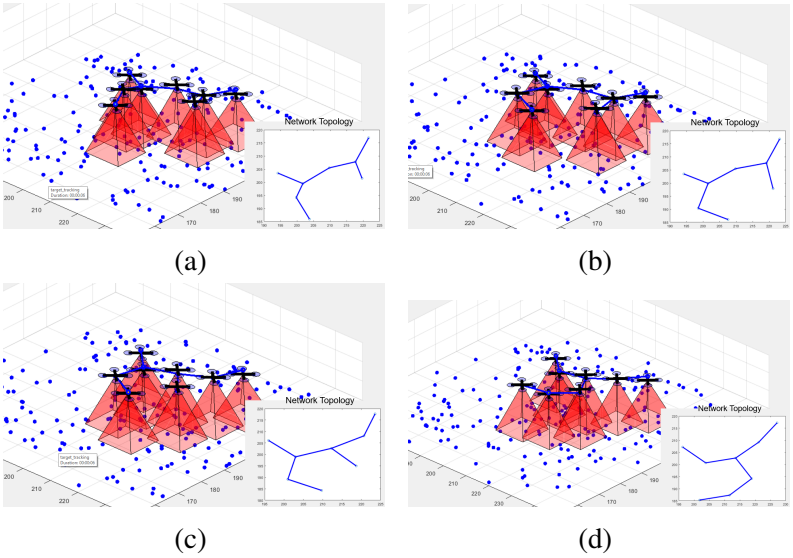
We will refer this approach as *Weight3* in the following. The intuition here is that a robot will check the drop of the number of targets observed if he deviates from its selection. If the drop is large, then it is less willing to deviate from its selection.

It should be noted that after setting w_i to be some non-negative constant, Problem 2 becomes a standard QP problem and can be solved using commercial solvers such as Gurobi.

2.5.0.1 Simulation Results

All experiments were performed on a Windows 64-bit laptop with 16 GB RAM and an 8-core Intel i5-8250U 1.6GHz CPU using MATLAB with Gurobi 9.0 [52], which can deal with both convex and non-convex quadratic constraints. We test three algorithms in this section: the proposed algorithm, the greedy algorithm without considering communication constraints, and SGG. Here the greedy algorithm corresponds to the greedy selection at the beginning of each

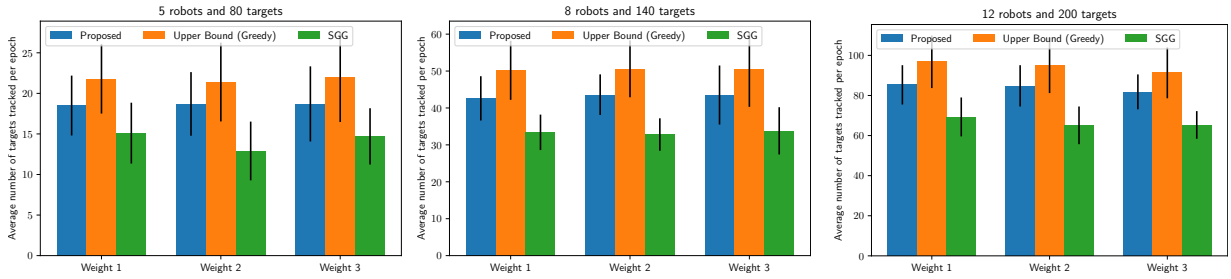
epoch without considering communication constraints, which can be viewed as an empirical upper bound of the team performance at each epoch. We compare with such results to show that the deviation minimization part of the proposed algorithm will only slightly reduce the performance of the team from the upper bound. The data on SGG is collected in the following way: we initialize the robots and targets in the same positions as that in the test setup of the proposed algorithm and run SGG to track targets. We use the results of SGG as a baseline.



(a) (b) (c) (d)

Figure 2.5: Screenshots of the target tracking process. There are eight robots tracking moving targets. The bottom right small graph in each figure show the network topology of the team at the start of that epoch. Sub-figures (a)-(d) show how the topology evolves over time.

As shown in Fig. 2.6, the proposed algorithm can achieve the 90% of the performance of the pure greedy strategy. Moreover, the proposed algorithm on average can track ten percent more targets compared to SGG. As for three ways to select weights in deviation minimization problem, they have similar performances. Readers are referred to multimedia submission for an animation that shows the connectivity during operation of the team and one screenshot is shown in Fig. 2.5.



(a) (b) (c)

Figure 2.6: Average number of targets observed over each epoch. (a) There are 5 robots and 80 targets. (b) There are 8 robots and 140 targets. (c) There are 12 robots and 200 targets. The proposed algorithm, greedy algorithm without considering communication at each epoch, and SGG are denoted in blue, orange, and greed bars respectively. The black line above bars represent one standard deviation.

2.6 Conclusion

In this chapter, we propose a problem named Communication-aware Submodular Maximization (CSM) for a class of multi-robot task planning and propose a heuristic algorithm consisting of two stages, topology generation and deviation minimization, to solve CSM. The performance of the proposed heuristic algorithm is empirically evaluated in Section 2.5.

We plan to further our research by finding bounded-approximation algorithms for the proposed problem. We will explore in two directions. The first direction to consider some special cases of the CSM problem. For example, if the all robot share the same reachable points for the next epoch, the problem becomes a assignment problem over a graph. The research question then is that whether we find some approximation algorithm with performance guarantee for such a case. The second direction is to try to identify the approximation ratio of the proposed heuristic algorithm. In the proposed two-heuristic heuristic algorithm, the operations in the second stage will destroy the property of the greedy algorithm. One question we are interested in is whether we can quantify the deviation of the second stage causes to the solution of the greedy algorithm.

If we can achieve this, some approximation ratios for the proposed heuristic algorithm can be found.

Chapter 3: Multi-Robot Information Gathering with Partial Knowledge of the environmental model: Non-Bayesian Case

3.1 Overview

In this chapter, we consider the multi-robot information gathering problem in uncertain environments where some robots may fail during tasks, and we have only partial knowledge about the failures: there are at most α failures. Once a robot fails, all the information collected as well as the robot will be lost. Our goal is to find a coordinate strategy to make the robot team robust to failures when they operate in such environments. We formulate such problems as robust submodular maximization problems. Specifically, We introduce the Robust Multiple-path Orienteering Problem (RMOP) where we seek worst-case guarantees against an adversary that is capable of attacking at most α robots. We consider two versions of this problem: RMOP offline and RMOP online. In the offline version, there is no communication or replanning when robots execute their plans, and our main contribution is a general approximation scheme with a bounded approximation guarantee that depends on α and the approximation factor for single robot orienteering. In particular, we show that the algorithm yields a (i) constant-factor approximation when the cost function is modular; (ii) $\log$ factor approximation when the cost function is submodular; and (iii) constant-factor approximation when the cost function is submodular but the robots are allowed

to exceed their path budgets by a bounded amount. In the online version, RMOP is modeled as a two-player sequential game and solved adaptively in a receding horizon fashion based on Monte Carlo Tree Search (MCTS).

The Orienteering Problem (OP) is that of determining a path, whose length is less than a given budget, from a given starting vertex that maximizes the total reward collected along the path [53]. The reward depends on the vertices visited along the path. The OP[1] naturally models informative-path planning: a robot is tasked to gather as much information from the environment as possible within a given time or energy budget. For example, in [26, 54, 55], ocean monitoring,

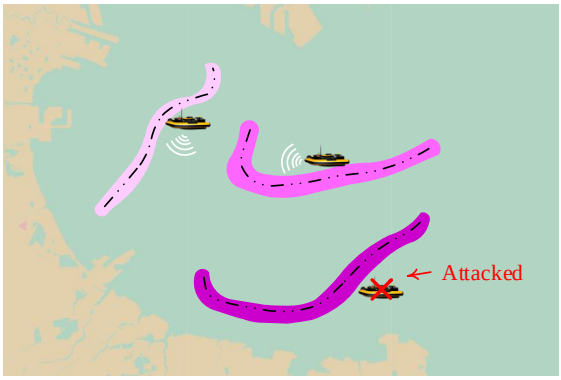


Figure 3.1: Case study of monitoring a marine environment with aquatic robots. The robots are tasked with finding informative paths to gather data. The darker the color of the path is, the more valuable the path since it gathers information from a more important region. We investigate the question of how the robots should plan their paths if we expect some of the robots to fail due to adversarial elements or natural causes.

opportunistic surveillance, and 3D reconstruction tasks are formulated as the OP or its variants. In general, the OP is NP-hard, but there are constant-factor approximation algorithms for many variants [56]. This includes the Multiple-path Orienteering Problem (MOP) [56] where the goal is to design paths for N robots such that the sum of the rewards collected by all the robots is maximized. In this paper, we introduce the robust variant of OP. Specifically, we introduce

[1]Unless specified otherwise, OP refers to single robot orienteering.

the Robust Multiple-Path Orienteering Problem (RMOP) motivated by scenarios where robots operate in adversarial or failure-prone environments.

Figure 3.1 shows a motivating scenario where a team of underwater robots is tasked with gathering data in an ocean. However, some robots in the team may fail to complete their paths either due to adversarial attacks [57] or hardware malfunction [58]. If a robot fails, then the data gathered by it is lost. Our goal is to provide efficient planning and coordination algorithms that are resilient to such failures.

Building robot teams that are robust to adversarial attacks is emerging as an important research area [59–62]. Our approach differs from classical fault-tolerant frameworks [63–65] that focus on making individual robots robust to failures. Instead, we focus on the question of how the team should coordinate their actions to improve redundancy in their plans such that even if some robots fail, the overall performance of the team will not drop significantly. As such, our work is completely different from the work on making individual robots robust.

In this chapter, we focus on the RMOP to make progress toward the aforementioned broader goal. The RMOP seeks plans for a team of N robots that guard against worst-case failures. Of course, in the worst case, all N robots may fail. To make it more meaningful, we study the case where at most a given number $\alpha < N$ robots may fail. What we seek is to understand how the performance of the team will be affected as a function of α. We consider two types of RMOP, in both of which we seek to find a path consisting of multiple steps for each robot. In the offline RMOP in which robots cannot communicate with each other or the base station during tasks, our main contribution is an algorithmic scheme that uses a single robot OP solution as a subroutine. Choosing an appropriate subroutine allows us to investigate three variants of the original problem. In the general version, the reward collected by an individual robot is a

submodular function of the vertices along the path. Submodularity is the property of diminishing returns [66]. Many information gathering measures such as mutual information [33] and coverage area [28] are known to be submodular. We also study special cases where the reward function is strictly modular (i.e., additive) and where the budget constraint for each robot can be relaxed by a bounded amount. In the online RMOP, we model the problem as a sequential two-player game and propose an adaptive strategy based on MCTS, and the problem is solved in a receding horizon fashion with the history of the observed attack taken into account.

The orienteering problem has been researched extensively by both theoretical computer science and operations research communities. The review by Vansteenwegen [53] summarizes various algorithms for OP and its variants. We highlight the results most closely related to our work. Blum et al. [56] presented a polynomial-time 4–approximation for OP when the objective function is modular. This result is then extended to yield a 5–approximation for the MOP assuming all robots start at different vertices. If the reward function is submodular, Chekuri and Pal [67] present a recursive greedy algorithm for a single robot that yields a $O(\log(OPT))$ approximation algorithm, where OPT is the reward collected by the optimal algorithm. The algorithm runs in quasi-polynomial time.

Singh et al. [68] showed how to use OP and MOP for active information gathering to learn a spatial model of the environment represented by Gaussian Processes. Their algorithms sequentially find paths for each robot using the single-robot algorithms by Blum et al. [56] and Chekuri and Pal [67] as subroutines. Atanasov et al. [69] recently presented a decentralized version for multi-robot information gathering along similar lines as [68]. They use a submodular objective function but solve a finite horizon planning problem as opposed to OP. However, none of these works account for potential failures of the robots, as we do in RMOP.

Recently, Jorgensen et al. introduced the Matroids Team Surviving Orienteers Problem (MTSO) [26] which does account for individual robot failures. They assume that there is some given probability of failure associated with every edge in the environment. The goal is to maximize the expected rewards while ensuring each path satisfies some survival chance constraints. MTSO is appropriate when the failures of robots are random and follow a known distribution. The version we study, the RMOP, accounts for worst-case failures which makes it better suited when operating in adversarial conditions or in stochastic conditions when worst-case guarantees are sought due to unknown probability distributions.

Our work builds on recent work on robust submodular maximization [23, 37, 70–73] which selects sets that are robust to worst-case removal of some subset of items. The challenge in this framework is to solve the trade-off between too much overlap, thereby not enough coverage (i.e., reward) and too little overlap, thereby not enough redundancy. The conceptual idea in these papers is similar — the final solution consists of two subsets, one that has enough redundancy to ensure robustness against worst-case removal and the other that has enough coverage to get good overall performance. Orlin et al. [70] term the former as "copies" whereas it is called "baits" in [23]. The robust submodular maximization formulation has been applied for multi-robot, multi-target tracking in centralized [23] and decentralized settings [73] as well as for active information gathering with multi-robot teams [37].

We seek similar robustness guarantees as in the works mentioned in the previous paragraph. The key technical advancement we make is that these prior works solve a single-step selection problem whereas we solve a multi-step planning problem. As a result, the single robot problem in the prior work can be trivially solved optimally (amounts to selecting the best amongst a finite set of options), whereas in the RMOP the single robot problem (OP) itself is NP-Hard.

While both [23] and [37] use their results for planning over a finite horizon, they make key assumptions that are limiting. Schlotfeldt [37] considers the continuous counterpart of the combinatorial problem considered in this paper, and they formulate the problem under the optimal control framework with one key assumption that the single robot, as well as multi-robot information gathering problem (without attacks), can be solved optimally (c.f. Proposition 1). Zhou et al. [23] repeatedly solve the one-step problem at each time step. Instead, we show how to use an approximate solution to the OP to yield a bounded approximation solution to the combinatorial problem RMOP.

3.2 Preliminaries

We use calligraphic fonts to denote sets (e.g. $\mathcal{A}$). Given a set $\mathcal{A}$, $2^{\mathcal{A}}$ denotes the power set of $\mathcal{A}$ and $|\mathcal{A}|$ denotes the cardinality of $\mathcal{A}$. Given another set $\mathcal{B}$, the set $\mathcal{A} \setminus \mathcal{B}$ denotes the set of elements in $\mathcal{A}$ but not in $\mathcal{B}$. Given a set $\mathcal{V}$, a set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$, and an element $x \in \mathcal{V}$, $f(x)$ is a shorthand that denotes $f(\{x\})$. We use $f_{\mathcal{A}}(\mathcal{B})$ to denote $f(\mathcal{A} \cup \mathcal{B}) - f(\mathcal{A})$.

We now define two useful properties of set functions.

Definition 3 (Normalized Monotonicity)**.** For a set $\mathcal{V}$, a function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is called as normalized, monotonically non-decreasing if and only if for any $\mathcal{A} \subseteq \mathcal{A}' \subseteq \mathcal{V}$, $f(\mathcal{A}) \leq f(\mathcal{A}')$ and $f(\mathcal{A}) = 0$ if and only if $A = \emptyset$.

As a short hand, we refer to a normalized, monotonically non-decreasing function as simply a monotone function.

Definition 4 (Submodularity)**.** For a set $\mathcal{V}$, a function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is submodular if and only if for any sets $\mathcal{A} \subseteq \mathcal{V}$ and $\mathcal{A}' \subseteq \mathcal{V}$, we have $f(\mathcal{A}) + f(\mathcal{A}') \geq f(\mathcal{A} \cup \mathcal{A}') + f(\mathcal{A} \cap \mathcal{A}')$.

Let $G(\mathcal{V}, \mathcal{E})$ be a graph. A path $\mathcal{P}$ in G is an ordered sequence of non-repeated vertices. As a shorthand, we use $\mathcal{P}$ to denote both the path (ordered set) as well as the unordered set of vertices along the path. When we use $\mathcal{P}$ as the path (ordered set), $\mathcal{P}(i)$ denotes i_{th} vertex in $\mathcal{P}$. Let $\mathcal{T} = 2^{\mathcal{V}}$ denote the power set of $\mathcal{V}$. Intuitively, $\mathcal{T}$ is the superset of all possible sets of vertices that a robot may visit along its path. The cost of a path $\mathcal{P}$, denoted by $C(\mathcal{P})$, is the sum of the edge weights along the path. We assume that the edge weights are metric. We study the *rooted* version of the problem where the path for robot i, denoted by $\mathcal{P}_i$, must begin at a specific vertex v_{s_i}.

We consider the case that the *reward function*, $g(\mathcal{P}) : \mathcal{T} \rightarrow \mathbb{R}_+$, of a single robot is a monotone submodular function. We also study the special version where the function is modular (i.e., the reward of a path is the sum of rewards of the vertices along the path).

Let $\mathcal{S}$ be some collection of N paths corresponding to the N robots in the team, $\mathcal{S} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$. The team reward collected by any subset $\mathcal{S}' \subseteq \mathcal{S}$ is given by,

$$f(\mathcal{S}') = g\left(\bigcup_{\mathcal{P}_i \in \mathcal{S}'} \mathcal{P}_i\right). \tag{3.1}$$

Note that the reward function of the team $f(\mathcal{S}')$ is a submodular function irrespective of whether the single robot reward function $g(\mathcal{P}_i)$ is submodular or not. Multiple robots can visit the same vertex, but only one visit is accounted for when computing the reward of the team. That is, there can be no double counting of the rewards. We are now ready to define our problem formally.

3.3 Problem Formulation—Offline RMOP

Problem 3 (Offline RMOP)**.** *Given a metric graph $G(\mathcal{V}, \mathcal{E})$, N robots with starting positions $\{v_{s_1}, v_{s_2}, \ldots, v_{s_N}\}$, budget constraint B, a robot reward function $g(\mathcal{P}) : \mathcal{T} \to \mathbb{R}_+$, and a team reward function f as defined in Equation 3.1, the offline Robust Multiple-Path Orienteering Problem seeks to find a collection of N paths, $\mathcal{S} = \{\mathcal{P}_1, \ldots, \mathcal{P}_N\}$ that are robust to the worst-case failure of α robots:*

$$\begin{aligned} &\max_{\mathcal{S}\subseteq\mathcal{T}} \min_{\mathcal{A}\subseteq\mathcal{S}} f(\mathcal{S} \setminus \mathcal{A}) \\ \textit{s.t.} \quad &|\mathcal{A}| \leq \alpha, 0 < \alpha < N \\ &|\mathcal{S}| = N \\ &C(\mathcal{P}_j) \leq B. \end{aligned} \tag{3.2}$$

where additionally v_{s_j} must be the starting vertex when constructing a path $\mathcal{P}_j$ for robot j.

The offline RMOP is suited to model the scenarios where we need to plan paths for all the robots before they are deployed and they cannot communicate with the base station to transmit collected rewards or with each other to replan during execution. Therefore, once a robot fails during the task, the reward of the whole path of that robot will be lost, which corresponds to the set removal of paths, and the team cannot adapt to the failures of robots. One practical example of the offline RMOP is the naval mine countermeasure mission [74], in which a team of robots is deployed to detect undersea mine information. In such a case, reliable communication is usually not available, and the robot may fail due to mines or other adversaries. Mathematically, the offline RMOP can be interpreted as a two-stage perfect information sequential one-step game, where the first player (i.e., the team of robots) chooses a set $\mathcal{S}$, and the second player (i.e., the adversary),

knowing $\mathcal{S}$, chooses a subset $\mathcal{A}$ to remove from $\mathcal{S}$. We seek worst-case guarantees — in practice, the adversary may not know the paths for each robot. By playing against this stronger adversary, we guarantee that the performance against a weaker one will be even better. We evaluate this empirically by considering attack models other than the worst one.

The adversarial model considered in this paper is the same as that in prior work on robust submodular optimization [23, 71, 72, 75]. However, the offline RMOP is even harder since even at the single robot level, the optimization problem we need to solve (i.e., OP) is NP-Hard. Nevertheless, we present a constant-factor approximation algorithm for this problem next.

3.4 Algorithm for Offline RMOP

In this section, we present the general algorithm to solve the offline RMOP (Algorithm 3). The algorithm uses a generic subroutine for solving OP. In the next section, we show examples of three subroutines that can be used and show how they affect the performance of the algorithm.

Our algorithm builds on those in [71, 75]. The key idea in these algorithms is to construct two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \cup \mathcal{S}_2$ is a feasible approximation solution to the corresponding problem and $f(s_1) \geq f(s_2), \forall s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2$. The main difference in our work lies in how these sets are computed. The problem in [71] considers only choosing elements from a given set, with a cardinality constraint. The algorithm in [71] finds $\mathcal{S}_1$ by sorting the elements and finds $\mathcal{S}_2$ by greedily adding elements with the largest marginal gain. A more general problem, with matroid constraints instead of a cardinality constraint, in considered in [75]. To find $\mathcal{S}_1$ from a given ground set $\mathcal{V}$, the algorithm loops over all elements in $\mathcal{V}$ and adds elements to $\mathcal{S}_1$ by considering the value of a single element ($f(y)$ in [75] line 3) and checking the matroid constraint

(lines 4-5 in [75]). To find $\mathcal{S}_2$, the algorithm in [75] loops over $\mathcal{V} \setminus \mathcal{S}_1$ and adds elements to $\mathcal{S}_2$ incrementally by considering the marginal gain and matroid constraints (lines 10-12 in [75]). In contrast, in our problem, the ground set $\mathcal{V}$ itself (the set of all paths) is not readily available. Enumerating this ground set is infeasible. Instead, we can use the set of all vertices in the graph as the ground set. However, finding one path that maximizes the reward function is an NP-Hard problem. Thus, the simple selection step that can be solved just by looping over all elements (e.g., line 3 [75]) requires solving an NP-Hard problem. We first let all robots to solve OP individually with a bounded approximation algorithm (lines 2-5). Then, we find a candidate set for $\mathcal{S}_1$ by sorting (lines 8-9). Next, we use a Sequential Greedy Assignment (SGA) paradigm to construct a candidate set for $\mathcal{S}_2$. To guarantee $f(s_1) \geq f(s_2), \forall s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2$, we use the while loop to improve the candidate sets for $\mathcal{S}_1$ and $\mathcal{S}_2$.

Before we describe the algorithm, we present additional notation. If $\mathcal{S}'$ is a set of $N' \leq N$ paths, then let $\mathcal{R}(\mathcal{S}')$ denote the set of corresponding N' robots whose paths are contained in $\mathcal{S}'$. We use $\mathcal{A}^*(\mathcal{S}) \triangleq \arg\min_{\mathcal{A}} f(\mathcal{S} \setminus \mathcal{A})$ to denote the worst-case set of paths that are removed from a given set of path $\mathcal{S}$. Therefore, $\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})$ denotes the set of paths that are not attacked from $\mathcal{S}$ with $|\mathcal{A}^*(\mathcal{S})| \leq \alpha$.

The algorithm consists of two main steps: first, it calls a subroutine for solving OP N times to compute a path for each robot independently. It then chooses α paths (denoted by $\mathcal{S}_1$) with the highest individual rewards without considering overlap with other robots; Second, it uses sequential greedy assignment to find paths for the rest of the robots (denoted by $\mathcal{S}_2$) by querying the OP subroutine $N - \alpha$ times. The **while** loop is used to maintain an invariant that the paths in $\mathcal{S}_1$ are always better than the paths in $\mathcal{S}_2$.

As described earlier, there is a tradeoff between redundancy and coverage in RMOP. The

Algorithm 3: Algorithm for Problem 3

Input : Per problem 3 requires following inputs:

- set of robots $\mathcal{R} = \{1, \ldots, N\}$
- metric graph G
- starting vertices $\{v_{s_1}, v_{s_2}, \ldots, v_{s_N}\}$
- number of maximum potential attacks α and budget B

Output: Set $\mathcal{S}$ of paths for each robot

$\mathcal{S}_1 \leftarrow \emptyset, \mathcal{S}_2 \leftarrow \emptyset, \mathcal{M} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** N **do**
  $\mathcal{P}_i \leftarrow OP(G, v_{s_i}, B)$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathcal{P}_i\}$
end
flag $\leftarrow$ True
while *flag* **do**
  Sort elements in $\mathcal{M}$ such that $\tilde{\mathcal{M}} = \{\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_N\}$ and $f(\{\mathcal{P}'_1\}) \geq f(\{\mathcal{P}'_2\}) \geq \ldots \geq f(\{\mathcal{P}'_N\})$
  $\mathcal{S}_1 \leftarrow \{\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_\alpha\}$
  //extract starting positions for the rest of the robots $\tilde{v}_s \leftarrow \{v_{s_j} | \forall j \in \mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)\}$
  //Sequential greedy assignment $\mathcal{S}_2 \leftarrow SGA(G, \tilde{v}_s, B)$
  //while loop control
  if $f(\mathcal{P}_i) \geq f(\mathcal{P}_j), \forall \mathcal{P}_i \in \mathcal{S}_1, \mathcal{P}_j \in \mathcal{S}_2$ **then**
    $flag \leftarrow$ False
  else
    Find all robots $j \in \mathcal{R}(\mathcal{S}_2)$ such that
    $\exists i \in \mathcal{R}(\mathcal{S}_1), f(\{\mathcal{P}_j \in \mathcal{S}_2\}) > f(\{\mathcal{P}_i \in \mathcal{S}_1\})$
    Replace the path stored in $\mathcal{M}$ corresponding to robot j with the better path found when constructing $\mathcal{S}_2$
  end
end
$\mathcal{S} \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$

two sets of paths are constructed so that $\mathcal{S}_1$ adds redundancy and $\mathcal{S}_2$ adds coverage, together yields a provably good solution for RMOP. We explain each step in Algorithm 3 next.

Constructing $\mathcal{S}_1$ Each of the α paths in $\mathcal{S}_1$ are better than those in $\mathcal{S}_2$. The paths in $\mathcal{S}_1$ may overlap with each other and also overlap with those in $\mathcal{S}_2$. Thus, these paths serve to add redun-

dancy to the team. Constructing the best α paths with respect to f itself is NP-hard. Therefore, Algorithm 3firstly solves the orienteering problem for each robot independently and stores in $\mathcal{M}$ the (approximately optimal) paths for individual robots (lines 2–5). Then Algorithm 3sorts the paths in $\mathcal{M}$ based on their collected rewards (line 8) and chooses the α best paths to be $\mathcal{S}_1$ (line 9).

Constructing $\mathcal{S}_2$ After finding $\mathcal{S}_1$, Algorithm 3 needs to find the best paths for the rest of robots $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$. Unlike $\mathcal{S}_1$, here the algorithm explicitly considers overlap when finding the paths. Thus, $\mathcal{S}_2$ serves to add coverage to the solution.

However, selecting optimal paths for $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$ is a multiple-path orienteering problem and is also NP-hard. Therefore, Algorithm 3 approximates the solution by employing the sequential greedy algorithm (line 11). For completeness, we present the pseudocode for SGA in Algorithm 4.

Specifically, for robots in $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$, Algorithm 4 finds a path using an approximation algorithm for OP (line 4). Then, Algorithm 4 sets the reward for the vertices visited by that robot to be zero (lines 6–8). This process repeats until we find a path for all robots. Here we implicitly assume that there is at least one path for each robot satisfying the budget constraints.

The paths in $\mathcal{S}_1 \cup \mathcal{S}_2$ form the solution to RMOP. However, we also have an outer **while** loop which we explain next.

Invariant Our analysis requires the paths in $\mathcal{S}_1$ and $\mathcal{S}_2$ to have the following property: $f(\{\mathcal{P}_i\}) \geq f(\{\mathcal{P}_j\}), \forall \mathcal{P}_i \in \mathcal{S}_1, \mathcal{P}_j \in \mathcal{S}_2$. This condition is trivially met if the single robot problem has to just choose the best amongst a fixed set of trajectories as in the prior work [23, 71]. However, when

solving RMOP, we employ a subroutine for solving OP which gives us the paths in $\mathcal{S}_1$ and $\mathcal{S}_2$. Since the subroutine uses an approximation algorithm for OP instead of an exact optimal one, we cannot guarantee that this invariant holds. For example, if the subroutine uses randomness, then running the same algorithm twice may give different results. In any case, all we can guarantee is that the paths found by the subroutine will be no more than a constant from the optimal.

We fix this problem by utilizing a **while** loop (lines 7–20). When the condition of the **while** loop holds (lines 13–15), the loop flag is set to be false and the while loop terminates. Otherwise (lines 16–19), Algorithm 3 will find those robots that violate the above inequality and update their corresponding paths in the set $\mathcal{M}$. Recall that $\mathcal{M}$ is used to store the best path corresponding to each robot. Then, while loop will restart to construct $\mathcal{S}_1$ using the updated $\mathcal{M}$ and $\mathcal{S}_2$ for the remaining robots, again. We show that this loop will eventually terminate.

Corollary 1. *The **while** loop in Algorithm 3 will terminate in a finite number of steps.*

Proof. If the flag is not set to false after an iteration of the **while** loop, then it must mean that at least one new path found when constructing $\mathcal{S}_2$, say for robot j, is better than some path in $\mathcal{S}_1$. Suppose this better path is $\mathcal{P}_j^{'}$. Note that the set $\mathcal{M}$ includes a path for the robot j, say $\mathcal{P}_j$. Since $\mathcal{S}_1$ consists of the best α paths in $\mathcal{M}$ and $\mathcal{P}_j \notin \mathcal{S}_1$, then it must mean that the path $\mathcal{P}_j^{'}$ is strictly better than $\mathcal{P}_j$. Thus, after every iteration of the **while** loop, if the flag is not set, then at least one path in $\mathcal{M}$ has improved. For each robot given a fixed budget, there is a maximum amount of reward that it can collect. We cannot keep increasing the rewards of paths in $\mathcal{M}$. Therefore, the while loop must terminate after finite iterations. ■

Remark 1. In practice, the loop in Algorithm 3 typically terminates after just one iteration. Paths in $\mathcal{S}_1$ are found without considering overlap. On the other hand, when solving SGA the robots

Algorithm 4: Sequential Greedy Assignment

Function SGA(G, v_s, B)**:**
  Input :
    - A graph G representing the environment
    - Budget B for each robot
    - Starting positions v_s

  Output: a collection $\mathcal{A}$ of paths
  $\mathcal{A} \leftarrow \emptyset, G' \leftarrow G, N \leftarrow length(v_s)$
  for $j \leftarrow 1$ **to** N **do**
    $\mathcal{P}_j \leftarrow OP(G', v_{s_j}, B)$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathcal{P}_j\}$
    foreach $v \in \mathcal{P}_j$ **do**
      Set the reward of $v \in G'$ to be zero
    end
  end
  return $\mathcal{A}$
end

find their paths by taking into account overlap with the previously found paths. The conditions in the latter are a subset of the former. Furthermore, none of the three subroutines that we employ for OP include any randomness. Therefore, it is unlikely that the paths in $\mathcal{S}_2$ will be better than that in $\mathcal{S}_1$. As such, it is less likely that the **while** loop will take more than one iteration. Nevertheless, we give the full algorithm for completeness.

So far, we have not discussed the subroutine used to solve OP. In Sec. 3.7, we present the analysis of the algorithm and then present the three subroutines.

3.5 Problem Formulation—Online RMOP

In the offline RMOP, we consider the scenario in which robots cannot communicate with each other or the base station when they execute tasks. As a result, once a robot fails or is attacked, we will lose all the rewards collected by that robot, i.e., lose the path and we corre-

spondingly plan for the worst-case attacks. In the online RMOP, we consider the scenario where robots can communicate with the base station to send collected rewards during execution and with teammates to replan in response to the attacks. Attackers' behaviors are assumed to be the same: there are α attacks in total, and the attacks can happen at any node on the map. But the difference is that for the offline case, it doesn't matter when the robot fails because we will lose the whole path as long as it fails. For the online case, when the robot fails, it will influence how much reward robots can collect. For example, if a robot is attacked at the very first node, we can get only the reward of that particular node; but if the robot is attacked when it almost uses up the budget, we can collect most reward along its path. Therefore, the attacker's behaviors are characterized by two types of decision variables: when to attack and which to attack. Without loss of generality, we assume that it takes one unit of time (not necessarily the same amount of budget) to traverse one edge of the graph such that robots are able to replan synchronously. Such a graph can be obtained by carefully designing motion primitives and discretizing the environment or by adding some virtual nodes on the edges of a graph. With this assumption, the online RMOP can be formulated as follows.

Definition 5 (Attacker behavior set)**.** An attack behavior set $\mathcal{A} = \{(t_1, r_1), \ldots, (t_\beta, r_\beta)\}$ is a set of tuples, each of which consists of two elements: the first element indicates when to attack and the second element indicates which robot to attack.

Problem 4 (Online RMOP)**.** *Given a metric graph $G(\mathcal{V}, \mathcal{E})$, N robots with starting positions $\{v_0^1, v_0^2, \ldots, v_0^N\}$, budget constraint B, a robot path reward function $g(\mathcal{P}) : \mathcal{T} \to \mathbb{R}_+$, the online*

Robust Multiple-Path Orienteering Problem seeks to find a collection of N paths

$$\mathcal{S} = \left\{ \begin{array}{l} \mathcal{P}_1 = \{v_0^1,\ \ldots, v_m^1, \ldots,\ v_{end}^1\} \\ \vdots \\ \mathcal{P}_N = \{v_0^N,\ \ldots, v_m^N, \ldots,\ v_{end}^N\} \end{array} \right\}$$

that are robust to the worst-case failure of α robots:

$$\begin{aligned} & \max_{\mathcal{S}\subseteq\mathcal{T}} \min_{\mathcal{A}=\{(t_j,r_j)\}} g(\bigcup_{i=1}^{N} \mathcal{P}_i \setminus \bigcup_{j=1}^{|\mathcal{A}|} \mathcal{P}_{r_j}[t_j+1:end]) \\ \textit{s.t.} \quad & |\mathcal{A}| \leq \alpha, 0 < \alpha < N \\ & |\mathcal{S}| = N \\ & C(\mathcal{P}_j) \leq B, \end{aligned} \tag{3.3}$$

where $\mathcal{A} = \{(t_1, r_1), \ldots, (t_{|\mathcal{A}|}, r_{|\mathcal{A}|})\}$ is an attacker behavior set; $\mathcal{P}_{r_j}[t_j + 1 : end]$ is the path segment of robot r_j's path $\mathcal{P}_{r_j}$ from the node $\mathcal{P}_{r_j}(t_j + 1)$ to the node $\mathcal{P}_{r_j}(end)$; additionally v_0^j must be the starting vertex when constructing a path $\mathcal{P}_j$ for robot j.

Intuitively, in Problem 4, we want to find paths for robots while α failures in total can happen at any nodes along the paths. If a robot fails at a particular node $\mathcal{P}(t)$, then it cannot collect reward after that node anymore. We model this problem as a discrete, sequential, two-player zero-sum game between the attackers and the robots. Since robots can communicate with each other, we aim to find one adaptive planning strategy that can adapt to the attacks.

Proposition 1. *It's not the optimal strategy for attackers to always launch all attacks at the very first step.*

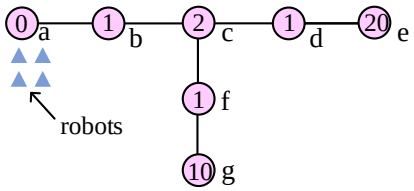


Figure 3.2: One example to show that rational attackers will not always launch attacks at the first step. Four robots start from node a, and each has a budget of 4. There are $\alpha = 2$ attacks in the environment. If attackers launch all attacks at the first step, the robots can collect 35 rewards in total by planning path (a, b, c, d, e) and (a, b, c, f, g) while robots can collect at most 25 if the attackers choose to wait until they know how robots move after node c.

Proof. One example is given in Fig. 3.2. ■

It should be noted that a rational attacker will not always launch attacks at the first step. For example, in Fig. 3.2, there are four robots that are initially located in node a, and each of them has a budget of $B = 4$. There are $\alpha = 2$ attackers in the environment. If the attackers attack two robots at the first step. The surviving two robots can certainly collect 35 rewards in total by planning path (a, b, c, d, e) and (a, b, c, f, g). By contrast, if the attackers choose to wait until they know how robots move after node c, the robots can collect at most 25 rewards. Here is the explanation. After four robots reach node c and all survive, the best strategy for robots is to send three of them to follow the path (c, d, e) and another robot to follow (c, f, g) considering that there are still $\alpha = 2$ attacks. In the worst case where the robot following (c, f, g) and one robot following (c, d, e) got attacked, robots can collect 25 rewards in total. If robots don't adopt the best strategy, they will get less than 25 rewards in the worst case. In the next section, we demonstrate how to find the optimal solution for this game using two-player MCTS.

3.6 MCTS for Online RMOP

MCTS is an approach for finding optimal actions by randomly selecting samples from search space and incrementally building the search tree [4] and is widely applied in robotics applications, including scouting [76], active parameter estimation [77], environment monitoring [78], and multi-robot active perception [79]. As shown in Fig. 3.3, there are four basic steps in each iteration of an MCTS process: selection, expansion, simulation, and backpropagation [80].

In the paper, we model the Problem 4 as a sequential, discrete two-player game and we adopt the MCTS algorithm to solve the game in an online fashion. At each step, attackers use their strategies to take one action first, and then robots take one action. Intuitively, it means that attackers observe the states of robots to decide whether to attack or not, and then robots respond to that. To choose one action, the robots will incrementally grow the search tree with some computational budget and then select one action to take from the root of the search tree based on the average reward of each action. Such a process continues until all robots run all of the budgets.

When our algorithm grows the search tree, the state of each robot is a tuple $s = (v, I)$ where $v \in \mathcal{V}$ represents the current position of the robot and I is an indicator on whether the robot has been attacked ($I = 1$) or not ($I = 0$). The joint state of the team is the product of the individual states of robots and is stored in each node of the search tree. Robots and attackers alternate turns in growing the tree. When it's the robots' turn, they will decide the transition of the positions while the attackers can decide the value of the indicator state in the attackers' turn. Once one indicator state is set to be one, which means a robot is attacked, it will remain one for the rest of the game, and that robot cannot move anymore, i.e., the only action that the robot can take in the game is to stay there. Similarly, if a robot has run out of budget, the only action

available is to stay in the current position. It should be noted that we present Algorithm 5 from the perspective of robots but attackers can also use similar strategies. In the following, we refer to two-player MCTS (if there are two alternating turns in the search) as MCTS with adversaries and one-player MCTS without considering opponents as naive MCTS. Details of the Algorithm 5 are given below.

1. *Selection* (Line 4 in Algorithm 5; Line 1-11 in Algorithm 6): Starting from the root node, a selection procedure is recursively applied until some leaf node is reached. In each recursion, a child node is selected based on Upper Confidence Bound for Trees (UCT) Kocsis and Szepesvári [81]. There are two parts to the UCT value: exploitation and exploration. The exploitation part corresponds to the average rollout reward obtained and the exploration part is decided by the number of times that the node has been visited ($n(v')$) and the number of times that the current node's parent has been visited (n_p). If a node is less visited, the exploration value will increase which encourages the selection of that node. It should be noted that if it's the robots' turn robots will select the node with the highest UCT value (Line 6) while the attacker will select the node with the lowest UCT value if it's the attacker's turn (Line 8).

2. *Expansion* (Line 5 in Algorithm 5; Line 12-19 in Algorithm 6): One (or more) child nodes are added to the tree based on the available actions. If the node has reached the terminal level, e.g., running out of budget, the current node will be returned (Lines 13-15). Otherwise, add all children of the current node to the tree and return the first child node (Lines 15-18).

3. *Simulation* (Line 6 in Algorithm 5; Line 20-29 in Algorithm 6): A rollout is conducted

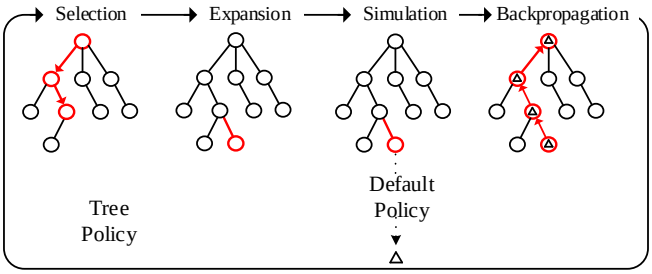


Figure 3.3: One iteration of the general MCTS approach [4].

Algorithm 5: Monte Carlo Tree Search

```
1  Function MCTS(s_1, s_2, ..., s_N):
       Input : Initial states of robots, which include information on attacks.
       Output: A search tree
2      Create a tree with root node v_0^t with initial states (s_1, ..., s_N)
3      while computational budget not used up do
           // selection
4          v_sel^t ← Selection(tree, v_0^t)
           // expansion
5          v_exp^t ← Expansion(tree, v_sel^t)
           // rollout
6          Reward ← Simulation(tree, v_exp^t)
           // backpropagation
7          Backpropagation(tree, Reward, v_exp^t)
8      end
9      return tree
10 end
```

from the chosen node using the default policy until some terminal condition is met. The obtained reward will be returned.

4. *Backpropagation* (Line 7 in Algorithm 5; Line 30-34 in Algorithm 6): The simulation result is propagated back to the root and update node information along the propagation path.

Algorithm 6: Monte Carlo Tree Search Subroutines

Function `Selection`($tree,\ v$)**:**
  if *level(v) = TERMINAL* **then**
    return v
  end
  if *turn(v) = ROBOT* **then**
    // n_p is the number of times that the parent of v has been visited
    $$v \leftarrow \underset{v' \in \text{children}(v)}{\operatorname{argmax}} \frac{Q(v')}{n(v')} + c\sqrt{\frac{2\ln n_p}{n(v')}}$$
  else
    $$v \leftarrow \underset{v' \in \text{children}(v)}{\operatorname{argmin}} \frac{Q(v')}{n(v')} - c\sqrt{\frac{2\ln n_p}{n(v')}}$$
  end
  return `Selection`($tree,\ v$)
end
Function `Expansion`($tree,\ v$)**:**
  if *level(v) = TERMINAL* **then**
    return v
  else
    Add all child nodes of v to $Tree$
    return the first child node
  end
end
Function `Simulation`($tree,\ v$)**:**
  while *level(v) $\neq$ TERMINAL* **do**
    if *turn(v)=ROBOT* **then**
      $v \leftarrow$ RobotDefaultPolicy(v)
    else
      $v \leftarrow$ AttackerDefaultPolicy(v)
    end
  end
  return CollectReward
end
Function `Backpropagation`($tree,\ Reward,\ v$)**:**
  while $v \neq$ *NULL* **do**
    // update total reward value $tree.v.Q \leftarrow tree.v.Q + Reward$
    $tree.v.n \leftarrow tree.v.n + 1$
  end
end

3.7 Performance Analysis

In this section, we quantify the performance of the proposed Algorithm 3. We first present a new analysis for the Sequential Greedy Assignment (SGA) and then show the performance bound for our algorithm. The performance is based on the notion of curvature of the set functions.

Definition 6 (Total Curvature)**.** Consider a finite ground set $\mathcal{V}$ and a monotone submodular set function $h : 2^{\mathcal{V}} \mapsto \mathbb{R}$. The curvature of h is defined as

$$k_h = 1 - \min_{v \in \mathcal{V}^\dagger} \frac{h(\mathcal{V}) - h(\mathcal{V} \setminus v)}{h(v)}, \tag{3.4}$$

where $\mathcal{V}^\dagger = \{v \in \mathcal{V} \mid h(v) > 0\}$.

The curvature takes values $0 \leq k_h \leq 1$ and measures how far h is from modularity. When $k_h = 0$, h is modular since for all $v \in \mathcal{V}$, we get $h(\mathcal{V}) - h(\mathcal{V} \setminus \{v\}) = h(v)$. On the other extreme, when $k_h = 1$ there exists some element v that makes no unique contribution to the rest of the set, since we get $h(\mathcal{V}) = h(\mathcal{V} \setminus \{v\})$. We assume that the curvature k_g of the reward function and that k_f of the objective function is strictly less than 1. This is reasonable since it implies every vertex and path in the environment makes some non-zero unique contribution over the rest.

We first analyze the SGA and then use that analysis to prove the performance bound of our algorithm.

3.7.1 Sequential Greedy Assignment

SGA was first proposed in [68] to solve the MOP. Note that the MOP is the same as RMOP if we consider $\alpha = 0$. SGA solves the problem by finding the path for the i^{th} robot in the i^{th}

iteration, by considering the paths found in the previous $i-1$ iterations. As part of our analysis, we also find a more general approximation bound for SGA, given in Theorem 1, generalizing the one in [68] using the curvature of the submodular function.

We assume that there is an $\eta \geq 1$ approximation algorithm for submodular OP.

Theorem 1. *Algorithm 4 (SGA) gives a $k_f + \eta$ approximation for MOP, where η is the approximation factor for OP and $k_f \in [0,1]$ is the total curvature for the reward function $f(\cdot)$.*

The proof of Theorem 1 is given in the appendix. Note that this bound $k_f + \eta$ generalizes the $1 + \eta$ bound in [68]. We now use this result to prove our main result.

3.7.2 Analysis for Algorithm 3

Theorem 2. *Algorithm 3 returns a set $\mathcal{S}$ such that*

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{\max[1-k_f, \frac{1}{\alpha+1}, \frac{1}{N-\alpha}]}{k_f+\eta} f^*$$

where η, k_f are the same as that defined in Theorem 1; k_f is the curvature of objective function f; and $\mathcal{A}^*(S)$ is the optimal removal set of $\mathcal{S}$; and f^* is the optimal solution to RMOP.

The omitted proofs can be found in the appendix. Our proof builds on the ones in [71,75]. The main difference is that the proof in [71,75] is suitable for picking a subset of elements directly from a given ground set, while our proof (and algorithm) deals with finding the set of paths, each of which consists of several elements (vertices). All the proofs relies on the property that for any $s_1 \in \mathcal{S}_1$ (*bait set*), $s_2 \in \mathcal{S}_2$ (*reward set*), the inequality $f(s_1) \geq f(s_2)$ is always satisfied. Besides, we need to consider the fact that the single robot submodular OP, which is NP-hard, can only be

solved approximately whereas this step is trivial in the earlier work.

Now, we describe the three subroutines that can be employed for solving OP. We start with the most general case where the reward function g is submodular and the budget for each robot must be strictly enforced.

Corollary 2. *If recursive greedy algorithm [67] is used as a subroutine to solve OP and additionally each robot has a predefined terminal vertex, then η in Theorem 2 equals to $\log(OPT)$. Here OPT is the reward collected by the optimal algorithm. The running time of the resulting algorithm is quasi-polynomial since the running time of recursive greedy is quasi-polynomial.*

Next, consider the variant where g is still submodular, but each robot is allowed to exceed its predefined budget by a bounded amount.

Corollary 3. *If General Cost-Benefit (GCB) approximation algorithm [82] is used as a subroutine for OP and we are allowed to relax given budget to $\frac{\psi(n)K_c}{\beta(1+\beta(K_c-1)(1-k_c))}B$, then η in Theorem 2 equals to $2(1-e^{-1})^{-1}$. Here, $\psi(n), \beta, K_c, k_c$ as defined in [82]. The GCB algorithm runs in polynomial time.*

Finally, consider the case where g is modular. Here, we get the strongest guarantee with no relaxations to RMOP.

Corollary 4. *If the reward function g is modular, then using the approximation algorithm for OP [56] yields an $\eta = 4$ in Theorem 2. The running time of the algorithm [56] is polynomial.*

3.7.3 Running Time

MCTS is an anytime algorithm and can converge to the optimal solution as computational time increases. In applications, the running time is decided by the available computational budget.

Next, we will mainly focus on the running time analysis of the proposed Algorithm 1.

Let t_{OP} be the time needed to solve a single robot OP and t_f be the time to evaluate the submodular function of a robot path. Suppose that the basic operations like sorting and comparison take one unit of time. Line 2-5 involves solving OP for N times, and it takes $O(Nt_{OP})$. Inside the while loop, for line 8, the sorting will take $O(N\lg N)$, and evaluation of the submodular function will take $O(Nt_f)$. Lines 9 and 10 take constant time. SGA (line 11) will take $O((N-\alpha)t_{OP})$. Line 13-19 involves $O((N-\alpha)\alpha)$ comparisons and takes $O(Nt_f)$ to evaluate submodular functions. Since sorting and comparing operations is much faster than computing OP and evaluating submodular functions, the overall running time inside the while loop will be dominated by $O(N(t_f + t_{OP}))$. Suppose that the while loop terminates after n_w loops, which can be upper-bounded as follows. Let n_i be the number of feasible paths for robot i and $n_p = \max_i n_i$. By Corollary 1, the while loop will surely terminate when the reward of each path in $\mathcal{M}$ cannot be increased anymore. The path in $\mathcal{M}$ corresponding to robot i can be improved at most n_i times. As a result, the total number of while loops can be upper-bounded by

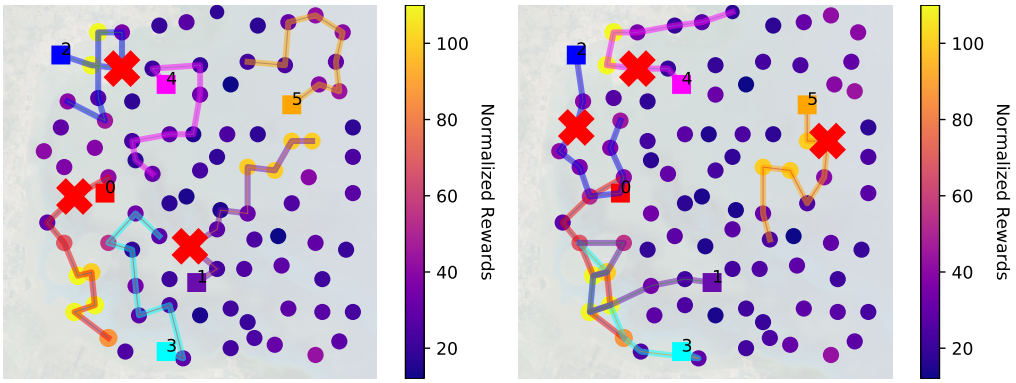
$$n_w \leq \sum_{i=1}^{N} n_i \leq n_p N.$$

Therefore, the running time for the whole while loop can be upper-bounded by $O(n_p N^2(t_{OP}+ t_f))$. It should be noted that as mentioned in Remark 1, n_w is usually very small in practice. Combining with the running time for lines 2-5 ($O(Nt_{OP})$), the running time of the algorithm is $O(n_p N^2(t_{OP} + t_f))$.

3.8 Case Study for Offline RMOP: Ocean Monitoring

In this section, we validate the performance of Algorithm 3 through numerical simulations. In particular, (1) we compare the performance of our algorithm with two baseline strategies; (2) demonstrate the robustness of the proposed algorithm against attacks that are not necessarily the worst-case ones; and (3) investigate the running time as a function of the size of the input graph and the number of robots. All experiments were performed on a Windows 64-bit laptop with 16 GB RAM and an 8-core Intel i5-8250U 1.6GHz CPU using Python 3.7.

3.8.1 Simulation Setup



(a) SGA paths (b) RMOP paths

Figure 3.4: Case study of monitoring macroalgal blooms using $N = 6$ robots assuming $\alpha = 3$ failures. Colored dots indicate locations to be monitored along with their importance (i.e., rewards). Red crosses indicate worst-case attacks found using brute force. Paths returned by the proposed algorithm manage to cover one of the three important areas (lower left corner) while SGA loses all three. The background map is part of the Yellow Sea, where green tides have prevailed every summer since 2007.

Application case study We use the application of monitoring a marine environment for mapping oil leaks, macroalgal blooms, or pH values. Specifically, as explained in [83], such tasks usually

call for the collaboration of multiple sensors including satellites, which can provide coarse prior information on the concentration of the phenomenon of interest, and mobile robotic sensors, which can use the prior information for targeted data collection. Using this as motivation, we consider a scenario where prior information from satellites (for example) can be used to define an importance map of the environment to be monitored. Fig. 3.4 shows the setup which consists of 96 vertices placed in the environment. The vertex's color reflects the vertex's importance, which gives the reward associated with visiting that vertex. Here, the single robot function, $g(\mathcal{P}_i)$, is a modular reward. The cost along the edges is the Euclidean distance between the vertices. Assuming the unit speed of travel, the cost of a path reflects the travel time of the robot. In all the instances, each robot is given a budget of $B = 60$ units.

Baseline algorithms Since we introduce RMOP in this paper, there is no other efficient algorithm to compare the performance with directly. One option is to compute the optimal solution (using, for example, brute-force enumeration) which quickly becomes intractable. Instead, we choose two approximation algorithms for MOP, the non-adversarial version, as baselines. The first one uses the sequential greedy assignment for all N robots (we refer to it as SGA) where the path for robot i is based on the paths computed for robots 1 through $i-1$. The second baseline is the naive greedy algorithm where each robot naively (without considering the travel cost) and greedily (without considering other robots) maximizes its rewards (we refer to it as NG).

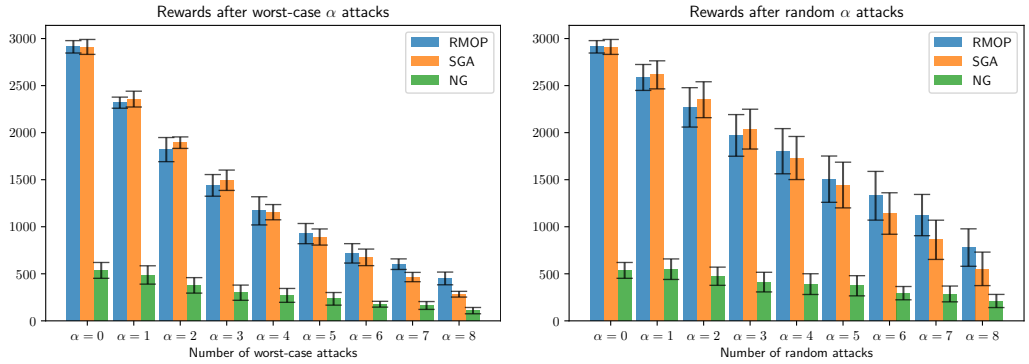
For both SGA and the proposed algorithm, we use the GCB algorithm solving OP due to its efficiency and ease of implementation. Specifically, we implement GCB using details provided in [82]. When running GCB, we simply set the relaxed budget itself to be B.

Attack models Our algorithm is designed to give performance guarantees against worst-case attacks. However, in practice, we would like for any algorithm to be robust to not just the worst-case attacks but also other attacks. Therefore, we evaluate two other types of attacks besides worst-case attacks. The details are provided in the next subsection.

3.8.2 Results

Fig. 3.4 shows a qualitative example comparing our proposed algorithm and SGA with $N = 6$ and $\alpha = 3$. Not surprisingly, the six paths found by SGA do not have any overlap, but the ones found by our algorithm do. As a result, the worst-case attack takes away all three robots covering the important regions in SGA, whereas one of the three regions is still covered with our algorithm. The worst-case attacks were computed using brute force.

Next, we present quantitative results. In all the following figures, the error bar shows the variance of 20 trials where the starting robot positions are randomly chosen.



(a) (b)

Figure 3.5: Results for Algorithm 3 and the baseline algorithms. (a) Rewards after worst-case attack with increasing α and $N = 10$. (b) Rewards after random α attacks and $N = 10$.

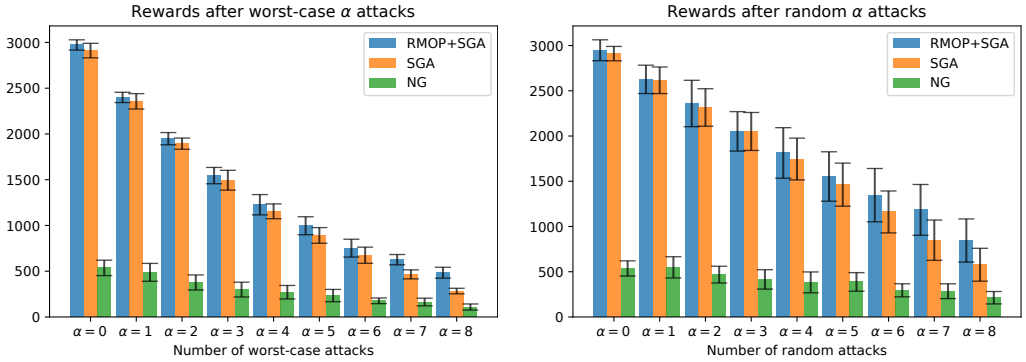
Fig. 3.5a shows the comparison between our algorithm for RMOP with SGA and NG as α increases with $N = 10$. The bars show the rewards collected by the robots after attacks. Our

algorithm returns paths that are slightly worse than SGA when α is small. This is not surprising since our algorithm will have overlapping paths whereas SGA will not. NG is the other extreme since each robot plans for itself which can lead to a high degree of overlap. Though our algorithm has only comparable performance to SGA when α is relatively small, the reward gap is small. As α increases, our algorithm gradually significantly outperforms SGA. For example, when $\alpha = 8$ our algorithm yields a reward of 451 whereas SGA only yields 283 on average. As a result, in general, the overall performance of the Algorithm 1 can be trusted especially for the large α. Moreover, in practice, since we cannot decide the threshold above which Algorithm 1 significantly outperforms SGA, we can run SGA and Algorithm 1 in parallel for offline planning if the planning budget permits and select the one that returns higher rewards.

Specifically, for a given offline RMOP instance, we first use Algorithm 1 to find a solution $\mathcal{S}$ and compute the paths left after worst-case failures $\mathcal{S} \setminus \mathcal{S}(\mathcal{A}^*)$. Then, we know that the lower bound of the reward that we are guaranteed to obtain is $f(\mathcal{S} \setminus \mathcal{S}(\mathcal{A}^*))$. Similarly, we can use SGA to compute a solution $\mathcal{S}'$ and find what is left $\mathcal{S}' \setminus \mathcal{S}'(\mathcal{A}^*)$ after the worse-case failures. In theory, $f(\mathcal{S}' \setminus \mathcal{S}'(\mathcal{A}^*))$ can be arbitrarily bad compared to $f(\mathcal{S} \setminus \mathcal{S}(\mathcal{A}^*))$. In practice, $f(\mathcal{S}' \setminus \mathcal{S}'(\mathcal{A}^*))$ may be greater than $f(\mathcal{S} \setminus \mathcal{S}(\mathcal{A}^*))$ in some cases. The mixed strategy is that if $f(\mathcal{S}' \setminus \mathcal{S}'(\mathcal{A}^*)) > f(\mathcal{S} \setminus \mathcal{S}(\mathcal{A}^*))$, we will use $\mathcal{S}'$ otherwise we will use $\mathcal{S}$. In this way, we can not only preserve the performance guarantee but also improve the empirical performance.

We conducted an experiment to compare SGA and the mixed SGA+Algorithm 3. The result is shown in Fig. 3.6. As shown in Fig. 3.6a, the mixed strategy collects more reward on average as compared to the SGA, across all α when the worst-case attack happens. We observe the same trend for the random attack case shown in Fig. 3.6b.

Next, we evaluate the performance of our algorithm when the attack model does not match



(a) (b)

Figure 3.6: Results for the mixed strategy and the baseline algorithms. (a) Rewards after worst-case attack with increasing α and $N = 10$. (b) Rewards after random α attacks and $N = 10$.

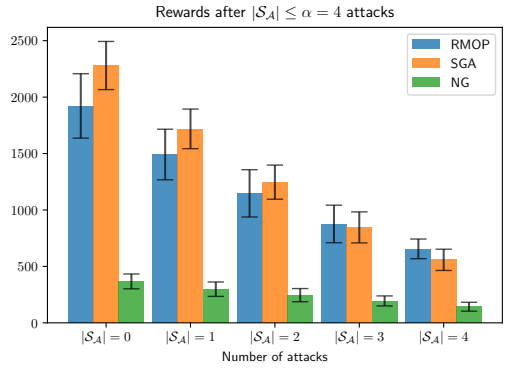
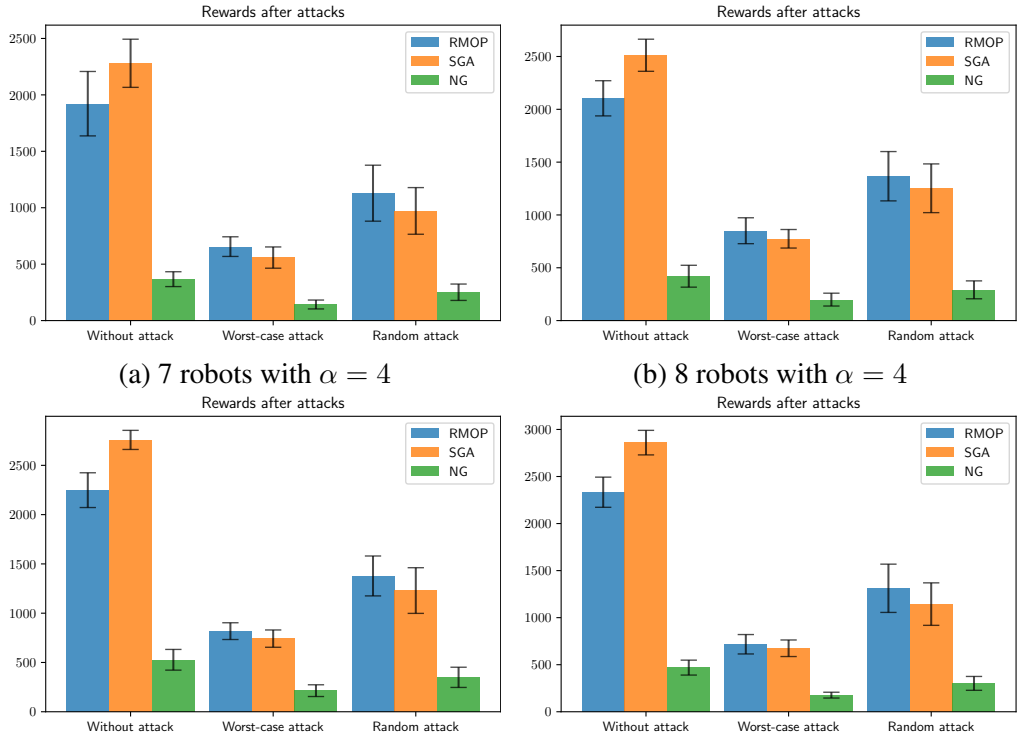


Figure 3.7: Rewards after worst-case attacks of increasing sizes, $|\mathcal{S}_\mathcal{A}| \leq \alpha$. Here the planner uses $N = 7$ and $\alpha = 4$.

the worst-case one assumed during planning. The goal is to verify the robustness of the algorithm to other attack models. Fig. 3.5b shows the comparison between our algorithm and the two baselines as α varies when the attacked robots are randomly chosen. Our algorithm still plans to assume worst-case attacks. We observe the same trend with random attacks as with the worst-case ones — as α increases, our algorithm outperforms SGA.

Fig. 3.7 shows results for the case where we construct paths assuming $\alpha = 4$ robots will be attacked but in practice only $|\mathcal{S}_\mathcal{A}| \leq \alpha$ robots suffer from worst-case attacks. SGA performs better than our algorithm when the number of robots actually attacked $|\mathcal{S}_\mathcal{A}|$ is far from the designed value of α. As the actual number of robots attacked increases and $|\mathcal{S}_\mathcal{A}|$ approaches



(a) 7 robots with $\alpha = 4$

(b) 8 robots with $\alpha = 4$

(c) 9 robots with $\alpha = 5$

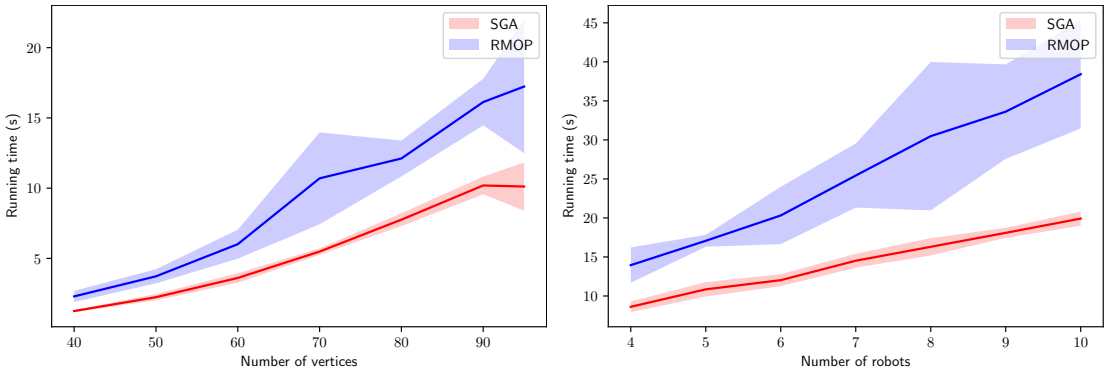
(d) 10 robots with $\alpha = 6$

Figure 3.8: Rewards after: (1) no attacks; (2) α out of N robots under worst-case attack; (3) α out of N robots under random attack.

α, our algorithm outperforms SGA. This suggests that we need to have an accurate estimate of the α before applying our algorithm, which is one limitation of this paper. However, in many robotic applications, it's possible to estimate α using historical data. Take the information gathering in the marine environment for example. We can use the number of robots that survived in the previous executions of the mission to estimate α.

Fig. 3.8 shows the evaluation when there are (1) no attacks; (2) worst-case attacks; and (3) random attacks for four configurations of N and α. In all three cases, our algorithm still plans the paths assuming worst-case attacks for the given value of α. When there are no attacks (first set of bars in each subfigure), SGA outperforms our algorithm as observed in previous charts. When worst-case attacks do happen (middle set of bars), the average rewards collected by the

unattacked robots employing our algorithm are better than that of SGA. This is also the case when the α attacked robots are chosen randomly (third set of bars). This trend holds for various values of N and α as shown.



(a) Running time w.r.t. vertices (b) Running time w.r.t. robots

Figure 3.9: Running time of Algorithm 3 and SGA

The running time comparisons between our algorithm and SGA are shown in Fig. 3.9. We vary the number of robots as well as the size of the graph. Our algorithm takes longer than SGA which is expected since it uses SGA as a subroutine. Nevertheless, we observe similar trends in the runtime.

Discussion of Results The results show the proposed algorithm works in practice as intended. As the number of attacked robots increases (either $\mathcal{S}_{\mathcal{A}}$ or α), it outperforms SGA. Furthermore, we observe that the margin between our algorithm and SGA increases as the number of attacked robots increases. Even when our algorithm finds worse paths than SGA, they are still comparable to SGA and are significantly better than NG. We also observe that our algorithm is robust to the actual attack models — even if the attacks are not the worst-case ones, we see similar trends.

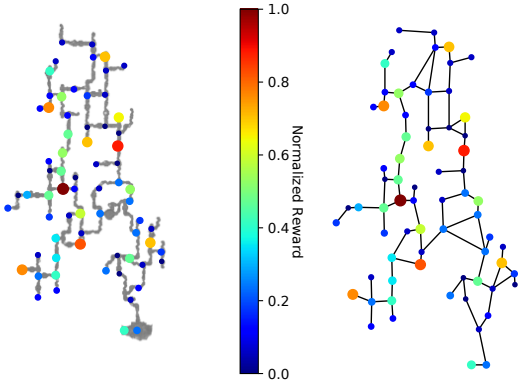
3.9 Case Study for Online RMOP: Cave Exploration

In this section, we validate the performance of Algorithm 5 for the online RMOP. In particular, we present a case study on information gathering in a tunnel and compare the performance of the team when robots and attackers adopt different strategies.

3.9.1 Simulation Setup

Application case study We use the application of information gathering in a tunnel in which we assume that some coarse prior information on the rewards of some locations is known, and a team of robots is sent to gather detailed information but at most α attackers may attack them at any locations. We also assume that communication, though maybe degraded, is available. We use a tunnel map from the Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge and use an image skeletonization algorithm and corner detection algorithm [84] to identify several points as locations of interest. Fig. 3.10a shows the setup which consists of 69 vertices placed in the environment. The color and the size of the node reflect the importance of that vertex which gives the reward associated with visiting that vertex. The corresponding graph abstraction is shown in Fig. 3.10b in which we assume that it takes one unit of time to transit between two adjacent nodes. Even though adding some virtual nodes on the long edges will make this assumption better justified, for simplicity, we ignore these virtual nodes. Here, the single robot function, $g(\mathcal{P}_i)$, is a modular reward. The cost along the edge is the distance between two adjacent nodes which is defined in the image coordinate as the shortest distance in the skeleton image (739×520 pixels). In this case study, there are four robots in the environment and $\alpha = 2$ attackers, and each robot has a budget of $B = 500$ units.

We also test the performance of our strategy compared to other strategies in the randomly generated graphs. We generate four 15×15 grid graphs, whose edges are of unit length, to represent the environments. To account for the sparsity of the tunnel environment, half of the edges are randomly removed in each graph, but the graph remains connected. For each graph, the reward of each node is generated by sampling a number from one exponential distribution. We use different rate parameters for different graphs. It should be noted that our algorithm doesn't depend on the particular distribution of rewards. We choose the exponential distribution just for its non-negative support. In all instances, there are four robots in the environment and $\alpha = 2$ attackers, and each robot has a budget of $B = 8$ units.



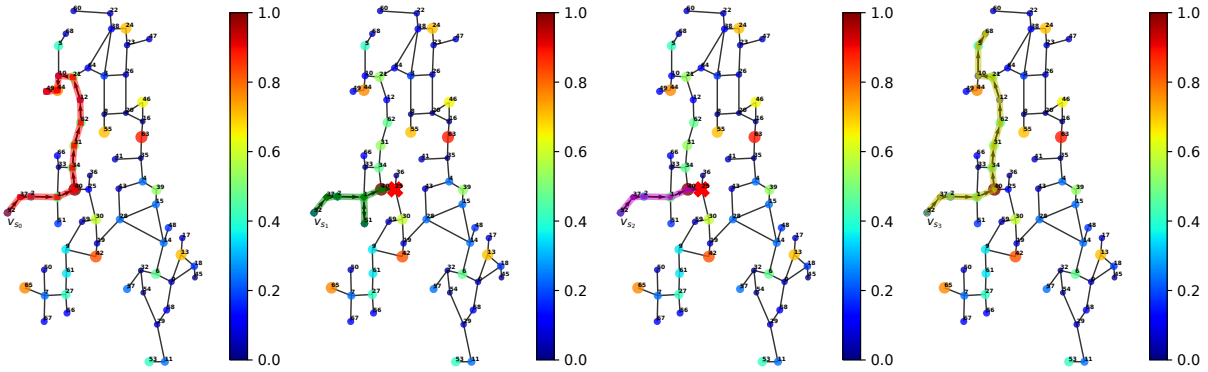
(a) Tunnel map (b) Tunnel graph

Figure 3.10: A tunnel map for a case study of information gathering. Colored dots with various sizes indicate locations to be visited along with their importance, i.e., rewards. (a) A tunnel map with 69 locations of interest. (b) The graph abstraction of the tunnel map. The tunnel map is from the Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge.

Strategies We consider two strategies for robots including MCTS with adversaries (Algorithm 5, two-player search, we refer it as MA) and naive MCTS (don't consider failures/attacks, one-player search, we refer it as M) and two strategies for attackers including MCTS with adversaries (Similar to Algorithm 5, two-player search, we refer it as MA and RandomMove (randomly

choose an available action each time, we refer it as R). Problem 4 is simulated as a two-player game in which at each step attackers first use their strategy to choose one available action, and then robots choose one action. Such a process continues until all robots run out of budget.

3.9.2 Results



(a) robot 0 (b) robot 1 (c) robot 2 (d) robot 3

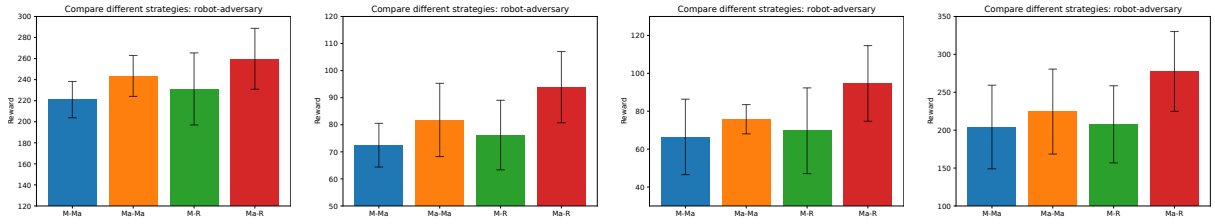
Figure 3.11: A case study of information gathering in a tunnel with $N = 4$ robots assuming $\alpha = 2$ failures/attacks. Colored dots indicate locations to be visited along with their importance (i.e., rewards). The red cross indicates the attacks found when robots and attackers play the two-player sequential game, and both use MCTS with adversaries. v_{s_i} represents the starting position of the robot i. Attackers launch the first attack when they observe that robots 0 and 3 reach node 34 and robot 2 reach node 25 at the fifth step and launch another attack later when robot 1 reaches node 25 at the seventh step. (a) robot 0 follows the path $52 \to 37 \to 2 \to 1 \to 40 \to 34 \to 31 \to 62 \to 12 \to 21 \to 10 \to 44 \to 49$. (b) robot 1 follows the path $52 \to 37 \to 2 \to 1 \to 51 \to 1 \to 40 \to 25$ and is attacked after two steps. (c) robot 2 follows the path $52 \to 37 \to 2 \to 1 \to 40 \to 25$ and is attacked after three steps. (d) robot 3 follows the path $52 \to 37 \to 2 \to 1 \to 40 \to 34 \to 31 \to 62 \to 12 \to 21 \to 10 \to 5 \to 68$.

Fig. 3.10 shows the tunnel map from the DARPA subterranean challenge. Fig. 3.10a is the original tunnel map with 69 locations of interest and Fig. 3.10b is the corresponding abstract graph representation of the tunnel map. Fig. 3.11 shows a qualitative example of how robots and attackers behave in a two-player sequential game when both of them use MCTS with adversaries. In each step, attackers will first grow the search tree based on what they observed so

far and choose an action. Then, robots will grow the search tree and choose one action. Such a process continues until the budgets of robots are used up. As a result, robot 0 follows the path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 34 \rightarrow 31 \rightarrow 62 \rightarrow 12 \rightarrow 21 \rightarrow 10 \rightarrow 44 \rightarrow 49$; robot 1 follows a path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 51 \rightarrow 1 \rightarrow 40 \rightarrow 25$ and is attacked at node 25; robot 2 follows a path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 25$ and is attacked at node 25; robot 3 follows a path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 34 \rightarrow 31 \rightarrow 62 \rightarrow 12 \rightarrow 21 \rightarrow 10 \rightarrow 5 \rightarrow 68$. As shown in Fig. 3.11, attackers launch the first attack when they observe that robots 0 and 3 reach node 34 and robot 2 reaches node 25 because if they don't attack at that moment robot 2 will move downward to collect more rewards. Attackers launch another attack later when robot 2 reaches node 25 and may collect more rewards from the nodes below.

Fig. 3.12 shows the results when robots and attackers use different strategies in four graphs. Robots are randomly initialized in different vertices, and for each initialization, the two-player game is conducted 20 times. The collected reward of the team is the sum of the rewards of nodes visited. As shown in Fig. 3.12, when attackers use MCTS with adversaries (first two bars blue and orange), robots can collect more reward on average if they also use the MCTS with adversaries (orange) compared to the case where they use a naive MCTS without considering attacks. If attackers use a random strategy (last two bars green and red), the MCTS with adversaries strategy (red) is also on average better than a naive MCTS without considering attacks (green). Moreover, robots can collect more rewards on average if attackers just select an action randomly (green and red compared to blue and orange).

We empirically evaluate how the number of iterations influences the performance of MCTS. As shown in Fig. 3.13, as the number of iterations increases, the average reward also improves. However, the rate of improvement shows diminishing returns. When the number of iterations



(a) (b) (c) (d)

Figure 3.12: Collected rewards for different strategies in four different maps: (1) M-MA: robots use one-player MCTS (M) and attackers use MCTS with adversaries (MA); (2) MA-MA: both robots and attackers use MCTS with adversaries; (3) M-R: robots use one-player MCTS (M) and attackers attacks randomly (R); (4) Ma-R: robots use MCTS with adversaries and attackers attacks randomly (R).

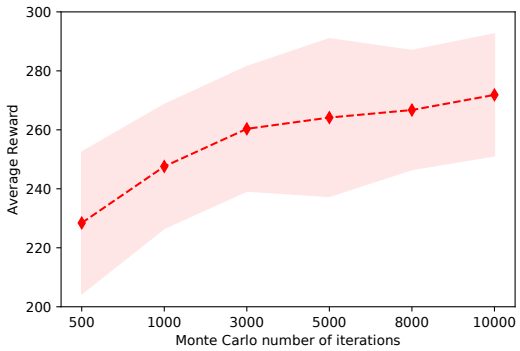


Figure 3.13: The effect of increasing the number of iterations in MCTS with $N = 4$ robots and $\alpha = 2$ failures/attacks.

increases from 500 to 10000, the average collected reward increases about 18% (from 229 to 271), suggesting that too many iterations may not be necessary in practice.

3.10 Conclusion

We introduced a new problem, termed the Robust Multiple-Path Orienteering Problem, in which we seek to construct a set of paths for robots such that even if a subset of robots fails, the rest of the team still performs well. We consider two types of RMOP. In the offline RMOP in which robots cannot communicate with each other or the base station during the execution of tasks, we provided a general approximation framework for the offline RMOP, which builds on

bounded approximation algorithms for OP and the sequential greedy assignment framework. We showed three variants of the general algorithm that use three different subroutines for OP and still yield a bounded approximation for RMOP. In addition to theoretical results, we presented empirical results showing that our algorithm is robust to attacks other than the worst-case ones. We also compare our performance with baseline algorithms and show that our algorithm yields better performance as more and more robots are attacked. In the online version, RMOP is modeled as a two-player sequential game and solved adaptively in a receding horizon fashion based on Monte Carlo Tree Search (MCTS). Simulation results show that MCTS with adversaries performs better on average than the MCTS without considering attacks/failures.

Chapter 4: Multi-Robot Information Gathering with Partial Knowledge of the Environmental Model: Bayesian Case

4.1 Overview

In this chapter, we consider using a heterogeneous team of Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV) for long-term information gathering tasks, e.g., surveillance, and environment monitoring. The UAVs and UGVs need to finish the task cooperatively, and the UGVs need to recharge the UAVs as mobile rechargers periodically. Due to the disturbances (e.g., wind) in the environment, the energy consumption of a UAV is stochastic, and we need to take such disturbances into account in the planning phase. Generally, we are interested in such problems: given the distribution of the stochastic disturbances in the environment, how to coordinate UAVs and UGVs to achieve high task performance and low failure rate (UAVs being out of charge)? We studied this type of problem for a team consisting of one UAV and one UGV as well as a team of multiple UAVs and UGVS.

4.2 Risk-aware UAV-UGV Recharging Rendezvous: One UAV and One UGV

Unmanned Aerial Vehicles (UAVs) are increasingly being sought in applications such as surveillance, environmental monitoring, and agriculture due to their ability to monitor large areas

in a short period of time. One bottleneck in practice that limits their application is the limited battery capacity, especially for multi-rotor UAVs. One way to overcome this bottleneck is to use a team of aerial and ground vehicles for such tasks, in which the UGV can work as a mobile recharging station and will recharge the UAV during long-range operations. The key to achieving such cooperation on the decision-making level is to design efficient routing algorithms that can tell robots *which task node to visit next*, and *when and where the UAV should be recharged*. Moreover, the rate of battery discharge of a UAV is stochastic in the real world. The routing algorithm should be able to deal with such uncertainties, e.g., trade-off task performance with failure risks.

In this section, we consider the cooperative routing problem with a team of a single UGV that can work as a mobile charger and a single energy-constrained UAV, in which the UAV and UGV need to complete a task by visiting task nodes distributed throughout the task area. The UGV can only move on the road network, but the UAV can directly fly between any pair of nodes (assuming it has enough charge). Given the task nodes to visit and the stochastic energy consumption model of the UAV, we are interested in finding a routing strategy for the UAV and the UGV such that the expected time to finish the task is minimized and the probability of running out of charge is less than a user-defined tolerance. Such problems can be formulated within the stochastic programming (SP) framework [85]. However, since we need to consider not only routes but also recharge decisions and chance constraints, SP-based formulation would involve too many variables, rendering the formulation only solvable for very small instances.

To this end, we propose to find the routing strategy in two decoupled phases. In the first phase, a higher-level planner finds deterministic routes for the UAV and the UGV without considering stochasticity in energy consumption based on the task requirement. In the second phase,

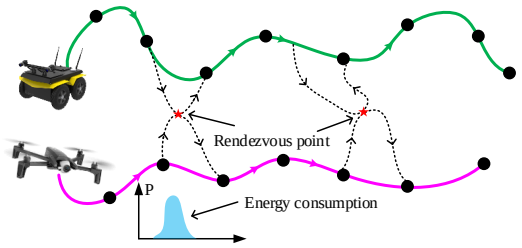


Figure 4.1: An illustrative example of the rendezvous problem considered in this paper. When the UAV and UGV are executing tasks, they need to decide when and where to rendezvous to replenish the battery of the UAV while minimizing the travel time of the UAV and satisfying the risk constraint induced by stochastic energy consumption. When they need to rendezvous, they will deviate from their task and meet at a chosen rendezvous location.

a risk-aware planner will refine the routes generated from the previous phase to find when and where to rendezvous to satisfy the chance constraint while minimizing the time to finish the UAV task[1]. Our focus in this paper is mainly on the second phase. We formulate our risk-aware refinement as a Markov Decision Process (CMDP), in which the chance constraint is modeled as the secondary cost in the constraint. To the best of our knowledge, this is the first CMDP-based formulation for the UAV-UGV routing problems under energy consumption uncertainty. We use Linear Programming (LP) to find the optimal stationary policy. We validate our formulation and the solution in an Intelligence Surveillance and Reconnaissance (ISR) mission.

The routing of energy-constrained UAVs with stationary recharging stations or assistive UGVs has been studied extensively [87–90]. Even with deterministic environmental changes or stationary conditions, this problem can be reduced to the Traveling Salesman Problem (TSP), [91, 92] making it an NP-hard problem.

The cooperative UAV and UGV routing problem has been studied from different perspectives and thus received various formulations in the literature. It is most commonly formulated

[1]We focus on minimizing the time taken for the UAV task instead of the total time which would be the maximum of the UAV and UGV travel times. If the UGV task takes longer than the UAV, the UGV simply executes the remaining portion of its route once the UAV's task is done. Therefore, optimizing the UAV time is appropriate. Furthermore, In many applications [86], the UGV's task is to act as a mobile recharging station simply

as a type of vehicle routing problem. Manyam et al. [93] use a team of one UAV and one UGV with communication constraints to cooperatively visit targets. Along with an exact solver to Mixed Integer Linear Programming (MILP) formulation, they also provide heuristic reduction to the generalized traveling salesman problem (GTSP). Maini et al. [94] present a two-fold strategy: first, they identify feasible rendezvous points and then formulate a MILP to find the optimal routes for the UAV and UGV. Thayer et al. [95] present a solution to the Stochastic Orienteering Problem, where the objective is to maximize the sum of rewards associated with each visited node while constrained by the maximum budget over edges with stochastic cost.

Murray and Chu [96] introduced the flying sidekick TSP (FSTSP) for parcel delivery systems, which was later adopted in last-mile delivery applications using drones [97, 98]. In literature, the term multi-echelon scheme is often used for systems where delivery consists of multiple layers. Specifically, the two-echelon vehicle routing problem (2E-VRP) is concerned with finding minimal-cost routes to deliver packages with trucks/UGVs and drones [88, 99]. An important differentiation from the original vehicle routing problem is the synchronization of UAV and UGV tasks.

Learning-based approaches have been used to address cooperative UAV and UGV routing problems. Ermugan et al. [100] also propose a two-phase approach. First, they find a route for UAVs without taking into account the energy constraints. Then, the planner learns to insert into the route recharging stations and replans a new TSP route. Reinforcement learning has also proven to be a possible approach to solving this problem [101].

In our previous work, we studied cooperative planning with a single UGV and an energy-constrained UAV as well [86, 91, 102]. Our proposed approach in [102] demonstrated how to maximize the number of sites visited in a single charge in conjunction with the ability to land

a UAV on top of a UGV to be transported to the next take-off site. We extended this in [86] to allow the UAV also to be recharged while either being transported or stationary on the UGV. We extended the latter to the area coverage path planning problem by formulating it as a GTSP [91]. Here, we extend this body of work by introducing the stochasticity of the UAV's energy consumption and by assuming that the UGV has its own required set of tasks to be carried out. To the best of our knowledge and based on the presented literature review, none of the works takes into account the stochastic nature of energy consumption.

4.2.1 Preliminaries

The cooperative routing problem studied in this section involves one UAV and one UGV. The UAV and the UGV are executing tasks, which are given by some task planners as shown in Section 4.2.4.1. The UAV needs to visit a sequence of task nodes to finish the task, but its battery may not be enough to finish the task in a single flight without recharging. Also, the energy consumption of the UAV is stochastic. The UAV needs to decide *when* and *where* it should rendezvous with the UGV to replenish the battery while minimizing the total travel time to finish the task. When the UAV decides to rendezvous with the UGV to replenish power, both the UAV and the UGV will take a detour (as defined in Sec. 4.2.1.4) from their respective tasks and go back to their tasks after recharging.

At a high level, the problem studied in this paper is stated below.

Problem 5 (Risk-aware UAV-UGV rendezvous)**.** *Given a route of nodes for the UAV $\mathcal{T}_A$, a route of nodes for the UGV $\mathcal{T}_G$, and the stochastic energy consumption model and battery capacity of the UAV, find a policy for the UAV to decide when and where to rendezvous with the UGV*

for recharging such that the total travel time is minimized and the probability of running out of charge during flight is less than a given tolerance.

Next, we will present the setup and assumptions used in this paper. Then we will present our CCMDP-based formulation and show how to transform a CCMDP into a CMDP.

4.2.1.1 Environment and Task Model

Our problem considers a two-dimensional Euclidean space, which consists of a road network graph $G = (V_r, E)$ and a set of task points V_t for the UAV to visit.

The UGV has to move on the road network and its task is specified as a sequence of road network nodes. UAV's task is specified by some task planners using nodes in V_t. More details on the task planner will be discussed in Section 4.2.4.1. Both UAV and UGV should follow the task specification to visit the task nodes in order, and they will deviate from the task route to rendezvous when necessary.

4.2.1.2 Vehicle Motion Model

The UGV will move at a fixed speed v_g when it transits between two nodes in the road network. When the UAV transits between two nodes, it will fly with either the best endurance speed, v_{be}, or the best range speed, v_{br}. The best endurance speed is the speed at which the energy consumption rate is minimized. At this speed, the propellers of the multirotor operate more efficiently than in hover, and the UAV is capable of the greatest flight duration. By contrast, when a UAV flies at the best range speed, it minimizes the derivative of energy consumption rate with respect to velocity. This flight speed results in a lower flight duration than operation at v_{be},

but will allow a greater range to be traveled per unit of energy. For a no-wind condition, the velocity of the best range is always better than the velocity of the best endurance.

4.2.1.3 Recharging and Stochastic Energy Consumption Model

We assume that it takes constant time T to finish the recharging process, including the landing/take-off and battery-swapping times.

In this paper, we only consider the power consumption when a UAV traverses the route, assuming that the power management system has reserved the power needed for computation, takeoff, and landing. As described in the transition model, the UAV will fly at a fixed speed when it transits between two nodes in the environment. However, given that constant speed, the energy consumption is stochastic considering the disturbances in the environment.

Given the distance l between two task nodes and the flying speed v, the energy consumption can be computed as

$$e_{l,v} = \int_{t=0}^{\frac{l}{v}} P_{\mathbf{\Theta}}(v)dt, \tag{4.1}$$

where $P_{\mathbf{\Theta}}(v)$ is the power consumption of the UAV when it flies at a speed v, and $\mathbf{\Theta}$ is a vector of parameters for stochastic variables.

4.2.1.4 Rendezvous Model

When the UAV reaches a node in $\mathcal{T}_A$ and decides to rendezvous with the UGV, the UAV and UGV will deviate from their task temporarily to finish the rendezvous process. There are two steps in the rendezvous process. In the first step, the UAV and UGV will meet at a rendezvous point as shown in Fig. 4.2, and in the second step, they will go to the next task node in $\mathcal{T}_a$ and

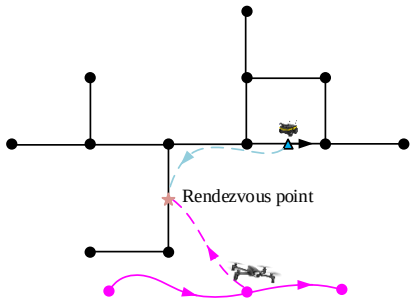


Figure 4.2: First step in the rendezvous process. The UGV (blue triangle) needs to deviate from its task node to rendezvous with the UAV at the rendezvous point (pink star). The rendezvous paths are in dashed lines.

$\mathcal{T}_g$ respectively. We want to optimize the time consumed in these two steps to find the optimal rendezvous points.

Let $d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_+$ be the distance metric between two points in the Euclidean space. We use $d_G : V_r \times V_r \to \mathbb{R}_+$ to denote the shortest path length between two road network nodes. We use $\mathcal{T}_a(k)$ to denote the position of the UAV when it decides to rendezvous at the kth node in its task route and $\mathcal{T}_a(k+1)$ to denote the next position to visit for UAV after the rendezvous. With a slight abuse of notation, we use $\mathcal{T}_g(k)$ to denote the position of the UGV in the road network when the UAV decides to rendezvous at the kth node in its task route. With the above notations, the problem of finding the rendezvous point can be stated below.

Problem 6 (Where to rendezvous)**.** *Given the positions of UAV ($\mathcal{T}_a(k)$) and UGV ($\mathcal{T}_g(k)$) at the beginning of the rendezvous process, UAV's next position to go $\mathcal{T}_a(k+1)$, UAV's flight speed v_a, UGV's transition speed v_g, and the road network G, we want to find a rendezvous point $p_r \in G$*

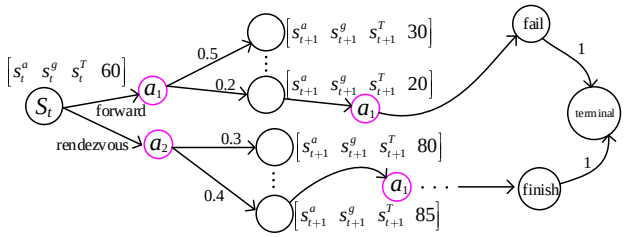


Figure 4.3: State transition graph in CMDP.

such that the time consumed in the rendezvous process is minimized. Mathematically,

$$\min_{\Delta \geq 0,\ p_r \in G} \Delta + \frac{d(p_r,\ \mathcal{T}_a(k+1))}{v_a} \tag{4.2}$$

$$s.t.\ \Delta = \max\left(\frac{d_G(\mathcal{T}_g(k),\ p_r)}{v_g},\ \frac{d(\mathcal{T}_a(k),\ p_r)}{v_a}\right). \tag{4.3}$$

In the first step of the rendezvous process, if the UAV or UGV reaches the rendezvous first, it has to wait for the other vehicle. Therefore, the time consumed in the first step is decided by the vehicle that reaches the rendezvous point later than the other. We encode this fact in the optimization problem by introducing the variable Δ, which describes the maximum time needed for both UAV and UGV to reach the rendezvous point. The time consumed in the second step of the rendezvous process is the time needed for the UAV to fly back to its next task node.

Problem 6 can be solved by iterating over the nodes in the road network as we do in the case study. However, such a method will increase the time to extract transition information for the CMDP. A more efficient way to solve Problem 6 is left for our future work.

4.2.2 Problem Formulation

4.2.2.1 Chance-Constrained Markov Decision Process

One natural choice to model the sequential decision-making problems described in Problem 5 is to use MDP. In this section, we first show how to formulate Problem 5 as a CCMDP and then show how to transform a CCMDP into a CMDP in the following section.

The rigorous definition of an MDP can be found in [103]. Here we define the MDP from the perspective of the application. The MDP corresponding to Problem 5 is defined as a tuple $\mathcal{M} = (S, A, T, C, s_0)$, where

- $S = \mathcal{T}_a \times S_g \times \mathcal{T}_g \times \mathcal{B} \cup \{s_{ob},\ s_l\}$ is the state space of the problem, where $\mathcal{T}_a$ here is used as an un-ordered set, which describes all possible positions of UAV in a task route; S_g is the set of positions of UGV and this information is needed when we compute the rendezvous points; $\mathcal{T}_g$ here is used as an un-ordered set, which describes the task nodes UGV will visit. $\mathcal{T}_g$ is included in the state space to inform the MDP about the next node the UGV needs to visit after a rendezvous. Without this information, the system will be non-Markovian; $\mathcal{B}$ is a discretized variable for describing the state of the charge of the UAV; s_{ob} is one failure state representing the out-of-charge state and the UAV will transit to this state whenever it cannot finish its task route; s_l is added as an absorbing state and UAV will transit to this state when it either finishes UAV's route or runs into a failure state. One illustrative example of state transitions is given in Fig. 4.3.

- A is the action space of the UAV. If the UAV has not finished its route and is not in a failure state, there are four actions for the UAV to choose from: 1. v_{be}: move to the next node in

$\mathcal{T}_a$ with the best endurance velocity. 2. v_{br}: move to the next node in $\mathcal{T}_a$ with the best range velocity. 3. v_{be_be}: rendezvous with the best endurance velocity. 4. v_{br_br}: rendezvous with the best range velocity. When a UAV is in a failure state or has finished its route, there is only one action that makes the system transit to the terminal state s_l.

- $T_a(s, s', a) = P(s' \mid s, a)$ is the transition function, which depends on the stochastic energy consumption model. When UAV chooses to move forward to its next task node, its battery state at the destination node is a random variable that depends on the current battery state and Equation (1). Since we have discretized the battery charging levels at each node, the probability of reaching the destination node with a given battery charge can be calculated using Equation (4.1). When it cannot reach the next task node, i.e., with non-zero probability, it will run out of charge, it transits to the failure state s_{ob}. When a UAV chooses to rendezvous, a rendezvous point is first computed by solving the Problem 6. Then the distribution of the battery remaining when it reaches the rendezvous point can be computed based on Equation (4.1). The non-positive portion of the distribution corresponds to the failure probability. After recharging, the UAV will transit to its next task node starting with a full battery. When the UAV transits to the failure state or it finishes the task route, it will transit to the terminal state s_l with probability 1 as shown in Fig. 4.3. In the terminal state s_l, the system will loop over this state.

- $C(s, s', a)$ is the cost function for the UAV. We define it as the time needed to transit between two states. If the UAV chooses to move to the next node, the cost will be time consumed during that transition. If a UAV chooses to rendezvous, the cost will be the sum of the time consumed in two steps of the rendezvous process. When the state transits to the failure state or to the terminal state, it takes zero cost.

- s_0 is the initial state of the system.

Definition 7 (Risk)**.** Let π be a policy, the risk of the policy given initial state s_0 is defined as

$$\rho^{\pi}(s_0) = \mathbf{P}(\exists\, t\; s_t = s_{ob} \mid s_0). \tag{4.4}$$

We seek the optimal policy π^* that satisfies

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{i=0}^{\infty} C(s_i, \pi(s_i))\right] \tag{4.5}$$

$$\text{s.t.}\;\; \rho^{\pi}(s_0) \leq \delta, \tag{4.6}$$

where δ is the user-specified risk tolerance.

4.2.2.2 Constrained Markov Decision Process

We can transform a CCMDP into a CMDP by introducing a new cost function $\overline{C} : S \times S \times A \to \{0, 1\}$ [104]. As shown in Fig. 4.3, when the system transits from a non-failure state to the failure state s_{ob}, it will incur a cost of one and other transitions will incur zero cost. The new cost function $\overline{C}$ is defined as

$$\overline{C}(s, a, s') = \begin{cases} 1 & \text{if } s \neq s_{ob} \text{ and } s' = s_{ob} \\ 0 & \text{else.} \end{cases} \tag{4.7}$$

As shown in [104] [Proposition 4.1], the risk can be defined using the new cost function $\overline{C}$

as

$$\rho^{\pi}(s_0) = \mathbb{E}\left[\sum_{i=0}^{\infty} \overline{C}(S_i, \pi(S_i)) \mid s_0\right]. \tag{4.8}$$

As a result, the CCMDP problem can be formulated as

$$\pi^* = \underset{\pi}{\operatorname{argmin}}\, \mathbb{E}\left[\sum_{i=0}^{\infty} C(s_i, \pi(s_i))\right] \tag{4.9}$$

$$\text{s.t. } \mathbb{E}\left[\sum_{i=0}^{\infty} \overline{C}(S_i, \pi(S_i)) \mid s_0\right] \leq \delta. \tag{4.10}$$

4.2.3 Solutions to CMDP

A CMDP can be solved using Linear Programming (LP) [105, 106]. The decision variables y in LP are the occupancy measure for each state-action pair and are defined as

$$y(s, a) = \sum_{t} \Pr(S_t = s, A_t = a). \tag{4.11}$$

The LP is formulated as:

$$\min_{y(s,a), \forall s,a} \sum_{(s,a) \in S \times A} y(s, a) C(s, a) \tag{4.12}$$

$$\text{s.t. } \sum_{s,a} y(s, a) \overline{C}(s, a) \leq \delta \tag{4.13}$$

$$\sum_{a'} y(s', a') = \mathbb{I}(s', s_0) + \sum_{s,a} y(s, a) \Pr(\mathrm{s}' \mid \mathrm{s}, \mathrm{a}) \quad \forall s' \in S \setminus \{s_l\} \tag{4.14}$$

$$y(s, a) \geq 0 \; \forall s, a, \tag{4.15}$$

where $\mathbb{I}(s', s_0)$ is a Dirac delta function that returns 1 when $s' = s_0$ and 0 otherwise. This LP corresponds to the dual linear program for MDPs [105] with one extra cost constraint (4.13), which enforces that the cost of entering the failure state be lower than the predefined risk tolerance. Constraint (4.14) is a flow conservation constraint to define valid occupancy measures and is defined by the initial state and the transition probability (see [105], ch. 8 for details). The last constraint (4.15) is added to guarantee that $y(s, a)$ is non-negative.

If LP admits a solution, we can construct the policy from the occupancy measures by normalizing them:

$$\pi^*(s, a) = \frac{y(s, a)}{\sum_{a'} y(s, a')} \forall (s, a) \in S \times A, \tag{4.16}$$

where $\pi^*(s, a)$ is the probability of taking action a in the state s in the optimal stationary randomized policy. If Eq. (4.16) has a zero denominator, which suggests that state s is not reachable from s_0, the policy for (s, a) can be defined arbitrarily.

4.2.4 Simulation

In this section, we first present a qualitative example to show what the input and output look like for our problem. Next, we study how system parameters (different risk tolerances) influence the rendezvous behaviors between the UAV and the UGV. Then, we present quantitative results for the ISR application that motivates our research. Specifically, we will use Monte Carlo (MC) simulations to evaluate 1. the satisfaction of the risk constraint for the policy constructed from LP; 2. the effectiveness of the policy in minimizing the expected task duration; 3. the risk tolerance-task duration Pareto curves. Moreover, the running time of LP for CMDP is empirically evaluated. All experiments are conducted using Python 3.8 on a PC with the i9-8950HK

processor. LP is solved using Gurobi 9.5.0.

4.2.4.1 Task Route Planner

The task routes $\mathcal{T}_a$ and $\mathcal{T}_g$ used in Problem 5 can be either generated jointly by some existing task planners [86, 93] or can be generated by separately by different task planners. In our case study, the task for the UGV is to persistently monitor nodes A, B, and C (blue squares in Fig. 4.5). The task nodes for the UAV are red dots in Fig. 4.5 and the task route (from node 0 to 18 and back to 0) is generated by a planner for Traveling Salesman Problem (TSP).

4.2.4.2 System Models

Table 4.1: Coefficients for stochastic energy consumption model

	b_0	b_1	b_2	b_3	b_4	b_5
Value	-88.77	3.53	-0.42	0.043	107.5	-2.74

The UAV task and UGV tasks are from our ongoing project on intelligence, surveillance, and reconnaissance (ISR) as shown in Fig. 4.5a. In this project, we are interested in the case where $\delta = 0.1$. UAV has about 240 KJ energy; its best range speed and best endurance speed are 14 m/s and 9.8 m/s, respectively. UGV moves at 4.5 m/s. The rendezvous process will take 300 seconds.

We consider two sources of stochasticity in the energy consumption model of UAVs: weight and wind velocity contribution to longitudinal steady airspeed. The deterministic energy consumption model of the UAV is a polynomial fit constructed from analytical aircraft modeling

data, given as

$$P(\boldsymbol{v_\infty}) = b_0 + b_1\boldsymbol{v_\infty} + b_2\boldsymbol{v_\infty^2} + b_3\boldsymbol{v_\infty^3} + b_4\boldsymbol{w} + b_5\boldsymbol{v_\infty}\boldsymbol{w}, \tag{4.17}$$

where b_0 to b_5 are coefficients, and their experimental values are listed in Table 4.1. Figure 4.4 shows the agreement between the polynomial regression fit model and the analytical data that it was derived from.

Weight is randomly selected following a normal distribution with a mean of 2.3 kg and a standard deviation of 0.05 kg, $\boldsymbol{w} \sim \mathcal{N}(\mu_{\boldsymbol{w}}, \sigma^2_{\boldsymbol{w}})$. Vehicle airspeed, v_∞, is the sum of the vehicle ground speed, v, and the component of the wind velocity that is parallel to the vehicle ground speed, ignoring sideslip angle and lateral wind components.

$$v_\infty = |\bar{v_g} + \cos(-\psi)\boldsymbol{\xi}_{\mathrm{a,b}}| \tag{4.18}$$

The longitudinal wind speed contribution is derived from two random parameters: wind speed and direction. Wind speed is modeled using the Weibull probability distribution model of wind speed distribution, $\boldsymbol{\xi}_{a,b}$, with a characteristic velocity $a = 1.5$ m/s and a shape parameter $b = 3$. This is representative of a fairly mild steady wind near ground level. Wind direction ψ is the heading direction of the wind and is uniformly randomly selected on a range of $[0, 360)$ degrees.

4.2.4.3 Simulation Results

An illustrative example of the input and the output of the problem considered is shown in Fig. 4.5. The input of the problem is shown in Fig. 4.5a, which consists of UAV task nodes (red dots), UGV task nodes (blue square), and road network (black nodes). Fig. 4.5b shows one

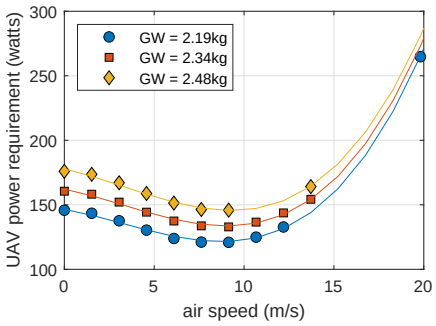
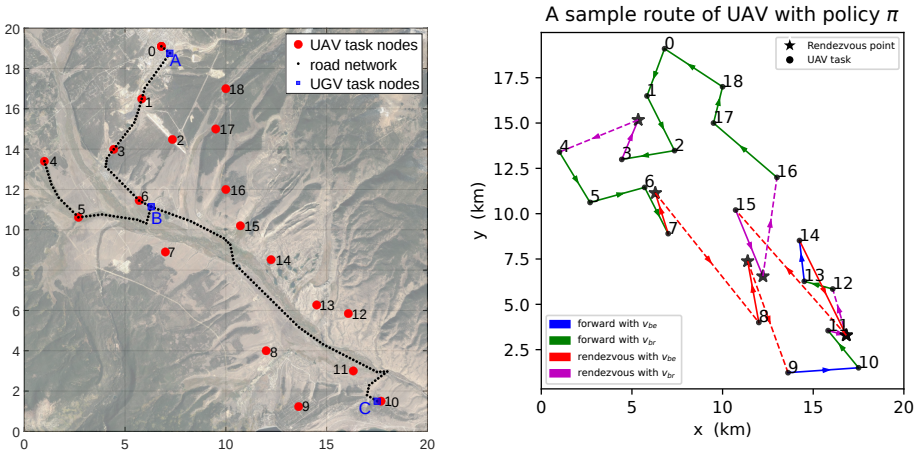


Figure 4.4: Comparison of analytical data used to derive the polynomial regression fit model of UAV power requirement at three weights and across 11 airspeeds.

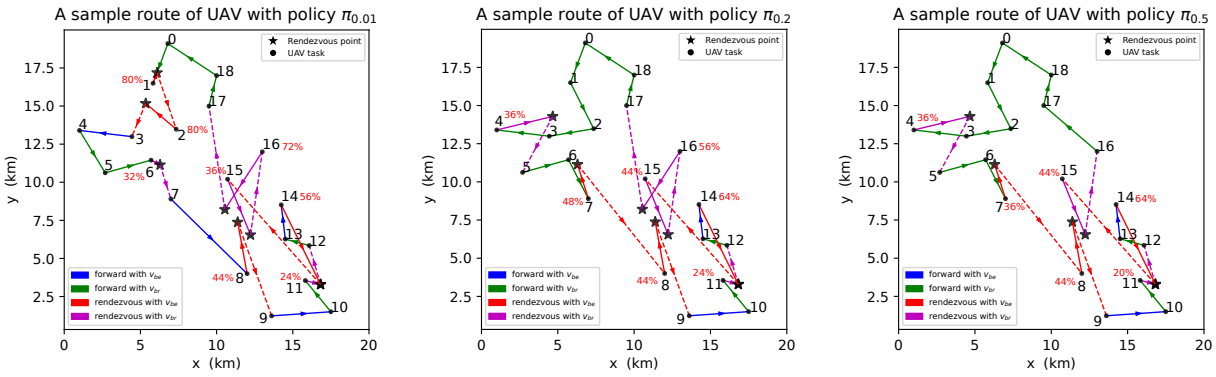


(a) (b)

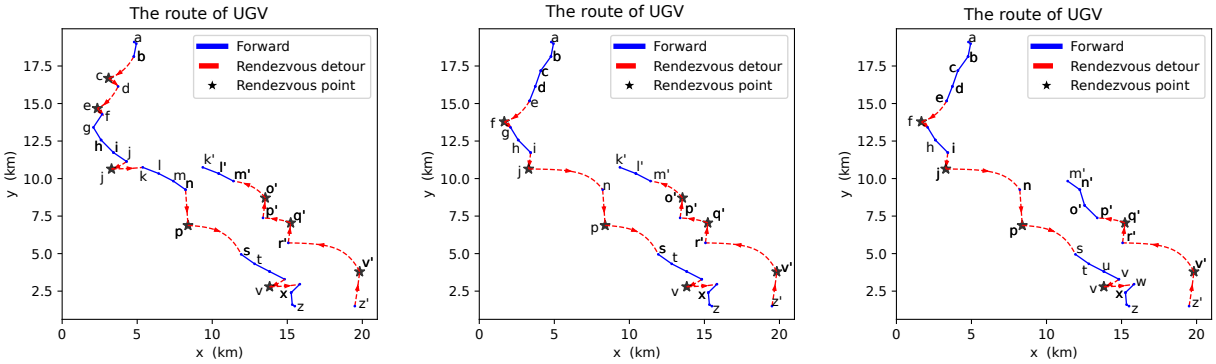
Figure 4.5: A qualitative example illustrates how UAV and UGV rendezvous with each other under the policy π obtained by solving the CMDP. The risk tolerance is set to be $\delta = 0.1$ in this case study. (a) The input of the risk-aware rendezvous problem. (b) One sample route of UAV when it executes the policy π.

sample route of UAV when the system executes the policy computed by LP. UAV's route starts from node 0. When the UAV reaches node 4, it will choose to rendezvous with UGV using the best range speed in a rendezvous point, which is denoted as a star, and then go to its next task node 5. Similarly, the UAV will rendezvous with the UGV when it reaches nodes 7, 8, 11, 14, and 15.

Next, we show how different risk tolerances influence rendezvous behaviors under our CMDP formulation. In these experiments, we set the risk tolerance δ to be 0.01, 0.2, and 0.5.



(a) Low risk tolerance ($\delta = 0.01$) (b) Medium risk tolerance ($\delta = 0.2$) (c) High risk tolerance ($\delta = 0.5$)



(d) UGV route corresponding to Fig. 4.6a (e) UGV route corresponding to Fig. 4.6b (f) UGV route corresponding to Fig. 4.6c

UAV data	$\delta = 0.01$	$\delta = 0.2$	$\delta = 0.5$
Empirical failure rate	0.00975	0.201	0.497
Average route travel time	11330 s	10418 s	10404 s
route travel time overhead	160.2%	139.2 %	138.9%
Average # of rendezvous	8.5	6.4	6.3

(g) Quantitative results.

Figure 4.6: How different risk thresholds influence the rendezvous behaviors. UAV route time with the best range speed is 4354 s, and the route distance is 61.0 km. (a) UAVs are very risk-averse to failures with a risk threshold equal to 0.01. (b) UAV is less risk-averse to failures with a risk threshold equal to 0.2. (c) UAV is neutral to the failures with a risk threshold equal to 0.5.

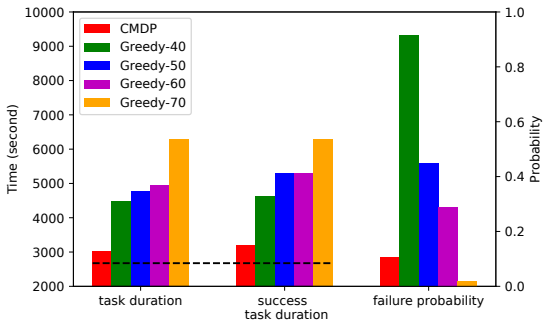


Figure 4.7: Results comparisons for the CMDP and greedy policies with $\delta = 0.1$. The dashed black line represents the task duration if the UAV moves with the best range speed without considering battery limitation.

Results shown in Fig. 4.6 include sample routes for the UAV and the UGV and statistical data of the policies. Fig. 4.6a, 4.6b, and 4.6c are sample routes for the UAV when it executes the policy. Fig. 4.6d, 4.6e, and 4.6f are corresponding routes of the UGV. The rendezvous point is denoted as a star. The SOC is annotated in red text close to the task node where the UAV decides to rendezvous. Some statistical data are summarized in table 4.6g. Generally, we observe that when the risk tolerance is set to be small, the UAV tends to rendezvous more often, and the average route travel time is higher. Here the average route travel time is computed by considering only trials where the UAV finishes its task route. By contrast, as the risk tolerance is relaxed to a larger value, the average route travel time will decrease, which comes at the cost of a high failure probability.

We also conducted several quantitative experiments to validate our formulation. The first experiment is to use MC simulation to check whether the failure probability is upper bound by the set risk tolerance of 0.1. We use FR to denote the empirical failure rate. As can be seen in Table 4.2, as MC increases, the empirical failure rate is close to and below the theoretical PF 0.1. In the following experiments, we will use $N_{MC} = 2000$ for simulation.

Table 4.2: Empirical evaluation of failure probability $\delta = 0.1$.

N_{MC} (# of MC trials)	500	1000	3000	5000
Failure rate	0.108	0.105	0.099	0.097

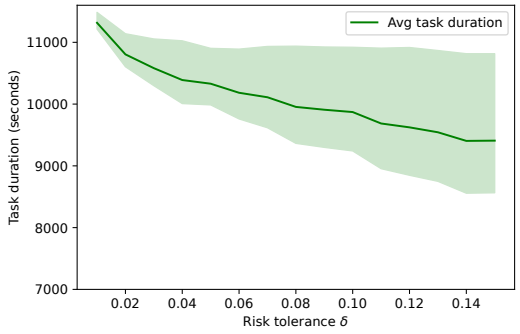


Figure 4.8: Risk tolerance vs task duration Pareto Curve.

To validate that the policy constructed from LP can minimize the expected travel distance. We compare our policy with a greedy baseline. The greedy policy is set as always flies with the best range speed and chooses to rendezvous when state-of-charge drops below a set value. What we observe in experiments is that when the route of a UAV is long, for example, there are more than 15 nodes, the probability of finishing the route is close to zero for the greedy baseline no matter what threshold we set. For a more informative comparison, we use only nodes 0 to 11 for the task route in the following experiments. We consider four-set values 40%, 50%, 60%, and 70%, and the corresponding policies are denoted as *Greedy-40*, *Greedy-50*, *Greedy-60*, and *Greedy-70*. As shown in Fig.4.7, our policy can guarantee success probability above the set value of 0.9 and the expected travel time of UAV is shorter compared to the baseline. Though the baseline can achieve a higher success probability in some cases as shown in Fig.4.7, its expected task duration is still longer than our policy.

The empirical Pareto curve for risk tolerance and the task duration is shown in Fig. 4.8. The green curve is the mean value, and the shaded area is formed using one standard deviation

from the mean. When the risk level is set to be a higher value, the UAV will tend to make more risky decisions, leading to a lower travel time at the cost of a higher failure probability.

The running time for the proposed routing problem consists of three parts. The first part is devoted to extracting transition information for LP. The second part is about constructing an LP model with Gurobi, and the last part is about solving the LP. In our case study, there are about 54000 states, and it takes about 6 min to extract transition information, 9 minutes to create an LP model, and about 1 second to solve the LP.

4.3 Risk-aware UAV-UGV Recharging Rendezvous: Multiple UAVs and UGVs

The formulation presented in Sec. 4.2 is not scalable to the number of robots and the planning horizon. In this section, we will present a more scalable formulation based on graph matching for a team of multiple UAVs and UGVs.

4.3.1 Problem Formulation

4.3.1.1 Problem Statement

Consider a team of N_a UAVs and N_g UGVs persistently monitoring a set of locations in an environment. The UAVs and UGVs move on the graphs $G_a = (U_a, E_a)$ and $G_g = (U_g, E_g)$ with their deterministic speeds v_a and v_g respectively. The vertex sets U_a and U_g represent the locations to be monitored by the aerial and ground vehicles respectively. The edge set E_a may be complete since the UAVs can move between any of the tasks, whereas the edge set E_g represents the road network on which the ground vehicles can move. We assume that the ordering of the task nodes for the UAVs and the UGVs is given, i.e., we have pre-defined persistent monitoring tours

for the UAVs and UGVs. These tours can be generated by planners that either do not consider recharging [107–109] or assume deterministic discharge [94, 110]. The tours for i^{th} UAV and k^{th} UGV are denoted by $\mathcal{T}_i^a$ and $\mathcal{T}_k^g$ respectively. Similar to our recent work for the single robot case [19], we show how to refine these tours in a risk-aware fashion.

A UAV i can take a *detour* from its monitoring tour $\mathcal{T}_i^a$ at any point along the tour to rendezvous with a UGV k and land on it, for recharging. The UGV keeps moving along its tour $\mathcal{T}_k^g$. We only model the case where only UAVs take detours, and not UGVs since UAVs are typically much faster and not restricted to the road network. The UAV can also wait at a rendezvous location for the UGV if it reaches there before the UGV. The number of UAVs that can simultaneously charge on a given UGV is d. Once recharged, the UAV leaves the UGV and goes to the next task node along its monitoring tour.

We assume that the UGVs do not run out of charge since typically they have much larger battery capacity than UAVs or can be easily refueled. We consider a stochastic energy discharge model for the UAVs that is given. We assume that the probability of a UAV running out of charge within the next t time units given the current charge level can be calculated, as shown in [19]. The stochastic battery discharge is monotonic in the time traveled by the UAV.

Since we consider a persistent monitoring mission, we solve RRRP in a receding-horizon manner. Given time horizon T, we seek a policy for the UAVs to recharge at most once in the horizon. Moreover, as the battery discharge rate of UAVs is stochastic, there may be a non-zero probability of some UAVs running out of charge. Hence, we also need to have a notion of risk aversion for the UAVs. On the other hand, to avoid frequent recharging, we also need to reduce the detour time spent by the UAVs for recharging.

At a high level, we consider the following problem: *Given a time horizon T, a risk-*

tolerance probability $\rho \in (0, 1)$, task routes for the N_a UAVs and N_g UGVs along with their current locations and state of charge, find a recharging schedule for each UAV such that:

1. *each UAV recharges at most once during T,*
2. *no UGV can charge more than d UAVs at a time,*
3. *the probability that no UAV runs out of charge during T is at least ρ, and*
4. *the total detour time of the UAVs is minimized.*

Next, we will show how to model this problem formally.

4.3.1.2 Graph Matching Based Formulation

Given the tour $\mathcal{T}_k^g$ for UGV k, we discretize it by introducing vertices every f units of time starting from the UGV's current position where f is the maximum time the UGV needs to travel for the UAV to recharge in the worst-case. These vertices representing possible rendezvous locations are denoted by $V(\mathcal{T}_k^g)$ and are shown as green nodes in Figure 4.9. Similarly, the set $V(\mathcal{T}_i^a)$ represents the set of locations from which UAV i can leave its monitoring tour $\mathcal{T}_i^a$ for a recharging detour (Figure 4.9). The set $V(\mathcal{T}_i^a)$ contains the current position of UAV i and its task nodes that can be visited within the next T time by UAV i. We do not need to discretize UAV tours further as it can be shown [20, Lemma 6] that there exists an optimal solution where the UAVs will leave their tours for recharging from either their current position or a task node.

We define the problem formally on a bipartite graph $G = (V_a \cup V_g, E)$ as follows.

UAV vertices: The vertex set V_a consists of N_a disjoint node sets, i.e., $V_a = \cup_{i=1}^{N_a} \mathcal{V}_i$ where $\mathcal{V}_i = V(\mathcal{T}_i^a) \cup a_i^{\emptyset}$. The vertex $a_i^{\emptyset}$ represents the scenario where the UAV i chooses not to rendezvous in

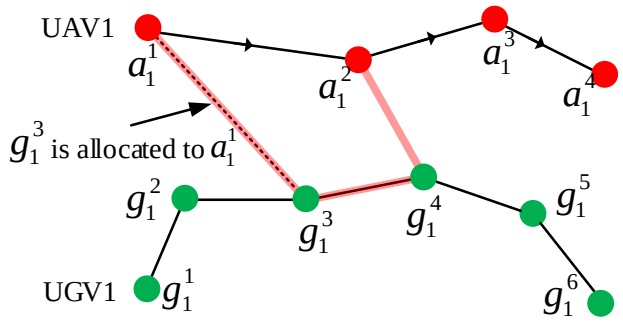


Figure 4.9: Recharging detour example. The UAV 1 leaves its route at the node a_1^1 to rendezvous with UGV 1 at the node g_1^3. The recharging occurs when they reach the node g_1^4 and the UAV moves to its next task node a_1^2. Note that if $d = 1$, no other UAV can recharge on this UGV between g_1^3 and g_1^4.

the current horizon.

UGV vertices: The vertex set V_g consists of $\{g_1^\emptyset, \ldots, g_i^\emptyset, \ldots, g_{N_a}^\emptyset\}$ and d copies of the set $\cup_{k=1}^{N_g} V(\mathcal{T}_k^g)$. The vertex $g_i^\emptyset$ for UAV i represents the scenario where UAV i chooses not to rendezvous for the current horizon. The d copies of each vertex in $V(\mathcal{T}_k^g)$ are to represent up to d different UAVs recharging at a time on a UGV.[2]

Edges: The edge set E denotes the set of all feasible recharging detour options. If the UAV i, starting from a node $j \in V(\mathcal{T}_i^a)$, is able to rendezvous with the UGV k at $l \in V(\mathcal{T}_k^g)$, an edge exist between the corresponding two nodes in V_a and V_g. The vertex $a_i^\emptyset$ is connected to $g_i^\emptyset$ for all $i \in [N_a]$.

Edge cost: For an edge $(i, j) \in E$ where $i \in V_a$ and $j \in V_g$, the edge cost c_{ij} represents the time needed to complete the recharging detour along the edge (i, j). The time needed for the recharging detour consists of three parts: the time to reach the rendezvous node, the waiting and recharging times at the rendezvous node, and the time to go to the next node on the tour. The edge cost along edge $(a_i^\emptyset, g_i^\emptyset)$ is zero.

[2]We use d copies of each vertex in $V(\mathcal{T}_k^g)$ to keep the analysis of the algorithm simple. The problem can be defined without having d copies for each $v \in V(\mathcal{T}_k^g)$ by changing Constraint (4.22) in Problem 7 to $\sum_i x_{ij} \leq d$.

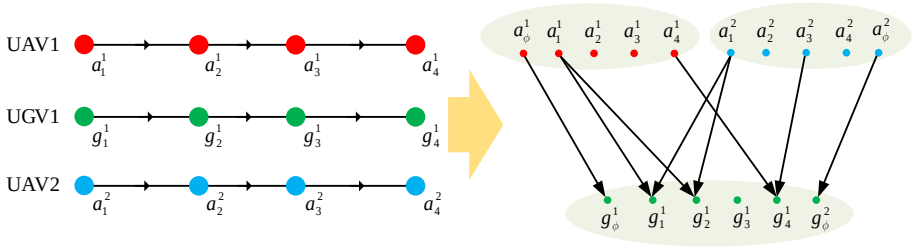


Figure 4.10: An example to show how to construct a bipartite graph for one planning horizon from the UAV and UGV route segments.

Edge success probability: For an edge $(i, j) \in E$ where $i \in V_a$ and $j \in V_g$, the edge success probability p_{ij} is defined as the overall probability to finish the task route for the current horizon given that the recharging detour along edge (i, j) is taken. It is the product of the two success probabilities: the probability of successfully completing the recharging detour, and the probability of finishing the rest of the route after recharging. The probability along edge $(a_i^{\emptyset}, g_i^{\emptyset})$ is the probability of reaching the end of the current horizon without recharging.

The finite horizon recharge rendezvous problem can now be defined as the following combinatorial optimization problem.

Problem 7 (Risk-aware Recharging Rendezvous Problem (RRRP))**.** *Given a bipartite graph* $G = (V_a \cup V_g, E)$ *where* $V_a = \cup_{i=1}^{N_a} \mathcal{V}_i$*, with edge costs* c_{ij} *and probabilities* p_{ij} *for edge* $(i, j) \in E$*,*

solve:

$$\min \sum_{i,j} c_{ij} x_{ij} \tag{4.19}$$

$$\textit{s.t.} \sum_{i,j} \log \frac{1}{p_{ij}} x_{ij} \leq \log \frac{1}{\rho} \tag{4.20}$$

$$\sum_{i \in \mathcal{V}_r} \sum_{j} x_{ij} = 1, \qquad \forall r \in [N_a] \tag{4.21}$$

$$\sum_{i} x_{ij} \leq 1, \qquad \forall j \in V_g \tag{4.22}$$

$$x_{ij} \in \{0, 1\} \tag{4.23}$$

The variable x_{ij} indicates whether edge (i, j) is in the solution or not. The objective is to minimize the total time incurred by the recharging detours of all UAVs. Constraint (4.20) enforces that the probability of no UAV running out of charge during the current horizon is at least ρ. We can write this as a linear constraint since the stochastic discharging processes of the UAVs are independent. For ease of notation, we will use the following inequality instead of Constraint (4.20).

$$\sum_{i,j} a_{ij} x_{ij} \leq B \tag{4.24}$$

where $B = \log 1/\rho$ and $a_{ij} = \log 1/p_{ij}$. Constraint (4.22) enforces that each UGV can recharge at most d UAVs at a time (by making sure that at most one UAV recharges at one of the d copies of a UGV vertex). Constraint (4.21) enforces that each UAV should be recharged at most once. Given a solution x_l, let M_l represent the corresponding solution on graph G. Let us define $c(M) = c(x) = \sum c_{ij} x_{ij}$, $a(M) = a(x) = \sum a_{ij} x_{ij}$ for ease of notation.

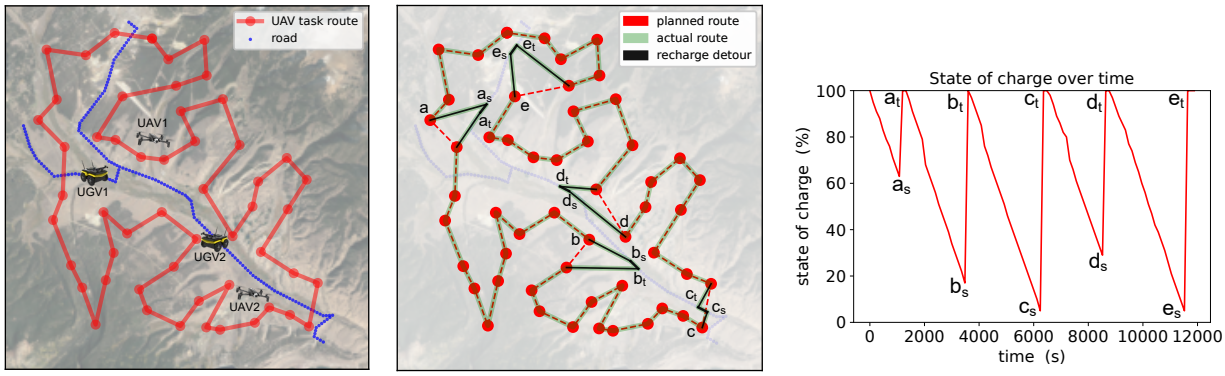
4.3.2 Simulation

In this section, we first present a qualitative example of the persistent monitoring mission. Next, we study how the value of risk tolerance influences the recharging behaviors and task performances of the UAVs. Then, we compare the performance of our scheduling strategy with a baseline (greedy strategy). Moreover, we empirically evaluate the performance of the proposed heuristic algorithm. All experiments are conducted on a PC with the i9-8950HK processor unless specified otherwise. The baseline solver is Gurobi 9.5.0.

4.3.2.1 Experimental Setup

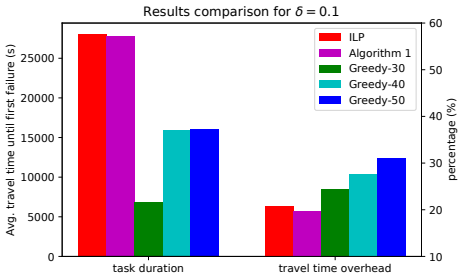
We consider a team consisting of two UAVs and two UGVs. The task routes $\mathcal{T}_a$ and $\mathcal{T}_g$ used in the problem can be either generated jointly by some task planners similar to those in [86, 93] or can be generated separately by different task planners. The UAV and UGV move at $v_a = 9.8$ m/s and $v_g = 4.5$ m/s respectively based on the field test data collected and used in our previous work [19]. The recharging process (swapping battery) takes 100s. The UAV and UGV need to monitor the task nodes on the route persistently. We apply our recharging strategy in a receding horizon fashion: every two minutes, the UAVs-UGVs team solves the RRRP problem to decide the UAVs' recharging schedule for the next $T = 2500$ seconds. For each UAV, the current position will be the first node when we construct the bipartite graph. If some UAV is on a detour, we do not replan until the UAV has finished its detour.

We consider two sources of stochasticity in the energy consumption model of UAVs: weight and wind velocity contribution to longitudinal steady airspeed. The energy consumption model of the UAV is the same as that in [19].



(a) (b) (c)

Figure 4.11: A qualitative example to illustrate how UAV and UGV rendezvous with each other solving the RRRP. The risk tolerance is set to be $\rho = 0.1$ in this case study. Subscriptions s and t denote the start and the terminal of the recharging process. (a) The input of the RRRP problem includes the UAV and UGV tasks and the road network. (b) One sample tour of UAV 1 when it persistently monitors the route. (c) One sample history of SOC for UAV 1.



(a)

Figure 4.12: Quantitative results. (a) Comparisons of the RRRP scheduling and the greedy strategies with $\rho = 0.1$.

4.3.2.2 Simulation Results

Qualitative Example The input of the problem consists of UAV task nodes and nodes of the road network (Figure 4.11a). Figure 4.11b shows one tour route of one UAV when the system executes the proposed strategy in a receding horizon fashion. The UAV monitors the task route persistently. When the UAV reaches node a, it doesn't move forward to its next task node (connected through a dashed red line). Instead, the new schedule is to rendezvous with UGV at a_s and takes off from the UGV at a_t, and then go to its next task node. Similarly, the UAV will rendezvous with the

Table 4.3: Statistical results for UAVs

UAV data	$\rho = 0.01$	$\rho = 0.1$	$\rho = 0.3$
Mean time before failure (s)	39660	27600	24360
Avg. travel time overhead	19.7 %	18.5 %	17.8 %
Avg. # of task nodes visited	158	110	105
Avg. # of rendezvous per T	1.4	1.3	1.3

UGV when it is close to nodes b, c, d and e. Subscriptions s and t denote the start and the terminal of the recharging. A sample of the history of the state of charge (SOC) is shown in Figure 4.11c. We can observe in Figure 4.11c that the UAV's recharging strategy is more than a simple rule, such as for example getting recharged when the SOC is below 50 $\%$ and may get recharged at various values of SOC.

Effect of Risk Tolerance We study how various risk tolerances influence the strategy (see Table 4.3). We set the risk tolerance ρ to be 0.01, 0.1, and 0.3 and use four metrics to quantify the performance of the strategy:

(1) Mean time before the first failure: The time before the first UAV runs out of charge and needs human intervention. *(2) Travel time overhead*:

$$\frac{\texttt{actual travel time} - \texttt{task time}}{\texttt{task time}},$$

where `task time` is the travel time of the route without any recharging. A lower overhead is desired since it accounts for the delay in visiting task nodes. *(3) Average number of task nodes visited:* by the UAV before its first failure. Similar to (1), a higher number reflects a better strategy. *(4) Average number of rendezvous per planning horizon* T. If this number is too large,

it suggests that the UAV takes too many recharging detours, which should be avoided.

In general, we observe that when the risk tolerance is set to be smaller, the mean time before the first failure will be longer. Similarly, the travel time overhead and the average number of rendezvous per planning horizon will be greater, which implies the UAV spends more portion of flight time in the recharging detours.

4.4 Conclusion

We consider the cooperative routing problems in which the UAVs and UGVs need to finish the task cooperatively and the UGVs need to recharge the UAVs as mobile rechargers periodically. We study this type of problem for a team consisting of one UAV and one UGV as well as a team of multiple UAVs and UGVS. For the one UAV and one UGV case, we formulate the problem as a CMDP and use LP to find the optimal policy for the CMDP. For a team of multiple UAVs and UGVS, we give one graph-matching-based formulation.

For the proposed work in Section 4.2, we mainly plan to improve the computational time to enable online replanning. The computational time consists of two parts: the time to extract transition probability and the time to solve LP. To accelerate the process of obtaining the transition probability, we will combine the supervised learning and the high-fidelity simulation to find the mapping from the environmental disturbances (e.g., wind field) to the transition probability matrix. To accelerate the computation for the optimal policy, we will resort to feature-based representation methods to reduce the size of the decision variables.

For the proposed work in Section 4.3, the existing commercial solver is efficient for only small problem instances. Our main focus for the next step is to find some approximation algo-

rithms that work faster than the existing solver in the large problem instance.

Chapter 5: Multi-Robot Information Gathering with Decision-oriented Learning of the Environmental Model

5.1 Overview

In Chapters 2, 3, and 4, we consider the decision-making problems in which the environmental model is given. In this chapter, our focus is on learning an environmental model for decision-making problems. We are interested in the case where the environmental model is encoded as parameters that affect the optimization objective. We want to learn the mapping from the environmental observation to the parameters. For example, in the Traveling Salesman Problem (TSP), the environmental model is the travel cost for all edges which depends on the environmental factors such as rain, terrain conditions, etc. which are available as environmental observations. We need to learn a mapping from the observations to the parameters (i.e., edge costs), to solve the TSP instance. Classically, such a learning process is independently conducted without considering the downstream problem. In contrast, we propose to incorporate the downstream decision-making problem into the learning process. Such integration will help reduce the misalignment between the prediction model and the downstream task. The misalignment refers to a predictor that despite achieving high predictive accuracy in the learning phase may not necessarily result in good decisions in the downstream task. Instead, by making the combinatorial

optimization differentiable, we can reduce such misalignment and learn a predictor that results in good downstream decisions. By making the combinatorial optimization differentiable, we can treat it as a differentiable module in the learning process.

In this chapter, we present a *Decision-Oriented Learning* framework that integrates two types of submodular maximization into the learning process: where the objective is a parameterized (1) monotone non-negative submodular function and (2) possibly non-monotone and possibly non-positive submodular function. In both cases, the input to the learning module is the observation of the environment, and the output is the parameters for the objectives. To achieve such integration, we design two differentiable algorithms to make the corresponding submodular maximization differentiable. We experimentally demonstrate the advantages of such a decision-oriented learning framework by considering variants of the Vehicle Routing Problem (VRP).

5.2 Learning a Context-aware Objective for Information Gathering: Monotone Non-negative Submodular Objective Case

Many multi-robot decision-making problems can be formulated as combinatorial optimization problems, among which the objectives in some problems (e.g., mutual information [33], area explored, number of targets tracked [23], detection probability [111] etc.) have diminishing returns property, i.e., submodularity. Intuitively, submodularity formalizes the notion that adding more robots to a larger multi-robot team cannot yield a smaller marginal gain in the objective of adding the same robot to a smaller team.

If the submodular objective is known and fixed, the multi-robot decision-making problem boils down to a submodular maximization problem, which is NP-hard but can be solved with

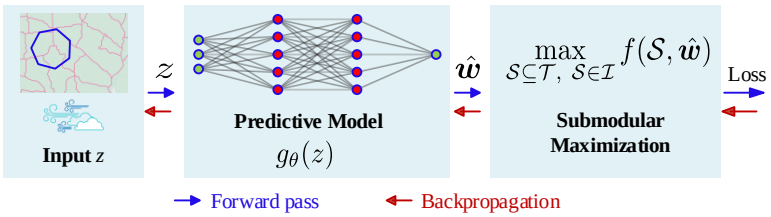


Figure 5.1: Decision-Oriented Learning framework. The training loss is defined after the downstream task.

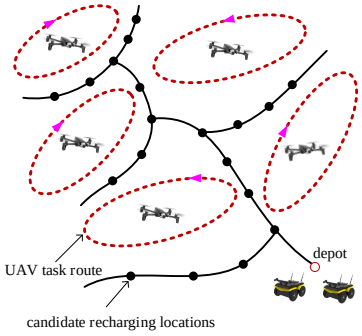


Figure 5.2: An illustrative example for vehicle routing problems.

an $(1 - \frac{1}{e})$-approximation by the greedy algorithm [15]. However, in practice, there are several parameters that affect the objective function that may not be known exactly.

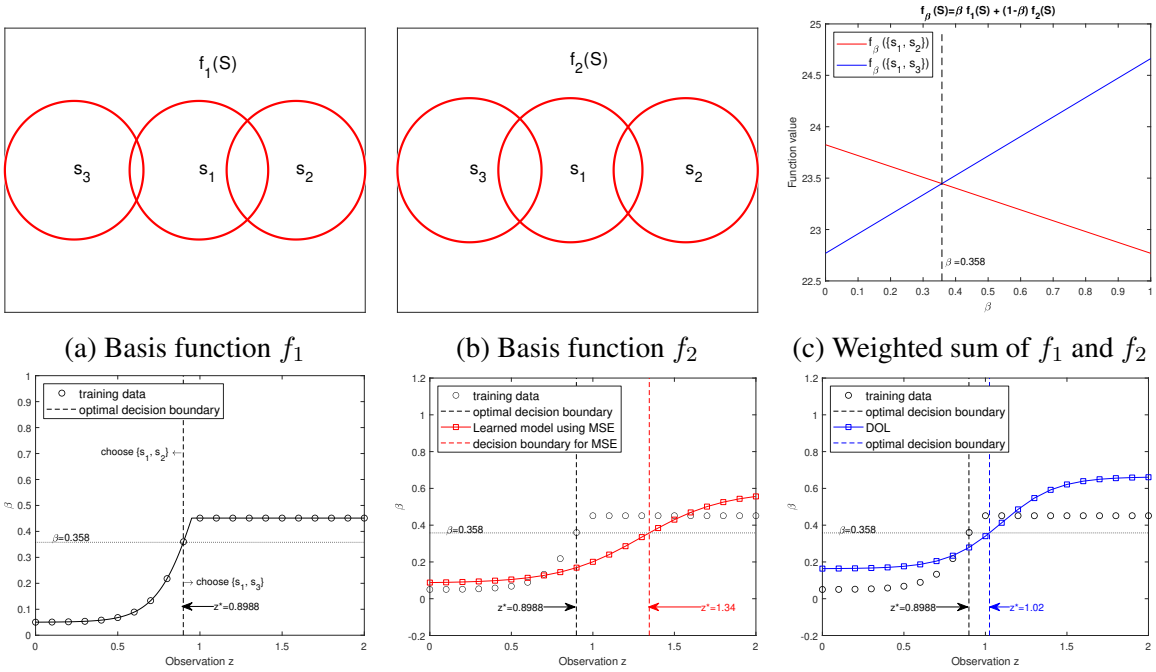
Consider the following illustrative example of a vehicle routing problem shown in Figure 5.2. Here, a team of Unmanned Ground Vehicles (UGVs) are tasked with servicing a set of requests that appear throughout the environment. We have a set of candidate routes of which we must select one for each UGV. The objective is to maximize the number of requests serviced. If a request location lies on more than one UGV path (the paths may overlap as the UGVs move on a road network), it only counts once in the objective function. Thus, the objective function is a coverage function, which is a special case of the submodular function.

If we know the location of the requests, then we can solve this problem greedily to obtain a $(1 - \frac{1}{e})$–approximation. The greedy algorithm requires the capability to compute the objective function $f(S)$. However, there are many scenarios where we may not know where the requests

show up and as such not know $f(S)$. For example, the requests could correspond to Unmanned Aerial Vehicles (UAVs) that are carrying out persistent monitoring missions that land when out of charge so as to be recharged by mobile recharging stations [19, 20, 91, 112–114]. Here, even if we know the routes followed by each UAV, we may not know their exact landing locations since the energy consumption is stochastic [19, 20] and communication between UAVs and UGVs is not available (e.g., due to stealth). In such cases, we may be able to predict $f(S)$ using all the available information. We call the latter as *context* z, which can include the routes of the UAVs, the environmental conditions including the wind conditions, etc.

The traditional pipeline here would be to use the context information and predict $f(S)$ and then solve the downstream UGV route selection problem, $\operatorname{argmax}_S \hat{f}(S)$, using this predicted $\hat{f}(S)$. However, as the following example shows a good predictor of $f(S)$ does not necessarily align with making good decisions on the downstream task. On the other hand, a predictor that does not necessarily yield the best predictions of $f(S)$ may still yield the best decisions for the downstream $\operatorname{argmax}_S \hat{f}(S)$ problem. Particularly, a good prediction of the parameters may be sufficient but unnecessary in making good route decisions. Further, the *loss* function used in learning to predict the parameters may be misaligned with the downstream task (i.e., finding good routes). An alternative is to solve this problem end-to-end, where we directly map the context input to the routes using, for example, deep neural networks. The objective in such an approach is not misaligned since the loss function for training the network will depend directly on the routes selected. However, such an approach faces two limitations. Training end-to-end a combinatorial optimization problem may require a lot of data [115]. Further, the black-box nature of neural networks will make the route selection hard to explain or interpret.

We present an illustrative example of such misalignment in Fig. 5.3. Let f_1, f_2 be two



(a) Basis function f_1 (b) Basis function f_2 (c) Weighted sum of f_1 and f_2

(d) Ground truth data (e) Learned model using MSE (f) Decision-Oriented-Learning

Figure 5.3: An illustrative example to show the misalignment between the prediction model that achieves high predictive accuracy and the one that results in good decisions.

coverage functions (coverage function is submodular by definition) defined over set $\{s_1, s_2, s_3\}$. Given a subset $\mathcal{S} \subseteq \{s_1, s_2, s_3\}$, $f_i(\mathcal{S}), i = 1, 2$ will return the area covered by the selection. The submodular objective that we are interested in is defined as $f_\beta(\mathcal{S}) = \beta f_1(\mathcal{S}) + (1-\beta) f_2(\mathcal{S}), \beta \in [0, 1]$, which is also submodular by definition. Suppose that we want to maximize f_β with a partition matroid: $|\mathcal{S} \cap \{s_1\}| \leq 1$, $|\mathcal{S} \cap \{s_2, s_3\}| \leq 1$. Then the optimal solution is either $\{s_1, s_2\}$ or $\{s_1, s_3\}$. In Fig. 5.3c, we show how the optimal decision changes w.r.t. β. When $\beta \geq 0.358$, the optimal decision is $\{s_1, s_3\}$ since $f_\beta(\{s_1, s_3\}) \geq f_\beta(\{s_1, s_2\})$ (the blue line is above the red line). By contrast, when $\beta < 0.358$, the optimal decision is $\{s_1, s_2\}$. Next, let us look at the learning problem for f_β. We want to find a mapping from the observation z to β. In Fig. 5.3d, we show the training data sampled from the ground truth and the optimal decision boundary $z^* = 0.8988$, which is obtained by finding the intersection between the ground truth curve and

$\beta = 0.358$. If we use Mean Square Error (MSE) as the objective for learning without considering the downstream task, we will get two lines as shown in Fig. 5.3e. The decision boundary (dashed vertical red line, $z^* = 1.34$, passing the intersection of the learned red line and $\beta = 0.358$) is on the right of the optimal boundary, thus not optimal. By contrast, if we consider the downstream optimization, we will get two lines as shown in Fig. 5.3f and the decision boundary (dashed vertical blue line, $z^* = 1.02$, passing the intersection of the learned blue line and $\beta = 0.358$) is closer to the optimal boundary, thus reducing the regions of suboptimal decisions. Such an observation motivates us to incorporate the decision process (submodular maximization) into the learning process.

To this end, we propose a Decision-Oriented-Learning (DOL) framework for learning context-aware parameterized submodular objectives. We focus on submodular functions that can be parameterized, i.e., $f(S, \boldsymbol{w})$, where the parameters are to be learned from the context. As described earlier and pointed out in [116–118], the best estimator of $\boldsymbol{w}$ does not necessarily yield the best decisions for the downstream task. Instead, in the proposed framework, the decision-making problem (submodular maximization) is treated as a differentiable layer that takes as input the output from the prediction module as shown in Fig. 5.1. The prediction module takes as input a context observation z and predicts $\boldsymbol{w}$, i.e., the parameterized submodular function. By using a differentiable submodular optimization layer, we can train the prediction module using the loss from the downstream task, thereby yielding aligned predictions.

Most existing multi-robot decision-making work consider the case where the optimization objective is well-defined and known. Wilde et al. [119] consider the case where the optimization objective is hard to quantitatively specify and may be subjective and proposed an interactive learning framework to learn the objectives. Our work shares a similar stance with [119] but dif-

fers in two aspects. First, we consider the fact that the task objective may change in different contexts, for example in different weather conditions, and aim at learning a context-aware objective. Second, our learning framework integrates the downstream decision-making process into the learning process.

Another line of research related to this work is decision-oriented learning. The key idea is to embed the decision-making problem as a differentiable layer in the learning pipelines. The main advantage is that it allows end-to-end training and reduces the engineering efforts to design some intermediate learning objectives. Such an idea was initially explored for continuous optimization problems [120, 121] and has gained popularity in control and robotics [122–125]. The idea was later extended to the combinatorial problems [117, 118, 126, 127]. Our work is inspired by [117, 126] and our framework integrates the decision-making process for mobile charging station routing, which is modeled as submodular maximization, into the learning process.

This work is also closely related to differentiable submodular maximization. Submodular maximization and its variants have been widely used in multi-robot decision-making problems including coverage, target tracking, exploration, and information gathering. These studies are all based on the fact that the greedy algorithm and its variants can solve submodular maximization problems its variants efficiently with a provable performance guarantee. Since the submodular objective and greedy algorithm are tightly coupled, it is better to take into account the influence of the greedy algorithm when we consider learning submodular functions [128]. To this end, several differentiable versions of the greedy algorithms have been proposed [128, 129]. The core idea behind these algorithms is stochastic smoothing, i.e., perturb the algorithm by introducing some probability distribution in the intermediate steps. Our framework is built on these differentiable greedy algorithms but is targeted specifically for context-dependent routing problems.

5.2.1 Problem Formulation

We are interested in parameterized submodular objective function $f(\mathcal{S}, \boldsymbol{w})$, where $\boldsymbol{w}$ is the parameter vector. Readers are referred to [119] for a formal definition. In practice, such an objective is usually unknown and context-dependent, i.e., the parameters $\boldsymbol{w} \in \mathcal{W}$ depend on the environment features. Our goal is to learn a function $g_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathcal{W}$ that maps the context observation $z \in \mathcal{Z}$ to the objective parameters $\boldsymbol{w}$. Traditionally, finding the mapping $g_{\boldsymbol{\theta}}$ and optimizing the downstream objective $f(S, \boldsymbol{w})$ are considered separately: given the training data $\mathcal{D} = \{(\boldsymbol{z}_1, \boldsymbol{w}_1), (\boldsymbol{z}_2, \boldsymbol{w}_2), \ldots, (\boldsymbol{z}_{|\mathcal{D}|}, \boldsymbol{w}_{|\mathcal{D}|})\}$, first find the mapping $g_{\boldsymbol{\theta}}$ by optimizing over $\boldsymbol{\theta}$ in a supervised fashion, and then use the parameter $\boldsymbol{w} = g_\theta(\boldsymbol{z})$ to optimize $f(S, \boldsymbol{w})$.

By contrast, the proposed paradigm that integrates downstream optimization is given below.

Problem 8. *Given the training data $\mathcal{D} = \{(\boldsymbol{z}_1, \boldsymbol{w}_1), (\boldsymbol{z}_2, \boldsymbol{w}_2), \ldots, (\boldsymbol{z}_{|\mathcal{D}|}, \boldsymbol{w}_{|\mathcal{D}|})\}$, learn a function $g_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$ such that the learning cost $L = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \ell_i(\boldsymbol{w}_i, \hat{\boldsymbol{w}}_i)$ is minimized, where $\ell_i(\boldsymbol{w}_i, \hat{\boldsymbol{w}}_i)$ is defined through Eq. (5.1) to Eq. (5.3):*

$$\hat{\boldsymbol{w}}_i \coloneqq g_{\boldsymbol{\theta}}(\boldsymbol{z}_i) \tag{5.1}$$

$$\hat{\mathcal{S}} \coloneqq \mathcal{S}^*(\hat{\boldsymbol{w}}_i) \;\; \textit{by solving (5.5) with } \boldsymbol{w} = \hat{\boldsymbol{w}}_i \tag{5.2}$$

$$\ell_i(\hat{\boldsymbol{w}}_i, \boldsymbol{w}_i) \coloneqq f(\mathcal{S}^*(\boldsymbol{w}_i), \boldsymbol{w}_i) - f(\hat{\mathcal{S}}, \boldsymbol{w}_i), \tag{5.3}$$

where $\mathcal{S}^*(\boldsymbol{w}_i)$ denotes the solution of (5.5) returned by some approximation algorithms with $\boldsymbol{w} = \boldsymbol{w}_i$; $f(\mathcal{S}^*(\boldsymbol{w}_i), \boldsymbol{w}_i)$ denotes the decision quality when we use the ground truth parameter $\boldsymbol{w}_i$ for decisions; $f(\hat{\mathcal{S}}, \boldsymbol{w}_i)$ denotes the decision quality when we use the predicted parameter $\hat{\boldsymbol{w}}_i$ for decisions, i.e., use $\hat{\boldsymbol{w}}_i$ to obtain the decision $\hat{\mathcal{S}}$, but the decision is evaluated w.r.t. the true

parameter $\boldsymbol{w}_i$.

The intuition for Eq. (5.3) is that we want to minimize the gap between the decision quality of the true parameters and that of the predicted parameters. One challenge is when we use the chain rule to compute the gradient of the loss function, we need to differentiate through the optimization problem (the first term on the r.h.s. of Eq. (5.4)) as shown in the illustrative computational graph in Fig. 5.1.

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \frac{\partial \ell_i}{\partial \hat{\boldsymbol{w}}_i} \cdot \frac{\partial \hat{\boldsymbol{w}}_i}{\partial \boldsymbol{\theta}} \tag{5.4}$$

In the following sections, we will show how to approximately compute the first term on the r.h.s. of Eq. (5.4).

5.2.2 Case Study: Ground Vehicle Routing Problem

Suppose that there is a set of candidate routes, $\mathcal{T}$, each of which starts and terminates at the same depot. Our goal is to select a subset from $\mathcal{T}$ for UGVs to traverse and recharge the UAV along the way such that the total number of UAVs that UGVs will recharge is maximized.

environmental model: As shown in Fig. 5.2, the working environment is described by an area $\mathcal{E} \subseteq \mathbb{R}^2$. There are n_a UAVs that are executing persistent monitoring. The energy consumption of UAVs will be affected by the wind. The wind field is represented as a tuple $(\boldsymbol{\omega}_s, \omega_o)$, where $\boldsymbol{\omega}_s$ and ω_o denote the description vectors for the speed and the orientation of the wind, respectively. There are n_g UGVs in $\mathcal{E}$, denoted by the set $\{1, \ldots, n_g\}$.

UAV Behavior: There are three components defining the behavior of each UAV. The first one is the task route, which is defined as a sequence of ordered locations projected on the ground,

and the UAV will persistently monitor these locations. The UAV will fly at a fixed speed v_a between two task locations, and the wind will affect its energy consumption. The second component is the recharging strategy dealing with the depletion of the battery. The third component is the energy consumption model. We use the same model as that in [130].

Context Observation: Each observation $\boldsymbol{z}$ consists of two components: the task routes of all UAVs; and the wind field $(\boldsymbol{\omega_s}, \omega_o)$ of the working area.

If such submodular objective f is known, the problem boils down to a submodular maximization problem with a matroid constraint: let $\mathcal{T}$ be set of all candidate routes, the problem is to select a subset from $\mathcal{T}$ to maximize the objective, i.e.,

$$\max_{\mathcal{S}\subseteq\mathcal{T},\ |\mathcal{S}|\leq n_g} f(\mathcal{S}, \boldsymbol{w}), \tag{5.5}$$

where $\boldsymbol{w}$ denotes the parameters in the objective function.

Parameterization of the Objective Function: In general, such applications have no closed-form expression of the objective function. In this paper, we consider the case where the objective function f is the linear combination of a set of basis functions. Such parameterization techniques are commonly used in the literature on learning submodular functions [111, 119, 131]. Similar to [119], we assume without loss of generality that each basis function is characterized by a subset $W_i \subseteq V$. That is, for any W_i, let $\psi_i(\mathcal{S})$ be a count of how many vertices of the tours $\mathcal{S}$ lie in W_i, then f_i is a functional of $\psi_i(\mathcal{S})$.

The overall objective function is:

$$f(\mathcal{S}, \boldsymbol{w}) = \sum_{i=1}^{n} \sum_{j=1}^{|\Gamma|} w_{i,j} f_{i,j}(\mathcal{S}), \tag{5.6}$$

where $\boldsymbol{w} = [w_{i,j}]$ denotes the matrix of unknown parameters of the function; basis functions are defined as $f_{i,j}(\mathcal{S}) = \sum_{\alpha=1}^{\psi_i(\mathcal{S})} \gamma_j^{(\alpha-1)}$; and $\gamma_j \in (0, 1]$ comes from a known set Γ. As shown in [119], the objective function $f(\mathcal{S}, \boldsymbol{w})$ proposed in Eq. (5.6) is a normalized, monotone, and submodular set function.

5.2.3 Learning Algorithm

In this section, we present the main learning algorithm that solves the submodular maximization in a differentiable manner.

5.2.3.1 Smoothed Greedy Algorithm

Algorithm 7: Smoothed Greedy

Input : $f(\mathcal{S}, \boldsymbol{w})$ and independent set $\mathcal{I}$
Output: Set $\mathcal{S}$ of tours for each robot

$\mathcal{S} \leftarrow \emptyset$
for $k \leftarrow 1$ **to** N **do**
  // find all addable elements in the current round
  $U_k = \{u_1, \ldots, u_{n_k}\} \leftarrow \{T \notin \mathcal{S} \mid \mathcal{S} \cup \{T\} \in \mathcal{I}\}$
  // marginal gain for all addable elements
  $\boldsymbol{m}_k(\boldsymbol{w}) \leftarrow (f_{\mathcal{S}}(u_1, \boldsymbol{w}), \ldots, f_{\mathcal{S}}(u_{n_k}, \boldsymbol{w}))$
  // compute a probability distribution
  $\boldsymbol{p}_k(\boldsymbol{w}) \leftarrow \operatorname{argmax}_{\boldsymbol{p} \in \Delta^{n_k}} \{\langle \boldsymbol{m}_k(\boldsymbol{w}),\ \boldsymbol{p} \rangle - \Omega_k(\boldsymbol{p})\}$
  $s_k \leftarrow$ sample $u \in U_k$ with probability $p_k(u, \boldsymbol{w})$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_k\}$
end
return $\mathcal{S}$

The Smoothed Greedy (SG) algorithm is given in Algorithm 7, which was first proposed in [129]. For a given $\boldsymbol{w} \in \mathcal{W}$, In each iteration step, we compute marginal gain $f_{\mathcal{S}}(u, \boldsymbol{w})$ for each

candidate element $u \in U_k$ (line 3); we define $n_k \coloneqq |U_k|$. Let

$$\boldsymbol{m}_k(\boldsymbol{w}) = (m_k(u_1, \boldsymbol{w}), m_k(u_2, \boldsymbol{w}), \ldots, m_k(u_{n_k}, \boldsymbol{w})) \in \mathbb{R}^{n_k}$$

denote the marginal gain vector. The probability vector, $\boldsymbol{p}_k(\boldsymbol{w}) = (p_k(u_1, \boldsymbol{w}), \ldots, p_k(u_{n_k}, \boldsymbol{w}))$, is computed as:

$$\boldsymbol{p}_k(\boldsymbol{w}) = \underset{\boldsymbol{p} \in \Delta^{n_k}}{\operatorname{argmax}} \{\langle \boldsymbol{m}_k(\boldsymbol{w}),\ \boldsymbol{p} \rangle - \Omega_k(\boldsymbol{p})\}, \tag{5.7}$$

where $\Delta^{n_k} \coloneqq \{\boldsymbol{p} \in \mathbb{R}^{n_k} \mid \boldsymbol{p} \geq \boldsymbol{0}_{n_k}, \langle \boldsymbol{p}, \boldsymbol{1}_{n_k} \rangle = 1\}$ is the $(n_k - 1)$-dimensional probability simplex; $\Omega_k : \mathbb{R}^{n_k} \to \mathbb{R}$ is a strictly convex function and is a regularization function.

Next, we will show the theoretical results for Algorithm 7. Detailed explanations and proofs can be found in [129]. Let $\delta \geq 0$ be a constant that satisfies $\delta \geq \Omega_k(\boldsymbol{p}) - \Omega_k(\boldsymbol{q})$ for all $k = 1, \ldots, |\mathcal{S}|$ and $\boldsymbol{p}, \boldsymbol{q} \in \Delta^{n_k}$. We will use δ to quantify the performance of SG.

As shown in Theorem 1 in [129], in expectation, the output of SG satisfies that $\mathbb{E}\left[f(\mathcal{S}, \boldsymbol{w})\right] \geq (1 - \frac{1}{e}) f(OPT, \boldsymbol{w}) - \delta n_g$, where OPT denotes the optimal solution. This result suggests that the SG algorithm in expectation almost preserves the performance of the deterministic greedy algorithm, whose approximation factor is $(1 - \frac{1}{e})$, with one extra term δn_g, which is the price for differentiability. It should be noted that by using SG, the output is stochastic and we focus on the expected result of the output. The regularization functions Ω_k are chosen to guarantee the expected outputs of SG differentiable. Examples for Ω_k will be discussed in the Sec. 5.2.4.

5.2.3.2 Gradient Estimation

Let $\mathcal{O}_{\mathcal{I}}$ be the set of all possible solutions returned by SG. Let $p(\mathcal{S}, \boldsymbol{w}) \in [0, 1]$ be the probability for $\mathcal{S} \in \mathcal{O}_{\mathcal{I}}$. Specifically, for a returned sequence $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\} \in \mathcal{O}_{\mathcal{I}}$, the associated probability can be computed as $p(\mathcal{S}, \boldsymbol{w}) = \prod_{k=1}^{|\mathcal{S}|} p_k(s_k, \boldsymbol{w})$, where $p_k(s_k, \boldsymbol{w})$ is the element of $\boldsymbol{p}_k(\boldsymbol{w})$ defined Eq. (5.7) corresponding to $s_k \in U_k$.

Next, we will show how to construct a gradient estimator based on the output distribution. Let $Q(\mathcal{S})$ be any scalar- or vector-valued function. We want to compute $\nabla_{\boldsymbol{w}} \mathbb{E}_{\mathcal{S} \sim p(\boldsymbol{w})} [Q(\mathcal{S})] = \sum_{\mathcal{S} \in \mathcal{O}_{\mathcal{I}}} Q(\mathcal{S}) \nabla_{\boldsymbol{w}} p(\mathcal{S}, \boldsymbol{w})$. Since the size of the independent set will increase exponentially w.r.t. the size of the ground set, it is computationally expensive to compute this gradient exactly. Instead, we will use the following unbiased estimator for the gradient in training.

As shown in Proposition 1 in [129], let $\mathcal{S}_j = (s_1, \ldots, s_{|\mathcal{S}_j|}) \sim p(\boldsymbol{w}) (j = 1, \ldots, N)$ be outputs of SG. Then,

$$\frac{1}{N} \sum_{j=1}^{N} Q(\mathcal{S}_j) \otimes \nabla_{\boldsymbol{w}} \ln p(\mathcal{S}_j, \boldsymbol{w}) \tag{5.8}$$

is an unbiased estimator of $\nabla_{\boldsymbol{w}} \mathbb{E}_{\mathcal{S} \sim p(\boldsymbol{w})} [Q(\mathcal{S})]$, where $\otimes$ denotes the outer product.

5.2.4 Simulation

environmental model: There are n_a UAVs and n_g UGVs. The global wind ω is represented as $[a, b, \omega_o]$, where a and b are the shape and scale parameters of Weibull distribution, respectively, and ω_o is the wind direction.

UAV and UGV Behavior: In this case study, we consider the case that the task route is defined as a sequence of ordered locations uniformly sampled from a circle whose center is $[C_x, C_y]$ and

radius is r. Using this geometric information, each route can be represented as a vector C_x, C_y, r. As for the recharging strategy, we use a simple strategy in the simulation: whenever the state of charge drops below 30%, fly to the nearest recharging location and wait for the UGVs. We use the same energy model as that in [130]. We generate UGV routes by first randomly selecting a set of nodes and then solving a TSP to get a route.

Context Input $\boldsymbol{z}$ and mapping $g_{\boldsymbol{\theta}}(\boldsymbol{z})$: As shown in Fig. 5.1, each $\boldsymbol{z}$ consists of two components: the task routes of all UAVs and the wind field vector $[a, b, \omega_o]$ of the working area. Since the route of the UAV can be parameterized by a circle (C_x, C_y, r), z can be represented as $[C_x^1, C_y^1, r^1, C_x^2, C_y^2, r^2, \ldots, C_x^{n_a}, C_y^{n_a}, r^{n_a}, a, b, \omega_o]$. $g_{\boldsymbol{\theta}}(\boldsymbol{z})$ is instantiated using neural networks with one hidden layer (of size 64) with ReLu activation function.

Regularization and Basis Function We choose the entropy function for experiments. Specifically, when $\Omega_k(\boldsymbol{p}) = \epsilon \sum_{i=1}^{n_k} p(u_i) \ln p(u_i)$, where $p(u_i)$ is the $i-$th entry of $\boldsymbol{p} \in [0,1]^{n_k}$ and $\epsilon > 0$ is an arbitrary constant, δ can be set to $\epsilon \ln n_k$. For the road graph $G = (V, E)$, we use graph partition algorithms to generate nine sets of nodes, i.e., $\{W_1, \ldots, W_9\}$. For each partition $W_i \subset V$, we define three basis functions for decay parameters $\gamma \in \{0.001, 0.5, 1\}$.

Raw Data: Based on the UAV UGV behavior models, we build a simulator for the routing problems. Given simulation parameters (e.g., UAV routes, and wind conditions), it will simulate UAVs' execution of persistent tasks. If we provide several selected routes to the simulator, it will return the number of UAVs recharged if UGV follows these routes. Based on this simulator, for each context observation $\boldsymbol{z}$, we test multiple possible route selections and obtain a set of the actual number of UAVs that UGV recharged. We will use this raw data to obtain the training data.

To train the proposed decision-oriented framework, we need the i-th data point to be in the form $(\boldsymbol{z}_i, \boldsymbol{w}_i)$, where $\boldsymbol{z}_i$ denotes the context input and $\boldsymbol{w}_i$ is the corresponding parameter

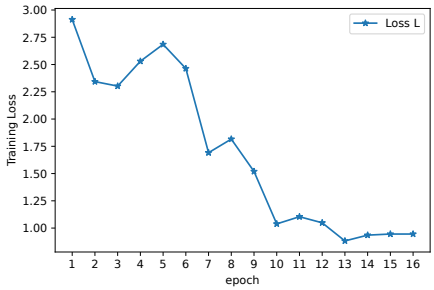


Figure 5.4: Per epoch loss curve of DOL.

Table 5.1: Test results for learned models.

Avg # of UAV recharged	DOL	two-stage	random
$n_a = 6, n_g = 3$	**5.3** $\pm$ 0.5	4.6 $\pm$ 0.4	2.1 $\pm$ 1.4
$n_a = 10, n_g = 3$	9.2 $\pm$ 0.7	8.5 $\pm$ 0.6	4.5 $\pm$ 1.3
$n_a = 15, n_g = 4$	14.1 $\pm$ 0.8	13.2 $\pm$ 0.7	7.5 $\pm$ 1.6

vector of the objective function. However, $\boldsymbol{w}_i$ is not directly available and we need to do some pre-processing of the raw data. Specifically, for each $\boldsymbol{z}$, we have a set of values for different selections, i.e., $\mathcal{F} = \{f(\mathcal{S}_1), \ldots, f(\mathcal{S}_{|\mathcal{F}|})\}$. We find the corresponding $\boldsymbol{w}$ by solving the following regularized least square optimization problem [131], i.e., $\min_{\boldsymbol{w} \geq 0} \sum_{i=1}^{|\mathcal{F}|} \|f(\mathcal{S}_i, \boldsymbol{w}) - f(\mathcal{S}_i)\|_2^2 + \xi \|\boldsymbol{w}\|_2^2$, where ξ is a user-specified regularization parameter.

Fig. 5.4 shows the learning curves over epochs. In each epoch, we compute the gradient by sampling a batch size of 40 in each iteration. We can see that as the training epoch increases, the loss will gradually decrease to a steady value. It should be noticed that the loss here represents the solution quality gap between the solution obtained using ground truth parameters and the solution obtained using predicted parameters as defined in Problem 8. Therefore, such a decrease suggests that the decision quality is improving.

After training, we test the performance of the learned models using the simulator. We generate a set of context observations $\{\boldsymbol{z}_i\}_{\text{test}}$ and compute the corresponding predicted weights

$\{\hat{\boldsymbol{w}}_i\}_{\text{test}}$. Then, we use $\{\hat{\boldsymbol{w}}_i\}_{\text{test}}$ to select routes and feed the route to the simulator to obtain the actual number of UAVs recharged. The result is shown in Table 5.1. We compare three approaches: DOL (our), two-stage (classic supervise learning with MSE loss), and random (select routes randomly without any learning). As shown in Table 5.1, our approach on average can result in better route selection and recharge more UAVs.

5.3 Learning a Context-aware Objective for Information Gathering: (Non)-monotone and Possibly Non-positive Submodular Objective Case

In Section 5.2, we consider the problem of finding the mapping from context observations to the parameters of a monotone non-negative submodular objective by integrating the downstream decision-making problem. In this section, we study the case for a possibly non-monotone and possibly non-positive submodular objective.

We consider a specific VRP scenario where a team of UAVs along with a team of Unmanned Ground Vehicles (UGVs) are persistently monitoring an environment (Fig. 5.5). The UAVs occasionally land on the UGVs for recharging. However, due to the uncertainty inherent in the environment, failures and malfunctions are inevitable. For example, as shown in Fig. 5.5, the tracking camera of a UAV may stop working, and it may not reliably land on the UGV for recharging. Likewise, due to the stochastic wind field, the UAV may end up out of charge before reaching the UGV. In such cases, the human operator may have to intervene to recover the UAV (either by directly going to the UAV's location or by teleoperating the UGV to get to the UAV). Each intervention is costly as it requires the human operator's attention and causes an interruption to the monitoring task. We need to account for the cost of intervention in the route selection.

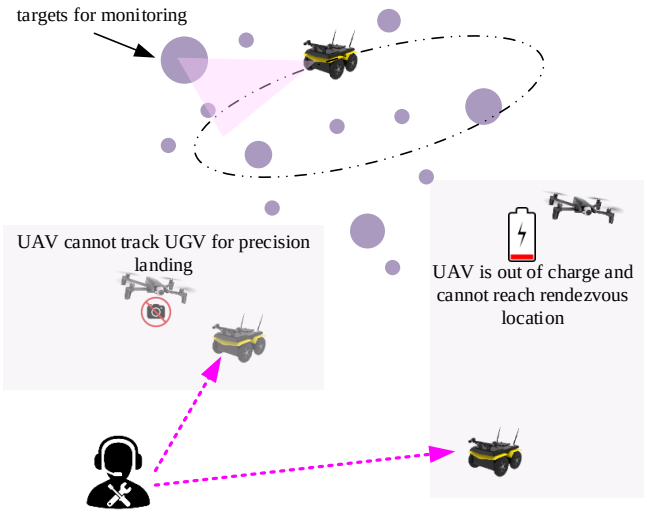


Figure 5.5: The motivating case study of Intervention-aware UGVs routing.

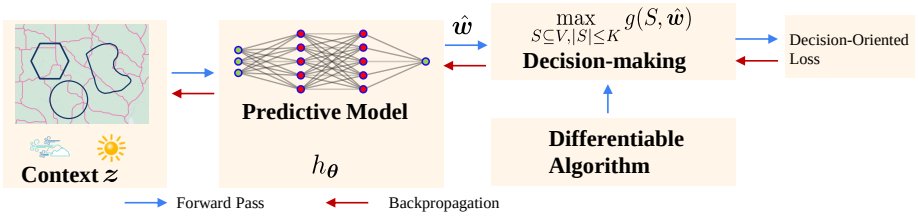


Figure 5.6: The proposed framework that incorporates the non-monotone submodular maximization into the learning process.

The intervention cost depends on the UAV and UGV routes as well as the environmental factors (e.g., wind conditions). We call all these factors together as *context*. We are interested in finding such a mapping from context observation to the intervention cost.

We consider the optimization objective to be a combination of the monitoring performance and the intervention cost. The monitoring performance is a monotone, submodular function for which a greedy algorithm yields a constant-factor approximation [15]. In Section 5.2, we presented a DOL framework for differentiable submodular maximization of monotone functions. However, by including the intervention cost in the objective, we cannot guarantee that the objective is monotone or even non-negative anymore. To accommodate such non-monotone optimization, we present a Differentiable Cost-Scaled Greedy (D-CSG) algorithm. The differentiability

of the non-monotone submodular maximization is achieved by using the multi-linear extension of the set function along with a novel differentiable algorithm, which expands and approximates the existing non-differentiable algorithm [132] as a differentiable computational graph.

Such research is also closely related to differentiable submodular maximization. Since the submodular objective and greedy algorithm are tightly coupled, it is necessary to consider the influence of the greedy algorithm when we consider learning submodular functions [128]. For the non-monotone submodular objective considered in this section, the simple greedy algorithm [15] does not have a performance guarantee, and we need to use a variant called the CSG algorithm to maximize the objective. As a result, the differentiable versions of the simple greedy algorithm [128, 129] cannot be directly used in our learning framework, and we need to develop our differentiable version of the CSG algorithm. Besides, our D-CSG algorithm is technically different from the existing differentiable greedy algorithm. The approach in [128, 129] is based on adding stochastic disturbances to the algorithm and using a gradient estimator. Our algorithm is based on the relaxation of the non-differentiable operation to a differentiable operation and the relaxation of the set function to a continuous counterpart.

5.3.1 Problem Formulation

Let $f : \{0,1\}^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ and $c : \{0,1\}^{\mathcal{V}} \to \mathbb{R}_{\geq 0}$ be a normalized monotone submodular function and a non-negative linear function, respectively. We are interested in a special type of submodular function $g : \{0,1\}^{\mathcal{V}} \to \mathbb{R}$, which is defined as

$$g(\boldsymbol{x}, \boldsymbol{w}) = f(\boldsymbol{x}) - \lambda c(\boldsymbol{x}, \boldsymbol{w}), \tag{5.9}$$

where $\boldsymbol{w}$ is cost vector for the set $\mathcal{V}$; $\boldsymbol{x} \in \{0,1\}^{\mathcal{V}}$; λ is a user-specified parameter for their cost tolerance level; and $c(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$.

It should be noted that g is still a submodular function, but it can take both positive and negative values and may not be monotone [132, 133]. Such a function is suitable to model the scenario where we need to balance the task performance ($f(\boldsymbol{x})$) with the cost needed to achieve the performance ($c(\boldsymbol{x}, \boldsymbol{w})$).

The decision-making is to solve the following problem:

$$\max_{\boldsymbol{x} \in \{0,1\}^{\mathcal{V}}} g(\boldsymbol{x}, \boldsymbol{w}) \tag{5.10}$$

$$\text{s.t. } \mathbf{1}^T \boldsymbol{x} \leq K, \tag{5.11}$$

where K is the number of elements that can be selected.

Our goal is to learn a function $h_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathbb{R}_+^N$ that maps the context observation $\boldsymbol{z} \in \mathcal{Z}$ to the objective parameters $\boldsymbol{w} \in \mathbb{R}_+^N$. Traditionally, finding the mapping $h_{\boldsymbol{\theta}}$ and optimizing the downstream objective $g(S, \boldsymbol{w})$ are considered separately: given the training data $\mathcal{D} = \{(\boldsymbol{z}_1, \boldsymbol{w}_1), (\boldsymbol{z}_2, \boldsymbol{w}_2), \ldots, (\boldsymbol{z}_{|\mathcal{D}|}, \boldsymbol{w}_{|\mathcal{D}|})\}$, find the mapping $h_{\boldsymbol{\theta}}$ by optimizing over $\boldsymbol{\theta}$ in a supervised fashion. After optimization, use the parameter $\boldsymbol{w} = h_\theta(\boldsymbol{z})$ for decision-making using (solve Eq. (5.10)) when we get an observation $\boldsymbol{z}$.

However, in robotic applications, the available training data is usually limited. Such a pipeline may result in a $h_{\boldsymbol{\theta}}$ that either overfits the data or cannot generalize well when deployed, i.e., leads to low-quality decisions in the downstream task. At a high level, the question that we will explore in this paper is:

Can we improve the decision quality in the downstream tasks if we explicitly incorporate the

downstream optimization into the process of learning $h_{\boldsymbol{\theta}}$?

Our answer is that optimizing the following decision-oriented loss can improve the decision quality compared to the baseline approach.

Decision-Oriented Loss: for a given training data $(\boldsymbol{z}_i, \boldsymbol{w}_i)$, the decision-oriented loss $\ell_{DOL}(\boldsymbol{w}_i, \hat{\boldsymbol{w}}_i)$ is defined through Eq. (5.12) to Eq. (5.14):

$$\hat{\boldsymbol{w}}_i \coloneqq h_{\boldsymbol{\theta}}(\boldsymbol{z}_i) \tag{5.12}$$

$$\hat{\boldsymbol{x}} \coloneqq \boldsymbol{x}^*(\hat{\boldsymbol{w}}_i) \text{ solving (5.10) with } \boldsymbol{w} = \hat{\boldsymbol{w}}_i \tag{5.13}$$

$$\ell_{DOL}(\hat{\boldsymbol{w}}_i, \boldsymbol{w}_i) \coloneqq g(\boldsymbol{x}^*(\boldsymbol{w}_i), \boldsymbol{w}_i) - g(\hat{\boldsymbol{x}}, \boldsymbol{w}_i), \tag{5.14}$$

where $\boldsymbol{x}^*(\boldsymbol{w}_i)$ denotes the solution of (5.10) returned by some approximation algorithms with $\boldsymbol{w} = \boldsymbol{w}_i$; $g(\boldsymbol{x}^*(\boldsymbol{w}_i), \boldsymbol{w}_i)$ denotes the decision quality when we use the ground truth parameter $\boldsymbol{w}_i$ for decisions; $g(\hat{\boldsymbol{x}}, \boldsymbol{w}_i)$ denotes the decision quality when we use the predicted parameter $\hat{\boldsymbol{w}}_i$ for decisions, i.e., use $\hat{\boldsymbol{w}}_i$ to obtain the decision $\hat{\boldsymbol{x}}$, but the decision is evaluated w.r.t. the true parameter $\boldsymbol{w}_i$.

The intuition for Eq. (5.14) is that we want to minimize the gap between the decision quality of the true parameters and that of the predicted parameters. One challenge is when we use the chain rule to compute the gradient of the loss function we need to differentiate through the optimization problem (the first term on the r.h.s. of Eq. (5.4)) as shown in the illustrative computational graph in Fig. 5.6.

$$\frac{\partial \ell_{DOL}}{\partial \boldsymbol{\theta}} = \frac{\partial \ell_{DOL}}{\partial \hat{\boldsymbol{w}}_i} \cdot \frac{\partial \hat{\boldsymbol{w}}_i}{\partial \boldsymbol{\theta}} \tag{5.15}$$

Next, we will show how to approximately compute the first term on the r.h.s. of Eq. (5.15).

5.3.2 Case Study: Intervention-Aware UGV Routing

As described earlier, we consider the case of UAVs and UGVs persistently monitoring the environment with occasional interventions to recover the nonoperational UAVs. We focus on selecting the routes for $n_g = K$ UGVs and assume that the UAV routes are fixed. Specifically, we consider that we are given a set of candidate routes, $\mathcal{T}$, and we need to select a subset from $\mathcal{T}$ for the UGVs. Here, $\mathcal{T}$ corresponds to the ground set $\mathcal{V}$. For any subset $S \in \mathcal{T}$, $f(S)$ gives the weighted coverage achieved by the UGVs executing the routes in S. Similarly, $c(S, w)$ gives the expected cost of intervention for the selected routes S where w are the unknown cost vectors. The objective then is:

$$\max_{S \in \mathcal{T}} g(S, w) \tag{5.16}$$

$$\text{s.t. } |S| \leq K. \tag{5.17}$$

We observe that this is the same problem as described earlier in Eq. 5.10.

The cost $c(S, w)$, i.e., how often and how much time we expect the human operators to be involved are closely related to how fast the UAVs consume energy and how close the UAVs and UGVs will be to each other when a recharging rendezvous is needed. From the UGVs' remote supervisor's perspective, some routes can cover many import task locations, but the UGV may be interrupted frequently to assist the UAVs. Since we plan the routes before knowing if/when/where the interventions will need to occur, our combined objective $g(S)$ aims to achieve a balanced performance between task coverage by the UGVs and expected human intervention

costs.

We can compute the intervention cost if we know the value of cost vector $\boldsymbol{w}$. If $\boldsymbol{w}$ is known, we can solve problem (5.10) using the existing algorithm [132]. However, in practice, we do not know the intervention cost in advance when we make decisions. Instead, we have some indirect information. We call the latter as *context* $\boldsymbol{z}$, which can include the geometric information of the routes of the UAVs and UGVs, the environmental conditions (e.g., wind conditions), etc. Hence, we solve two problems (simultaneously): predicting $\boldsymbol{w}$ from $\boldsymbol{z}$ and using the predicted $\boldsymbol{w}$ to find routes S by solving Eq. 5.10.

5.3.3 Learning Algorithm

In this section, we present the differentiable algorithm for solving our problem building on the cost-scaled greedy algorithm.

5.3.3.1 Cost-Scaled Greedy (CSG) Algorithm

Algorithm 8: Cost-Scaled Greedy (CSG) Algorithm

Input : Ground set V, scaled objective $\tilde{g}(S, \boldsymbol{w}) = f(S) - 2c(S, \boldsymbol{w})$, cardinality K
Output: A set $S \subseteq V$

$S \leftarrow \emptyset$
for i=1 to K **do**
  $e_i \leftarrow \text{argmax}_{e \in V} \tilde{g}(e \mid S)$
  if $\tilde{g}(e_i \mid S) \leq 0$ **then**
    break
  end
  $S \leftarrow S \cup \{e_i\}$
end
return S

The classic greedy algorithm cannot provide a performance guarantee for the objective in

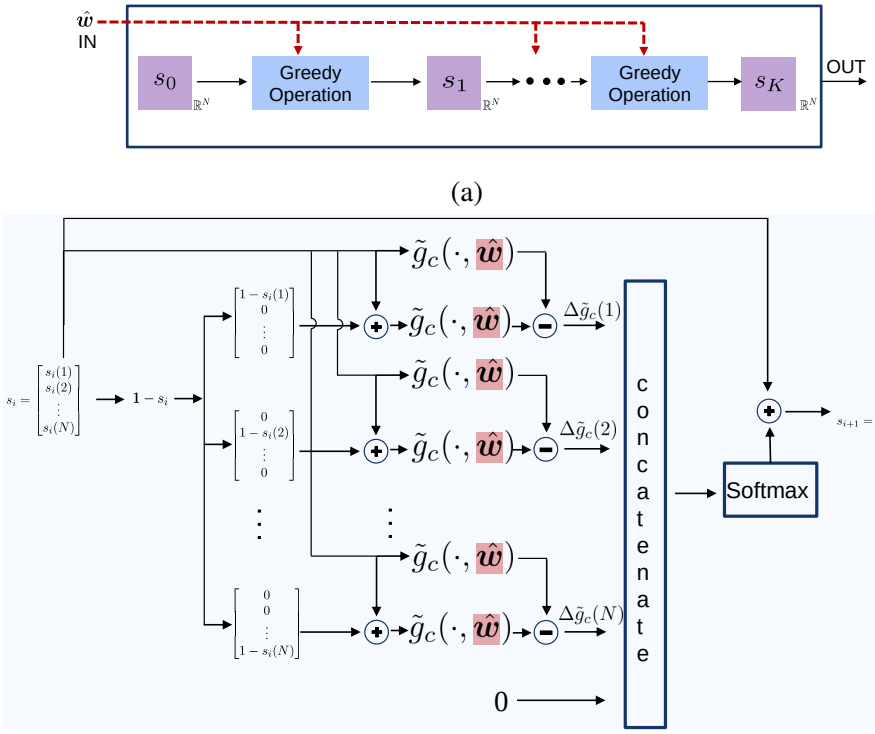


Figure 5.7: Computational graph of the proposed differentiable algorithm. (a) The structure of the algorithm. (b) The internal structure of the differentiable cost-scaled greedy operation.

Eq. (5.9). Instead, a modified version of the greedy algorithm, CSG, was proposed in [132] and was shown to achieve an approximation satisfying that $f(Q) - c(Q, \boldsymbol{w}) \geq \frac{1}{2} f(OPT) - c(OPT, \boldsymbol{w})$, where Q is the solution returned by Algorithm 8 and OPT refers to the optimal solution. It should be noted that the output of such an algorithm is not differentiable w.r.t. the parameter $\boldsymbol{w}$.

5.3.3.2 Multilinear Extension of Submodular Function

A prerequisite for D-CSG to work is that we need to have a continuous and differentiable relaxation of the objective in Eq. (5.10). The linear part, $c(\boldsymbol{x}, \boldsymbol{w})$, can be directly relaxed to a continuous version. As for the submodular part, $f(\boldsymbol{x})$, We use the multilinear extension to relax

the submodular part.

For a submodular function $f : \{0,1\}^N \to \mathbb{R}_{\geq 0}$, its multilinear extension $F : [0,1]^N \to \mathbb{R}_{\geq 0}$ is defined as

$$F(\boldsymbol{x}) = \sum_{\mathcal{S} \subseteq \mathcal{T}} f(\mathcal{S}) \prod_{i \in \mathcal{S}} x_i \prod_{i \notin \mathcal{S}} (1 - x_i), \tag{5.18}$$

which is a unique multilinear function agreeing with f in the vertices of the hypercude $[0,1]^N$.

Let $\boldsymbol{q}$ denote a random vector in $\{0,1\}^N$, where each coordinate is independently rouned to 1 with probability x_i or 0 otherwise. It can be shown that the derivative $\frac{\partial F}{\partial x_i}$ is

$$\frac{\partial F}{\partial x_i} = \mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{x}} \left[f([\boldsymbol{q}]_{i=1}) \right] - \mathbb{E}_{\boldsymbol{q} \sim \boldsymbol{x}} \left[f([\boldsymbol{q}]_{i=0}) \right], \tag{5.19}$$

where $[\boldsymbol{q}]_{i=1}$ and $[\boldsymbol{q}]_{i=0}$ are equal to the vector $\boldsymbol{q}$ with the i-th coordinate set to be 1 and 0, respectively.

5.3.3.3 Differentiable-Cost-Scaled Greedy (D-CSG) Algorithm

Based on the CSG algorithm, we developed one differentiable version of CSG. The key idea is to expand the computation steps as one computational graph, as shown in Fig. 5.7a. Suppose we must select up to K elements from a ground set whose size is N. We abstract the CSG algorithm as a K step computational graph as shown in Fig. 5.7a. The selection vector is initially an all-zero vector, i.e., $s_0 = \mathbf{0}, s_0 \in \mathbb{R}^N$. In each step, the greedy operation will try to set one element in the selection vector from 0 to 1 approximately. The details of the greedy operation are given in Fig. 5.7b. For an input vector s_i, we must first identify the elements that are not selected yet by doing $\mathbf{1} - s_i$. Then, we separate $\mathbf{1} - s_i$ into N vectors, each of which has one

element from $\mathbf{1} - s_i$ and the rest is zero. Each vector represents selecting an element from what is left in the ground set. If an element $s_i(j) \approx 1$ (j is already selected), $1 - s_i(j)$ is approximately zero, and the sum of this vector with s_i implies adding no new element to s_i. By contrast, if an element $s_i(j) \approx 0$ (j is not selected yet), $1 - s_i(j)$ is approximately one, and the sum of this vector with s_i implies adding one new element to s_i. Then, we feed the selection result to the continuous relaxation of the cost-scaled objective function, $\tilde{g}_c$, to compute the marginal gain. To account for the branch control in Algorithm 8 (line 4-6), we add one dummy element with zero marginal gain when we concatenate all the marginal values. Then, this concatenated vector will be fed into one argmax operator to select the one with the largest marginal gain (similar to line 3 in Algorithm 8). If all the marginal gains are less than zero, then the output of the argmax will choose the dummy element. As a result, the first N elements of the output of the argmax will all approximately to be zero, and the last element corresponding to the dummy selection will be one. Therefore, if we add the result of the first N elements to s_i to get a new N-dimensional vector, s_{i+1}, the s_{i+1} will be the same as s_i, which is in effect equivalent to skipping selection in this step. Such skipping step will also happen in the following steps since all marginal gains will be less than zero. It is equivalent to the branch control statement in Algorithm 8 (lines 4-6). It should be noted that the argmax operator itself is not differentiable and cannot be used during training. Instead, we use Gumbel-softmax [134], which uses a temperature parameter τ to scale how it is close to the argmax operator. A larger τ will make the approximate smoother, but the approximation error will also be larger. In experiments, this parameter is set empirically.

Remark 2. The greedy operation has two non-matrix operations: evaluation of $\tilde{g}_c$ ($2N$ times) and softmax. The latter is much faster than the former. As a result, the time for evaluation of $\tilde{g}_c$ will

dominate the forward pass the greedy operation.

5.3.4 Simulation

We evaluate the performance of the proposed framework for intervention cost prediction using synthetic data. We will first compare the performances of different algorithms to solve the randomly generated instances of Problem (5.10) to show the necessity and correctness of the D-CSG algorithm. Then, we will present one qualitative example of why the proposed framework is better than the classic one based on MSE loss. Next, we will present some quantitative results to show that the proposed framework leads to better decisions.

5.3.4.1 Simulation Setup

There are n_a UAVs and n_g UGVs. The global wind ω is represented as $[a, b, \omega_o]$, where a and b are the shape and scale parameters of Weibull distribution [135], respectively, and ω_o is the wind direction.

UAV and UGV Behavior: We consider the case that each task route of UAVs and UGVs is a circle, whose center is $[C_x, C_y]$ and radius is r. This geometric information can represent each route as a vector $[C_x, C_y, r]$. Our framework is also applied to more general types of task routes as long as we have a compact representation. We use circular routes for simplicity. Both vehicles need to monitor their assigned task routes persistently. Our main focus is on the UGV's side. As shown in Fig. 3.1, several targets of different values are scattered in the environment. The team of UGVs needs to cover some of them persistently. The team can get that value of the target if a target is covered by one UGV when the UGV traverses along the task route (when

two or more UGVs cover a target, we can only get the value once). The team of UGVs will try to get a higher value in sum. At a high level, the team is trying to maximize a generalized coverage function [136], which is submodular. As for the UAVs, when the UAV is about to be out of charge, i.e., the state of charge drops below 30%, it will fly toward the closest UGV. We use the same energy model as that in [130] with a stochastic wind model. If it can successfully rendezvous with the UGV, it will be recharged. If it has to land before it can rendezvous with the UGV. In such a case, a human operator must take over until all vehicles return to their tasks.

Raw Data: Based on the UAV UGV behavior models, we build a simulator for the routing problems as in our previous work [19, 21, 137]. Given simulation parameters (e.g., UAV and UGV routes and wind conditions, λ), it will simulate vehicles' execution of persistent tasks. Based on this simulator, we randomly generate a set of context observations, and for each context $\boldsymbol{z}_i$, there are a set of candidate task routes, $\mathcal{T}_i$, for UGVs. The size of $\mathcal{T}_i$ is fixed to be N. Then, we will get the data by making a team of n_g UGVs execute different combinations of routes: collect simulation data and compute a corresponding $\boldsymbol{w}_i$ (average intervention time in minutes per hour) for $\boldsymbol{z}$ for all routes in $\mathcal{T}_i$. Such a process is repeated many times to generate the training data. In the test phase, we generate a set of context observations and test the performance of the algorithm by directly running the simulator.

We test the performance of the proposed differentiable algorithm in synthetic instances of the problem in (5.10). We compare with two baselines: one is the Naive Greedy [15] and CSG.

Objective Value For each instance, we set the objective value returned by CSG as the denominator and scale the outputs of D-CSG and NG. As shown in Fig. 5.8, our D-CSG achieves comparable performance compared to CSG, which suggests that the differentiability does not sacrifice much optimization performance. By contrast, the performance of the NG is, on aver-

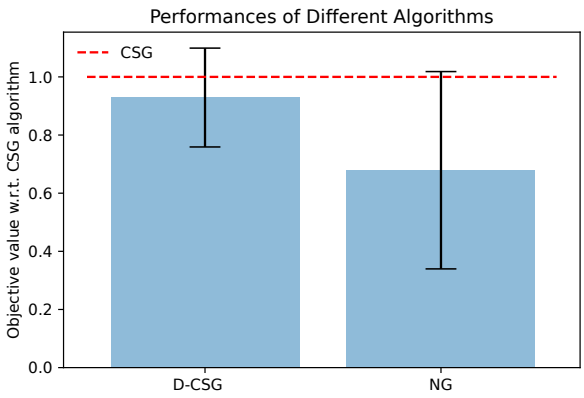


Figure 5.8: Simulation results for the network (D-CSG).

age, worse than that of D-CSG, which justifies our motivation to develop a novel differentiable algorithm rather than using the differentiable version of NG.

Running Time The price of differentiability is mainly reflected in the running time. In experiments, we observe that the D-CSG is usually 20-30 times slower than CSG. This is mainly because the evaluation of the continuous relaxation of the submodular objective is time-consuming, which can be viewed as a polynomial with exponentially many terms w.r.t. the size of the route set. The running time can be improved by using an estimator for function evaluation and gradient computation [138]. We leave this for future work.

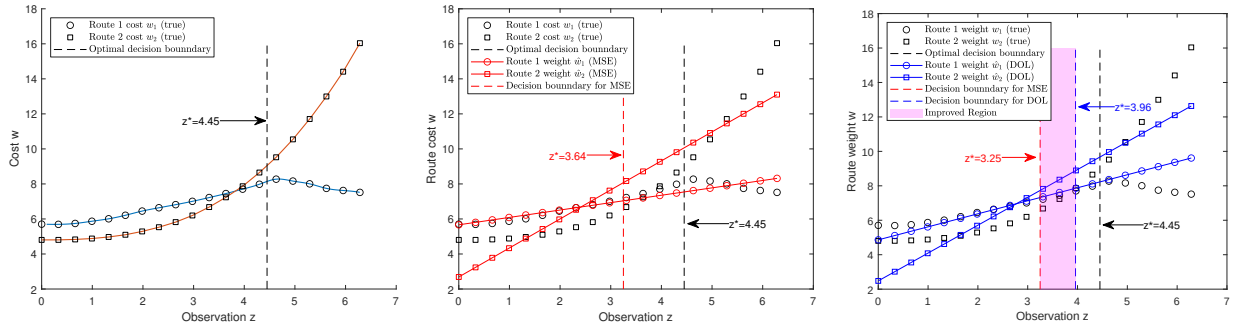
Table 5.2: Submodular Function for Qualitative Example

	s_1	s_2	s_3	s_1, s_2	s_1, s_3	s_2, s_3	s_1, s_2, s_3
$f(\cdot)$	16	17	25	21	37	38	41

A Qualitative Example Let us consider a normalized submodular function f ,i.e., $f(\emptyset) = 0$, defined over a ground set $\mathcal{S} = \{s_1, s_2, s_3\}$. Each element in $\mathcal{S}$ can be viewed as a task route for UGVs. The values of f for choosing different elements are shown in Table 5.2. Verifying the submodularity of f is easy using the definition. Each $\mathcal{S}$ element is associated with a context-dependent cost. Suppose that the cost of s_3 is constantly to be one, and the ground truth costs

Table 5.3: Test Results of Objective Values

Training Parameters	DOL	two-stage	random
$n_a = 10, n_g = 3, N = 10$	**203** $\pm$ 30	167 $\pm$ 26	81 $\pm$ 53
$n_a = 15, n_g = 5, N = 12$	**325** $\pm$ 27	291 $\pm$ 27	102 $\pm$ 69
$n_a = 20, n_g = 7, N = 15$	**471** $\pm$ 34	419 $\pm$ 23	227 $\pm$ 51



(a) (b) (c)

Figure 5.9: A case study with three candidate routes and two UGVs. (a) Application scenario. (b) Ground truth data and the optimal decision boundary. (c) Learned linear models using MSE loss. (d) Learned linear models using the DOL framework.

for s_1 and s_2 are shown in Fig.5.9a. We are interested in solving a problem as defined in Eq. (5.10) with $K = 2$. When we make decisions, we can only see the context, and we need to infer the route costs. The optimal decision is either $\{s_1, s_3\}$ or $\{s_2, s_3\}$. If we know the ground truth context-to-cost function as shown in Fig.5.9a, the optimal decision boundary is $z = 4.45$ at which the cost choosing s_2 is greater than that of s_1 by 1. Namely, if the context observation z is less than 4.45, we should choose s_2 and s_3. When z exceeds 4.45, we should choose s_1 and s_3. Next, let us look at the result if learning is involved. We want to find a mapping from the observation z to costs. Suppose we obtain the training data by sampling from the ground truth as shown in Fig.5.9a. MSE as the objective for learning without considering the downstream task, we will get two lines as shown in Fig. 5.9b. The decision boundary (dashed vertical red line, $z^* = 3.64$, at which $w_2 - w_1 = 1$) is on the left of the optimal boundary, thus not optimal. By contrast,

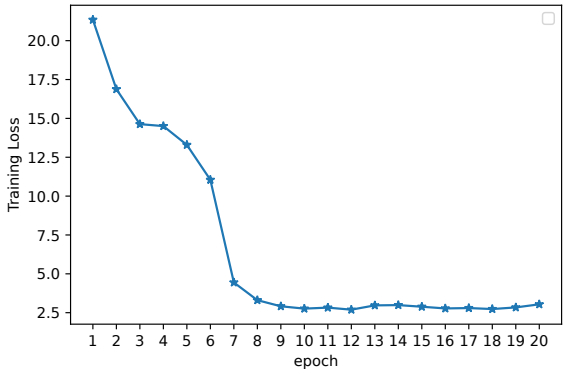


Figure 5.10: Per epoch loss curve of DOL.

if we consider the downstream optimization, we will get two lines as shown in Fig. 5.9 and the decision boundary (dashed vertical blue line, $z^* = 3.96$, at which $w_2 - w_1 = 1$) is closer to the optimal boundary, thus reducing the regions of suboptimal decisions.

We use a three-layer neural network to approximate the mapping $g_{\boldsymbol{\theta}}$. The number of parameters in the neural network will vary according to the number of UAVs (n_a) and the number of routes (N). In training, we use MSE loss first for a few epochs to as a warm start. Then, we will use DOL loss defined in Eq. (5.3). Fig. 5.10 shows one typical learning curve using the DOL loss over epochs. The loss will decrease first and reach a steady value, which suggests that the decision quality gap between the reference solution and the one obtained using predictions from $g_{\boldsymbol{\theta}}$ is decreasing.

After training, we test the performance of the learned models using the simulator. We generate a set of context observations $\{\boldsymbol{z}_i\}_{\text{test}}$ and compute the corresponding predicted weights $\{\hat{\boldsymbol{w}}_i\}_{\text{test}}$. Then, we use $\{\hat{\boldsymbol{w}}_i\}_{\text{test}}$ to select routes and feed the route to the simulator to obtain the actual intervention time and the coverage value. Then, we will use these two values to compute the objective value as defined in (5.9). The result of the objective value is shown in Table 5.3. We compare three approaches: DOL (our), two-stage (classic supervise learning with MSE loss),

and random (select routes randomly without any learning). As shown in Table 5.3, our approach, on average, can result in a more balanced route selection (trade-off between task coverage and intervention time) and get a higher objective value for the route selections.

5.4 Conclusion

We propose two decision-oriented learning frameworks that explicitly incorporate downstream tasks into the learning process. The key to achieving such integration is to make the combinatorial optimization differentiable. For the case where the objective in the task optimization is a parameterized monotone non-negative submodular function, we design a stochastic perturbation-based approach to make the task optimization differentiable. For the case where the objective in the task optimization is a parameterized (non)-monotone and possibly non-positive submodular function, we show how to make task optimization a differentiable layer by using the proposed D-CSG algorithm and the multilinear extension of the objective function. Both frameworks are validated in VRP-related case studies.

Chapter 6: Conclusion and Future Work

6.1 Summary of Contributions

This dissertation addressed the problem of how to coordinate multi-robot teams to gather information effectively given the available knowledge about the environment and the capability constraints in robots. Our research spans the whole spectrum of the knowledge of the environment model: from one extreme where the environment model is fully known to another extreme where the environment model is unknown but can be learned from empirical data, and considers diverse constraints in the robotic platform, including communication radius, battery capacity, and possible platform failures (i.e., robots are out of service). Our contributions can be summarized as follows.

- When the environment model is fully known, we proposed a problem named Communication-aware Submodular Maximization (CSM) for a class of multi-robot information gathering problems that have submodular objectives and require intermittent connectivity between robots and proposed an efficient two-stage algorithm to solve the CSM problem.

- When the multi-robot team has to perform information gathering tasks in an uncertain environment where some of them may fail and we only have one pessimistic estimation about the failures: at most α failures, we proposed a novel formulation named Robust

Multiple-path Orienteering Problem (RMOP) and proposed one efficient algorithm with worst-case guarantees to solve RMOP.

- When the multi-robot team has to perform information gathering tasks in an uncertain environment where we know the distribution of the uncertainty, we design two risk-aware routing algorithms for the coordination of a heterogenous team of UAVs and UGVs.

- When the environment model is unknown and context-dependent, and we need to learn an environment model from the empirical data, we proposed novel decision-oriented-learning frameworks that explicitly incorporate the downstream decision-making into the learning process and technically show how to make submodular maximization and its variants differentiable.

6.2 Future Directions

Although we have explored several aspects of multi-robot coordination and learning algorithms for information gathering tasks, we have only touched the tip of the iceberg, and there are still many aspects that are worth more research efforts. Specifically, this dissertation can be extended and augmented in the following aspects.

6.2.1 Decentralized and Anytime Decision-making Algorithms

In this dissertation, the coordination algorithms are designed to be executed in a centralized fashion. Such algorithms are suitable for scenarios where we need to make decisions once at a central server and then let the robots follow the decisions. However, in the field, we may need to make decisions many times for a given task, and it is more desirable for robots to be able to

make decisions in a decentralized fashion. Although there is already some research going in this direction, the scalability of the algorithm, performance guarantee, and robustness to unreliable communication are still largely unsolved. Similarly, when we deploy the robots in the field, the computational power of the robot is limited. It is desirable to design anytime algorithms for robots such that robots can have a good enough solution when they do not have enough time to solve the decision-making problem (near)-optimally. Machine learning-based techniques may be used in parallel to boost the performance of the anytime algorithm in two ways: first, the machine learning model may help the algorithm to guess a good enough warm start for the algorithm; second, the machine learning model may help to configure the parameters of the anytime algorithm based on the online observations of the environments.

6.2.2 Human-robot Teaming

A growing trend in the robotics community nowadays is to consider the human factor in robot autonomy. This dissertation can also be extended in this direction. If humans work as remote supervisors of a large robotic team (i.e., whenever the robots encounter some unsolvable deadlock or failure, human supervisors need to take over), for a given long-term information gathering task, what is the minimum number of human supervisors to guarantee the relatively *smooth* operation of the robotic team considering that malfunction or failures of robots are inevitable? If human also needs to be part of the team for a cooperative task, how do we design task allocation and communication mechanism such that we can fully exploit human advantages (e.g., capture and analyze semantic information) without overburdening them? Besides, if human feedback has to be incorporated into the online decision-making loop, how do we evaluate the human exper-

tise level based on the history record and how do we incorporate the human feedback into the decision-making framework based on our evaluation? Besides, human expertise for information gathering tasks may gradually increase, but the willingness for repeated tasks may decrease, how do we account for these factors in the decision-making pipelines? The studies to answer these research questions will be a good complement to this dissertation.

6.2.3 Decision-oriented Learning for a Broader Class of Combinatorial Optimization

In the decision-oriented learning chapter, we consider the problem of learning a mapping from context observation to the parameters only in the objective function. Besides, for the decision-making problem, we only consider the matroid constraints (e.g., cardinality constraint). What if we also need to predict the parameters in the constraint (e.g., budget constraint) and the constraint is not a matroid (e.g., routing constraint)? In such a case, we need to design new, differentiable algorithms to incorporate decision-making problems into the learning process. Moreover, the objective may not be exactly submodular but approximately submodular. How to design differentiable algorithms for such problems is an important and interesting future research direction. Besides, the learning algorithm in Chapter 5 is suitable for the case where the data is already collected and we just need to learn the model offline. What if the data streams in and we need to learn the model online and simultaneously make decisions? Such a problem can be further complicated if the streaming data is incomplete (part of the data is missing) and noisy. All these questions need further research efforts after this dissertation.

Appendix A: Proofs for RMOP

A.1 Proof of Theorem 1

Let $\mathcal{V}$ be a finite ground set and $f : 2^{\mathcal{V}} \to \mathbb{R}^+$ a normalized nondecreasing submodular set function. In our setup, the ground set $\mathcal{V}$ represents the collection of all feasible paths for all robots[1] and f represents the sensing function. Given a set $\Omega \subseteq \mathcal{V}$ and an ordered set $\mathcal{S} = \{s_1, s_2, \ldots, s_t\}$, we define $\mathcal{S}_0 = \emptyset$ and $\mathcal{S}_i = \{s_1, s_2, \ldots, s_i\}$ for $1 \leq i \leq t$, and

$$k_0 = 1 - \min_{i:s_i \in \mathcal{S}^\dagger} \frac{f_{\mathcal{S}_{i-1} \cup \Omega}(s_i)}{f_{\mathcal{S}_{i-1}}(s_i)}, \tag{A.1}$$

where $\mathcal{S}^\dagger = \{s_i \in \mathcal{S} \setminus \Omega \mid f_{\mathcal{S}_{i-1}}(s_i) > 0\}$ and $i : s_i \in \mathcal{S}^\dagger$ denotes $i \in \{j \mid s_j \in \mathcal{S}^\dagger\}$.

Recall that the definition of the total curvature is

$$k_f = 1 - \min_{s_i \in \mathcal{V}^\dagger} \frac{f_{\mathcal{V} \setminus \{s_i\}}(\{s_i\})}{f(\{s_i\})},$$

where $\mathcal{V}^\dagger = \{s_i \in \mathcal{V} \mid f(\{s_i\}) > 0\}$.

Lemma 1.

$$k_0 \leq k_f. \tag{A.2}$$

[1] Note that we do not actually require this full set as input for the algorithm.

Proof.

$$k_0 = \max_{i:s_i \in \mathcal{S}^\dagger} \frac{f_{\mathcal{S}_{i-1}}(s_i) - f_{\mathcal{S}_{i-1} \cup \Omega}(s_i)}{f_{\mathcal{S}_{i-1}}(s_i)} \tag{A.3}$$

$$= \max_{i:s_i \in \mathcal{S}^\dagger} 1 - \frac{f_{\mathcal{S}_{i-1} \cup \Omega}(s_i)}{f_{\mathcal{S}_{i-1}}(s_i)} \tag{A.4}$$

$$\leq \max_{i:s_i \in \mathcal{S}^\dagger} 1 - \frac{f_{\mathcal{G} \setminus \{s_i\}}(s_i)}{f(s_i)} \tag{A.5}$$

$$\leq \max_{i:s_i \in \mathcal{G}^\dagger} 1 - \frac{f_{\mathcal{G} \setminus \{s_i\}}(s_i)}{f(s_i)} = k_f. \tag{A.6}$$

Eq. A.5 follows Eq. A.4 due to the monotonicity of the submodular function. Eq. A.6 follows Eq. A.5 since $\mathcal{S}^\dagger \subseteq \mathcal{G}^\dagger$. ■

Lemma 2.

$$f(\Omega \cup \mathcal{S}) = f(\Omega) + \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\Omega \cup \mathcal{S}_{i-1}}(s_i). \tag{A.7}$$

Proof. By definition, $f(\Omega \cup \mathcal{S})$ can be computed by first computing $f(\Omega)$ and then summing over the marginal gain of each element $s_i \in \mathcal{S} \setminus \Omega$. Suppose we sum these marginal gain using the same order as those elements show in $\mathcal{S}$, then $f_{\Omega \cup \mathcal{S}_{i-1}}(s_i)$ represents the marginal gain of s_i. ■

Lemma 3.

$$f(\Omega) \leq k_0 \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{i:s_i \in \Omega \cap \mathcal{S}} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{i:\omega_i \in \Omega \setminus \mathcal{S}} f_{\mathcal{S}}(\omega). \tag{A.8}$$

Proof. By definition and submodularity,

$$f(\Omega \cup \mathcal{S}) = f(\mathcal{S}) + f_{\mathcal{S}}(\Omega \setminus \mathcal{S}) \tag{A.9}$$

$$\leq f(\mathcal{S}) + \sum_{\omega \in \Omega \setminus \mathcal{S}} f_{\mathcal{S}}(\omega). \tag{A.10}$$

By Lemma 2 and the definition of k_0,

$$f(\Omega \cup \mathcal{S}) = f(\Omega) + \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\Omega \cup \mathcal{S}_{i-1}}(s_i) \tag{A.11}$$

$$\geq f(\Omega) + (1 - k_0) \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\mathcal{S}_{i-1}}(s_i). \tag{A.12}$$

Combining Eq. (A.10) and Eq. (A.12), we have

$$\begin{aligned} f(\Omega) \leq k_0 \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\mathcal{S}_{i-1}}(s_i) \quad & + f(\mathcal{S}) - \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\mathcal{S}_{i-1}}(s_i) \\ & + \sum_{\omega \in \Omega \setminus \mathcal{S}} f_{\mathcal{S}}(\omega) \end{aligned} \tag{A.13}$$

$$= k_0 \sum_{i:s_i \in \mathcal{S} \setminus \Omega} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{i:s_i \in \mathcal{S} \cap \Omega} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{\omega \in \Omega \setminus \mathcal{S}} f_{\mathcal{S}}(\omega). \tag{A.14}$$

■

Let $\eta \geq 1$ be the approximate factor for SOP and $\mathcal{O} = \{\mathcal{O}_1^*, \mathcal{O}_2^*, \ldots, \mathcal{O}_i^*, \ldots, \mathcal{O}_N^*\}$ be the optimal solution to MOP. The SGA solution up to the stage i is denoted as $\mathcal{A}_i = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_i\}$ and we use $\mathcal{A} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_i, \ldots, \mathcal{P}_N\}$ to denote the solution returned by SGA. It should be noted that $\mathcal{A}$ is an ordered set and $\mathcal{O}$ is an unordered set.

Let $\sigma_{\mathcal{S}}(s_i)$ denote the index of the element s_i in the ordered set $\mathcal{S}$. For example, $\sigma_{\mathcal{A}}(\mathcal{P}_i) = i$.

In the following proof, we will treat $\mathcal{O}$ as an ordered set (different orders of $\mathcal{O}$ will not change $f(\mathcal{O})$) and order the elements in $\mathcal{O}$ in such a way:

$$\text{if } \mathcal{O}_i^* \in \mathcal{A} \cap \mathcal{O}, \;\; \sigma_{\mathcal{O}}(\mathcal{O}_i^*) = \sigma_A(\mathcal{O}_i^*), \tag{A.15}$$

i.e., if an element is in the intersection of two ordered set, it has the same index in these two sets. One direct result with such ordered set is that

$$\{i \mid o_i \in \mathcal{O} \cap \mathcal{A}\} = \{i \mid a_i \in \mathcal{A} \cap \mathcal{O}\}, \tag{A.16}$$

and

$$\begin{aligned} \{i \mid o_i \in \mathcal{O} \setminus \mathcal{A}\} &= \{1, 2, \ldots, N\} - \{i \mid o_i \in \mathcal{O} \cap \mathcal{A}\} \\ &= \{1, 2, \ldots, N\} - \{i \mid a_i \in \mathcal{A} \cap \mathcal{O}\} \\ &= \{i \mid a_i \in \mathcal{A} \setminus \mathcal{O}\}. \end{aligned} \tag{A.17}$$

Next we will start our proof of Theorem 1.

Proof. By Lemma 3,

$$\begin{aligned} f(\mathcal{O}) \leq k_0 \sum_{i:a_i \in \mathcal{A} \setminus \mathcal{O}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:a_i \in \mathcal{O} \cap \mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) \\ + \sum_{i:o_i \in \mathcal{O} \setminus \mathcal{A}} f_{\mathcal{A}}(o_i). \end{aligned} \tag{A.18}$$

Considering the first term on the right hand side, by monotonicity, we have

$$k_0 \sum_{i:a_i \in \mathcal{A} \setminus \mathcal{O}} f_{\mathcal{A}_{i-1}}(a_i) \leq k_0 \sum_{i:a_i \in \mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) = k_0 f(\mathcal{A}).$$

For the second and third term,

$$\sum_{i:a_i \in \mathcal{O} \cap \mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{o_i \in \mathcal{O} \setminus \mathcal{A}} f_{\mathcal{A}}(o_i) \tag{A.19}$$

$$\leq \sum_{i:a_i \in \mathcal{O} \cap \mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{o_i \in \mathcal{O} \setminus \mathcal{A}} f_{\mathcal{A}_{i-1}}(o_i) \tag{A.20}$$

$$\leq \sum_{i:a_i \in \mathcal{O} \cap \mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:o_i \in \mathcal{O} \setminus \mathcal{A}} f_{\mathcal{A}_{i-1}}(\mathcal{P}_i^{*\text{local}}) \tag{A.21}$$

$$\leq \sum_{i:a_i \in \mathcal{O} \cap \mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:a_i \in \mathcal{A} \setminus \mathcal{O}} \eta f_{\mathcal{A}_{i-1}}(a_i) \tag{A.22}$$

$$= \sum_{i:a_i \in \mathcal{O} \cap \mathcal{A}} \eta f_{\mathcal{A}_{i-1}}(a_i) + \sum_{a_i \in \mathcal{A} \setminus \mathcal{O}} \eta f_{\mathcal{A}_{i-1}}(a_i) \tag{A.23}$$

$$= \eta f(A), \tag{A.24}$$

where $i : o_i \in \mathcal{O} \setminus \mathcal{A}$ ($i : a_i \in \mathcal{A} \setminus \mathcal{O}$) denotes $i \in \{i \mid o_i \in \mathcal{O} \setminus \mathcal{A}\}$ ($i \in \{i \mid a_i \in \mathcal{A} \setminus \mathcal{O}\}$).

Eq. (A.20) follows from Eq. (A.19) due to the submodularity. Eq. (A.21) follows from Eq. (A.20) due to the definition of $\mathcal{P}_i^{*\text{local}}$:

$$\mathcal{P}_i^{*\text{local}} = \max_{\mathcal{P}} \ f_{\mathcal{A}_{i-1}}(\mathcal{P}).$$

Eq. (A.22) follows from Eq. (A.21) based on Eq. (A.17) and the property of the approximation algorithm. Eq. (A.23) transits to Eq. (A.24) by definition.

Combining the inequalities for all terms on the right hand side, we have

$$f(\mathcal{O}) \leq k_0 f(\mathcal{A}) + \eta f(\mathcal{A}) \tag{A.25}$$

$$\leq (k_f + \eta) f(\mathcal{A}). \tag{A.26}$$

■

A.2 Proof of Theorem 2

Proof. Our proof relies on the following three inequalities:

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq (1 - k_f) f(\mathcal{S}_2) \tag{A.27}$$

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{1}{\alpha + 1} f(\mathcal{S}_2) \tag{A.28}$$

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{1}{N - \alpha} f(\mathcal{S}_2) \tag{A.29}$$

The proof for inequality (A.27) is similar to the proof (Eq. 16-20) in [71] when combined with the invariant maintained by the **while** loop in Algorithm 3. Likewise, the proof for inequality (A.28) resembles that given in [71] (from Eq. 21 to Eq. 25) and inequality (A.29) can be proved in the same fashion as that in [75] (Eq. 57 to Eq. 58).

From Theorem 1 we have,

$$f(\mathcal{S}_2) \geq \frac{1}{k_f + \eta} f(\mathcal{Q}^*) \tag{A.30}$$

where $\mathcal{Q}^*$ is the optimal solution to the multi-path orienteering problem for robots $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$. Similar to Lemma 9 in [75], we have another inequality:

$$f(\mathcal{Q}^*) \geq f^* = f(\mathcal{S}^* \setminus \mathcal{A}^*(\mathcal{S}^*)), \tag{A.31}$$

where f^* is the optimal solution to RMOP. For completeness, we give the proof in the supplementary document. Combining inequalities (A.27) – (A.31), we get the statement for Theorem 2. ■

A.3 Proof of Inequality in Equation (A.27)

We will prove the following,

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq (1 - k_f) f(\mathcal{S}_2) \tag{A.32}$$

where $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ is the solution returned by our algorithm; $\mathcal{A}^*(S)$ is the optimal removal of $\mathcal{S}$; k_f is the curvature of function f. Next, we prove inequality A.32 and we will use Lemma 1 from [71] without proof. Proof here is essentially the same as that in [71] but with different notations for better understanding.

Lemma 4. *Consider a finite ground set $\mathcal{V}$ and a monotone set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ such that f is a non-negative and $f(\emptyset) = 0$. For any set $\mathcal{A} \subseteq \mathcal{V}$,*

$$f(\mathcal{A}) \geq (1 - k_f) \sum_{a \in \mathcal{A}} f(a) \tag{A.33}$$

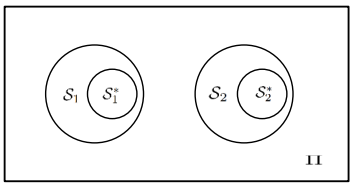


Figure A.1: Venn diagram, where $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_1^*, \mathcal{S}_2^*$ are defined as follows: Per run of proposed algorithm for RMOP, $\mathcal{S}_1$ and $\mathcal{S}_2$ are intermediate results such that $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, and $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. Let $\mathcal{S}_{\mathcal{A}}^*(\mathcal{S})$ be the optimal removal from $\mathcal{S}$. Then $\mathcal{S}_1^*, \mathcal{S}_2^*$ are defined such that $\mathcal{S}_1^* = \mathcal{A}^*(\mathcal{S}) \cap \mathcal{S}_1$ and $\mathcal{S}_2^* = \mathcal{A}^*(\mathcal{S}) \cap \mathcal{S}_2$. By definition, $\mathcal{S}_1^* \cap \mathcal{S}_2^* = \emptyset$ and $\mathcal{S}_{\mathcal{A}}^*(\mathcal{S}) = \mathcal{S}_1^* \cup \mathcal{S}_2^*$.

Let $\mathcal{S}_1^+ = \mathcal{S}_1 \setminus \mathcal{S}_1^*$ and $\mathcal{S}_2^+ = \mathcal{S}_2 \setminus \mathcal{S}_2^*$

$$f(\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})) = f(\mathcal{S}_1^+ \cup \mathcal{S}_2^+) \tag{A.34}$$

$$\geq (1 - k_f) \sum_{s \in \mathcal{S}_1^+ \cup \mathcal{S}_2^+} f(s) \tag{A.35}$$

$$\geq (1 - k_f)(\sum_{s \in \mathcal{S}_2 \setminus \mathcal{S}_2^+} f(s) + \sum_{s \in \mathcal{S}_2^+} f(s)) \tag{A.36}$$

$$\geq (1 - k_f)(f(\mathcal{S}_2 \setminus \mathcal{S}_2^+) + f(\mathcal{S}_2^+)) \tag{A.37}$$

$$\geq (1 - k_f)((\mathcal{S}_2 \setminus \mathcal{S}_2^+) \cup \mathcal{S}_2^+) \tag{A.38}$$

$$= (1 - k_f) f(\mathcal{S}_2) \tag{A.39}$$

where (A.34) holds by definition; (A.34) to (A.35) holds due to Lemma 4; (A.36) follows from (A.35) since all paths $s \in \mathcal{S}_1^+$ and all paths $s' \in \mathcal{S}_2 \setminus \mathcal{S}_2^+$, the inequality $f(s) \geq f(s')$ holds, i.e. paths in $\mathcal{S}_1$ have more rewards compared to that in $\mathcal{S}_2$ (note that by definitions of sets $\mathcal{S}_1^+$ and $\mathcal{S}_2^+$ it is $|\mathcal{S}_1^+| = |\mathcal{S}_2^*| = |\mathcal{S}_2 \setminus \mathcal{S}_2^+|$, i.e. the number of non-removed paths in $\mathcal{S}_1$ is equal to the number of removed paths in $\mathcal{S}_2$); from (A.36) to (A.37), it is due to submodularity; and (A.39) follows from (A.38) by definition.

A.4 Proof of Inequality in Equation (A.28)

We will prove the following,

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{1}{\alpha+1} f(\mathcal{S}_2) \tag{A.40}$$

We will use Lemma 2 from [71] without proof. Proof here is essentially the same as that in [71] but with notations consistent with our paper for better understanding.

Lemma 5. *Consider any finite ground set $\mathcal{V}$, a monotone submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$ such that f is a non-negative and $f(\emptyset) = 0$. Consider two non-empty sets $\mathcal{Y}, \mathcal{P} \subseteq \mathcal{V}$ such that for all elements $y \in \mathcal{Y}$ and all elements $p \in \mathcal{P}$ it is $f(y) \geq f(p)$. Then,*

$$f_{\mathcal{Y}}(\mathcal{P}) \leq |\mathcal{P}| f(\mathcal{Y}) \tag{A.41}$$

First we introduce one notation:

$$\xi = \frac{f_{\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*)}{f(\mathcal{S}_2)} \tag{A.42}$$

To prove (A.40), we still need to discuss two cases: $\mathcal{S}_2^* = \emptyset$ and $\mathcal{S}_2^* \neq \emptyset$. When $\mathcal{S}_2^* = \emptyset$, we have $f(\mathcal{S} \setminus \mathcal{A}^*(S)) = f(\mathcal{S}_2)$, and (A.40) holds. Next, we consider the case where $\mathcal{S}_2^* \neq \emptyset$ holds. The proof starts with one observation that

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \max\{f(\mathcal{S} \setminus \mathcal{A}^*(S)), f(\mathcal{S}_1^+)\}, \tag{A.43}$$

and then prove the following three inequalities:

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq (1-\xi) f(\mathcal{S}_2) \tag{A.44}$$

$$f(\mathcal{S}_1^+) \geq \xi \frac{1}{\alpha} f(\mathcal{S}_2) \tag{A.45}$$

$$\max\{(1-\xi, \xi\frac{1}{\alpha})\} \geq \frac{1}{\alpha+1} \tag{A.46}$$

Next, if substitute (A.44), (A.45), and (A.46) to (A.43), then (A.40) is proved.

1) *Proof of inequalities* $0 \leq \xi \leq 1$: Since f is non-negative and therefore by definition $\xi \geq 0$. For numerator of ξ, by submodularity, $f_{\mathcal{S}\setminus\mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*) \leq f(\mathcal{S}_2^*)$ and notice that $\mathcal{S}_2^*$ is a subset of $\mathcal{S}_2$. Therefore,

$$\begin{aligned} \xi &= \frac{f_{\mathcal{S}\setminus\mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*)}{f(\mathcal{S}_2)} \\ &\leq \frac{f(\mathcal{S}_2^*)}{f(\mathcal{S}_2)} \leq 1 \end{aligned}$$

2) *proof of inequality A.44*: The proof can be done in two steps. Firstly, it can be verified using $f_{\mathcal{A}}(\mathcal{B}) = f(\mathcal{A} \cup \mathcal{B}) - f(\mathcal{A})$ that

$$\begin{aligned} f(\mathcal{S} \setminus \mathcal{A}^*(S)) = f(\mathcal{S}_2) - f_{\mathcal{S}\setminus\mathcal{A}^*(S)}(\mathcal{S}_2^*) \\ + f_{\mathcal{S}_2}(\mathcal{S}_1) - f_{\mathcal{S}\setminus\mathcal{S}_1^*}(\mathcal{S}_1^*) \end{aligned} \tag{A.47}$$

It should be noted that $f_{\mathcal{S}_2}(\mathcal{S}_1) - f_{\mathcal{S}\setminus\mathcal{S}_1^*}(\mathcal{S}_1^*) \geq 0$ for two following observations: i) $f_{\mathcal{S}_2}(\mathcal{S}_1) \geq f_{\mathcal{S}_2}(\mathcal{S}_1^*)$ since f is monotone and $\mathcal{S}_1^* \subseteq \mathcal{S}_1$; ii) $f_{\mathcal{S}_2}(\mathcal{S}_1^*) \geq f_{\mathcal{S}\setminus\mathcal{S}_1^*}(\mathcal{S}_1^*)$ since f is submodular and

$\mathcal{S}_2 \subseteq \mathcal{S} \setminus \mathcal{S}_1^*$ (see also Fig. A.1). Then we have

$$\begin{aligned} f(\mathcal{S} \setminus \mathcal{A}^*(S)) &\geq f(\mathcal{S}_2) - f_{\mathcal{S}\setminus\mathcal{A}^*(S)}(\mathcal{S}_2^*) \\ &= f(\mathcal{S}_2) - \xi f(\mathcal{S}_2) \end{aligned}$$

Inequality A.44 proved.

3) *Proof of inequality A.45*: Since it is $\mathcal{S}_2^* \neq \emptyset$ which suggests that $\mathcal{S}_1^+ \neq \emptyset$ and for all paths $a \in \mathcal{S}_1^+$ and all elements $b \in \mathcal{S}_2^*$ it is $f(a) \geq f(b)$, from Lemma 5, we have

$$\begin{aligned} f_{\mathcal{S}_1^+}(\mathcal{S}_2^*) &\leq |\mathcal{S}_2^*| f(\mathcal{S}_1^+) \\ &\leq \alpha f(\mathcal{S}_1^+) \end{aligned}$$

Since $|\mathcal{S}_2^*| \leq \alpha$. Overall,

$$\begin{aligned} f(\mathcal{S}_1^+) &\geq \frac{1}{\alpha} f_{\mathcal{S}_1^+}(\mathcal{S}_2^*) \\ &\geq \frac{1}{\alpha} f_{\mathcal{S}_1^+ \cup \mathcal{S}_1^+}(\mathcal{S}_2^*) \\ &= \frac{1}{\alpha} f_{\mathcal{S}\setminus\mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*) \\ &= \xi \frac{1}{\alpha} f(\mathcal{S}_2) \end{aligned} \tag{A.48}$$

where inequalities flow from top to down for submodularity, the definition of $\mathcal{S}_1^+ \cup \mathcal{S}_1^+$, and the definition of ξ.

4) *Proof of inequality A.46*: Let $b = \frac{1}{\alpha}$. We complete the proof first for the case where $(1-\xi) \geq \xi b$, and then for the case where $(1-\xi) < \xi b$: when $(1-\xi) \geq \xi b$, $\max\{(1-\xi), \xi b\} = 1-\xi$ and $\xi \leq \frac{1}{1+b}$; due to the latter, $1-\xi \geq \frac{b}{1+b} = \frac{1}{\alpha+1}$, which suggests inequality A.46 holds; Finally, when $1-\xi < \xi b$, $\max\{(1-\xi), \xi b\} = \xi b$ and $\xi > \frac{1}{1+b}$; due to the latter, $\xi b > \frac{b}{1+b} = \frac{1}{1+\alpha}$,

which also suggests inequality A.46 holds. Thus the inequality A.40 is proved.

A.5 Proof of Inequality in Equation (A.29)

We will prove the following,

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{1}{N-\alpha} f(\mathcal{S}_2) \tag{A.49}$$

where $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ is the solution returned by our algorithm; $\mathcal{A}^*(S)$ is the optimal removal of $\mathcal{S}$; $\alpha = \max |\mathcal{A}|$ is the maximum number of removal from set $\mathcal{S}$ and N is the number of robots.

The following two cases are enough to explain why (A.49) holds.

- when $\mathcal{S}_2^* = \emptyset$, i.e. all paths in $\mathcal{S}_1$ are removed and correspondingly no paths are removed in $\mathcal{S}_2$. Then $f(\mathcal{S} \setminus \mathcal{A}^*(S)) = f(\mathcal{S}_2)$, and (A.49) holds.

- when $\mathcal{S}_2^* \neq \emptyset$, that is there is at least one path left in $\mathcal{S}_1$ and choose one s from any of them, then

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq f(s) \tag{A.50}$$

since f is non-decreasing. Moreover,

$$f(\mathcal{S}_2) \leq \sum_{v \in \mathcal{S}_2} f(v) \leq (N-\alpha) f(s) \tag{A.51}$$

where the first inequality holds due to submodularity and the second holds because the proposed algorithm will construct $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $f(v') \geq f(v), \forall v' \in \mathcal{S}_1, v \in \mathcal{S}_2$.

Combine two cases, inequality (A.49) is proved.

A.6 Proof of Inequalities in Equation (A.31)

We will prove the following

$$f(\mathcal{Q}^*) \geq f^*, \tag{A.52}$$

where $\mathcal{Q}^*$ is the optimal MOP solution to robots $i \in \mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$, i.e. the optimal paths for robots corresponding to $\mathcal{S}_2$; f^* is the optimal solution to RMOP.

We first prove the inequality A.52. Let Π_i be the set of all feasible paths for robot i, which is hard to compute but is assumed here somehow known for analysis purposes. Then the ground set is defined as

$$\Pi = \bigcup_{i=1}^{N} \Pi_i$$

By definition, $(\Pi, \mathcal{I})$ is a partition matroid if

$$\mathcal{I} = \{I \subseteq \Pi \mid |I \cap \Pi_i| \leq 1, \forall i = 1, 2, \ldots, N\} \tag{A.53}$$

where independent set $\mathcal{I}$ represents all possible results of finding paths to N robots.

Similarly, we have another partition matroid $(\Pi, \mathcal{I}')$ with

$$\begin{aligned} \mathcal{I}' = \{I \subseteq \Pi \mid |I \cap \Pi_i| \leq a, a = 1, \forall i \in \mathcal{R}(\mathcal{S}_1); \\ a = 0, \forall i \in \mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)\} \end{aligned} \tag{A.54}$$

where independent set $\mathcal{I}'$ represents all possible results of finding paths to robots in $\mathcal{R}(\mathcal{S}_1)$ and $\mathcal{I}' \subseteq \mathcal{I}$.

In inequality (A.52), L.H.S, for any set $\mathcal{S}_1 \subseteq \Pi$ returned by algorithms with $|\mathcal{S}_1| =$

$|\mathcal{R}(\mathcal{S}_1)| = \alpha$ such that $\mathcal{S}_1 \in \mathcal{I}$ and $\mathcal{S}_1 \in \mathcal{I}'$. By definition of $\mathcal{Q}^*$

$$f(\mathcal{Q}^*) = \max_{\mathcal{S}_2 \subseteq \Pi, \mathcal{S}_2 \cup \mathcal{S}_1 \in \mathcal{I}} f(\mathcal{S}_2) \tag{A.55}$$

$$= \max_{\mathcal{S}_2 \subseteq \Pi \setminus \mathcal{S}_1, \mathcal{S}_2 \cup \mathcal{S}_1 \in \mathcal{I}} f(\mathcal{S}_2) \tag{A.56}$$

$$\geq \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} \max_{\mathcal{S}_2 \subseteq \Pi \setminus \tilde{\mathcal{S}}_1, \mathcal{S}_2 \cup \tilde{\mathcal{S}}_1 \in \mathcal{I}} f(\mathcal{S}_2) \tag{A.57}$$

$$= \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \subseteq \in \mathcal{I}'} \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}, \tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \tag{A.58}$$

$$= \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \subseteq \in \mathcal{I}'} \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \tag{A.59}$$

$$\triangleq h \tag{A.60}$$

(A.55) is the definition of $\mathcal{Q}^*$; (A.55) to (A.56) holds since we have a partition matroid with independent set defined as (A.53) and the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$ will be empty; In (A.56), $\mathcal{S}_1$ is a specific subset of Π and in the independent set $\mathcal{I}'$. If we think $\mathcal{S}_1$ as an instantiation of a certain 'set variable' $\tilde{\mathcal{S}}_1$, we can find a minimal value (R.H.S of (A.57)) through optimizing over $\tilde{\mathcal{S}}_1$ and (A.56) should be greater than minimal value, i.e. (A.56) to (A.57) holds; next we use the trick of changing of variables: let $\tilde{\mathcal{S}} = \tilde{\mathcal{S}}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_2 = \tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1$ due to the fact that $\tilde{\mathcal{S}}_1$ and $\mathcal{S}_2$ are disjoint. As a result, (A.57) to (A.58) holds; notice that in (A.58) the minimization operation over $\tilde{\mathcal{S}}_1$ can guarantee that the solution satisfies $\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}$ and we can remove the redundant constraint $\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}$ in maximization, i.e. (A.58) to (A.59) holds; and we define (A.59) as h.

In the following, we basically show that min-max function is no less than max-min function. Notice that for any $\mathcal{S} \subseteq \Pi$ such that $\mathcal{S} \in \mathcal{I}$, and any set $\tilde{\mathcal{S}}_1 \subseteq \Pi$ such that $\tilde{\mathcal{S}}_1 \in \mathcal{I}'$, it

holds

$$\max_{\tilde{\mathcal{S}}\subseteq\Pi,\tilde{\mathcal{S}}\in\mathcal{I}} f(\tilde{\mathcal{S}}\setminus\tilde{\mathcal{S}}_1) \geq f(\mathcal{S}\setminus\tilde{\mathcal{S}}_1) \tag{A.61}$$

which implies:

$$\begin{aligned} h &\geq \min_{\tilde{\mathcal{S}}_1\subseteq\Pi,\tilde{\mathcal{S}}_1\in\mathcal{I}'} f(\mathcal{S}\setminus\tilde{\mathcal{S}}_1) \\ &= \min_{\tilde{\mathcal{S}}_1\subseteq\mathcal{S},\tilde{\mathcal{S}}_1\in\mathcal{I}'} f(\mathcal{S}\setminus\tilde{\mathcal{S}}_1) \end{aligned} \tag{A.62}$$

Notice that (A.62) holds for all $\mathcal{S}\in\mathcal{I}$. As a result,

$$h \geq \max_{\tilde{\mathcal{S}}\subseteq\Pi,\tilde{\mathcal{S}}\in\mathcal{I}} \min_{\tilde{\mathcal{S}}_1\subseteq\tilde{\mathcal{S}},\tilde{\mathcal{S}}_1\in\mathcal{I}'} f(\tilde{\mathcal{S}}\setminus\tilde{\mathcal{S}}_1) \tag{A.63}$$

Next we consider the minimization operation of (A.63). For any $\tilde{\mathcal{S}}\in\mathcal{I}$,

$$\min_{\tilde{\mathcal{S}}_1\subseteq\tilde{\mathcal{S}},\tilde{\mathcal{S}}_1\in\mathcal{I}'} f(\tilde{\mathcal{S}}\setminus\tilde{\mathcal{S}}_1) \geq \min_{\tilde{\mathcal{S}}_1\subseteq\tilde{\mathcal{S}},|\tilde{\mathcal{S}}_1|\leq\alpha} f(\tilde{\mathcal{S}}\setminus\tilde{\mathcal{S}}_1) \tag{A.64}$$

Reasons for (A.64) to hold: the L.H.S of (A.64) only allows remove $\tilde{\mathcal{S}}_1\in\mathcal{I}'$ and $|\tilde{\mathcal{S}}_1|$ can go up to $|\mathcal{R}(\mathcal{S}_1)|=\alpha$ (refer to definition of $\mathcal{I}'$); by contrast, R.H.S of (A.64) is less constrained and can also remove up to α elements from $\tilde{\mathcal{S}}$. Thus, the R.H.S can get the result no greater than L.H.S.

As a result,

$$h \geq \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} \min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \tag{A.65}$$

$$\geq \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} \min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, |\tilde{\mathcal{S}}_1| \leq \alpha} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \tag{A.66}$$

$$= f(\mathcal{S}^* \setminus \mathcal{A}^*(\mathcal{S}^*)) \tag{A.67}$$

$$= f^* \tag{A.68}$$

In sum,

$$f(\mathcal{Q}^*) \geq h \geq f^*.$$

Bibliography

[1] Geoffrey A Hollinger, Sunav Choudhary, Parastoo Qarabaqi, Christopher Murphy, Urbashi Mitra, Gaurav S Sukhatme, Milica Stojanovic, Hanumant Singh, and Franz Hover. Underwater data collection using robotic sensor networks. *IEEE Journal on Selected Areas in Communications*, 30(5):899–911, 2012.

[2] Marija Popović, Teresa Vidal-Calleja, Gregory Hitz, Jen Jen Chung, Inkyu Sa, Roland Siegwart, and Juan Nieto. An informative path planning framework for uav-based terrain monitoring. *Autonomous Robots*, 44(6):889–911, 2020.

[3] Maaike Harbers, Joachim de Greeff, Ivana Kruijff-Korbayová, Mark A Neerincx, and Koen V Hindriks. Exploring the ethical landscape of robot-assisted search and rescue. In *A World with Robots*, pages 93–107. Springer, 2017.

[4] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[6] Mustafa M Aral, Jiabao Guan, Morris L Maslia, et al. Optimal design of sensor placement in water distribution networks. *Journal of Water Resources Planning and Management*, 136(1):5, 2010.

[7] William E Hart and Regan Murray. Review of sensor placement strategies for contamination warning systems in drinking water distribution systems. *Journal of Water Resources Planning and Management*, 136(6):611–619, 2010.

[8] Shoichi Koyama, Gilles Chardon, and Laurent Daudet. Optimizing source and sensor placement for sound field control: An overview. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:696–714, 2020.

[9] YL Guo, YQ Ni, and SK Chen. Optimal sensor placement for damage detection of bridges subject to ship collision. *Structural Control and Health Monitoring*, 24(9):e1963, 2017.

[10] Minwoo Chang and Shamim N Pakzad. Optimal sensor placement for modal identification of bridge systems considering number of sensing nodes. *Journal of Bridge Engineering*, 19(6):04014019, 2014.

[11] Boston Dynamics. `https://www.bostondynamics.com/`.

[12] Shenzhen DJI Sciences and Technologies Ltd. `https://www.dji.com/`.

[13] Clearpath Robotics, Inc. `https://clearpathrobotics.com/`.

[14] Unitree. `https://m.unitree.com/`.

[15] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[16] Guangyao Shi, Ishat E Rabban, Lifeng Zhou, and Pratap Tokekar. Communication-aware multi-robot coordination with submodular maximization. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8955–8961. IEEE, 2021.

[17] Guangyao Shi, Lifeng Zhou, and Pratap Tokekar. Robust multiple-path orienteering problem: Securing against adversarial attacks. In *Robotics: Science and Systems (RSS)*, 2020.

[18] Guangyao Shi, Lifeng Zhou, and Pratap Tokekar. Robust multiple-path orienteering problem: Securing against adversarial attacks. *IEEE Transactions on Robotics*, 2023.

[19] Guangyao Shi, Nare Karapetyan, Ahmad Bilal Asghar, Jean-Paul Reddinger, James Dotterweich, James Humann, and Pratap Tokekar. Risk-aware uav-ugv rendezvous with chance-constrained markov decision process. *2022 IEEE 61th Annual Conference on Decision and Control (CDC)*, 2022.

[20] Ahmad Bilal Asghar, Guangyao Shi, Nare Karapetyan, James Humann, Jean-Paul Reddinger, James Dotterweich, and Pratap Tokekar. Risk-aware resource allocation for multiple uavs-ugvs recharging rendezvous. *http://arxiv.org/abs/2209.06308*, 2022.

[21] Guangyao Shi and Pratap Tokekar. Decision-oriented learning with differentiable submodular maximization for vehicle routing problem. *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[22] Guangyao Shi, Chak Lam Shek, Nare Karapetyan, and Pratap Tokekar. Decision-oriented intervention cost prediction for multi-robot persistent monitoring. *arXiv preprint arXiv:2310.01519*, 2023.

[23] Lifeng Zhou, Vasileios Tzoumas, George J Pappas, and Pratap Tokekar. Resilient active target tracking with multiple robots. *IEEE Robotics and Automation Letters*, 4(1):129–136, 2018.

[24] Micah Corah and Nathan Michael. Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice. *Autonomous Robots*, 43(2):485–501, 2019.

[25] Ryan K Williams, Andrea Gasparri, and Giovanni Ulivi. Decentralized matroid optimization for topology constraints in multi-robot allocation problems. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 293–300. IEEE, 2017.

[26] Stefan Jorgensen, Robert H Chen, Mark B Milam, and Marco Pavone. The matroid team surviving orienteers problem: Constrained routing of heterogeneous teams with risky traversal. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5622–5629. IEEE, 2017.

[27] Philip Dames, Pratap Tokekar, and Vijay Kumar. Detecting, localizing, and tracking an unknown number of moving targets using a team of mobile robots. *The International Journal of Robotics Research*, 36(13-14):1540–1553, 2017.

[28] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):1–20, 2011.

[29] Javier Alonso-Mora, Eduardo Montijano, Tobias Nägeli, Otmar Hilliges, Mac Schwager, and Daniela Rus. Distributed multi-robot formation control in dynamic environments. *Autonomous Robots*, 43(5):1079–1100, 2019.

[30] Eric Cristofalo, Eduardo Montijano, and Mac Schwager. Consensus-based distributed 3d pose estimation with noisy relative measurements. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 2646–2653. IEEE, 2019.

[31] Xinmiao Sun, Christos G Cassandras, and Xiangyu Meng. Exploiting submodularity to quantify near-optimality in multi-agent coverage problems. *Automatica*, 100:349–359, 2019.

[32] Ragesh K Ramachandran, Lifeng Zhou, and Gaurav S Sukhatme. Resilient coverage: Exploring the local-to-global trade-off. *arXiv preprint arXiv:1910.01917*, 2019.

[33] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb):235–284, 2008.

[34] Lifeng Zhou and Pratap Tokekar. Sensor assignment algorithms to improve observability while tracking targets. *IEEE Transactions on Robotics*, 35(5):1206–1219, 2019.

[35] Lifeng Zhou, Vasileios Tzoumas, George J Pappas, and Pratap Tokekar. Distributed attack-robust submodular maximization for multi-robot planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2479–2485. IEEE, 2020.

[36] Pratap Tokekar, Volkan Isler, and Antonio Franchi. Multi-target visual tracking with aerial robots. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3067–3072. IEEE, 2014.

[37] Brent Schlotfeldt, Vasileios Tzoumas, Dinesh Thakur, and George J Pappas. Resilient active information gathering with mobile robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4309–4316. IEEE, 2018.

[38] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 2–10, 2006.

[39] Bahman Gharesifard and Stephen L Smith. Distributed submodular maximization with limited information. *IEEE transactions on control of network systems*, 5(4):1635–1645, 2017.

[40] David Grimsman, Mohd Shabbir Ali, Joao P Hespanha, and Jason R Marden. The impact of information in greedy submodular maximization. *IEEE Transactions on Control of Network Systems*, 2018.

[41] Lorenzo Sabattini, Nikhil Chopra, and Cristian Secchi. Decentralized connectivity maintenance for cooperative control of mobile robotic systems. *The International Journal of Robotics Research*, 32(12):1411–1423, 2013.

[42] Michael M Zavlanos, Ali Jadbabaie, and George J Pappas. Flocking while preserving network connectivity. In *2007 46th IEEE Conference on Decision and Control*, pages 2919–2924. IEEE, 2007.

[43] Michael M Zavlanos, Magnus B Egerstedt, and George J Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9):1525–1540, 2011.

[44] Meng Ji and Magnus Egerstedt. Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Transactions on Robotics*, 23(4):693–703, 2007.

[45] Francis Bach. Learning with submodular functions: A convex optimization perspective. *arXiv preprint arXiv:1111.6453*, 2011.

[46] Chien-Chung Huang, Mathieu Mari, Claire Mathieu, Joseph SB Mitchell, and Nabil H Mustafa. Maximizing covered area in the euclidean plane with connectivity constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[47] Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai. Maximizing submodular set function with connectivity constraint: Theory and application to networks. *IEEE/ACM Transactions on Networking*, 23(2):533–546, 2014.

[48] Rohan Ghuge and Viswanath Nagarajan. Quasi-polynomial algorithms for submodular tree orienteering and other directed network design problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1039–1048. SIAM, 2020.

[49] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic foundations of robotics XI*, pages 109–124. Springer, 2015.

[50] Michael S Branicky, Ross A Knepper, and James J Kuffner. Path and trajectory diversity: Theory and algorithms. In *2008 IEEE International Conference on Robotics and Automation*, pages 1359–1364. IEEE, 2008.

[51] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[52] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

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

[54] Dinesh Thakur, Maxim Likhachev, James Keller, Vijay Kumar, Vladimir Dobrokhodov, Kevin Jones, Jeff Wurz, and Isaac Kaminer. Planning for opportunistic surveillance with multiple robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5750–5757. IEEE, 2013.

[55] Armin Sadeghi, Ahmad Bilal Asghar, and Stephen L Smith. On minimum time multi-robot planning with guarantees on the total collected reward. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 16–22. IEEE, 2019.

[56] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 46–55, Oct 2003.

[57] Efrat Sless, Noa Agmon, and Sarit Kraus. Multi-robot adversarial patrolling: facing coordinated attacks. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1093–1100. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[58] Jennifer Carlson, Robin R Murphy, and Andrew Nelson. Follow-up analysis of mobile robot failures. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 5, pages 4987–4994. IEEE, 2004.

[59] Amanda Prorok, Brian M Sadler, Magnus Egerstedt, and Vijay Kumar. Guest editorial: Special section on "foundations of resilience for networked robotic systems". *Autonomous Robots*, 43(3):741–741, 2019.

[60] Luis Guerrero-Bonilla, Amanda Prorok, and Vijay Kumar. Formations for resilient robot teams. *IEEE Robotics and Automation Letters*, 2(2):841–848, 2017.

[61] Amanda Prorok. Redundant robot assignment on graphs with uncertain edge costs. In *Distributed Autonomous Robotic Systems*, pages 313–327. Springer, 2019.

[62] Kelsey Saulnier, David Saldana, Amanda Prorok, George J Pappas, and Vijay Kumar. Resilient flocking for mobile robot teams. *IEEE Robotics and Automation letters*, 2(2):1039–1046, 2017.

[63] Mark W Spong. On the robust control of robot manipulators. *IEEE Transactions on automatic control*, 37(11):1782–1786, 1992.

[64] Yung Ting, Sabri Tosunoglu, and Benito Fernandez. Control algorithms for fault-tolerant robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 910–915. IEEE, 1994.

[65] Didier Crestani and Karen Godary-Dejean. Fault tolerance in control architectures for mobile robots: Fantasy or reality? In *CAR: Control Architectures of Robots*, 2012.

[66] Andrew Clark, Basel Alomair, Linda Bushnell, and Radha Poovendran. *Submodularity in dynamics and control of networked systems*. Springer, 2016.

[67] Chandra Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 245–253, Oct 2005.

[68] Amarjeet Singh, Andreas Krause, Carlos Guestrin, and William J Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, 34:707–755, 2009.

[69] Nikolay Atanasov, Jerome Le Ny, Kostas Daniilidis, and George J Pappas. Decentralized active information acquisition: Theory and application to multi-robot slam. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4775–4782. IEEE, 2015.

[70] James B Orlin, Andreas S Schulz, and Rajan Udwani. Robust monotone submodular function maximization. In *Proceedings of the 18th International Conference on Integer Programming and Combinatorial Optimization-Volume 9682*, pages 312–324. Springer-Verlag, 2016.

[71] Vasileios Tzoumas, Konstantinos Gatsis, Ali Jadbabaie, and George J Pappas. Resilient monotone submodular function maximization. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1362–1367. IEEE, 2017. arXiv version.

[72] Vasileios Tzoumas, Ali Jadbabaie, and George J Pappas. Resilient monotone sequential maximization. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 7261–7268. IEEE, 2018.

[73] Lifeng Zhou, Vasileios Tzoumas, George J Pappas, and Pratap Tokekar. Distributed attack-robust submodular maximization for multi-robot planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, to appear.

[74] Sanem Sariel, Tucker Balch, and Nadia Erdogan. Naval mine countermeasure missions. *IEEE Robotics & Automation Magazine*, 15(1):45–52, 2008.

[75] Vasileios Tzoumas, Ali Jadbabaie, and George J Pappas. Resilient non-submodular maximization over matroid constraints. *arXiv preprint arXiv:1804.01013*, 2018.

[76] Zhongshun Zhang, Jonathon M Smereka, Joseph Lee, Lifeng Zhou, Yoonchang Sung, and Pratap Tokekar. Game tree search for minimizing detectability and maximizing visibility. *Autonomous Robots*, pages 1–15, 2021.

[77] Patrick Slade, Preston Culbertson, Zachary Sunberg, and Mykel Kochenderfer. Simultaneous active parameter estimation and control using sampling-based bayesian reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 804–810. IEEE, 2017.

[78] Roman Marchant, Fabio Ramos, Scott Sanner, et al. Sequential bayesian optimisation for spatial-temporal monitoring. In *UAI*, pages 553–562, 2014.

[79] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. Dec-mcts: Decentralized planning for multi-robot active perception. *The International Journal of Robotics Research*, 38(2-3):316–337, 2019.

[80] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[81] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[82] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[83] Qianguo Xing, Deyu An, Xiangyang Zheng, Zhenning Wei, Xinhua Wang, Lin Li, Liqiao Tian, and Jun Chen. Monitoring seaweed aquaculture in the yellow sea with multiple sensors for managing the disaster of macroalgal blooms. *Remote Sensing of Environment*, 231:111279, 2019.

[84] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[85] Bin Du, Dengfeng Sun, Satyanarayana Gupta Manyam, and David W Casbeer. Cooperative air-ground vehicle routing using chance-constrained optimization. In *2020 American Control Conference (ACC)*, pages 392–397. IEEE, 2020.

[86] Kevin Yu, Ashish Kumar Budhiraja, Spencer Buebel, and Pratap Tokekar. Algorithms and experiments on routing of unmanned aerial vehicles with mobile recharging stations. *Journal of Field Robotics*, 36(3):602–616, dec 2018.

[87] Alena Otto, Niels Agatz, James Campbell, Bruce Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018.

[88] Hongqi Li, Jun Chen, Feilong Wang, and Ming Bai. Ground-vehicle and unmanned-aerial-vehicle routing problems from two-echelon scheme perspective: A review. *European Journal of Operational Research*, 294(3):1078–1095, 2021.

[89] Shushman Choudhury, Kiril Solovey, Mykel J Kochenderfer, and Marco Pavone. Efficient large-scale multi-drone delivery using transit networks. *Journal of Artificial Intelligence Research*, 70:757–788, 2021.

[90] Parikshit Maini, Kevin Yu, PB Sujit, and Pratap Tokekar. Persistent monitoring with refueling on a terrain using a team of aerial and ground robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8493–8498. IEEE, 2018.

[91] Kevin Yu, Jason M. O'Kane, and Pratap Tokekar. Coverage of an environment using energy-constrained unmanned aerial vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[92] Neil Mathew, Stephen L Smith, and Steven L Waslander. Planning paths for package delivery in heterogeneous multirobot teams. *IEEE Transactions on Automation Science and Engineering*, 12(4):1298–1308, 2015.

[93] Satyanarayana G Manyam, Kaarthik Sundar, and David W Casbeer. Cooperative routing for an air–ground vehicle team—exact algorithm, transformation method, and heuristics. *IEEE Transactions on Automation Science and Engineering*, 17(1):537–547, 2019.

[94] Parikshit Maini, Kaarthik Sundar, Mandeep Singh, Sivakumar Rathinam, and PB Sujit. Cooperative aerial–ground vehicle route planning with fuel constraints for coverage applications. *IEEE Transactions on Aerospace and Electronic Systems*, 55(6):3016–3028, 2019.

[95] Thomas C Thayer and Stefano Carpin. An adaptive method for the stochastic orienteering problem. *IEEE Robotics and Automation Letters*, 6(2):4185–4192, 2021.

[96] Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54:86–109, 2015.

[97] Niels Agatz, Paul Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, 52(4):965–981, 2018.

[98] Quang Minh Ha, Yves Deville, Quang Dung Pham, and Minh Hoàng Hà. On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 86:597–621, 2018.

[99] Yao Liu, Zhihao Luo, Zhong Liu, Jianmai Shi, and Guangquan Cheng. Cooperative routing problem for ground vehicle and unmanned aerial vehicle: The application on intelligence, surveillance, and reconnaissance missions. *IEEE Access*, 7:63504–63518, 2019.

[100] Umut Ermağan, Barış Yıldız, and F Sibel Salman. A learning based algorithm for drone routing. *Computers & Operations Research*, 137:105524, 2022.

[101] Guohua Wu, Mingfeng Fan, Jianmai Shi, and Yanghe Feng. Reinforcement learning based truck-and-drone coordinated delivery. *IEEE Transactions on Artificial Intelligence*, 2021.

[102] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. Sensor planning for a symbiotic uav and ugv system for precision agriculture. *IEEE Transactions on Robotics*, 32(6):1498–1511, 2016.

[103] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

[104] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108, 2005.

[105] Eitan Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

[106] Sylvie Thiébaux, Brian Williams, et al. Rao*: An algorithm for chance-constrained pomdp's. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[107] Nikhil Nigam and Ilan Kroo. Persistent surveillance using multiple unmanned air vehicles. In *2008 IEEE Aerospace Conference*, pages 1–14. IEEE, 2008.

[108] Ahmad Bilal Asghar, Stephen L Smith, and Shreyas Sundaram. Multi-robot routing for persistent monitoring with latency constraints. In *2019 American Control Conference (ACC)*, pages 2620–2625. IEEE, 2019.

[109] SKK Hari, S Rathinam, S Darbha, K Kalyanam, SG Manyam, and D Casbeer. Efficient computation of optimal uav routes for persistent monitoring of targets. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 605–614. IEEE, 2019.

[110] Kaarthik Sundar and Sivakumar Rathinam. Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots. *IEEE Transactions on Automation Science and Engineering*, 11(1):287–294, 2013.

[111] Kuo-Shih Tseng and Bérénice Mettler. Near-optimal probabilistic search via submodularity and sparse regression. *Autonomous Robots*, 41(1):205–229, 2017.

[112] Derek Mitchell, Nilanjan Chakraborty, Katia Sycara, and Nathan Michael. Multi-robot persistent coverage with stochastic task costs. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3401–3406. IEEE, 2015.

[113] Jason Derenick, Nathan Michael, and Vijay Kumar. Energy-aware coverage control with docking for robot teams. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3667–3672. IEEE, 2011.

[114] Lantao Liu and Nathan Michael. Energy-aware aerial vehicle deployment via bipartite graph matching. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 189–194. IEEE, 2014.

[115] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[116] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

[117] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

[118] Jayanta Mandi, Peter J Stuckey, Tias Guns, et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1603–1610, 2020.

[119] Nils Wilde, Armin Sadeghi, and Stephen L Smith. Learning submodular objectives for team environmental monitoring. *IEEE Robotics and Automation Letters*, 7(2):960–967, 2021.

[120] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[121] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[122] Simon Muntwiler, Kim P Wabersich, and Melanie N Zeilinger. Learning-based moving horizon estimation through differentiable convex optimization layers. In *Learning for Dynamics and Control Conference*, pages 153–165. PMLR, 2022.

[123] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

[124] Steven Chen, Kelsey Saulnier, Nikolay Atanasov, Daniel D Lee, Vijay Kumar, George J Pappas, and Manfred Morari. Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American control conference (ACC)*, pages 1520–1527. IEEE, 2018.

[125] Mohak Bhardwaj, Byron Boots, and Mustafa Mukadam. Differentiable gaussian process motion planning. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10598–10604. IEEE, 2020.

[126] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

[127] Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

[128] Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. *Advances in Neural Information Processing Systems*, 30, 2017.

[129] Shinsaku Sakaue. Differentiable greedy algorithm for monotone submodular maximization: Guarantees, gradient estimators, and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 28–36. PMLR, 2021.

[130] Francesco Betti Sorbelli, Federico Corò, Sajal K Das, and Cristina M Pinotti. Energy-constrained delivery of goods with drones under varying wind conditions. *IEEE Transactions on Intelligent Transportation Systems*, 22(9):6048–6060, 2020.

[131] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. *Advances in neural information processing systems*, 27, 2014.

[132] Sofia Maria Nikolakaki, Alina Ene, and Evimaria Terzi. An efficient framework for balancing submodularity and cost. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1256–1266, 2021.

[133] Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In *International Conference on Machine Learning*, pages 2634–2643. PMLR, 2019.

[134] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net, 2017.

[135] JV Seguro and TW Lambert. Modern estimation of the parameters of the weibull wind speed distribution for wind energy analysis. *Journal of wind engineering and industrial aerodynamics*, 85(1):75–84, 2000.

[136] Thibaut Horel and Yaron Singer. Maximization of approximately submodular functions. *Advances in neural information processing systems*, 29, 2016.

[137] Ahmad Bilal Asghar, Guangyao Shi, Nare Karapetyan, James Humann, Jean-Paul Reddinger, James Dotterweich, and Pratap Tokekar. Risk-aware recharging rendezvous for a collaborative team of uavs and ugvs. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5544–5550. IEEE, 2023.

[138] Gözde Özcan, Armin Moharrer, and Stratis Ioannidis. Submodular maximization via taylor series approximation. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 423–431. SIAM, 2021.