Submodular Optimization in Multi-Robot Teams: Robustness, Resilience, and Decentralization

Jun Liu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Ryan K. Williams, Chair
Pratap Tokekar
Guoqiang Yu
Daniel Stilwell
Craig Woolsey

December 15, 2022
Blacksburg, Virginia

Keywords: Multi-robot systems, submodular optimization, decision-making, task allocation, robustness, resilience, decentralization.

Submodular Optimization in Multi-Robot Teams: Robustness, Resilience, and Decentralization

Jun Liu

ABSTRACT

Decision-making is an essential topic for multi-robot coordination and collaboration and is also the main topic of this thesis. Examples can be found in autonomous driving, environmental monitoring, intelligent transportation, etc. To study this problem, we first use multiple applications as motivating examples and then construct the general formulation and solution for those applications. Finally, we extend our investigation from the fundamental problem formulation to resilient and decentralized versions. All those problems are studied in the combinatorial optimization domain with the help of submodular and matroid optimization techniques.

As a motivating example, we use a multi-robot environmental monitoring problem to extract the general formulation of a multi-robot decision-making problem. Consider the problem of deploying multi-agent teams for environmental monitoring in a precision farming application. We want to answer the question of when and where to deploy our robots. This is a typical task allocation problem in multi-robot systems. Using the above problem as an example, we first focus on this decision-making problem, e.g., intermittent deployment problem, in a centralized scenario. Given a predictable agriculture environment, we want to make decisions for robots for this monitoring task. The problem is formulated as a combinatorial submodular optimization with matroid constraints. By utilizing the properties of submodularity, we aim to develop a solution with performance guarantees. This motivating example demonstrates how to use a submodular function and matroids to model and solve decision-making problems

in multi-robot systems. Based on this framework, we continue to explore the fundamental decision-making problem in several other directions in multi-robot systems, including the robust decision-making problem. All those problems and solutions are formulated and considered in a centralized scenario.

In the second part of this thesis, we switch our focus from centralized to decentralized scenarios. We first investigate a case where the robots in a distributed multi-robot system need to work together to guard the system against worst-case attacks while making decisions. By worst-case attacks, we refer to the case where the system may have up to K sensor failures. To increase resilience, we propose a fully distributed algorithm to guide each robot's action selection when the system is attacked. The proposed algorithm guarantees performance in a worst-case scenario where up to a portion of the robots malfunction due to attacks. Based on this specific task allocation problem in robotics, we then create a unified framework for a more general case in a decentralized scenario, e.g., asynchronous decentralized decision-making problems with matroid and knapsack constraints. Finally, several applications in decentralized scenarios are used to validate the theoretical guaranteed performance in robotics.[1]

[1]This dissertation was supported by the National Institute of Food and Agriculture under Grant 2018-67007-28380.

Submodular Optimization in Multi-Robot Teams: Robustness, Resilience, and Decentralization

Jun Liu

GENERAL AUDIENCE ABSTRACT

Robots have been widely used as mobile sensing agents nowadays in various applications. Especially with the help of multi-robot systems and artificial intelligence, our lives have changed dramatically in the last decades. One of the most fundamental questions is how to utilize multi-robot systems to finish tasks successfully. To answer this, we need first to formulate the problem from applications and then find theoretically guaranteed answers to those questions. Meanwhile, the robustness and resilience of the solution also need to be taken care of, as cyber-attacks or system failures can happen everywhere. Motivated by those two main goals, this thesis will first use multiple applications to introduce the thesis's topic. We then provide solutions to those problems in centralized and decentralized scenarios. Meanwhile, to increase the system's ability to handle failures, we need to answer how to improve the robustness and resilience of the proposed solutions. Therefore, the topic of this thesis spread from problem formulation to failure-proof solutions. The result of this thesis can be widely used in multi-robot decision-making applications, including autonomous driving, intelligent transportation, and other cyber-physical systems.

To my parents.

Acknowledgments

Thanks to my advisor, Ryan K. Williams, my family, and anyone who helped me on this long, challenging, yet rewarding journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of robotics research, one of the core problems is to determine efficient and effective strategies for deploying robots in an environment, such as motion planning [1], multi-scale coordination [2], etc. However, the large body of work in these areas is conditioned on the assumption that an environment should be sensed. Our suggestion is that there are settings in which we should first ask when it is appropriate to deploy our robots. Specifically, we aim to select an intermittent deployment strategy in order to minimize the cost of deployment while maintaining a high accuracy in our understanding of an environment. This is an action selection problem in robotics. Based on this basic setting, we will first investigate this intermittent deployment problem. Then, we will investigate distributed resilient action selection problem and general action selection problem in this thesis. The details are as follows.

Chapter 3: Intermittent Deployment Strategies for Multi-Robot Teams

In this chapter, we formulate and solve the intermittent deployment problem, which yields strategies that couple when heterogeneous robots should sense an environmental process, with where a deployed team should sense in the environment. As a motivation, suppose that a spatiotemporal process is slowly evolving and must be monitored by a multi-robot team, e.g., unmanned aerial vehicles monitoring pasturelands in a precision agriculture context. In such a case, an intermittent deployment strategy is necessary as persistent deployment or

monitoring is not cost-efficient for a slowly evolving process. At the same time, the problem of where to sense once deployed must be solved as process observations yield useful feedback for determining effective future deployment and monitoring decisions. In this context, we model the environmental process to be monitored as a spatiotemporal Gaussian process with mutual information as a criterion to measure our understanding of the environment. To make the sensing resource-efficient, we demonstrate how to use matroid constraints to impose a diverse set of homogeneous and heterogeneous constraints. In addition, to reflect the cost-sensitive nature of real-world applications, we apply budgets on the cost of deployed heterogeneous robot teams. To solve the resulting problem, we exploit the theories of submodular optimization and matroids and present a greedy algorithm with bounds on sub-optimality. Finally, Monte Carlo evaluations are used to demonstrate the correctness of the proposed method.

The result of the work where the constraint is an intersection of matroids appeared [1]. The result of the work where the constraint is a combination of matroids and knapsacks appeared in [2]. A result of a Markov decision process (MDP) formulation of this problem appeared in [3].

Chapter 4: Submodular Optimization for Coupled Task Allocation and Intermittent Deployment Problems

In this chapter, we demonstrate a formulation for optimizing coupled submodular maximization problems with provable sub-optimality bounds. In robotics applications, it is quite common that optimization problems are coupled with one another and therefore cannot be solved independently. Specifically, we consider two problems coupled if the outcome of the first problem affects the solution of a second problem that operates over a longer time scale. For example, in our motivating problem of environmental monitoring, we posit that multi-robot

task allocation will potentially impact environmental dynamics and thus influence the quality of future monitoring, here modeled as a multi-robot intermittent deployment problem. The general theoretical approach for solving this type of coupled problem is demonstrated through this motivating example. Specifically, we propose a method for solving coupled problems modeled by submodular set functions with matroid constraints. A greedy algorithm for solving this class of problem is presented, along with sub-optimality guarantees. Finally, practical optimality ratios are shown through Monte Carlo simulations to demonstrate that the proposed algorithm can generate near-optimal solutions with high efficiency.

The result of this work appeared in [4].

Chapter 5: A Fast Algorithm for Robust Action Selection in Multi-Agent Systems

In this chapter, we consider a robust action selection problem in multi-agent systems where performance must be guaranteed when the system suffers a worst-case attack on its agents. Specifically, agents are tasked with selecting actions from a common ground set according to individualized objective functions, and we aim to protect the system against attacks. In our problem formulation, attackers attempt to disrupt the system by removing an agent's contribution after knowing the system solution and thus can attack perfectly. To protect the multi-agent system against such attacks, we aim to maximize the minimum performance of all agents' individual objective functions under attacks. Thus, we propose a fast algorithm with tunable parameters for balancing complexity and performance, yielding substantially improved time complexity and performance compared to recent methods. Finally, we provide Monte Carlo simulations to demonstrate the performance of the proposed algorithm.

The result of this work appeared in arXiv [5] and is currently under review.

Chapter 6: Distributed Resilient Action Selection in Adversarial Environments

In this chapter, we consider a distributed submodular maximization problem for multi-robot systems when attacked by adversaries. One of the major challenges for multi-robot systems is to increase resilience against failures or attacks. This is particularly important for distributed systems under attack as there is no central point of command that can detect, mitigate, and recover from attacks. Instead, a distributed multi-robot system must coordinate effectively to overcome adversarial attacks. In this work, our distributed submodular action selection problem models a broad set of scenarios where each robot in a multi-robot system has multiple action selections that may fulfill a global objective, such as exploration or target tracking. To increase resilience in this context, we propose a fully distributed algorithm to guide each robot's action selection when the system is attacked. The proposed algorithm guarantees performance in a worst-case scenario where up to a portion of the robots malfunction due to attacks. Importantly, the proposed algorithm is also consistent, as it is shown to converge to the same solution as a centralized method. Finally, a distributed resilient multi-robot exploration problem is presented to confirm the performance of the proposed algorithm.

The result of this work appeared in [6].

Chapter 7: Asynchronous Distributed Decision-Making for Multi-robot Systems

In this chapter, we consider an asynchronous distributed submodular maximization algorithm with an application to the intermittent deployment problem. As a motivation, we consider a spatiotemporal environmental process that needs to be monitored. We then formulate an intermittent deployment problem, which couples when heterogeneous robots should sense an environmental process, with where a deployed team should sense in the environment. Meanwhile, the asynchronicity of the system is considered as different robots may have

diverse computations or transmission abilities. To address these issues, we formulate the problem as a submodular maximization with a matroid intersection (and knapsack) constraint problem. We propose an asynchronous distributed submodular maximization algorithm that finds a local solution for every robot greedily through asynchronous communications. We prove that every robot will eventually agree with each other with theoretical guarantees. Moreover, we demonstrate that the result of the proposed distributed algorithm converges to the centralized solution. Therefore, the deployment strategy is able to yield sensing times and sensing locations simultaneously. Since the proposed algorithm is a general framework that can be applied to other problems, we also demonstrate its practicality when applied to other problems. Finally, Monte Carlo simulations are presented to corroborate our theoretical results.

Chapter 8: Intermittent Deployment for Large-Scale Multi-Robot Forage Perception: Data Synthesis, Prediction, and Planning

Monitoring the health and vigor of grasslands is vital for informing management decisions to optimize rotational grazing in agriculture applications. To take advantage of forage resources and improve land productivity, we require knowledge of pastureland growth patterns that is simply unavailable at state of the art. In this chapter, we propose to deploy a team of robots to monitor the evolution of an unknown pastureland environment to fulfill the above goal. To monitor such an environment, which usually evolves slowly, we need to design a strategy for rapid assessment of the environment over large areas at a low cost. Thus, we propose an integrated pipeline comprising of data synthesis, deep neural network training and prediction along with a multi-robot deployment algorithm that monitors pasturelands intermittently. Specifically, using expert-informed agricultural data coupled with novel data synthesis in ROS Gazebo, we first propose a new neural network architecture to learn the spatiotemporal

dynamics of the environment. Such predictions help us to understand pastureland growth patterns on large scales and make appropriate monitoring decisions for the future. Based on our predictions, we then design an intermittent multi-robot deployment policy for low-cost monitoring. Finally, we compare the proposed pipeline with other methods, from data synthesis to prediction and planning, to corroborate our pipeline's performance.

This work is a collaboration work where the focus of this thesis is on planning, decision-making, and the pipeline integration. The full result appeared in [7]

Chapter 2

Elements from the Theories of Combinatorial Optimization and Networks

In this chapter, we introduce some concepts related to combinatorial optimization theories and network theories.

2.1 Combinatorial Optimization

Combinatorial optimization and its applications in robotics have a rich history. In this thesis, we will be mainly focusing on combinatorial optimization in robotics. Specifically, we utilize the properties of submodular functions in robotic applications since those properties capture the diminishing return property that applies to lots of applications. We direct the reader to [8] for a comprehensive overview of combinatorial optimization.

Definition 1 (Set function [8])**.** *A set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is a function that maps every subset $\mathcal{X} \subseteq \mathcal{V}$ into a real value $\mathbb{R}$. The set $\mathcal{V}$ is usually called the ground set.*

The ground set $\mathcal{V}$ contains all the elements that the system can choose from. The power set $2^{\mathcal{V}}$ is the set of all subsets of $\mathcal{V}$, including $\emptyset$ and $\mathcal{V}$ itself.

Definition 2 (Submodularity [8]). *A set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is*

- normalized, *if $f(\emptyset) = 0$;*
- non-decreasing, *if $f(\mathcal{X}) \leq f(\mathcal{Y})$ when $\mathcal{X} \subseteq \mathcal{Y} \subseteq \mathcal{V}$;*
- modular, *if $f(\mathcal{X}) = \sum_{x \in \mathcal{X}} f(x)$ for all $x \in \mathcal{X}$;*
- submodular, *if $f(\mathcal{X}) + f(\mathcal{Y}) \geq f(\mathcal{X} \cup \mathcal{Y}) - f(\mathcal{X} \cap \mathcal{Y})$ where $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{V}$.*

Another equivalent definition of submodular set function is that $f(\mathcal{X} \cup \{v\}) - f(\mathcal{X}) \geq f(\mathcal{Y} \cup \{v\}) - f(\mathcal{Y})$ holds for any $\mathcal{X} \subseteq \mathcal{Y} \subseteq \mathcal{V}$ and $v \in \mathcal{V} \setminus \mathcal{Y}$ with $\mathcal{V}$ as the ground set for the set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$. For this reason, submodular functions can be interpreted as exhibiting diminishing returns since the $f(\mathcal{X} \cup \{v\}) - f(\mathcal{X})$ becomes less when $\mathcal{X}$ is replaced by a larger set $\mathcal{Y}$.

Definition 3 (Marginal gain). *For a set function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$, let the marginal gain of adding element $v \in \mathcal{V}$ into set $\mathcal{X} \subseteq \mathcal{V}$ be $f_{\mathcal{X}}(v) \triangleq f(\mathcal{X} \cup \{v\}) - f(\mathcal{X})$*[1].

Using this definition, we can rewrite the above diminishing return property in another form $f_{\mathcal{X}}(v) \geq f_{\mathcal{Y}}(v)$. This diminishing returns property can be read as "the marginal gain becomes larger when the set becomes smaller". Examples of non-decreasing submodular functions include:

- $f(\mathcal{X}) = \max_{i \in \mathcal{X}} w_i$ with $\mathcal{X} \subseteq \mathcal{V}$ and $w_i \geq 0$;
- $f(\mathcal{X}) = |\bigcup_{i \in \mathcal{X}} \mathcal{S}_i|$ with $\mathcal{X} \subseteq \mathcal{V}$ and $\mathcal{S}_i \subset \mathcal{V}$;
- $f(\mathcal{X}) = \min\left\{\sum_{i \in \mathcal{X}} w_i, b\right\}$ with $\mathcal{X} \subseteq \mathcal{V}$, $w_i \geq 0$, $b \geq 0$.

[1]We will use v to represent $\{v\}$ if there is no confusion.

Notably, the submodularity of objective functions is a natural property and exists in many robotics applications, e.g., sensor placement [9], task allocation [10], environmental monitoring [1], etc. We refer the reader to [8, 11] for more details and applications of submodularity. With loss of generality, we restrict our discussion to normalized functions because any unnormalized function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ can be normalized to another function $f'(\mathcal{X})$ through $f'(\mathcal{X}) = f(\mathcal{X}) - f(\emptyset)$.

Definition 4 (Curvature [12]). *Let $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ be a monotone non-decreasing submodular function, the curvature of $f(\cdot)$ is defined as*

$$c_f \triangleq 1 - \min_{a \in \mathcal{A}} \frac{f(\mathcal{V}) - f\left(\mathcal{V} \setminus a\right)}{f(a)},$$

where $\mathcal{A} = \{a \in \mathcal{V} \mid f(a) > 0\}$ and $\mathcal{V}$ is the ground set.

This curvature represents the submodularity level of $f(\cdot)$. It holds that $0 \leq c_f \leq 1$. If $c_f = 0$, then $f(\cdot)$ is a modular function and $f(\mathcal{X} \cup \{v\}) - f(\mathcal{X}) = f(v)$. If $c_f = 1$, then $f(\mathcal{X} \cup \{v\}) - f(\mathcal{X}) = 0$, where $\mathcal{X} \subseteq \mathcal{V}$ and $v \in \mathcal{V} \setminus \mathcal{X}$. This means element v adds zero contribution to the function value $f(\cdot)$ if set $\mathcal{X}$ is already selected. More details and examples about curvature and submodularity can be found in [8, 11–13].

Definition 5 (Matroid [14]). *A matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is a set system, where $\mathcal{V}$ is a finite ground set and $\mathcal{I}$ is a collection of subsets of $\mathcal{V}$, with the following properties:*

1. *$\emptyset \in \mathcal{I}$;*
2. *If $\mathcal{X} \subseteq \mathcal{Y} \in \mathcal{I}$, then $\mathcal{X} \in \mathcal{I}$;*
3. *If $\mathcal{X}, \mathcal{Y} \in \mathcal{I}$ and $|\mathcal{Y}| < |\mathcal{X}|$, there exists a $e \in \mathcal{X} \setminus \mathcal{Y}$ such that $\mathcal{Y} \cup \{e\} \in \mathcal{I}$.*

A reason of interest for matroids is that it extends the concept of linear independence from linear algebra to set systems and easily represents independence constraints in combinatorial optimization. In linear algebra, a matrix's rank function is a measure of the independence of rows or columns. In set systems, matroid is a measure of the independence of sets. In many applications, we use this structure to model the independence requirements (constraints) on solutions chosen from the ground set. For example, in task allocation applications, if any task can only be taken by at most one robot, then the independence of matroid constraints will be violated if every task is taken by more than one robot.

A matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ defines the subsets of the ground set $\mathcal{V}$ that are admissible. In other words, a matroid constraint is a constraint that is represented by admissible subsets of the ground set that satisfy the above axioms. Thus, if a solution belongs to a matroid, it is one of the admissible sets. Also, the sets contained in $\mathcal{I}$ are called independent sets. Specifically, in this problem, given a ground set $\mathcal{V}$ and a matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$, we want to find a subset $\mathcal{S} \subseteq \mathcal{V}$ to maximize a submodular function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ such that S satisfies all three matroid axioms, i.e., $\mathcal{S} \in \mathcal{I}$. For example, given a ground set V, if $\mathcal{I}$ is a collection of subsets of V that are at most size ℓ, i.e., $\mathcal{I} = \{\mathcal{X} \subseteq V : |\mathcal{X}| \leq \ell\}$, then $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is a matroid constraint as Definition 5 is respected. In multi-robot task allocation, this simple matroid example could constrain each robot i to choose at most ℓ tasks from a ground set. Examples of matroid constraints $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ include:

- *Uniform matroid:* $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ where $\mathcal{I} = \{\mathcal{X} \subseteq \mathcal{V} : |\mathcal{X}| \leq \ell\}$. Examples can be found in resource limited applications in robotics, control, etc.. These resources can be batteries, communication bandwidths, computation resources, information about targets [15–17], etc..

- *Partition matroid:* $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ where $\mathcal{I} = \{\mathcal{X} \subseteq \mathcal{V} : |\mathcal{X} \cap V_i| \leq \ell_i, \forall\, i = 1, \ldots, N\}$,

$\mathcal{V} = \bigcup_{i=1}^{N} \mathcal{V}_i$, and $\bigcap_{i=1}^{N} \mathcal{V}_i = \emptyset$. Note that this is one matroid with several sub-ground set. Examples can be found in heterogeneous systems with limited resources in robotics, control, etc. For example, a robotic system with different types of robots, sensors, batteries, payloads [18–21], etc..

A generalization of a single matroid is a matroid intersection. That is, $\mathcal{I} = \{\mathcal{X} \subseteq \mathcal{V} : \mathcal{X} \in \bigcap_{i=1}^{N} \mathcal{I}_i\}$ is the intersection of N matroids, i.e., $\mathcal{M}_i = (\mathcal{V}, \mathcal{I}_i), \forall\, i = 1, \ldots, N$. The cardinality of this matroid intersection is $|\mathcal{I}| = N$. Note that matroid unions are matroidal, but matroid intersections are not necessarily matroidal [8]. Examples of matroids and their applications can be found in [4, 8, 14, 18, 22]. The reader is referred to [8] for a comprehensive overview of matroids.

The interesting aspect of a matroid is its ability to model constraints that allow for efficiently computable solutions, which is especially useful in robotic applications. And, an important class of problems that combine submodularity and matroid constraints is submodular maximization subject to a matroid constraint. In this setting, efficient greedy algorithms can be used to optimize submodular objective functions with bounded sub-optimality when the constraint is a matroid (or matroid intersection) [8, 23].

Definition 6 (Knapsack)**.** *A knapsack constraint is a linear constraint defined through a cost function $c : 2^{\mathcal{V}} \mapsto \mathbb{R}$ on the ground set $\mathcal{V}$. That is,*

$$\mathcal{Y} = \left\{ \mathcal{X} \subseteq \mathcal{V} \mid \sum_{e \in \mathcal{X}} c(e) \leq 1 \right\}.$$

This constraint is also called a budget constraint. In the definition, the budget is set to 1 without a loss of generality. The knapsack constraint get its name as it is the constraint of the classical knapsack problem: $\max \left\{ \sum_{e \in \mathcal{S}} c_e : \mathcal{S} \in \mathcal{X} \right\}$.

2.2 Robot and Network

In this section, we introduce some basic concepts of multi-robot network as they will be used in our multi-robot applications.

Consider a heterogeneous multi-robot system operating in $\mathbb{R}^2$. Each robot $i \in \mathcal{A}$ is associated with computation and communication capabilities, where $\mathcal{A}$ is a set of robots. To model the communication capabilities, we define an undirected graph $\mathcal{G} = (\mathcal{A}, \mathcal{E})$, having vertices v_i associated with robot $i \in \mathcal{A}$ and an edge set $\mathcal{E} = \{(i, j) \mid i, j \in \mathcal{A}\}$. The graph $\mathcal{G}$ is undirected if $(i, j) \in \mathcal{E} \Leftrightarrow (j, i) \in \mathcal{E}$. Let $\mathcal{N}_i = \{v_j \in \mathcal{A} \mid (i, j) \in \mathcal{E}\}$ be the neighbor set of $i \in \mathcal{A}$ when $v_i \in \mathcal{A}$ can communicate with $v_j \in \mathcal{A}$ through $(i, j) \in \mathcal{E}$. The diameter $d(\mathcal{G})$ of a graph $\mathcal{G}$ is defined as the largest distance between any pair of robots.

Chapter 3

Intermittent Deployment Strategies for Multi-Robot Teams

Multi-robot systems have a natural advantage over a single robot when tasks, especially monitoring, are distributed over space or in time. More robots means large environments can be monitored more effectively, and observations can be made with finer granularity. These advantages have been seen in various established methods for environmental sensing/modeling [19, 20, 24–26]. While these methods are certainly effective, we point out that they are all conditioned on the underlying assumption that a robotic team should be continuously deployed for a given task. That is, the majority of multi-robot problems belong to the post-deployment category, meaning we have already made our decision to deploy a multi-robot system for the desired task. Our suggestion is that in addition to considering the post-deployment problem, we must consider the deployment problem itself since little attention has been given to the concept of deploying multi-robot teams intermittently. Specifically, we want to answer the question "when is it the best time to deploy our robots and once deployed where should they sense?" In particular, we are motivated by problems from the precision agriculture context where a slowly evolving environmental process must be observed, and thus persistent monitoring is not cost-effective.

When our objective is to monitor a slowly evolving environmental process, such as forage in pastureland (Fig. 3.1), we cannot deploy robots frequently as the difference in environmental

Fig. 3.1. An example of a slowly evolving spatiotemporal process, pastureland forage, that needs to be monitored for effective land management.

information between deployments may be negligible. Instead, enough time must elapse so that sufficient variation in sensing information allows for an appropriate balance with the cost of deploying a robot team. Indeed, we require an intermittent deployment strategy that defines when it is appropriate to deploy a robotic team, or in other words, how often they should be deployed. Such a strategy should account for the cost of using differently composed robot teams and, if necessary, should not exceed a cost budget. At the same time, another problem that couples with the intermittent deployment problem is the sensing location selection problem [27, 28]. We not only need to know when to deploy our robots, but we also need to decide where these robots should sense once deployed. The reason for this coupling is that effective deployment timing yields a high sensing accuracy without excessive cost while selecting informative sensing locations aids in reducing the deployment needs while maintaining a high sensing accuracy. Since these two problems are naturally coupled, we propose to solve them simultaneously.

A motivating application for this work is precision pastureland management [29, 30]. Pasturelands are an integral part of agricultural production. To take full advantage of the forage resource, we should monitor pasturelands in a manner that is both cost-effective and sufficiently informative for practitioners. The scale of pasturelands is usually large, and the

evolution of forage processes is generally very slow. Hence, we argue our above-described deployment and sensing strategy is ideally suited for precision pastureland management, as well as problems with similar characteristics such as ocean monitoring [31] or adversarial surveillance [32].

Related Work

The idea of intermittence appears in certain robotic applications. In [33], the authors demonstrated the convergence of the covariance matrix for Kalman filtering when the observations arrive stochastically. In [34], the robots are required to communicate intermittently but with a pre-defined interval before the actual deployment. In contrast, we require the sensing interval to be determined intrinsically by the environment instead of artificially defined a priori. The work in [35–38] considered time window and precedence constraints in task allocation problems. However, our problem is more complex as an environmental process must be observed. Moreover, our motivation is also contrary to the idea of persistent monitoring [25, 26], which is usually not energy efficient (albeit necessary for fast-evolving processes).

Our proposed problem can be considered in the realm of (partially observable) Markov Decision Processes ((PO)MDPs) [39]. Under this framework, [40] proposed an optimal stopping method yielding plans for when to observe an MDP. [41, 42] proposed a method for solving the quickest detection/estimation problem with observation uncertainty. Similar work can also be found in [3]. However, as these frameworks require known state transition probabilities and measurement probabilities, their applicability is limited in real-world applications. This chapter aims to make the intermittent deployment and sensing problem more general by removing these constraints. Moreover, heterogeneous robot teams and more complex constraints in time are considered in the chapter.

Formally, we utilize Gaussian processes [43] to learn and predict the environment, and use a submodular objective function, mutual information [27], to measure the information content we anticipate to gather when deploying robot teams. To reflect the intermittent deployment requirements, we propose to use matroids [4, 14] to impose both homogeneous (all robots) and heterogeneous (robot-specific) constraints. Additionally, both homogeneous and heterogeneous budget constraints are considered to limit the cost of deployed robot teams. In general, our problem becomes a submodular optimization problem under matroid intersection and knapsack constraints. Generally, submodular maximization is hard even without any constraints [8]. Multi-linear extension methods [44] use a continuous greedy method [45] with an approximation bound. However, the running time of these methods is not practical. [46] proposes a modified greedy method which is an interpolation between the traditional greedy method [23] and the continuous greedy method [45]. It requires less running time to solve the submodular maximization problem with matroid intersection and knapsack constraints. In this chapter, we will demonstrate how to utilize this method to solve the intermittent deployment problem.

Contributions

1. We formalize the intermittent deployment problem for multi-robot systems while considering heterogeneity.

2. We demonstrate how to utilize matroids to model novel homogeneous and heterogeneous constraints for deployment over time.

3. We demonstrate how to solve the problem efficiently and provide optimality and complexity analyses.

3.1 Problem Formulation

3.1.1 Environment Modeling and Sensing

Generally, a Gaussian process $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), \kappa(\mathbf{x}^i, \mathbf{x}^j))$ is specified by a mean function $m(\mathbf{x}) = \mathbb{E}\left[f(\mathbf{x})\right]$ and a covariance kernel $\kappa\left(\mathbf{x}^i, \mathbf{x}^j\right) = \mathbb{E}\left[(f(\mathbf{x}^i) - m(\mathbf{x}^i))(f(\mathbf{x}^j) - m(\mathbf{x}^j))\right]$, where $\mathbf{x}^i, \mathbf{x}^j \in \mathbb{R}^q$. Given a training set $\mathcal{E} = \{(\mathbf{x}^i, z^i) \mid i = 1, \ldots, n\}$ with n observations with $\mathbf{x}^i \in \mathbb{R}^q$ as an input and $z^i \in \mathbb{R}$ as an output, the properties of $f(\mathbf{x})$ can be inferred through $\mathcal{E}$. Specifically[1], the goal is to predict $f' \in \mathbb{R}$ for the testing input $\mathbf{x}' \in \mathbb{R}^q$ using $\mathcal{E}$. This process is equivalent to calculating $p(f' \mid f, X, \mathbf{z}, \mathbf{x}')$, which follows a Gaussian distribution $\mathcal{N}(\mu', \Sigma')$ [43] where $X = \left[(\mathbf{x}^1)^\top, \ldots, (\mathbf{x}^n)^\top\right]^\top \in \mathbb{R}^{n \times q}$ and $\mathbf{z} = [z^1, \ldots, z^n]^\top \in \mathbb{R}^n$. Also, $\mu' = m' + K_{\mathbf{x}'X}(K_{XX} + \sigma_\epsilon^2 I)^{-1}(\mathbf{z} - m(X))$ and $\Sigma' = K_{\mathbf{x}'\mathbf{x}'} - K_{\mathbf{x}'X}(K_{XX} + \sigma_\epsilon^2 I)^{-1}K_{X\mathbf{x}'}$, where $\sigma_\epsilon^2 \in \mathbb{R}$ is the variance of an additive Gaussian noise with zero mean. $K_{\mathbf{x}'X}$ is the covariance between $\mathbf{x}'$ and X defined by $\kappa(\mathbf{x}', X)$ and similarly for $K_{X\mathbf{x}'}$, K_{XX}. A commonly used kernel function is the squared exponential (SE) kernel $\kappa\left(\mathbf{x}^i, \mathbf{x}^j\right) = \sigma^2 \exp\left(-(2\ell^2)^{-1}\left(\mathbf{x}^i - \mathbf{x}^j\right)^2\right)$, where σ^2 and ℓ are the variance and length-scale respectively.

If $\mathcal{V}$ is the ground set containing all available choices for collecting data for a GP, one criterion for measuring the quality of the collected information contained in dataset $\mathcal{D}$ is the mutual information [47] between $\mathcal{D}$ and $\mathcal{V} \setminus \mathcal{D}$. That is, $M(\mathcal{D}) = H(\mathcal{D}) - H(\mathcal{D} \mid \mathcal{V} \setminus \mathcal{D})$, where $H(\mathcal{D}) = \frac{1}{2}\log\det\left(2\pi e\Sigma(\mathcal{D})\right)$ is the entropy and $\Sigma(\mathcal{D})$ is the covariance of $\mathcal{D}$. Mutual information is symmetric, i.e., $M(\mathcal{D}) = M(\mathcal{V} \setminus \mathcal{D})$. Using the relationship between mutual information and entropy, we get the marginal gain [27] of element e under current set $\mathcal{D}$ as $M(\{e\} \mid \mathcal{D}) = M(\{e\} \cup \mathcal{D}) - M(\mathcal{D}) = H(\{e\} \mid \mathcal{D}) - H(\{e\} \mid \mathcal{V} \setminus (\mathcal{D} \cup \{e\}))$.

In this work, we will exploit the metric of mutual information on information collected over

[1]For notation clarity, we drop the parameters of the functions in the remainder of this section, i.e., we use f to represent $f(X)$ and use f', μ', Σ', m' to represent $f(\mathbf{x}'), \mu(\mathbf{x}'), \Sigma(\mathbf{x}'), m(\mathbf{x}')$.

space and in time during intermittent deployments. The addition of the time dimension leaves all of the typical properties and evaluation methodologies of mutual information intact, such as in [27]. Thus we omit those details here as it is not our novelty.

3.1.2 General Modeling

Consider a spatiotemporal Gaussian process $f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), \kappa(\mathbf{x}^i, \mathbf{x}^j)\right)$ evolving over a $P \times Q$ 2D field $\mathcal{P}$ with time indexed by t where $\mathbf{x}^i = [x^i, y^i, t]^\top \in \mathbb{R}^3$. $\mathbf{x}^i$ is composed of a 2D location $\mathbf{p}^i = (x^i, y^i)$ and the corresponding time t. Also, $z^i(\mathbf{x}^i) \in \mathbb{R}$ is the corresponding measurement. $\mathcal{P} = \{1, \ldots, P \times Q\}$ contains the indexes of the available locations for sensing the process. The initial training data set is $\mathcal{E} = \left\{ \left(\mathbf{x}^i, z^i\right) \mid i = 1, \ldots, n \right\}$ with n observations, where $X = \left[(\mathbf{x}^1)^\top, \ldots, (\mathbf{x}^n)^\top\right]^\top \in \mathbb{R}^{n \times 3}$ is the input and $\mathbf{z} = [z^1, \ldots, z^n]^\top \in \mathbb{R}^n$ is the output. Since $\mathcal{E}$ is only used for training an initial environment model, it can derive either from existing process data (e.g., from similar pastureland deployments) or from manual short-term deployments in the target environment before intermittent deployment begins. For the kernel function, we use a separable spatiotemporal kernel to build a composite covariance function as $\kappa\left(\mathbf{x}^i, \mathbf{x}^j\right) = \kappa_p\left(\mathbf{p}^i, \mathbf{p}^j\right) \cdot \kappa_t\left(t^i, t^j\right) + \sigma_\epsilon^2$, where κ_p and κ_t are the kernels for the $\mathbf{p}$ and t dimensions, respectively.

Remark 1. *It is worth noting that our formulation also applies if the initial process model is poor (high uncertainty). In this case, initial deployment decisions may be poor. However as data is collected and the model improves, deployment decisions will correspondingly improve.*

We first build the ground set $\mathcal{V}$, which contains all possible deployment decisions for a heterogeneous multi-robot team over space and in time. We denote by $\mathcal{T} = \{1, \ldots, T\}$ the set of time indices when robots can be deployed. Consider a set of heterogeneous robots $\mathcal{R}$ which are available to deploy over the time horizon $\mathcal{T}$. We denote by $\mathcal{R} = \{1, \ldots, R\}$ the set of

TABLE 3.1
THE GROUND SET OF THE INTERMITTENT DEPLOYMENT PROBLEM

	$t=1$	$\dots$	$t=T$
$r=\{1\}$	$(x_1^1, y_1^1, 1)$	$\dots$	(x_1^1, y_1^1, T)
	$\dots$	$\dots$	$\dots$
	$(x_1^N, y_1^N, 1)$	$\dots$	(x_1^N, y_1^N, T)
$r=\{r\}$	$(x_r^1, y_r^1, 1)$	$\dots$	(x_r^1, y_r^1, T)
	$\dots$	$\dots$	$\dots$
	$(x_r^N, y_r^N, 1)$	$\dots$	(x_r^N, y_r^N, T)
$r=\{R\}$	$(x_R^1, y_R^1, 1)$	$\dots$	(x_R^1, y_R^1, T)
	$\dots$	$\dots$	$\dots$
	$(x_R^N, y_R^N, 1)$	$\dots$	(x_R^N, y_R^N, T)

indices of all robots. Each robot $r \in \mathcal{R}$ has a different sensing capability, i.e., different sensing noise variance σ_r^2, and different deployment cost c_r^i with respect to different location index $i \in \mathcal{P}$. Every location index i is associated with a location (x^i, y^i). Also, when considering the heterogeneity of robot r at different time t, we can write the above sensing noise variance and cost as $\sigma_r^2(t)$ and $c_r^i(t)$. Now, our ground set per time contains all available choices at time t in the Cartesian product of $\mathcal{P}$ and $\mathcal{R}$ denoted by $\mathcal{V}_t = \big\{(x_r^i, y_r^i, t) : \forall\, i \in \mathcal{P}, r \in \mathcal{R}\big\}$. We interpret (x_r^i, y_r^i, t) as "location (x^i, y^i) is sensed by robot r at time t". Finally, the ground set of the intermittent deployment problem is $\mathcal{V} = \bigcup_{t\in\mathcal{T}} \mathcal{V}_t$. Importantly, the $\mathcal{V}_i$'s are disjoint, i.e., $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \forall\, i, j \in \mathcal{T}$. The cardinality of the ground set is $|\mathcal{V}| = P \cdot Q \cdot R \cdot T$. In Table 3.1, we show all elements in the ground set $\mathcal{V}$, where we use $N = P \cdot Q$ for brevity.

3.1.3 Constraint Modeling

1) Knapsack constraints.

The general form is $\mathcal{X} = \big\{\mathcal{S} \subseteq \mathcal{V} : c(S) \leq B\big\}$, where $c : 2^{\mathcal{V}} \mapsto \mathbb{R}$ is a cost function. This

form can be used in the following ways:

- $\mathcal{X}_1 = \{\mathcal{S} \subseteq \mathcal{V} : c(\mathcal{S}_r) \leq B, \forall\, r \in \mathcal{R}\}$, where $S_r = \{(x_k^i, y_k^i, t) \mid k = r\}$. In this case, we set a constraint B on the cost used by each robot.

- $\mathcal{X}_2 = \{\mathcal{S} \subseteq \mathcal{V} : c(\mathcal{S}(t)) \leq B, \forall\, t \in \mathcal{T}\}$, where $S(t) = \{(x_r^i, y_r^i, k) \mid k = t\}$. In this case, we set a constraint B on the cost in each time.

- $\mathcal{X}_3 = \{\mathcal{S} \subseteq \mathcal{V} : c(\mathcal{S}^i) \leq B, \forall\, i \in \mathcal{P}\}$, where $S^i = \{(x_r^k, y_r^k, t) \mid k = i\}$. Here, we set a constraint B on the cost used by all robots for sensing the location (x^i, y^i).

2) Homogeneous constraints.

The general form is a matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ with

$$\mathcal{I} = \left\{ \mathcal{S} \subseteq \mathcal{V} : \sum_{t \in \mathcal{T}} \mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) \leq L \right\},$$

where $\mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) = 1$ if $|\mathcal{S} \cap \mathcal{V}_t| \geq 1$ and $\mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) = 0$ if $|\mathcal{S} \cap \mathcal{V}_t| = 0$. We refer to these constraints as homogeneous as we do not differentiate the behavior of different robots under these constraints. In this case, the general form above can be used in the following ways:

- $\mathcal{I}_{21} = \{\mathcal{S} \subseteq \mathcal{V} : |\mathcal{S} \cap \mathcal{V}_t| \leq L(t)\}$.
  In this case, we can model the constraint on the number of deployed robots in each time $t \in \mathcal{T}$.

- $\mathcal{I}_{22} = \left\{\mathcal{S} \subseteq \mathcal{V} : \mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) \leq L(t)\right\}$.
  In this case, we can model the constraint on the deployment status in each time $t \in \mathcal{T}$. We use the deployment status to indicate if there is at least one deployment at time t. $L(t) \in \{0, 1\}$, where $L(t) = 0$ means there is no deployment at t and $L(t) = 1$ means

there is at least one deployment at t. In this way, the constraint can be used to control intermittence by restricting times when there can be no deployment.

- $\mathcal{I}_{23} = \left\{\mathcal{S} \subseteq \mathcal{V} : \sum_{t\in\mathcal{T}} \mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) \leq L\right\}$.

  This is the most general form in this case, as shown in the beginning. We can use this to model the constraint on the number of non-zero deployments from $t = 1$ to $t = T$. If there is at least one deployment at some time, we call it a non-zero deployment. This constraint is also an intermittence constraint.

3) Heterogeneous constraints.

The general form is a matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ with

$$\mathcal{I} = \left\{\mathcal{S} \subseteq \mathcal{V} : \sum_{t\in\mathcal{T}} \mathbb{1}\left(|\mathcal{S}_r \cap \mathcal{V}_t|\right) \leq L_r, \forall\, r \in \mathcal{R}\right\},$$

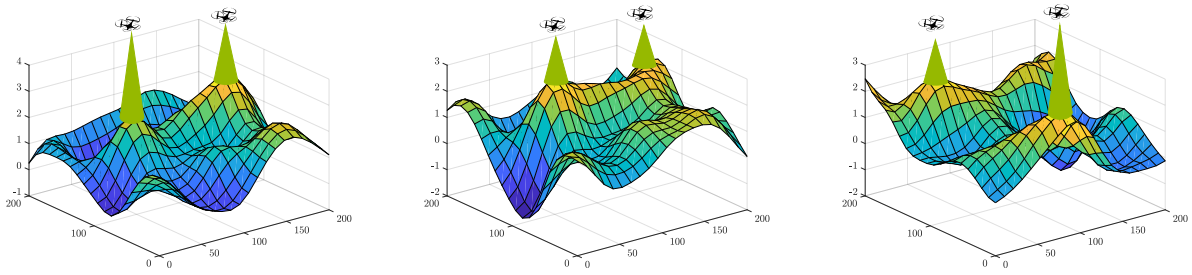
where $S_r = \{(x_k^i, y_k^i, t) \mid k = r\}$. Also, $\mathbb{1}\left(|\mathcal{S}_r \cap \mathcal{V}_t|\right) = 1$ if $|\mathcal{S}_r \cap \mathcal{V}_t| \geq 1$, and $\mathbb{1}\left(|\mathcal{S}_r \cap \mathcal{V}_t|\right) = 0$ if $|\mathcal{S}_r \cap \mathcal{V}_t| = 0$. We refer to these constraints as heterogeneous as different robots will have different constraints that are unique. In this case, this general form can be used in the following ways:

- $\mathcal{I}_{31} = \left\{\mathcal{S} \subseteq \mathcal{V} : |\mathcal{S}_r \cap \mathcal{V}_t| \leq L_r(t), \forall\, r \in \mathcal{R}\right\}$.

  In this case, we can model the constraint on the number of deployments for robot r in each time t.

- $\mathcal{I}_{32} = \left\{\mathcal{S} \subseteq \mathcal{V} : \mathbb{1}\left(|\mathcal{S}_r \cap \mathcal{V}_t|\right) \leq L_r(t), \forall\, r \in \mathcal{R}\right\}$.

  In this case, we can set a deployment status for robot r in each time t. As before, $L_r(t) = 0$ means there is no deployment for robot r at t and $L_r(t) = 1$ means there is at least one deployment for robot r at time t. Therefore, this constraint is especially



(a) Time $t = 1$. (b) Time $t = 2$. (c) Time $t = 5$.

Fig. 3.2. An example of intermittent deployment with 2 robots and time horizon 5.

useful for modeling any available/unavailable time window [35] for each robot $r \in \mathcal{R}$. Specifically, if the available deployment time for robot r is time $t = 1$ and $t = 2$, we can set $L_r(1) = 1$, $L_r(2) = 1$, and $L_r(t) = 0, \forall\, t \in \mathcal{T} \setminus \{1, 2\}$. Clearly, this constraint can be classified as an intermittence constraint.

- $\mathcal{I}_{33} = \left\{\mathcal{S} \subseteq \mathcal{V} : \sum_{t\in\mathcal{T}} \mathbb{1}\left(|\mathcal{S}_r \cap \mathcal{V}_t|\right) \leq L_r, \forall\, r \in \mathcal{R}\right\}$.
  This is the most general form in this case as shown in the beginning. We can use this to model the constraint on the number of non-zero deployments for robot r from $t = 1$ to $t = T$. This is also an intermittence constraint.

An illustrative example of the intermittent deployment problem is given in Fig. 3.2.

In the intermittent deployment problem, we desire high-quality predictions of our process, and thus we choose to maximize the mutual information we collect over deployments. We write the matroid intersection constraint as $\mathcal{I} = \left\{\mathcal{S} \subseteq \mathcal{V} : \mathcal{S} \in \bigcap_{k\in\mathcal{K}} \mathcal{I}_k\right\}$ where $\mathcal{K}$ represents the set indexing the matroids constraints. Also, we denote by $\mathcal{X} = \left\{\mathcal{S} \subseteq \mathcal{V} : S \in \bigcap_{\ell\in\mathcal{L}} \mathcal{X}_\ell\right\}$ and $\mathcal{L}$ represents the set indexing the knapsack constraints. Then, the general form of the

intermittent deployment problem is now given as:

$$\begin{aligned} \underset{\mathcal{S}\subseteq\mathcal{V}}{\text{maximize}} \quad & M(\mathcal{S}) \\ \text{subject to} \quad & \mathcal{S}\in\mathcal{X}, \mathcal{S}\in\mathcal{I}, \end{aligned} \tag{3.1}$$

Remark 2. *We have given general forms of various constraints in order to begin a library of models for the intermittent deployment problem that can meet various application requirements. In the simulation section, we will give an example to demonstrate this idea.*

3.2 Solution Method and Analyses

Theorem 1. *Both the homogeneous constraints and heterogeneous constraints are matroidal.*

The proof is omitted here due to space limitations. The reader is referred to our previous work [4, 22] for applicable proof methodologies.

Theorem 2. *Consider the intermittent deployment problem given by* (3.1)*. If the 2D space is discretized into $P\times Q$ cells, then the cardinality of the solution $|\mathcal{D}|$ should satisfy $|\mathcal{D}|\leq\frac{P\cdot Q\cdot R\cdot T}{2}$ to ensure the non-decreasing property of $M(\mathcal{D})$. If the maximum cardinality of the solution $\mathcal{D}$ is $|\mathcal{D}|$, then the discretization should satisfy $P\cdot Q\geq\frac{2|\mathcal{D}|}{R\cdot T}$ to ensure the non-decreasing property of $M(\mathcal{D})$.*

Proof. When the ground set is $\mathcal{V}$, we have from [27] that $M(\mathcal{D})$ is monotonic non-decreasing when $|\mathcal{D}|\in(0,\frac{|\mathcal{V}|}{2}]$ and monotonic non-increasing when $|\mathcal{D}|\in\left[\frac{|\mathcal{V}|}{2},|\mathcal{V}|\right]$. As the 2D space is discretized into $P\times Q$, the cardinality of the ground set $\mathcal{V}$ is $P\cdot Q\cdot R\cdot T$. Therefore, if $P\times Q$ is fixed, then $|\mathcal{D}|\leq\frac{P\cdot Q\cdot R\cdot T}{2}$ can ensure the non-decreasing property of $M(\mathcal{D})$. On the contrary, if $|\mathcal{D}|$ is fixed, i.e., the total number of deployments is fixed, a proper discretization of the 2D space, i.e., $P\cdot Q\geq\frac{2|\mathcal{D}|}{R\cdot T}$, ensures the non-decreasing property. □

Algorithm 1 The Intermittent Deployment Problem

Input: The inputs are as follows:

- matroid intersection constraint $\bigcap_{i=1}^{p} \mathcal{I}_i$, knapsack constraint $\bigcap_{j=1}^{\ell} \mathcal{X}_j$; mutual information function $M(\cdot)$

Output: The deployment set $\mathcal{D}$.

$\mathcal{T} \leftarrow \emptyset$;
$d \leftarrow \max_{e \in \mathcal{V}} M(\{e\})$;
for $(\rho = \frac{d}{p+\ell}; \rho \leq \frac{2d}{p+\ell}|\mathcal{V}|; \rho \leftarrow (1+\eta)\rho)$ **do**
    $\mathcal{S} \leftarrow \emptyset$;
    $M_\rho \leftarrow \max_{e \in \mathcal{V}} \{M(\{e\}) : \frac{M(\{e\})}{\sum_{j=1}^{\ell} c_j(e)} \geq \rho\}$;

    for $(\tau = M_\rho; \tau \geq \frac{\eta}{|\mathcal{V}|} M_\rho; \tau \leftarrow \frac{\tau}{1+\eta})$ **do**
        for $\forall\, e \in \mathcal{V}$ **do**
            if $\mathcal{S} \cup \{e\} \in \mathcal{I}$ **then**
                $m_e \leftarrow M(\{e\} \mid \mathcal{S})$;
                if $m_e \geq \tau$ and $\frac{m_e}{\sum_{j=1}^{\ell} c_j(e)} \geq \rho$ **then**

                    if $c_j(\mathcal{S} \cup \{e\}) \leq B, j = 1, \ldots, \ell$ **then**
                        $\mathcal{S} \leftarrow \mathcal{S} \cup \{e\}$;
                    else
                        $\mathcal{T} \leftarrow \mathcal{T} \cup \{e\}$;
                        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathcal{S}\}$;
                        restart with the next ρ;
                    end if

                end if
            end if
        end for
    end for
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathcal{S}\}$;
end for
$\mathcal{D} \leftarrow \arg\max_{\mathcal{S} \in \mathcal{T}} M(\mathcal{S})$.

We here provide a general greedy method from [46] to demonstrate how to solve the problem when considering p matroid constraints and ℓ knapsack constraints. We contextualize the algorithm to our problem for completeness and also tutorial value.

The Algorithm 1 has two loops: the outer loop and the inner loop to deal with knapsack constraints and matroid constraints, respectively. In each iteration of the outer loop (Line 3-26), the algorithm picks a threshold ρ as a marginal-to-cost ratio, i.e., $\frac{m_e}{\sum_{j=1}^{\ell} c_j(e)}, \forall\, e \in \mathcal{V}$. Again, $m_e = M\left(\{e\} \mid \mathcal{S}\right)$ is the marginal value of e under the current solution $\mathcal{S}$, and $e = \left(x_r^i, y_r^i, t\right)$ is one element of $\mathcal{V}$. Also, $c_j(e)$ is the cost of $e \in \mathcal{V}$ in jth knapsack constraint. The $\mathcal{S}$ chosen by the outer loop will serve as one candidate for the final solution. The threshold ρ ensures we do not exceed the budget constraint by discretizing the space of the marginal-to-cost ratio. The inner loop (Line 7-24) also has a threshold τ for the marginal value m_e and decreases in every iteration. This threshold τ acts as a lower bound for the marginal value of m_e in the current settings. Under different τ's, we can add different e's to the current solution $\mathcal{S}$ when constraints are satisfied (Line 14). The final $\mathcal{S}$ forms one candidate for the problem solution. There might be a case that only the knapsack constraints are not satisfied. We then need to keep track of this case (Line 16-17) when deciding the final problem solution $\mathcal{D}$.

Theorem 3 ([46])**.** *Let $\mathcal{D}$ be the greedy solution of the intermittent deployment. Also, denote by $\mathcal{D}^\star$ an optimal solution of Algorithm 1, then we have an optimality bound $M(\mathcal{D}) \geq \frac{1}{(1+\eta)(p+2\ell+1)} M(\mathcal{D}^\star)$ with $\eta \in (0, 1]$.*

Theorem 4. *The computational complexity of Algorithm 1 for solving the intermittent deployment problem is $\mathcal{O}\left(\frac{|\mathcal{V}|^4}{\eta^2} \log^2 \frac{|\mathcal{V}|}{\eta}\right)$.*

Proof. For computing marginal gains of mutual information, it takes $\mathcal{O}\left(|\mathcal{V}|^3\right)$ runs. If we denote each call of calculating the mutual information gain as an oracle call, it takes $\mathcal{O}\left(\frac{|\mathcal{V}|}{\eta^2} \log^2 \frac{|\mathcal{V}|}{\eta}\right)$ calls to get the final solution [46]. Therefore, the computational complexity is $\mathcal{O}\left(\frac{|\mathcal{V}|^4}{\eta^2} \log^2 \frac{|\mathcal{V}|}{\eta}\right)$. □

3.3 Numerical Evaluation

In this section, we study a problem to compare the greedy solution with an optimal solution. The optimal solution is computed by enumerating all combinations of deployments. Consider a 2D space with the size of $P \times Q$. The intermittent deployment we build here is one instance of our general form. Specifically, we use $\mathcal{I}_{21}$, $\mathcal{I}_{23}$, and the general knapsack constraints. Specifically, we put a constraint $L(t)$ on the number of deployed robots for $t \in \mathcal{T}$, a constraint L on the total number of non-zero deployments, and a constraint on the cost of robots used in each time. We denote by R the number of robots and set $L(t) = R, \forall\, t \in \mathcal{T}$. Finally, we get the number of combinations/cases that we need to consider when evaluating an optimal solution as $\binom{T}{1}\binom{R(PQ+1)}{R} + \ldots + \binom{T}{L}\binom{R(PQ+1)}{R}^L$. First, since there are T times available and we can collect samples at a maximum of L different times, there are $\binom{T}{\ell}, \forall\, 1 \leq \ell \leq L$ combinations. Then, if we can deploy robots at ℓ different times, we have $\binom{R(PQ+1)}{R}^\ell$ combinations as there are $\binom{R(PQ+1)}{R}$ combinations in each of ℓ times. $(PQ+1)$ is the number of combinations for every robot since every robot can be deployed to one of the $P \cdot Q$ positions and also can be not deployed. To make the problem tractable when computing an optimal solution, we simulate the problem with $\mathcal{T} \in \{4, \ldots, 8\}$, $L \in \{2, \ldots, 4\}$, $\mathcal{R} \in \{2, \ldots, 4\}$, $P \in \{3, \ldots, 5\}$ and $Q \in \{3, \ldots, 5\}$. For the environmental process, we use the SE kernel for both temporal and spatial kernels. The cost of every robot r is the multiplication of the traveling cost with a weight factor w_r, which is a random number generated in $(0, 1)$. In the simulation, we first collect training samples from an Gaussian mixture model (GMM) [48], and then train the GP using these samples. Finally, we make the deployment strategy based on the GP prediction. The 2D space is 200×200. The GMM is modeled as a linear combination of fixed basis functions and time-varying weights. In the simulation, we use 5 different basis functions. So, the output the GMM is $\varphi(x, y, t) = \sum_{i=1}^{5} w_i(t) b_i(x, y) = \mathbf{w}(t)^\top \mathbf{b}(x, y)$, where $w_i(t) \in \mathbb{R}$ is a weight and $b_i(x, y) \in \mathbb{R}$ is the output of a basis function. $\mathbf{w}(t) \in \mathbb{R}^5$ is the stacked weights at

time t and $\mathbf{b}(x, y) \in \mathbb{R}^5$ is the stacked basis functions for the location (x, y). The dynamics of weights are $\dot{\mathbf{w}} = A\mathbf{w} + \sigma(t)$, where $A = -I_{5\times5}$ and $\sigma(t) \in \mathbb{R}^5$ is a Gaussian noise. The initial settings of weights are $\{5, 5, 3, 8, 4\}$ and the initial locations of these basis functions are $(100, 100), (60, 80), (40, 30), (160, 160), (160, 30)$.

We run simulations 100 times under the settings described above. Fig. 3.3 shows the optimality ratios of the greedy solution to an optimal solution for different problem sizes. We denote the problem size as $(L \cdot R \cdot P \cdot Q)$. As we use two matroid constraints $\mathcal{I}_{21}$, $\mathcal{I}_{23}$ to constrain the time and a general knapsack constraint to constrain the cost, then $p = 2$ and $\ell = 1$. Also, we set the tunable parameter $\eta = 0.1$ in Algorithm 1. Therefore, the optimality ratio of the greedy method is $\frac{1}{(1+\eta)(p+2\ell+1)} = \frac{1}{(1+\eta)(2+2+1)} \approx 18\%$ according to Theorem 3. In Fig. 3.4, we show the mutual information of the greedy solution for different problem sizes. Since different simulations may have the same problem size, we here use the average of different utilities/mutual information if this case happens. From the result, we can see that as the set $\mathcal{D}$ becomes big, the mutual information become small. An intuitive example to show the connection between a deployment strategy and a GMM environment can be seen in Fig. 3.2. Any selected element from the ground set defines a sensing location and time, and all selected elements make the deployment strategy set.

3.4 Conclusions

In this chapter, we proposed a new intermittent deployment problem and demonstrated how to use matroid and knapsack constraints to model different instances of the problem. We also illustrated how to utilize a general submodular maximization method to solve our problem. Finally, the effectiveness of the solutions was demonstrated by Monte Carlo simulations.

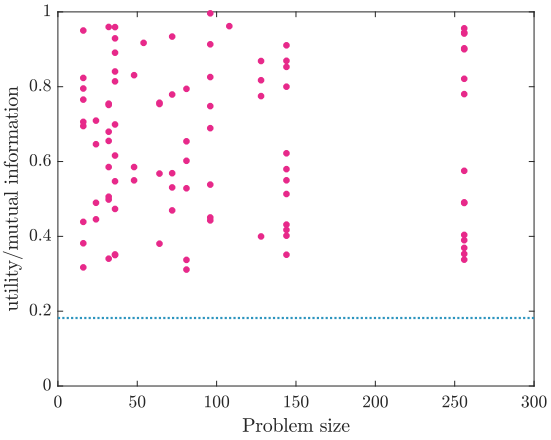


Fig. 3.3. The optimality ratio of the greedy method with respect to different problem sizes.

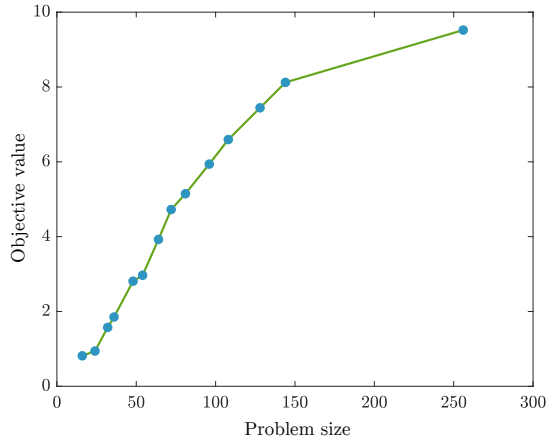


Fig. 3.4. The objective values with respect to different problem sizes.

Chapter 4

Submodular Optimization for Coupled Task Allocation and Intermittent Deployment Problems

It is common that multi-robot team objectives are intertwined or coupled, with an especially interesting example being objectives that operate sequentially over different time scales. Consider, for example, an environmental monitoring application where we first need to allocate a group of heterogeneous robots to perform a set of tasks, e.g., collecting samples or otherwise interacting with the environment. This problem is well-known as a multi-robot task allocation problem and occurs over a short time scale. However, the critical factor considered in this chapter is that the tasks themselves may impact underlying environmental dynamics, and thus future long-term objectives will be influenced. Here we consider the future long-term objective of multi-robot intermittent deployment, where we ask: when is it appropriate to deploy a robotic team for long-term monitoring? To account for the impact of short-term task allocation on long-term monitoring, we formulate a general coupled submodular optimization problem, which yields a bounded sub-optimal solution for a provably hard problem.

Related Work

Multi-robot task allocation problems have been studied for a long time [49, 50]. In this chapter, our focus will instead be on the intermittent deployment problem and its coupling with task allocation. The idea of intermittence in robotics applications can be either by design or intrinsic. In our motivating example of environmental monitoring, we design the system to intermittently deploy to reduce cost over the long-term. In [51], the mobile robot networks are required to be connected intermittently, which is achieved through a linear temporal logic method. In [34], the robots can only communicate periodically in predefined time steps. On the other hand, in [33], the authors studied the convergence of Kalman filtering when the measurement arrival time is intrinsically intermittent. A similar application to our deployment problem without the intermittent feature is the sensor scheduling problem [52], which needs to schedule sensors sequentially to estimate a linear system.

More generally, various multi-robot problems contain two or more sub-problems coupled in some way. In multi-robot motion planning applications, such as collaborative coordination [53], the movement of one robot will impact the others. In the environmental monitoring problem [54], different robots need to work together to cover/explore the environment more efficiently. In humanoid robot manipulation [55], the problem of finding an optimal grasp position and the problem of reaching the object is indeed strongly coupled. Thus, in general, it is necessary to model problem couplings and seek efficient algorithms to provide quality solutions. If the domain of a problem is discrete, we need to consider methods for combinatorial optimization, as we do in this work. For example, the sensor placement problem [27] seeks to find locations for sensors from a discrete location set to maximize the mutual information for estimating the environment. The abstract task allocation problem seeks to assign robots to tasks to maximize the reward [22]. Other applications can be found in the target tracking problem, the environmental monitoring problem, etc.. Generally, these problems are NP-hard [56]

and cannot be solved optimally by using a polynomial time algorithm. Moreover, if two or more problems are coupled with each other, it is even harder to have quality solutions. Therefore, previous work mainly focuses on generating approximation methods which yield sub-optimality bounds that are often used in practice. In particular, the key focus of late is on greedy algorithms for submodular function optimization. These greedy algorithms usually come with a performance guarantee. The first provable bound for submodular function optimization over general matroid constraints is shown in [23]. Combining the modular and the submodular result as a single result, submodular curvature is used in [12, 13, 57] for proving optimality bounds. A recently improved version of monotone submodular function maximization over general matroid constraints by using a multi-linear relaxation scheme is shown in [44]. Also, [58] proposed a multivariate version of submodular optimization using the multi-linear extension. This work focuses on minimizing or maximizing a single objective represented as a multivariate function. In our work, the objective function includes two sub-objective functions. For a comprehensive overview of submodular optimization, the reader is referred to [59] for additional details.

A common practice to solve coupled problems is to solve each problem separately and combine solutions. In this chapter, we instead propose to solve coupled submodular optimization problems with general matroid constraints. As an example of such a problem, we couple a task allocation problem [22] with an intermittent deployment problem [3] where robots are optimally deployed to monitor an environment over time.

Contributions

1. We formalize a modeling and solution method for coupled submodular optimization problems with general matroid constraints.

2. We demonstrate how to use matroids to model constraints in robotic applications;

3. We provide a greedy algorithm with bounded optimality for solving the general coupled optimization problem. We demonstrate this by using a combination of the task allocation problem and the intermittent deployment problem to show the performance in an environmental monitoring application.

Chapter Outline

The remainder of this chapter is organized as follows. We first introduce the problem formulation in Section 4.1. In Section 4.2, we present the details about our running example: the multi-robot task allocation problem and the multi-robot intermittent deployment problem. Then, we generalize the properties of our coupled problem formulation. In Section 4.3, we present a greedy algorithm with provable performance bounds. In Section 4.4, we demonstrate the result of the proposed algorithm using Monte Carlo simulations. Conclusions and directions for future work are stated in the final section.

4.1 Problem Formulation

Problem 1. *The multi-robot task allocation problem coupled with the multi-robot intermittent deployment is given by:*

$$\begin{aligned} \underset{\mathcal{A}\subseteq\mathcal{E}}{\textit{maximize}} \quad & g(\mathcal{A}) + \max_{\mathcal{B}\subseteq\mathcal{V}} f(\mathcal{A},\mathcal{B}) \\ \textit{subject to} \quad & \mathcal{A}\in\mathcal{I}_1, \mathcal{B}\in\mathcal{I}_2. \end{aligned}$$

where $\mathcal{A}$ is a multi-robot task allocation chosen from the finite ground set $\mathcal{E}$ with $\mathcal{A}$ satisfying the matroid intersection constraint $\mathcal{M}_1 = (\mathcal{E}, \mathcal{I}_1)$, i.e., $\mathcal{A} \in \mathcal{I}_1$. The function $g : 2^{\mathcal{E}} \mapsto \mathbb{R}$ is a utility function for the multi-robot task allocation problem. $\mathcal{B}$ is a multi-robot deployment policy chosen from the finite ground set $\mathcal{V}$ with $\mathcal{B}$ satisfying the matroid intersection constraint $\mathcal{M}_2 = (\mathcal{V}, \mathcal{I}_2)$, i.e., $\mathcal{B} \in \mathcal{I}_2$. The function $f : 2^{\mathcal{E} \times \mathcal{V}} \mapsto \mathbb{R}$ is a utility function for the intermittent deployment problem, where $\mathcal{E} \times \mathcal{V}$ is the Cartesian product of $\mathcal{E}$ and $\mathcal{V}$. $f(\cdot)$ is a function of both $\mathcal{A}$ and $\mathcal{B}$ because we assume that multi-robot task allocations (first phase, short-term) *have an impact on the multi-robot intermittent deployment action* (second phase, long-term)*. Then, the objective function is*

$$\begin{aligned} m(\mathcal{A}) &= g(\mathcal{A}) + h(\mathcal{A}) \\ &= g(\mathcal{A}) + \max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{A}, \mathcal{B}). \end{aligned}$$

In the following sections, we will detail how to build our modular/submodular functions, how to use matroids to model constraints, and how to solve Problem 1.

4.2 Coupled Multi-Robot Task Allocation and Intermittent Deployment Problem

Now, we present the details about each problem and then give the properties of the problem formulation.

4.2.1 The Multi-Robot Task Allocation Problem

The multi-robot task allocation model comes from our previous work [22]. Briefly, the formulation of this problem with matroid intersection constraint is:

$$\begin{aligned} \underset{\mathcal{A}\subseteq\mathcal{E}}{\text{maximize}} \quad & g(\mathcal{A}) \\ \text{subject to} \quad & \mathcal{A} \in \mathcal{I}_1, \end{aligned}$$

where $g : 2^{\mathcal{E}} \mapsto \mathbb{R}$ is the utility function and $\mathcal{M}_1 = (\mathcal{E}, \mathcal{I}_1)$ is the matroid intersection constraint. The element of the ground set $\mathcal{E}$ is represented by the triplet (r, d, e) for $r \in \mathcal{R}_1, d \in \mathcal{D}, e \in \mathcal{E}$, and $(d, e) \in \mathcal{O}$. Here, $\mathcal{R}_1$ is the robot ground set. $\mathcal{O}$ is the functionality-requirement ground set. (r, d, e) can be read as "robot i performs functionality d for the requirement e". Each functionality-requirement pair (d, e) is a task. Therefore, $\mathcal{O}$ can also be viewed as the task ground set. Each triplet (r, d, e) forms an element of an allocation set $\mathcal{A}$. The goal is to allocate tasks from $\mathcal{O}$ to the robots in $\mathcal{R}_1$ to form an allocation set $\mathcal{A}$ to maximize the utility $g(\mathcal{A})$.

To make it more clear, let's look at an example. In the example, we define $\mathcal{R}_1 = \{1, 2\}$, which means there are two robots available. Specifically, let's consider the case that the first robot is a ground robot and the second one is an aerial robot. Also, if functionality-requirement set is $\mathcal{O} = \big\{(d_1, e_1), (d_2, e_2)\big\}$, where (d_1, e_1) means flying ability (functionality: d_1) for a long distance package delivery (requirement: e_1), and (d_2, e_2) means moving/flying ability (functionality: d_2) for data collection (requirement: e_2). Then, we can build constraints for these two robots as follows. Specifically, the constraint for robot 1 is $\mathcal{I}_1 = \big\{\{(1, d_2, e_2)\}\big\}$ and the constraint for robot 2 is $\mathcal{I}_2 = \big\{\{(2, d_1, e_1)\}, \{(2, d_2, e_2)\}\big\}$. We construct these two constraints due to the reason that the aerial robot 2 can finish both (d_1, e_2) and (d_2, e_2) functionality-requirement pairs while the ground robot 1 can only finish the pair (d_2, e_2).

This *independence constraint* $\mathcal{M}_{11}$ with two other constraints, *uniqueness constraint* $\mathcal{M}_{12}$ and *topology constraint* $\mathcal{M}_{13}$, are matroidal as shown in [22]. The *uniqueness constraint* requires each functionality-requirement pair can only be allocated no more than once. The *topology constraint* requires the distance of adjacent elements of an allocation is less than a threshold to ensure robots can communicate with each other. The intersection of these matroid constraints forms the matroid intersection constraint $\mathcal{M}_1$ of this problem. For the utility function $g(\cdot)$, a simple example would be the sum of reward for each element a of allocation set $\mathcal{A}$, i.e., $g(\mathcal{A}) = \sum_{a \in \mathcal{A}} u_a$, where u_a is the reward of the allocation element $a = (r, d, e)$.

4.2.2 The Multi-Robot Intermittent Deployment Problem

The idea of intermittently deploying a multi-robot system is to render the system more efficient by asking: When is it appropriate to deploy a robotics team? Which combination of robots is suitable?

As we will deal with deployment constraints over time, we first partition the ground set $\mathcal{V}$ of this problem into disjoint sets $\mathcal{V}_1, \ldots, \mathcal{V}_K$ over a time horizon of K steps. The partition at time k is $\mathcal{V}_k$. Specifically, $\mathcal{V}_k = \big\{(r, d) \mid r \in \mathcal{R}_2, d \in \{0, 1\}\big\}$ with $\mathcal{R}_2$ the set of robots for this problem and d the deployment action, where 0 and 1 means not deploy and deploy, respectively. To capture the idea of intermittent deployment, we require the deployment policy to satisfy the following constraints:

1. No more than ℓ_k robots can be deployed at time k for $k = 1, \ldots, K$. This is our constraint $\mathcal{M}_{21}$.

2. The number of times where there is at least one robot deployed is less than or equal to ℓ. This is $\mathcal{M}_{22}$.

3. Each robot can only be selected or not selected at every time k for $k = 1, \ldots, K$. This is our constraint $\mathcal{M}_{23}$.

To satisfy Item 1, consider the constraint $\mathcal{M}_{21} = (\mathcal{V}, \mathcal{I}_{21})$ where

$$\mathcal{I}_{21} = \left\{ \mathcal{B} \subseteq \mathcal{V} : |\mathcal{B} \cap \mathcal{V}_k| \leq \ell_k \right\}. \tag{4.1}$$

To satisfy Item 2, consider the constraint $\mathcal{M}_{22} = (\mathcal{V}, \mathcal{I}_{22})$ where

$$\mathcal{I}_{22} = \left\{ \mathcal{B} \subseteq \mathcal{V} : \sum_{k=1}^{K} \mathbb{1}\left(|\mathcal{B} \cap \mathcal{V}_k| \right) \leq \ell \right\}, \tag{4.2}$$

and $\mathbb{1}(\cdot)$ is an indicator function that takes the form

$$\mathbb{1}(|\mathcal{B} \cap \mathcal{V}_k|) = \begin{cases} 0 & \text{, if } |\mathcal{B} \cap \mathcal{V}_k| = 0, \\ 1 & \text{, if } |\mathcal{B} \cap \mathcal{V}_k| \geq 1. \end{cases}$$

To satisfy Item 3, consider the constraint $\mathcal{M}_{23} = (\mathcal{V}, \mathcal{I}_{23})$, where

$$\mathcal{I}_{23} = \left\{ \mathcal{B} \subseteq \mathcal{V} : |\mathcal{B}_r \cap \mathcal{V}_k| = 1, \forall\, r \in \mathcal{R}_2 \right\}. \tag{4.3}$$

In Fig. 4.1, we illustrate the intermittent deployment concept. Finally, we must verify that the above constraints are indeed matroidal.

Theorem 5. *The constraints $\mathcal{M}_{21}, \mathcal{M}_{22}$ and $\mathcal{M}_{23}$ are matroidal.*

Proof. We only need to verify property ii and iii of matroid definition since property i holds by construction.

1) For constraint $\mathcal{M}_{21}$:

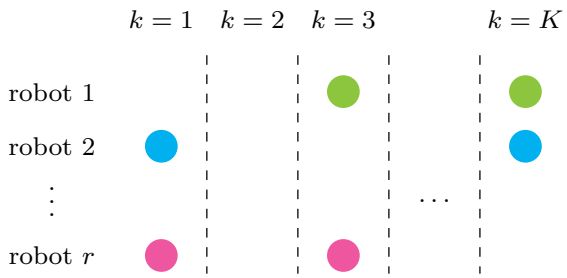


Fig. 4.1. An illustration of the intermittent deployment idea. $r \in \mathcal{R}_2$ and $k = 1, \ldots, K$. At time $k = 2$, there is no deployment. The constraint Item 1 and Item 3 are applied vertically for robots in each time. The constraint Item 2 is applied horizontally when viewing all robots as a group in each time.

Matroid axiom ii: Consider a set $\mathcal{B}_1 \subseteq \mathcal{V}$ and assume that for every element $(r, d) \in \mathcal{B}_1$ it satisfies that $|\mathcal{B}_1 \cap \mathcal{V}_k| \leq \ell_k$, i.e. $\mathcal{B}_1 \in \mathcal{I}_{21}$. Now, for any $\mathcal{B}_2 \subseteq \mathcal{B}_1$ and any element $(r, d) \in \mathcal{B}_2$, we know that if $|\mathcal{B}_1 \cap \mathcal{V}_k| < \ell_k$ then $|\mathcal{B}_2 \cap \mathcal{V}_k| < \ell_k$. So, if $\mathcal{B}_2 \subseteq \mathcal{B}_1 \in \mathcal{I}_{21}$, then $\mathcal{B}_2 \in \mathcal{I}_{21}$. The property ii is verified.

Matroid axiom iii: Now consider the case $\mathcal{B}_1, \mathcal{B}_2 \in \mathcal{I}_{21}$. Without loss of generality, we assume that $|\mathcal{B}_2| < |\mathcal{B}_1|$, i.e., $\mathcal{B}_1 \setminus \mathcal{B}_2 \neq \emptyset$. Let's assume that there exists no element $e \in (\mathcal{B}_1 \setminus \mathcal{B}_2)$ such that $\mathcal{B}_2 \cup \{e\} \in \mathcal{I}_{21}$. Our assumption implies that ℓ_k robots have been allocated in $\mathcal{B}_2$, which implies $|\mathcal{B}_2| = \ell_k$. However, since $\mathcal{B}_1 \in \mathcal{I}_{21}$, it implies $|\mathcal{B}_1| \leq \ell_k$, we have shown the contradiction as $|\mathcal{B}_2| < |\mathcal{B}_1|$. So, if $\mathcal{B}_1, \mathcal{B}_2 \in \mathcal{I}_{22}$ and $|\mathcal{B}_2| < |\mathcal{B}_1|$, there exists a $e \in (\mathcal{B}_1 \setminus \mathcal{B}_2)$ such that $\mathcal{B}_2 \cup \{e\} \in \mathcal{I}_{21}$. The property iii is verified.

2) For constraint $\mathcal{M}_{22}$:

Matroid axiom ii: Consider a set $\mathcal{B}_1 \subseteq \mathcal{V}$ and assume that for every element $(r, d) \in \mathcal{B}_1$ it satisfies that $\sum_{k=1}^{K} \mathbb{1}\left(|\mathcal{B}_1 \cap \mathcal{V}_k|\right) \leq \ell$, i.e. $\mathcal{B}_1 \in \mathcal{I}_{22}$. For elements $(r, d) \in \mathcal{B}_1$, there exist three cases: i) $|\mathcal{B}_1 \cap \mathcal{V}_k| \geq 1$ holds for every $(r, d) \in \mathcal{B}_1$; ii) $|\mathcal{B}_1 \cap \mathcal{V}_k| = 0$ holds for every $(r, d) \in \mathcal{B}_1$; iii) $|\mathcal{B}_1 \cap \mathcal{V}_k| \geq 1$ for some $(r, d) \in \mathcal{B}_1$ and $|\mathcal{B}_1 \cap \mathcal{V}_k| = 0$ holds for the rest of the elements in $\mathcal{B}_1$. *Case i)*: if $|\mathcal{B}_1 \cap \mathcal{V}_k| \geq 1$, then, either $|\mathcal{B}_2 \cap \mathcal{V}_k| \geq 1$ holds or $|\mathcal{B}_2 \cap \mathcal{V}_k| = 0$ holds. Therefore, we know $\sum_{k=1}^{K} \mathbb{1}\left(|\mathcal{B}_2 \cap \mathcal{V}_k|\right) \leq \ell$ holds. *Case ii)*: If $|\mathcal{B}_1 \cap \mathcal{V}_k| = 0$, then $|\mathcal{B}_2 \cap \mathcal{V}_k| = 0$, which

implies $\sum_{k=1}^{K} \mathbb{1}\left(|\mathcal{B}_2 \cap \mathcal{V}_k|\right) \leq \ell$ holds. *Case iii)*: this case is a combination of the first two cases, so $\sum_{k=1}^{K} \mathbb{1}\left(|\mathcal{B}_2 \cap \mathcal{V}_k|\right) \leq \ell$ still holds. So, if $\mathcal{B}_2 \subseteq \mathcal{B}_1 \in \mathcal{I}_{22}$, then $\mathcal{B}_2 \in \mathcal{I}_{22}$ holds. The property ii is verified.

Matroid axiom iii: Now consider the case where we assume $\mathcal{B}_1, \mathcal{B}_2 \in \mathcal{I}_{22}$ and $|\mathcal{B}_2| < |\mathcal{B}_1|$. In this case, we have that $\mathcal{B}_1 \setminus \mathcal{B}_2$ is non-empty. Let's assume that there exists no element $e \in (\mathcal{B}_1 \setminus \mathcal{B}_2)$, such that $\mathcal{B}_2 \cup \{e\} \in \mathcal{I}_{22}$. This implies that the number of times where there is at least one deployments for $\mathcal{B}_2$ has been reached ℓ. However, from the definition we know that the number of times where there is at least one deployment for $\mathcal{B}_1$ is at most ℓ. So, we have shown a contradiction as $|\mathcal{B}_1| < |\mathcal{B}_2|$. So, if $\mathcal{B}_1, \mathcal{B}_2 \in \mathcal{I}_{22}$ and $|\mathcal{B}_2| < |\mathcal{B}_1|$, then there exists a $e \in (\mathcal{B}_1 \setminus \mathcal{B}_2)$ such that $\mathcal{B}_2 \cup \{e\} \in \mathcal{I}_{22}$. The property iii is verified.

3) For constraint $\mathcal{M}_{23}$: The verification is similar to the verification for $\mathcal{M}_{21}$, and we omit it for brevity. □

The intersection of the above constraints forms the matroid intersection constraint $\mathcal{M}_2 = (\mathcal{V}, \mathcal{I}_2)$ of this problem with $\mathcal{I}_2 = \mathcal{I}_{21} \cap \mathcal{I}_{22} \cap \mathcal{I}_{23}$. The matroid modeling method is especially useful for robotics applications when constraints are abstract [22]. Here we use $\mathcal{M}_{21}$, $\mathcal{M}_{22}$, and $\mathcal{M}_{23}$ as examples to illustrate the idea of multi-robot intermittent deployment constraints. Other constraints for this problem can also be integrated easily into the problem formulation, e.g., the number of times for each robot that can be deployed or the composition of the robot teams that are deployed. It is straightforward to show that such constraints would also obey the matroid properties.

4.2.3 The Submodularity of the Objective Function

After giving the details about each problem, we now focus on the properties of the problem formulation in this section. The second part of the objective function is $h(\mathcal{A}) = \max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{A}, \mathcal{B})$,

which takes the task allocation's impact into consideration when evaluating the subsequent multi-robot deployment strategy. If we are interested in the best payoff a robot can experience from the first phase to the second phase, then we have that $f(\mathcal{A}, \mathcal{B}) = \max_{a \in \mathcal{A}} s(a, \mathcal{B})$. To begin with, we make some definitions as follows

$$\mathcal{B}_1 = \arg\max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{X}, \mathcal{B}), \qquad \mathcal{B}_3 = \arg\max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}), \tag{4.4}$$

$$\mathcal{B}_2 = \arg\max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{Y}, \mathcal{B}), \qquad \mathcal{B}_4 = \arg\max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}). \tag{4.5}$$

Following the definition $h(\mathcal{A}) = \max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{A}, \mathcal{B})$, we have

$$h(\mathcal{X}) = f(\mathcal{X}, \mathcal{B}_1), \qquad h(\mathcal{X} \cup \mathcal{Y}) = f\left(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3\right), \tag{4.6}$$

$$h(\mathcal{Y}) = f(\mathcal{Y}, \mathcal{B}_2), \qquad h(\mathcal{X} \cap \mathcal{Y}) = f\left(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}_4\right). \tag{4.7}$$

Using these definitions, the properties of the objective function is now formalized.

Theorem 6. *If $f(\mathcal{A}, \mathcal{B}) = \max_{a \in \mathcal{A}} s(a, \mathcal{B})$, the objective function $h(\mathcal{A}) = \max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{A}, \mathcal{B})$ is non-decreasing and submodular.*

Proof. 1) Non-decreasing

For proving this property, we need to show, for any $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{E}$, if $\mathcal{X} \subseteq \mathcal{Y}$ then $h(\mathcal{X}) \leq h(\mathcal{Y})$. When $\mathcal{X} \subseteq \mathcal{Y}$,

$$\begin{aligned} & h(\mathcal{Y}) - h(\mathcal{X}) \\ =& f(\mathcal{Y}, \mathcal{B}_2) - f(\mathcal{X}, \mathcal{B}_1) \\ =& \left(f(\mathcal{Y}, \mathcal{B}_2) - f(\mathcal{Y}, \mathcal{B}_1)\right) + \left(f(\mathcal{Y}, \mathcal{B}_1) - f(\mathcal{X}, \mathcal{B}_1)\right). \end{aligned}$$

The first equality holds because of (4.6) and (4.7). Following the definition of $\mathcal{B}_2$, it holds

that $f(\mathcal{Y}, \mathcal{B}_2) \geq f(\mathcal{Y}, \mathcal{B}_1)$. Since $\mathcal{X} \subseteq \mathcal{Y}$ and $f(\mathcal{A}, \mathcal{B}) = \max_{a \in \mathcal{A}} s(a, \mathcal{B})$, it holds that $f(\mathcal{Y}, \mathcal{B}_1) \geq f(\mathcal{X}, \mathcal{B}_1)$. Therefore, $h(\mathcal{Y}) - h(\mathcal{X}) \geq 0$.

2) Submodularity

For proving submodularity, we need to show that for any $\mathcal{X}, \mathcal{Y} \subseteq E$, the following two hold

$$\max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right) = f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3), \tag{4.8}$$

$$\min\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right) \geq f(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}_4). \tag{4.9}$$

Combining (4.8) and (4.9), we have $f(\mathcal{X}, \mathcal{B}_1) + f(\mathcal{Y}, \mathcal{B}_2) \geq f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3) + f(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}_4)$. This is equivalent to $h(\mathcal{X}) + h(\mathcal{Y}) \geq h(\mathcal{X} \cup \mathcal{Y}) + h(\mathcal{X} \cap \mathcal{Y})$ and satisfies the submodularity requirement for $h(A)$. So, we only need to prove (4.8) and (4.9).

Part a: For proving (4.8)

Since any equality $x = y$ can be proven by proving $x \leq y$ and $y \leq x$ for any $x, y \in \mathbb{R}$, we will prove (4.8) by proving:

$$f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3) \leq \max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right), \tag{4.10}$$

$$\max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right) \leq f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3). \tag{4.11}$$

Part a.1: For proving (4.10)

From the definition of $\mathcal{B}_1$, we have

$$f(\mathcal{X}, \mathcal{B}_3) \leq f(\mathcal{X}, \mathcal{B}_1) \leq \max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right)$$

These two hold because $\mathcal{B}_1 = \arg\max_{B \subseteq V} f(\mathcal{X}, \mathcal{B})$ and $x \leq \max(x, y), \forall\, x, y \in \mathbb{R}$. Similarly,

$$f(\mathcal{Y}, \mathcal{B}_3) \leq \max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right).$$

We know from the definition that $f(\mathcal{A}, \mathcal{B}) = \max_{a \in \mathcal{A}} s(a, \mathcal{B})$. Then, $f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3) = \max_{a \in \mathcal{X} \cup \mathcal{Y}} f_1(a, \mathcal{B}_3)$. Therefore, we have $f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3) \leq \max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right)$ as described in (4.10). This inequality holds because we know that $a \in \mathcal{X} \cup \mathcal{Y}$ means $a \in \mathcal{X}$ or $a \in \mathcal{Y}$. If $a \in \mathcal{X}$, the first inequality holds. If $a \in \mathcal{Y}$, the second inequality holds.

Part a.2: For proving (4.11)

It holds that $f(\mathcal{X}, \mathcal{B}_1) \leq f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_1) \leq f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3)$. The first inequality holds because of the monotonicity of $f(\mathcal{A}, \mathcal{B})$ on $\mathcal{A} \subseteq \mathcal{E}$ for any $\mathcal{B} \subseteq \mathcal{V}$, and the second inequality holds because $\mathcal{B}_3 = \arg\max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B})$. Similarly, $f(\mathcal{Y}, \mathcal{B}_2) \leq f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3)$. Combining these two inequalities, we get the result $\max\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right) \leq f(\mathcal{X} \cup \mathcal{Y}, \mathcal{B}_3)$.

Part b: For proving (4.9)

It holds that $f(\mathcal{X}, \mathcal{B}_1) \geq f(\mathcal{X}, \mathcal{B}_4) \geq f(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}_4)$. The first inequality holds since $\mathcal{B}_1 = \arg\max_{\mathcal{B} \subseteq \mathcal{V}} f(\mathcal{X}, \mathcal{B})$. The second inequality holds due to the monotonicity of $f(\mathcal{A}, \mathcal{B})$ on $\mathcal{A} \subseteq \mathcal{E}$ for any $\mathcal{B} \subseteq \mathcal{V}$. Similarly, $f(\mathcal{Y}, \mathcal{B}_2) \geq f(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}_4)$. Combining these two inequalities, we get the result $\min\left(f(\mathcal{X}, \mathcal{B}_1), f(\mathcal{Y}, \mathcal{B}_2)\right) \geq f(\mathcal{X} \cap \mathcal{Y}, \mathcal{B}_4)$. □

Remark 3. *If we are interested in the worst payoff that a robot can experience from the first phase to the second phase, we have* $f(\mathcal{A}, \mathcal{B}) = \min_{a \in \mathcal{A}} s(a, \mathcal{B})$. *Then,* $h(\mathcal{A})$ *is non-increasing on* $\mathcal{A}$.

It is worth mentioning that the purpose of this chapter is to find a method for solving coupled optimization problems in the robotics field, and we use the multi-robot task allocation problem and the multi-robot intermittent deployment problem as examples to illustrate this

idea. Other applications can also be applied in this formulation.

4.3 Algorithm Analysis

Algorithm 2 shows the greedy algorithm for solving the coupled problem when both $g(\cdot)$ and $s(\cdot)$ are set functions. If either $g(\cdot)$ or $s(\cdot)$ are a sequence, we only need to change the method slightly. Specifically, if $s(\cdot)$ is a sequence function, we need to change the line 6 in Algorithm 2 to $(j \leftarrow 0, \ldots, k \ldots, K-1)$ where k represents the sequence order. We refer to this as a modified version of Algorithm 2 for dealing with the sequence function case.

Theorem 7 (*Performance & complexity*)**.** *Let $\mathcal{A}^G$ and $\mathcal{A}^\star$ be greedy and optimal solutions, respectively. $m_1 = |\mathcal{M}_1|$, $m_2 = |\mathcal{M}_2|$. Algorithm 2 has the following performance:*

1. *If $g(\cdot)$ is a non-decreasing modular or submodular set function and*
   - *if $s(a, \mathcal{B})$ is a non-decreasing modular set function on $\mathcal{B}$ for any a, then, $m(\mathcal{A}^G) \geq 1/(m_2(m_1+1))m(\mathcal{A}^\star)$.*
   - *if $s(a, \mathcal{B})$ is a non-decreasing submodular set function on $\mathcal{B}$ for any a, then, $m(\mathcal{A}^G) \geq 1/((m_1+1)(m_2+1))m(\mathcal{A}^\star)$.*
2. *If $g(\cdot)$ is a non-decreasing modular or submodular set function and $s(a, \mathcal{B})$ is a non-decreasing sequence submodular function on $\mathcal{B}$, then*

$$m(\mathcal{A}^G) \geq (m_1+1)^{-1}(1 - e^{-1/(m_2+1)})m(\mathcal{A}^\star).$$

3. *Algorithm 2 has time complexity $\mathcal{O}\left(|\mathcal{E}|^3 \cdot |\mathcal{V}|^2\right)$.*

Proof. 1) Let $\mathcal{A}_i^G$ denote the greedy output for $\mathcal{A}$ at step i. In order to get $\mathcal{A}^G$, we need

to evaluate every $a \notin \mathcal{A}_i^G : \mathcal{A}_i^G \cup \{a\} \in \mathcal{I}_1$ and incrementally add a' to $\mathcal{A}_i$ in terms of maximizing the objective value $m(\mathcal{A}_i^G)$. That is, $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{a'\}$. In Algorithm 2, we omit the subscript and write it as $\mathcal{A} \leftarrow \mathcal{A} \cup \{a'\}$ for brevity. This omission also applies to other variables. We cannot evaluate $f(\cdot)$ without knowing $\mathcal{B}$ since $f : 2^{\mathcal{E}\times\mathcal{V}} \mapsto \mathbb{R}$ and $\mathcal{A} \in \mathcal{E}, \mathcal{B} \in \mathcal{V}$. If we can get a $\mathcal{B}^\star$ that gives the maximum objective function for each $\mathcal{A}_i^G \cup \{a\}$, where $a \notin \mathcal{A}_i^G : \mathcal{A}_i^G \cup \{a\} \in \mathcal{I}_1$, then it's easy to get a'. After adding a into A_i, we can construct the greedy solution A_{i+1} for $\mathcal{A}$ at step i, i.e. $\mathcal{A}_{i+1}^G = \mathcal{A}_i^G \cup \{a\}$. Here, $\mathcal{B}^\star$ is an optimal solution with respect to every $\mathcal{A}_i^G \cup \{a\}$ in terms of objective function value. Due to the intractability for getting $\mathcal{B}^\star$ for every $a \notin \mathcal{A}_i^G : \mathcal{A}_i^G \cup \{a\} \in \mathcal{I}_1$, we propose to use another greedy iteration to get $\mathcal{B}^G$ for replacing $\mathcal{B}^\star$.

Now, the only problem left is how to get $\mathcal{B}^G$. We construct $\mathcal{B}_j^G$ at step j for every $a \notin \mathcal{A}_i^G : \mathcal{A}_i^G \cup \{a\} \in \mathcal{I}_1$ using a similar greedy method. Notice that there is a $\mathcal{B}^G$ corresponding to every $\mathcal{A}_i^G \cup \{a\}$, and the final $\mathcal{B}^G$ is corresponding to $\mathcal{A}_{|\mathcal{E}|}^G$.

If $g(\mathcal{A})$ is non-decreasing and modular on $\mathcal{A} \subseteq \mathcal{E}$, then $g(\mathcal{A}^G) \geq \frac{1}{m_1} g(\mathcal{A}^\star)$; if $g(\mathcal{A})$ is non-decreasing and submodular on $\mathcal{A} \subseteq \mathcal{E}$, then $g(\mathcal{A}^G) \geq \frac{1}{m_1+1} g(\mathcal{A}^\star)$ [23]. For the second part of the objective function, it holds that $f(\mathcal{A}^G, \mathcal{B}^\star) \geq \frac{1}{m_1+1} f(\mathcal{A}^\star, \mathcal{B}^\star)$ since $h(\mathcal{A}) = \max_{\mathcal{B}\subseteq\mathcal{V}} f(\mathcal{A}, \mathcal{B})$ is a submodular function on $\mathcal{A}$ according to Theorem 6. Then,

- If $g(\mathcal{A})$ is non-decreasing modular on $\mathcal{A} \subseteq \mathcal{E}$ and we also use the greedy output $\mathcal{B}^G$ for $\mathcal{B}$, then $m(\mathcal{A}^G) \geq \frac{1}{m_1} g(\mathcal{A}^\star) + \frac{1}{m_1+1} f(\mathcal{A}^\star, \mathcal{B}^G)$.
- If $g(\mathcal{A})$ is non-decreasing submodular on $\mathcal{A} \subseteq \mathcal{E}$ and we also use $\mathcal{B}^G$ for $\mathcal{B}$, then $m(\mathcal{A}^G) \geq \frac{1}{m_1+1} g(\mathcal{A}^\star) + \frac{1}{m_1+1} f(\mathcal{A}^\star, \mathcal{B}^G)$.

When $s(a, \mathcal{B})$ follows the following property, we obtain

- If $s(a, \mathcal{B})$ is non-decreasing modular on $\mathcal{B} \subseteq \mathcal{V}$ for any $a \in \mathcal{A}$, then $f(\mathcal{A}^\star, \mathcal{B}^G) \geq$

$\frac{1}{m_2} f(\mathcal{A}^\star, \mathcal{B}^\star)$.

- If $s(\mathcal{A}, \mathcal{B})$ is non-decreasing submodular on $\mathcal{B} \subseteq \mathcal{V}$ for any $a \in \mathcal{A}$, then $f\left(\mathcal{A}^\star, \mathcal{B}^G\right) \geq \frac{1}{m_2+1} f\left(\mathcal{A}^\star, \mathcal{B}^\star\right)$.

Finally, combining one result regarding $f(\mathcal{A}, \mathcal{B})$ and one result regarding $m(\mathcal{A})$, we can get a corresponding result.

2) The analysis is similar to the analysis for the performance 1) except for a small change regarding how to get $\mathcal{B}^G$. When considering $\mathcal{M}_1$ and $\mathcal{M}_2$, we can use the result from [60] for analyzing the sequence submodular function $s(\cdot)$ and the above analysis. Then, we have the bound as shown in the statement.

3) Computational complexity: To get $\mathcal{A}^G$, we need to incrementally select an element $a \in \mathcal{E}$ for $\mathcal{O}\left(|\mathcal{E}|\right)$ times. For each $a \in \mathcal{E}$, we need to evaluate its objective function value $\mathcal{O}\left(|\mathcal{E}|\right)$ times. Also, we need to compute $\mathcal{B}^G$ corresponding to $\mathcal{A} \cup \{a\}$. However, since $f(\mathcal{A}, \mathcal{B}) = \max_{a \in \mathcal{A}} s(a, \mathcal{B})$, we need $\mathcal{O}\left(|\mathcal{E}| \cdot |\mathcal{V}|^2\right)$ times for getting $\mathcal{B}^G$. The finally computational complexity of Algorithm 2 is $\mathcal{O}\left(|\mathcal{E}|^3 \cdot |\mathcal{V}|^2\right)$. □

4.4 Numerical Evaluation

In this section, we will demonstrate the performance of the intermittent deployment problem. We will also demonstrate the performance of Algorithm 2 by using the combination of the task allocation problem and the intermittent deployment problem as a coupled example.

4.4.1 Simulation Setup

General Settings: The general setting is as follows. There is a 2D Gaussian mixture (GMM) environment that needs to be monitored. The task allocation problem and the intermittent deployment problem will operate in this environment in the time order. Different task allocation strategies will have different impacts on the environment, which leads to different initial conditions for the intermittent deployment problem.

The Task Allocation Problem: For this problem, we use the modular objective function $g(\mathcal{A}) = \sum_{a\in\mathcal{A}} u_a$, where $a = (r_1, d, e)$ is an assignment and u_a is the reward for robot $r_1 \in \mathcal{R}_1$ finishing the task $(d, e) \in \mathcal{O}$. For comparison, we set the parameters as follows to make sure that we can get the optimal solution. Specifically, we set the number of robot as $|\mathcal{R}_1| \in \{2, \ldots, 6\}$, the requirements cardinality as $|\mathcal{D}| \in \{2, \ldots, 6\}$, and functionality cardinality as $|\mathcal{E}| \in \{2, \ldots, 6\}$. In simulation, we use these parameters to generate a random problem instance as our ground set $\mathcal{E}$. The reward u_a is generated randomly for all $a \in \mathcal{E}$ before conducting the simulations. We use the independence constraint $\mathcal{M}_{11}$ and the uniqueness constraint $\mathcal{M}_{12}$ as the constraints. The problem size of this sub-problem is defined as $|\mathcal{S}_1| = |\mathcal{R}_1| \cdot |\mathcal{D}| \cdot |\mathcal{E}|$.

The Intermittent Deployment Problem: For this problem, we have $|\mathcal{R}_2|$ robots available at each time k, $k = 1, \ldots, K$, for monitoring this GMM environment along the time horizon K. The evolution of the weights of the GMM is modeled as a linear system. That is $x_{k+1} = Ax_k + w_k$, where $x_k \in \mathbb{R}^p$ is the state of the GMM weight and $w_k \in \mathbb{R}^p$ is the zero-mean Gaussian noise. p is the dimension of the GMM that needs to be estimated. The measurement model at time k is $y_{k+1} = C_k x_k + z_k$, where $z_k \in \mathbb{R}^q$ is the zero-mean Gaussian noise with noise covariance $Z_k \in \mathbb{R}^{q\times q}$. Different robots have different measurement abilities, which forms different measurement matrices C_k. Now, the intermittent deployment problem becomes

how to select robots from $\mathcal{R}_2$ to form the measurement matrix $C_k \in \mathbb{R}^{q \times p}$ at each time k to maximize the objective function $f(\mathcal{A}, \mathcal{B})$. The robots should satisfy the constraints in Section 4.2.2. The objective function is a combination of covariance reduction and the reward. We use the objective function from [52] since it has proven to be sequence submodular under assumptions which will be stated in the following. Because $f(A, B) = \max_{a \in A} s(a, \mathcal{B})$, we only need $s(a, \mathcal{B})$. Specifically, $s(a, \mathcal{B}) = \log(\det(P_1(a))/\det(P_K(a))) + \sum_{b \in \mathcal{B}} u_b$, where $P_1(a)$ is the GMM weight covariance at time $k = 1$. $P_1(a)$ is the crucial connection between the task allocation problem and the intermittent deployment problem. This is because different task allocations result in different $P_1(a)$ and $P_1(a)$ is also the initial condition for the intermittent deployment problem. $P_K(a)$ is the GMM weight covariance at time K. That is $P_K(a) = (A^{-\top})^K P_1(a) A^{-K} + \sum_{k=1}^{K} (A^{-\top})^{K-k} M_t A^{-(K-k)}$ [52] where $M_k = C_k^\top Z_k^{-1} C_k$. u_b is the reward associated with robot b. The assumptions for $s(a, \mathcal{B})$ to be sequence submodular is that [52] the state transition matrix A is full rank and $w_t = 0$. Also, A needs to satisfy $AP_1(a)A^\top \preceq P_1(a)$ and $A^\top M_k A \preceq M_k$. In the simulation, we use $A = I_p$. In each simulation, both $P_1(a)$ and the reward u_b are also generated randomly for each $b \in \mathcal{V}$ before starting the simulations. The dimension p is chosen from the set $\{2, \ldots, 5\}$. We set the time horizon as $K = \{2, \ldots, 5\}$ and the number of robots as $|\mathcal{R}_2| = \{2, \ldots, 4\}$. We then run the simulation and greedily select robots in each time. Also, the problem size of this sub-problem is defined as $|\mathcal{S}_2| = |\mathcal{R}_2| \cdot K$.

4.4.2 Performance Comparison

1) The Intermittent Deployment Problem Performance:

We first only evaluate the performance of the intermittent deployment problem. To evaluate, we need to know A from the task allocation to get the covariance matrix P_a. Here, we use a

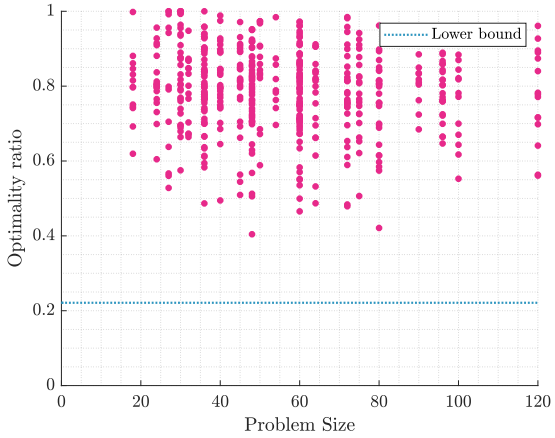


Fig. 4.2. The optimality ratio for the intermittent deployment problem. The problem size is $|\mathcal{S}_2| = |\mathcal{R}_2| \cdot K$.

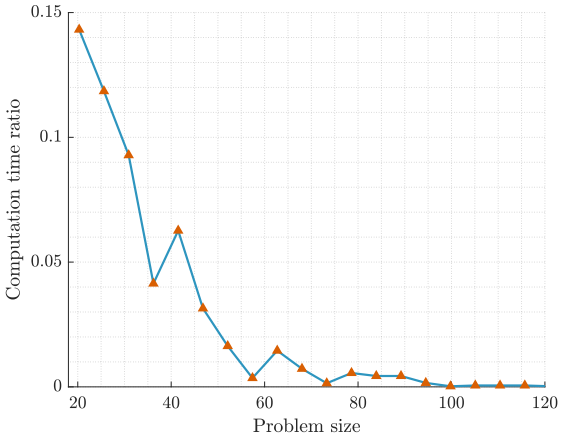


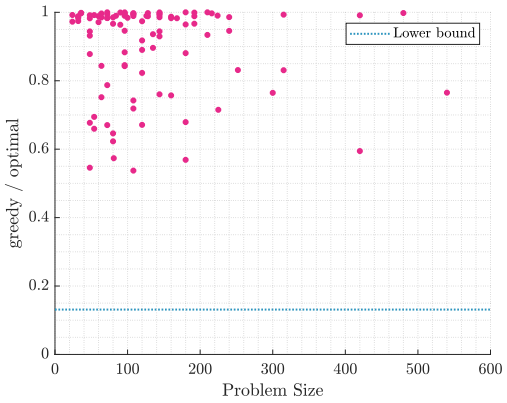
Fig. 4.3. The computation time ratio of the intermittent deployment problem. The problem size is $|\mathcal{S}_2| = |\mathcal{R}_2| \cdot K$.

random covariance matrix P_a as an initial condition. Then, the optimality ratio of the greedy algorithm is shown in Fig. 4.2. Also, we define the computation time ratio as $t(\mathcal{B}^G)/t(\mathcal{B}^\star)$, where $t(\mathcal{G}^G)$ is the time for computing the greedy solution and $t(\mathcal{B}^\star)$ is the time for computing an optimal solution. We then compute the average computation time ratio for each problem size as shown in Fig. 4.3. We see that the greedy method becomes more efficient as the problem size increases.

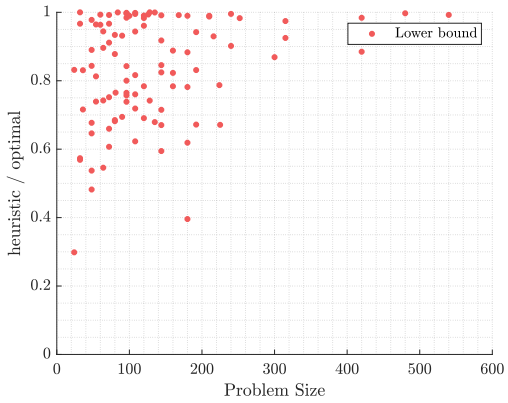
2) The Coupled Problem Performance:

Comparison criterion: We evaluate the performance of the coupled problem. Specifically, we compare the result from the proposed greedy solution with an optimal solution, a heuristic solution, and a random solution. The optimal solution is calculated through the brute force method. A heuristic to solve a coupled problem is to solve each problem separately. We generate the heuristic result using this manner. The random solution is used to demonstrate the effectiveness of the greedy method. The simulation runs 500 times.

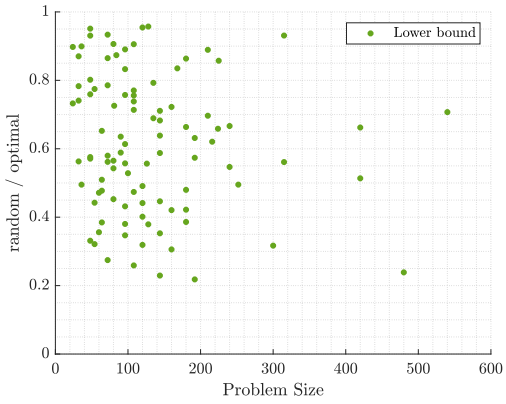
Optimality bound: To characterize the sub-optimality of the proposed greedy algorithm, we calculate the optimality ratio as $m(\mathcal{A}^G)/m(\mathcal{A}^\star)$. In the simulation, we evaluate the independence matroid constraint $\mathcal{M}_{11}$ and the uniqueness matroid constraint $\mathcal{M}_{12}$ for the multi-robot task allocation problem. For the multi-robot intermittent deployment problem, we use the constraint $\mathcal{M}_{23}$. As shown in Fig. 4.4(a), we observe that the proposed algorithm can generate better optimality ratios than the lower bound in most instances. At the same time, we also plot the result from the heuristic solution in Fig. 4.4(b) and random solution in Fig. 4.4(c). Further, it occurs often that the optimal solution is found in many instances for the greedy method. To make the comparison more clear, we also calculate the statistics of the optimality ratios of different methods. We calculate the mean and covariance of the optimality ratios for the above three methods. As shown in Table 4.1, the greedy method has a better performance on average. For the coupled problem, we define the problem size as



(a) Optimality ratio: greedy / optimal.



(b) Optimality ratio: heuristic / optimal.



(c) Optimality ratio: random / optimal.

Fig. 4.4. Monte Carlo simulation performance comparisons: (a) the optimality ratio for the greedy method. (b) the optimality ratio for the heuristic method. (c) the optimality ratio for the random method. The problem size of this coupled problem is $|\mathcal{S}_1| \cdot |\mathcal{S}_2|$.

TABLE 4.1
Statistics of the Optimality Ratios of Different Methods

method	mean	covariance
greedy	*0.89*	*0.018*
heuristic	0.83	0.024
random	0.61	0.040

$|\mathcal{S}_1| \cdot |\mathcal{S}_2|$. Because the complexity of coupled problems increases exponentially as the ground sets size increase, we also observe that even with moderate settings the problem size goes up to 600, necessitating efficient methods like those proposed in this work.

4.5 Conclusions

In this chapter, we presented a method for optimizing coupled problems by using submodular optimization. We demonstrated how to solve the proposed coupled problem by using the task allocation problem and the intermittent deployment problem as motivating examples. From the constraint perspective, we illustrated how to model general constraints as matroid constraints. From the objective function perspective, we demonstrated under which conditions the objective function is (sub)modular, which indicates the existence of an effective greedy algorithm. At the same, we analyzed the performance and computational complexity of our algorithm. In the end, Monte Carlo simulations demonstrated the effectiveness of the proposed algorithm.

Algorithm 2 The Greedy Method for Solving the Coupled Problem

Input: The inputs are as follows:

- The matroid constraints $\mathcal{M}_1$ and $\mathcal{M}_2$;
- The functions $g(\cdot)$ and $s(\cdot)$.

Output: Set $\mathcal{A}^G$ and set $\mathcal{B}^G$.

$\mathcal{A} \leftarrow \emptyset$;
for $i \leftarrow 0, \ldots, |\mathcal{E}| - 1$ **do** ▷ *step i*
  $\mathcal{C} \leftarrow \emptyset$;
  for $\forall\, a \in \mathcal{E} \setminus \mathcal{A}$ and $\mathcal{A} \cup \{a\} \in \mathcal{I}_1$ **do**
    $\mathcal{B} \leftarrow \emptyset$; $\mathcal{V}' \leftarrow \emptyset$;
    for $j \leftarrow 0, \ldots, |\mathcal{V}| - 1$ **do** ▷ *step j*
      $b' \leftarrow \arg\max_{b \in \mathcal{V} \setminus \mathcal{V}'} f(\mathcal{A} \cup \{a\}, \mathcal{B} \cup \{b\})$;
      if $\mathcal{B} \cup \{b'\} \in \mathcal{I}_2$ **then**
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{b'\}$;
      end if
      $\mathcal{V}' \leftarrow \mathcal{V}' \cup \{b'\}$;
    end for
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(a, \mathcal{B})\}$;
  end for
  if $\mathcal{C} = \emptyset$ **then** ▷ *no valid* $\mathcal{C}$
    break;
  end if
  $d(\mathcal{A} \cup \{a\}, \mathcal{B}) \leftarrow g(\mathcal{A} \cup \{a\}) + f(\mathcal{A} \cup \{a\}, \mathcal{B})$;
  $(a', \mathcal{B}^G) \leftarrow \arg\max_{\{(a,\mathcal{B})\} \in \mathcal{C}} d(\mathcal{A} \cup \{a\}, \mathcal{B})$;
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{a'\}$;
end for
$\mathcal{A}^G \leftarrow \mathcal{A}$.

Chapter 5

A Fast Algorithm for Robust Action Selection in Multi-Agent Systems

In multi-agent systems, action selection problems have been used in various applications, including task assignment [22], path planning [61, 62], coverage [6], environmental monitoring [1], rigidity evaluation [63], sensor selection [3], etc. In general, problems of this type are difficult combinatorial optimization problems, and recent work in this area has focused extensively on (non-)submodular optimization algorithms, which have proven tremendously useful as they offer performance guarantees coupled with efficient greedy selection.

In the general setting of submodular optimization, this chapter focuses on a scenario where a team of agents can select actions from a common ground set to fulfill a system-level objective. By using multiple agents for a single collaborative task, the system's redundancy can be increased to deal with possible attacks or failures. For example, in a collaborative task assignment problem, a team of agents (e.g., robots) may need to move an object from one location to another by applying different skills. If one agent is attacked, and the attack is a worst-case (the best possible attack), we desire a system that can continue operation with minimal impact on performance. By worst-case attack, we refer to the case when the attacker knows the best selection of the system and tries to remove the contribution from the agent with the maximum contribution. Here, the contribution is quantified by each agent's individual objective function value. To protect against such attacks, we must ensure that the

minimum contribution from any agent is maximized. In such a case, when the above attack occurs we can guarantee our system's performance. This chapter will tackle the problem by constructing an appropriate submodular function from the system objective function and providing a fast and tunable algorithm for finding *robust* action selections.

To provide context for our work and argue novelty, we now provide a review of relevant efforts in submodular optimization. Generally, submodular functions are a subset of set functions with a diminishing returns-like property. This property can be used to model many reward functions as the marginal reward of a system often decreases as the system makes more selections. For example, in the well-known sensor placement problem [9], the authors utilized the submodularity of the objective function, i.e., mutual information, and proposed a greedy method to solve the problem with guarantees. In the multi-agent task allocation problem [22], the marginal reward of the system is generally reduced when more tasks are assigned. Similar applications in robotics can also be found in the coverage problem [6], orienteering problem [18], interaction planning [64], target tracking [65], precision agriculture monitoring [7], etc.

While submodular functions (and thus submodular optimization) have broad applications, it is also essential to ensure a system's optimization performance when attacked, removing some parts or the entire contribution from one or more agents. Such scenarios have attracted significant attention as of late, and several robust/resilient algorithms have been proposed to mitigate the impact of attacks. In [57], the authors considered a general action selection (or task allocation) scenario. They proposed an algorithm to protect the system when an attacker knows the system's solution and can remove part of the system's solution under a partition matroid constraint. In [6], we proposed a distributed version of that algorithm that can be used in situations where no central point of command is available. The proposed algorithm has the same performance guarantees as to the centralized algorithm. In [66], the

author considered the case where a multi-agent team must select actions from a common ground set but with different individual objective functions. This is also the scenario that we consider in this chapter. To protect the system against attacks, the authors proposed an algorithm by utilizing the submodularity of a surrogate function instead of the original objective function. The proposed algorithm works for the cardinality constraint scenario, and the final performance is guaranteed when the constraint is relaxed. However, under some budget-critical circumstances, this relaxation cannot be allowed. In [67], the authors investigated this problem under a special case where the marginal gains of different agents' objective functions have a relation. The proposed algorithm's performance is guaranteed and is a function of the proposed marginal gain ratio. In contrast, we investigate this problem without any additional assumptions and aim to increase the speed of calculating a solution, which is useful when it is time-consuming to evaluate objective functions, a typical case.

In summary, the contributions of this chapter are as follows.

- We propose a fast algorithm for solving the robust multi-agent action selection problem.
- We prove the proposed algorithm's performance and computational complexity.
- We demonstrate the performance of the proposed algorithm using an action selection application.

5.1 Problem Formulation

Agents, actions, and constraints: Consider a multi-agent system $\mathcal{A}$ with $|\mathcal{A}| = N$, and a common action set $\mathcal{V}$ with $|\mathcal{V}| = M$. Each agent $a \in \mathcal{A}$ can select one or more actions from the action set $\mathcal{V}$. There is a reward associated with an agent's action selection set $\mathcal{S}$, where $\mathcal{S} \subseteq \mathcal{V}$. We denote by $h_i : 2^{\mathcal{V}} \mapsto \mathbb{R}$ a submodular function as agent i's *individual*

objective function for all $i \in \mathcal{A}$. This setting happens when all $i \in \mathcal{A}$ need to cooperate to finish a system objective, and the system redundancy is increased when the system uses multiple agents. We use a matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ to model the system constraint. For example, such a matroid constraint can be used to model the budget constraint for the system. e.g., $|\mathcal{S} \cap \mathcal{V}| \leq z$, where $\mathcal{S} \subseteq \mathcal{V}$ is a problem solution and $z \in \mathbb{R}$ is the system budget. Importantly, we make no assumptions about the type of matroid constraint used.

Attack: In this multi-agent system, where each agent $i \in \mathcal{A}$ can be deployed to fulfill our system objective, we want to make sure that the system's performance is guaranteed if an attacker attacks any agent by removing its contribution. In such a case, we want to ensure that the minimum performance of any individual agent's performance $h_i(\mathcal{S})$ is maximized. That is, the minimum performance of the system, $\min_i h_i(\mathcal{S})$, needs to be maximized. Considering this, we formulate the problem as follows.

Problem 2 (Robust action selection in multi-agent systems)**.** *Consider a multi-agent agent system $\mathcal{A}$ with N agents selecting actions from a common action set $\mathcal{V}$. Each agent has an individual objective function $h_i : 2^{\mathcal{V}} \mapsto \mathbb{R}$ for selecting actions from the action ground set $\mathcal{V}$. The system's constraint is a matroid constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$. The problem is formulated as*

$$\begin{aligned} \underset{\mathcal{S} \subseteq \mathcal{V}}{\textit{maximize}} \quad & \min_i h_i(\mathcal{S}), \\ \textit{subject to} \quad & \mathcal{S} \in \mathcal{I}. \end{aligned}$$

where $i = \{1, \ldots, N\}$, $|\mathcal{V}| = M$, and $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is matroid constraint of the multi-agent system.

In this problem formulation, different agents $a \in \mathcal{A}$ need to select a common action set from

$\mathcal{V}$ to fulfill a system objective to remain robust. For notation convenience, we denote by

$$g(\mathcal{S}) = \min_i h_i(\mathcal{S}), \quad \forall \mathcal{S} \subseteq \mathcal{V}$$

as the system objective function, and refer to $h_i(\mathcal{S})$ as the *individual* objective function for agent i. The system needs to ensure a performance guarantee if an attacker attacks any agent when knowing the system's common solution $\mathcal{S}$. On the contrary, if we do not require that different agents select a common action set to optimize a system objective, the problem becomes N independent optimization problems. An attacker needs to know more about the system when an attack happens as different agents hold different actions, and the corresponding contributions are different.

The problem formulation describes a game between the system and an attacker that can attack one agent. The system wants to select a common action set $\mathcal{S}$ for all agents to maximize any agent's performance, while an attacker wants to attack one of the agents' contributions in the worst case. Our mission is to ensure the system's performance if the attacker attacks the system in a worst-case scenario.

5.2 A Fast Algorithm for Robust Selection

In general, since the objective function $g(\mathcal{S}) = \min_i h_i(\cdot)$ is a set function and has no obvious property that we can utilize, we resort to the use of a surrogate function with a convenient property that can help service our maximizing purpose. Initially, this surrogate function idea is from [66] when the constraint is relaxed to yield a provable performance bound. In [68], the authors adopt the same idea and use it in the case when the constraint is a general matroid constraint instead of a cardinality constraint. In this paper, we use the same idea from the

previous two works and replace the procedure of finding the solution with a fast algorithm.

The surrogate function $f(\mathcal{S})$ that can be used for replacing the objective function $g(\mathcal{S}) = \min_i h_i(\cdot)$ is as follows:

$$f(\mathcal{S}) \leftarrow \frac{1}{N} \sum_{i=1}^{N} \min\left\{h_i(\mathcal{S}), \gamma\right\},$$

where N is the number of agents or the number of objective functions, and γ is used as the upper bound for the associated reward of different agents. It can be proven that the surrogate function $f(\mathcal{S})$ is submodular [69].

The proposed fast algorithm for solving Problem 2 is shown in Algorithm 3. First, we set an upper and lower bound for our surrogate function $f(\cdot)$. The lower bound ℓ is set to 0, while the upper bound is set to $f(\mathcal{V})$ as shown in Line 1. The upper bound equals the function value when all actions are selected, which is the maximum performance that the system can obtain.

Since the surrogate function $f(\cdot)$ is a monotone non-decreasing function, we can use a binary search method to reduce the gap between the upper bound u and the lower bound ℓ. First, we set the median of our surrogate function $f(\cdot)$ as $\gamma \leftarrow u + \ell$ as shown in Line 4. Then, we use this median γ as a temporary upper bound of the individual objective function $h_i(\mathcal{S})$ for different agents. Meanwhile, the corresponding surrogate function $f(\cdot)$ needs to be updated as $f(\mathcal{S}) \leftarrow \frac{1}{N} \sum_{i=1}^{N} \min\left\{h_i(\mathcal{S}), \gamma\right\}$. Based on the upper bound γ, we will then need to find an action set $\mathcal{S}$ that approximately maximizes the surrogate function $f(\cdot)$ as maximizing a submodular function $f(\cdot)$ with a matroid constraint is NP-hard [8].

In Line 7 to Line 9, we depict a fast method for maximizing a submodular function under a matroid constraint. Conventionally, maximizing a submodular function is achieved through a

simple greedy method [23],

$$\mathcal{S} \leftarrow \mathcal{S} \cup \left\{ \underset{\mathcal{S}\cup\{e\}\in\mathcal{I}}{\arg\max}\, f(e \mid \mathcal{S}) \right\}, \quad \forall\, e \in \mathcal{V} \setminus \mathcal{S}.$$

This method aims to build a solution set $\mathcal{S}$ by adding the element $e \in \mathcal{V} \setminus \mathcal{S}$ with the maximum marginal gain in each iteration. The computational complexity of this method is $\mathcal{O}(M^2)$ when the size of the action ground set is $|\mathcal{V}| = M$. However, when the size of the ground set is large, or it is time-consuming to evaluate the function $f(\cdot)$, we need a faster way to generate the final solution set. In general, our method can be summarized as follows:

$$\mathcal{S} \leftarrow \mathcal{S} \cup \left\{ e \mid f(e \mid \mathcal{S}) \geq \Delta, \mathcal{S} \cup e \in \mathcal{I} \right\}, \quad \forall\, e \in \mathcal{V} \setminus \mathcal{S}.$$

The parameter Δ is a lower bound for deciding whether to add $e \in \mathcal{V} \setminus \mathcal{S}$ to the solution $\mathcal{S}$ or not based on the current marginal gain $f(e \mid \mathcal{S})$. Note that the basic problem constraint $\mathcal{S} \cup \{e\} \in \mathcal{I}$ should also be followed in each iteration.

Specifically, from Line 7 to Line 9, we first set a lower bound for the marginal gain $f(e \mid \mathcal{S})$ based on the current system solution $\mathcal{S}$. If both this lower bound and the system matroid constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ are satisfied, we can add this e to the current solution set $\mathcal{S}$ as shown in Line 8. After all available $e \in \mathcal{V} \setminus \mathcal{S}$ are added to $\mathcal{S}$, we then update the marginal gain lower bound Δ as

$$\Delta \leftarrow \frac{\Delta}{1+\delta}.$$

This iterative process terminates when all $e \in \mathcal{V}$ are selected or when the lower bound of the marginal gain $f(e \mid \mathcal{S})$ achieves its predefined minimum value δF. Using this method, we do not need to calculate the maximum value in each iteration. Instead, we can add all elements that satisfy our requirements into the current solution set $\mathcal{S}$, making the algorithm faster.

Algorithm 3 A Fast Algorithm for Robust Selection

Input: The inputs are as follows:

- The individual objective function $h_i(\cdot)$;
- The action ground set $\mathcal{V}$.

Output: Set $\mathcal{S}^{\mathrm{G}}$.

1: $\ell \leftarrow 0, u \leftarrow \min_i h_i(\mathcal{V})$;
2: **while** $|u - \ell| > \epsilon$ **do**
3: $\quad \mathcal{S} \leftarrow \emptyset$;
4: $\quad \gamma \leftarrow \frac{1}{2}(u + \ell)$;
5: $\quad F \leftarrow \max_{i \in \mathcal{V}} f(i)$;
6:
7: $\quad$ **for** $\Delta = F, e \in \mathcal{V} \setminus \mathcal{S}; \Delta \geq \delta F; \Delta \leftarrow \frac{\Delta}{1+\delta}$ **do**
8: $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cup \{e \mid f(e \mid \mathcal{S}) \geq \Delta, \mathcal{S} \cup e \in \mathcal{I}\}$;
9: $\quad$ **end for**
10:
11: $\quad$ **if** $f(\mathcal{S}) < \frac{\gamma}{1+c_f+\delta}$ **then**
12: $\quad\quad u \leftarrow \gamma$;
13: $\quad$ **else**
14: $\quad\quad \ell \leftarrow \gamma, \mathcal{S}^{\mathrm{G}} \leftarrow \mathcal{S}$;
15: $\quad$ **end if**
16: **end while**
17: **return** $\mathcal{S}^{\mathrm{G}}$.

When compared with the conventional greedy maximization method, which has a complexity of $\mathcal{O}(M^2)$, the method (from Line 7 to Line 9) has a worst case complexity of $\mathcal{O}(M \log(\delta^{-1}))$.

Finally, we need to check the surrogate function value $f(\mathcal{S})$ against our predefined lower bound $\frac{\gamma}{1+c_f+\delta}$, where c_f is the curvature of $f(\cdot)$. Then, we use a binary search method to update those two bounds. If the function value $f(\mathcal{S})$ is lower than the predefined bound, we update the upper bound using γ. On the contrary, if the objective function value $f(\mathcal{S})$ is larger than the predefined bound $\frac{\gamma}{1+c_f+\delta}$, we need to update the lower bound ℓ using γ, and store the current solution $\mathcal{S}$ as the best solution $\mathcal{S}^{\mathrm{G}}$ as shown in Line 14.

Theorem 8. *For any fixed γ, let $\mathcal{S}^\star$ be an optimal solution for maximizing the surrogate*

function $f(\cdot)$, and $\mathcal{S}_m$ be the solution in the mth iteration in the for loop as shown in Line 8, then we have the following

$$f(e \mid \mathcal{S}_m) \geq \frac{1}{1+\delta} f(o \mid \mathcal{S}_m), \quad \forall\, o \in \mathcal{S}^\star \setminus \mathcal{S}_m, e \in \mathcal{V} \setminus \mathcal{S}_m.$$

Proof. Denote by Δ_n the lower bound of the marginal gain in the nth for loop as shown in Line 8. Note that it is possible that $n \neq m$ as for any Δ_n it is possible that $\mathcal{S}$ is updated multiple times. We also denote by $\mathcal{S}_{m-1}$ the solution before the updating using the current lower bound Δ_n. For any $e \in \mathcal{V} \setminus \mathcal{S}_{m-1}$ that can be added to the approximation solution as $\mathcal{S}_m \leftarrow \mathcal{S}_{m-1} \cup e$, there are two cases to be considered: (1). e is the first selected element that satisfies the above two requirements; (2). e is the element other than the first one that satisfies the above two requirements.

In the first case, for any $e \in \mathcal{V} \setminus \mathcal{S}_{m-1}$ that can be added to $\mathcal{S}_{m-1}$, it should satisfy the following condition

$$f(e \mid \mathcal{S}_{m-1}) \geq \Delta_n, \quad \forall\, e \in \mathcal{V} \setminus \mathcal{S}_{m-1}. \tag{5.1}$$

This is the necessary condition for $e \in \mathcal{V} \setminus \mathcal{S}_{m-1}$ to be considered as a candidate for adding to $\mathcal{S}_{m-1}$, as shown in Line 8. Now, for any $o \in \mathcal{S}^\star \setminus \mathcal{S}_{m-1}$, we have

$$f(o \mid \mathcal{S}_{m-1}) \leq \Delta_{n-1}, \quad \forall\, o \in \mathcal{S}^\star \setminus \mathcal{S}_{m-1}.$$

That is because if $o \in \mathcal{S}_{m-1}$, we have $f(o \mid \mathcal{S}_{m-1}) = 0$. Then, the above statement is true. If $o \notin \mathcal{S}_{m-1}$, it means $f(o \mid \mathcal{S}_{m-1})$ cannot satisfy the lower bound condition Δ_{n-1} in the $(j-1)$th iteration. Also, we have the relationship between lower bounds in different iterations as $\Delta_n \leftarrow \frac{\Delta_{n-1}}{1+\delta}$. We then have

$$f(o \mid \mathcal{S}_{m-1}) \leq \Delta_{n-1} = (1+\delta)\Delta_n, \quad o \in \mathcal{S}^\star \setminus \mathcal{S}_{m-1}. \tag{5.2}$$

Combining (5.1) and (5.2), for any $o \in \mathcal{S}^\star \setminus \mathcal{S}_{m-1}$, we have the result for the first case.

$$f(e \mid \mathcal{S}_{m-1}) \geq \frac{1}{1+\delta} f(o \mid \mathcal{S}_{m-1}), \quad \forall\, e \in \mathcal{V} \setminus \mathcal{S}_{m-1}.$$

In the second case, we consider the case where e is not the first element that satisfies the current marginal gain lower bound Δ_n. Denote by $\mathcal{S}_m$ the solution after the first eligible element e added to $\mathcal{S}$. Without loss of generality, we assume $e \in \mathcal{V} \setminus \mathcal{S}_m$. Following the same reasoning above, for any $o \in \mathcal{S}^\star \setminus \mathcal{S}_m$, we then have

$$f(e \mid \mathcal{S}_m) \geq \frac{1}{1+\delta} f(o \mid \mathcal{S}_{m-1}), \quad \forall\, e \in \mathcal{V} \setminus \mathcal{S}_m. \tag{5.3}$$

At the same time, by utilizing the submodularity of the objective function $f(\cdot)$, we have

$$f(o \mid \mathcal{S}_{m-1}) \geq f(o \mid \mathcal{S}_m). \tag{5.4}$$

Combining (5.3) and (5.4), for any $o \in \mathcal{S}^\star \setminus \mathcal{S}_m$, we have the following result for the second case.

$$f(e \mid \mathcal{S}_m) \geq \frac{1}{1+\delta} f(o \mid \mathcal{S}_m), \quad \forall\, e \in \mathcal{V} \setminus \mathcal{S}_m.$$

Finally, we can complete the proof by reduction based on the above two cases. □

Theorem 9. *Let c_f be the curvature of function $f(\cdot)$. For any fixed γ, let $\mathcal{S}^\star$ be an optimal solution for maximizing the surrogate function $f(\cdot)$ and $\mathcal{S}$ be the generated output using the approximation method (Line 7 to Line 9), we then have the following.*

$$f(\mathcal{S}) \geq \frac{1}{1+c_f+\delta} f(\mathcal{S}^\star).$$

Proof. First, by utilizing the submodularity of function $f(\cdot)$, we have

$$f(\mathcal{S}^\star \cup \mathcal{S}) \leq f(\mathcal{S}) + \sum_{o \in \mathcal{S}^\star \setminus \mathcal{S}} f(o \mid \mathcal{S}).$$

Also, we assume that $|\mathcal{S}^\star| = K$. Then, we have

$$f(\mathcal{S}^\star \cup \mathcal{S}) \leq f(\mathcal{S}) + \sum_{i=1}^{K} f(o \mid \mathcal{S}_{m-1}), \forall\, o \in \mathcal{S}^\star \setminus \mathcal{S}_{m-1}.$$

By using the result from Theorem 8, we have

$$\begin{aligned} & f(\mathcal{S}^\star \cup \mathcal{S}) \\ \leq & f(\mathcal{S}) + (1+\delta) \sum_{i=1}^{K} f(e \mid \mathcal{S}_{m-1}), \forall\, e \in \mathcal{V} \setminus \mathcal{S}_{m-1} \\ \leq & f(\mathcal{S}) + (1+\delta) \sum_{i=1}^{K} f(\mathcal{S}_m \mid \mathcal{S}_{m-1}). \end{aligned} \tag{5.5}$$

At the same time, for the element $e \in \mathcal{S}_m \setminus \mathcal{S}_{m-1}$, we have

$$\begin{aligned} & f(\mathcal{S}^\star \cup \mathcal{S}) \\ = & f(\mathcal{S}^\star) + \sum_{i=1}^{K} \left[f(\mathcal{S}^\star \cup \mathcal{S}_m) - f(\mathcal{S}^\star \cup \mathcal{S}_{m-1}) \right] \\ = & f(\mathcal{S}^\star) + \sum_{i=1}^{K} \left[f(\mathcal{S}^\star \cup \mathcal{S}_{m-1} \cup e) - f(\mathcal{S}^\star \cup \mathcal{S}_{m-1}) \right]. \end{aligned}$$

By utilizing the definition of curvature for submodular functions, we then have

$$\begin{aligned} & f(\mathcal{S}^\star \cup \mathcal{S}_{m-1} \cup e) - f(\mathcal{S}^\star \cup \mathcal{S}_{m-1}) \\ \geq & f(\mathcal{S}_{m-1} \cup e) - f(\mathcal{S}^\star \cup \mathcal{S}_{m-1}). \end{aligned}$$

By combining the above two results, we get

$$f(\mathcal{S}^\star \cup \mathcal{S}) \geq \sum_{i=1}^{K} \left[f(\mathcal{S}_{m-1} \cup e) - f(\cup \mathcal{S}_{m-1}) \right] \tag{5.6}$$

Finally, combining (5.5) and (5.6), we get the final result.

$$f(\mathcal{S}) \geq \frac{1}{1 + c_f + \delta} f(\mathcal{S}^\star).$$

□

Theorem 10. *In Algorithm 3, let $\mathcal{S}^\star$ be an optimal solution for $f(\cdot)$ with respect to the final γ, and $\mathcal{S}^G$ be the solution of Algorithm 3. When the algorithm terminates, we have the following.*

$$\min_i h_i(\mathcal{S}^G) \geq \frac{1}{1 + c_f + \delta} \min_i h_i(\mathcal{S}^\star).$$

Proof. From Theorem 9, for a fixed γ, we know that $f(\mathcal{S}) \geq \frac{1}{1+c_f+\delta} f(\mathcal{S}^\star)$. Also, since $f(\mathcal{S}) \leftarrow \frac{1}{N} \sum_{i=1}^{N} \min\left\{h_i(\mathcal{S}), \gamma\right\}$, we then have the above result. □

5.3 Evaluation

To evaluate the performance of the proposed algorithm, we use a sensor proximity maximization problem [67]. The problem settings are as follows. The robots and the sensors are located in $\mathbb{R}^2$ with a size of 100×100. The locations of both the robots and the sensors are randomly selected, and we want to use sensors to cover the robots. The idea of sensor proximity is to use the sensing radius instead of the covered area to calculate the effectiveness of the selected sensors. Based on the robots' locations, we need to choose the locations of sensors to maximize our objective function.

We denote by $\mathcal{A}$ and $\mathcal{V}$ the robot set and the sensor ground set. The objective function can be formulated as

$$h_i(\mathcal{S}) = \max_{j \in \mathcal{S}} d_{ij},$$

where $i \in \mathcal{A}$ is the ith robot, and $j \in \mathcal{S}$ is the jth sensor. The 2D distance function $d : \mathbb{R}^2 \times \mathbb{R}^2 \mapsto \mathbb{R}$ is a Euclidean distance function in $\mathbb{R}^2$. Given any selected sensor set $\mathcal{S} \subseteq \mathcal{V}$, the objective value of ith robot is defined as the maximum distance between the robot i and all the sensors in the set $\mathcal{S}$. Meanwhile, this 100×100 region is divided into four sub-regions with lines $x = 50$ and $y = 50$. Therefore, the sensor ground set is a combination of four sub-ground sets as $\mathcal{V} = \{\mathcal{V}_{s1}, \mathcal{V}_{s2}, \mathcal{V}_{s3}, \mathcal{V}_{s4}\}$. There is an upper limit on the number of sensors that each robot can select in each sub-region. This upper limit serves as a budget for the system in each sub-region and can be formulated as a matroid constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$. Thus, the problem formulation is

$$\begin{aligned} \underset{\mathcal{S} \subseteq \mathcal{V}}{\text{maximize}} \quad & \min_i h_i(\mathcal{S}), \\ \text{subject to} \quad & \mathcal{S} \in \mathcal{I}. \end{aligned}$$

where $h_i(\mathcal{S}) = \max_{j \in \mathcal{S}} d_{ij}$ is the individual objective function for robot i. Specifically, the matroid constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ of this problem is a partition matroid [8] and can be written as

$$\mathcal{I} : |\mathcal{S} \cap \mathcal{V}_{sj}| \leq z, \quad j = 1, \ldots, 4,$$

where $\mathcal{V}_{sj}$ is the sub-ground set of the problem associated with the jth sub-region with $j = 1, \ldots, 4$.

Next, we conduct Monte Carlo evaluations to test the performance. Specifically, the number of robots is set to $|\mathcal{A}| = 5$, and the number of sensors is set to $|\mathcal{V}| = 50$. The locations of robots and sensors are randomly located. Therefore, the number of sensors in each sub-region

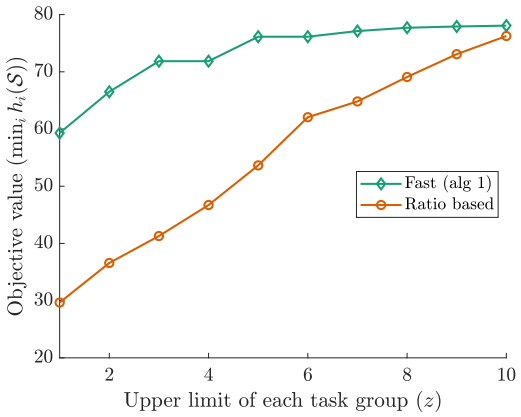


Fig. 5.1. The averaged objective value comparison between the proposed algorithm: "Fast (alg 1)" and the ratio based algorithm. Each averaged objective value is calculated based on 100 trials.

is not exactly the same. In the simulation, we change the upper limit z from 1 to 10. For each settled upper limit, we then run the simulation 100 times, where the locations of both the robots and the sensors are randomly generated for each simulation. The parameter c_f is set to 1, which is the upper bound. The tunable parameter δ is set to 10^{-3}. To test the performance, we compare the performance of the proposed Algorithm 3 with the current state-of-the-art algorithm from [67]. In that algorithm, the final solution is generated based on the ratios of the contributions of different sensors with respect to the maximum contribution of each robot's available selection. We refer to this method as the "Ratio based" method in the comparison. Meanwhile, we refer to the algorithm proposed in this paper as "Fast (alg 1)". Then, we compare the performance and the speed of the two different methods.

Since we run the simulation 100 trials for each random setting from $z = 1$ to $z = 10$, we calculate the averaged objective value to test the performance. From different z's, we get the averaged objective function value comparison, which is shown in Fig. 5.1. The result shows that the proposed algorithm performs much better when the limit z is set to lower values. Then, when the limit z approaches 10, we get almost identical performance.

Next, we compare the speed of the proposed method with the ratio-based method. Specifically,

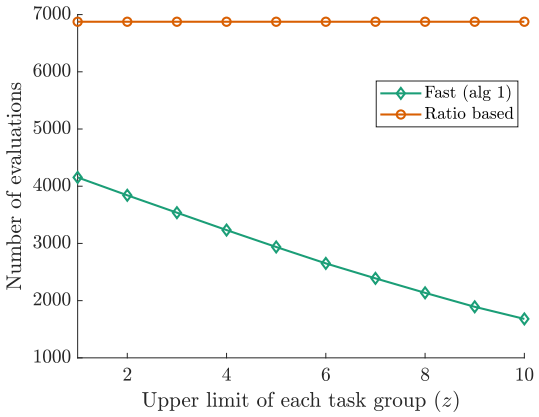


Fig. 5.2. The number of objective function evaluations comparison between the proposed algorithm: "Fast (alg 1)" and the ratio-based algorithm. Each averaged objective value is calculated based on 100 trials.

we compare the average number of objective function evaluations. For each upper limit setting z, we calculate the number of evaluations of $f(\cdot)$. Then, we have the speed comparison as shown in Fig. 5.2. This result indicates that the proposed fast method is significantly faster than the ratio-based method, even when z is set to a low value. Then, when we increase the upper limit z, we observe that the difference between the two speeds becomes more apparent. Meanwhile, when compared with the proposed method, we notice that the speed of the ratio-based method does not change too much when z is different. This is because we always need to find the sensor with the maximum contribution from the sensor ground set $\mathcal{V}$ in the corresponding method. This result shows the speed superiority of the proposed method.

5.4 Conclusions

This chapter proposed a fast algorithm for solving the robust action selection problem for multi-agent systems. Through the proposed method, we can significantly increase the algorithm's speed. The proposed algorithm can be used when it is time-consuming to calculate the objective function value or the size of the ground set is large. Finally, through Monte

Carlo simulation, we are able to evaluate the performance of the proposed algorithm through comparisons with a state-of-the-art algorithm.

Chapter 6

Distributed Resilient Action Selection in Adversarial Environments

Resilience is a crucial property for multi-robot systems. Consider, for example, a multi-robot exploration application where each robot selects exploration actions from an action candidate set, e.g., a motion primitive set. In adversarial environments, sensors may fail or get attacked, and depending on the contributions of the attacked sensors, the exploration performance may be seriously affected. This problem is more challenging in distributed multi-robot systems since each robot can only share its local information with neighbors to maximize the system reward subject to adversarial influences.

This chapter focuses on a scenario where the robots in a distributed multi-robot system need to work together to guard the system against worst-case attacks. By worst-case attacks, we refer to the case where the system may have up to K sensor failures. Robots operating in adversarial scenarios may get cyber-attacked or face failures, resulting in a temporary withdrawal of robots from the task (e.g., because of temporary deactivation of their sensors, blockage of their field of view). For example, in Fig. 6.1, each robot in the system is equipped with a downward-facing camera to explore an environment with different weights in different areas, where there is one robot whose *sensor* is blocked by an attacker. It is worth mentioning that robot failure and sensor failure are different. If a sensor is attacked, the corresponding robot may not know this attack and still perform other tasks/communications as planned.

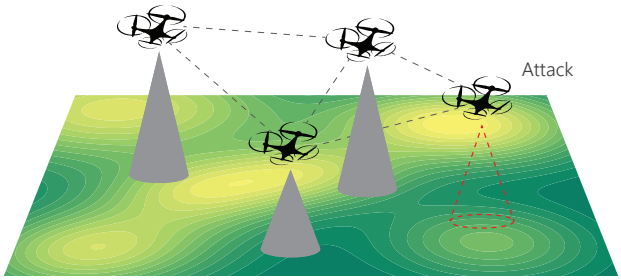


Fig. 6.1. In a multi-robot environmental exploration application, the robots are mounted with downward-facing cameras to explore the environment. An attacker blocks one of the robots' cameras (red).

Related Work

The resilience of multi-robot systems has received attention recently. In [70], the authors presented a resilient formation control algorithm that steers a team of mobile robots to achieve desired flocking even though some team members are non-cooperative (or adversarial) and broadcast deceptive signals. Deceptive or spoofing attacks were also considered in the wireless communication [71] and the target state estimation [72] of multi-robot teams. Another type of attack, called masquerade attack, was studied in a multi-agent path-finding problem [73]. In this chapter, we instead focus on defending multi-robot systems against the denial-of-service (DoS) attacks that can compromise the sensors' functionality [74]. For example, a polynomial-time resilient algorithm to counter adversarial denial-of-service (DoS) attacks or failures in a submodular maximization problem was proposed in [75]. Meanwhile, resilient coordination algorithms have been designed to cope with adversarial attacks in multi-robot target tracking [65], the orienteering problem [76], task assignment [77], etc. In [78], the authors proposed to solve the centralized resilient target tracking problem [65] in a distributed way. This method partitions robots into subgroups/cliques, and then the subgroups perform a centralized algorithm in parallel to counter the worst-case attacks. Thus, even if there exist communications between subgroups, these available communications are not utilized because each subgroup operates independently. Therefore, the proposed

algorithm in [78] has a worse approximation bound than its centralized counterpart.

The action selection problem falls into the combinatorial robotics application domain. The authors in [79] proposed a consensus-based method for the task allocation problem. In [10], the authors used matroids to model the task allocation constraints and provided a distributed approach with 1/2 optimality ratio. In [22], the authors extended the use of matroids to abstract task allocation constraints modeling and demonstrated the suboptimality through a sequential auction method in a decentralized scenario. In [1, 2], the authors applied submodular and matroids techniques in a multi-robot intermittent environmental monitoring problem, where the deployment actions are selected based on the environmental process. In [21], the authors utilized the submodularity of a mutual information function to prove the performance of a distributed multi-robot exploration method, while synchronization is needed. The authors in [4] considered two coupled action selection problems in an environmental monitoring application, where the selected tasks have an impact on the monitored environmental process behavior. In [80], the authors studied how the information from other robots impact the decisions of a multi-robot system in distributed settings. Similarly, the submodular properties were also utilized in the consensus problem [81], the leader selection problem [82], etc. However, resilience is not the primary consideration, especially when the system is under worst-case attacks. In this chapter, we propose a fully distributed resilient algorithm that requires no central point of command to solve the action selection problem in adversarial environments. The proposed distributed resilient method can perform as well as the corresponding centralized algorithm when subject to worst-case adversarial attacks.

Contributions

1. We formulate a fully distributed resilient submodular action selection problem.

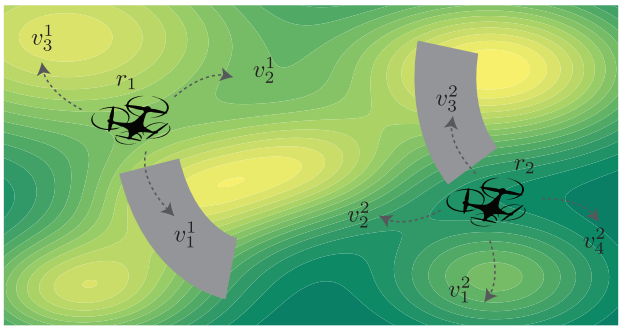


Fig. 6.2. Each robot chooses one motion primitive from its candidate motion primitive set to explore a region of the environment.

2. We demonstrate how to solve the problem in a *fully* distributed manner where each robot computes its action only and shares the decision with its neighbors to achieve convergence with performance guarantees.

3. We prove and evaluate the proposed algorithm's performance is consistent, as it is shown to converge to the same solution as a centralized method.

Chapter Outline

In Section 6.1, we introduce the problem formulation. In Section 6.2, we use two subsections to demonstrate the two phases of the proposed algorithm. Then, the performance analysis of the proposed algorithm is shown in Section 6.3. In Section 6.4, numerical evaluation is performed in a multi-robot exploration problem. We then close the chapter in Section 6.5.

6.1 The Distributed Resilient Action Selection Problem

Robots, actions, and rewards: Consider a team of N robots denoted by $\mathcal{R} = \{1, \ldots, N\}$. Each robot is equipped with one sensor. There is a (undirected) communication graph[1]

[1]It is worth noting that the proposed algorithm also works for directed communication graphs.

$\mathcal{G} = (\mathcal{R}, \mathcal{E})$ associated with nodes $\mathcal{R}$, and edges $\mathcal{E}$ such that $(i, j) \in \mathcal{E}$ if i and j can communicate with each other. We denote by $\mathcal{N}_i$ the neighbors of robot i. The diameter $d(\mathcal{G})$ of the communication $\mathcal{G}$ is the greatest length of the shortest paths between vertices. The communication can be synchronized or asynchronized. The performance analysis will be based on synchronized communication. Each robot $i \in \mathcal{R}$ has a set of candidate actions $\mathcal{V}_i$ and can only choose one action from $\mathcal{V}_i$ at each execution step. For example, in motion planning using motion primitives, the robot can only choose one motion primitive from its candidate motion primitives at a time. As shown in Fig. 6.2, robot 1 chooses action v_1^1 from its available action set $\mathcal{V}_1 = \left\{v_1^1, v_2^1, v_3^1\right\}$ and robot 2 chooses action v_3^2 from its available action set $\mathcal{V}_2 = \left\{v_1^2, v_2^2, v_3^2, v_4^2\right\}$, yielding the shaded explored area. We denote by $\mathcal{V} \triangleq \bigcup_{i \in \mathcal{R}} \mathcal{V}_i$ the ground set containing all robots' possible actions. There is an reward associated with any action. Also, the reward associated with action v_1^1 is the gray explored area. The function value $f(\mathcal{S})$ or the combined reward associated with v_1^1 and v_3^2 is the explored gray areas.

Objective Function: We use a non-decreasing and submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$ to model the quality of each valid action set $\mathcal{S} \subseteq \mathcal{V}$ since the diminishing return property of objective functions is common in robotics. For example, in Fig. 6.2, $f(\cdot)$ measures the extent of the joint area explored by chosen actions $\mathcal{S} = \left\{v_1^1, v_3^2\right\}$ (represented by the gray areas), which is a well-known coverage function that exhibits the submodularity property [9].

Assumption 1 (*Attacks*)**.** *We assume the robots encounter worst-case attacks that result in their sensor DoS failures. Thus, robots can still communicate with their neighbors even though their sensors are denied or blocked. The maximum number of anticipated attacks is upper bounded by $K, (K \leq N)$, where N is the number of robots.*

Problem 3 (*Distributed resilient multi-robot action selection in adversarial environments*)**.** *The robots, by communicating actions and rewards with their neighbors over the communication graph $\mathcal{G}$, choose action set $\mathcal{S}$ (per robot per action) to maximize a submodular objective $f(\cdot)$*

against K worst-case attacks. That is

$$\begin{aligned} \underset{\mathcal{S}\subseteq\mathcal{V}}{\textit{maximize}} \quad & \min_{\mathcal{F}\subseteq\mathcal{S}} f(\mathcal{S}\setminus\mathcal{F}) \\ \textit{subject to} \quad & |\mathcal{S}\cap\mathcal{V}_i| = 1, \forall\, i\in\mathcal{R}, \\ & |\mathcal{F}| \leq K, \end{aligned} \tag{6.1}$$

where $\mathcal{R}$ contains the indexes of the robots in the system, $\mathcal{F}$ denotes the action set associated with the attacked sensors, and $\mathcal{V}_i$ is the available action set for robot i.

The first constraint ensures that robot i only chooses one action from its action set $\mathcal{V}_i$. The "min" operator indicates the attacks we consider are the worst-case attacks. The constraint $|\mathcal{F}| \leq K$ captures the problem assumption that at most K sensors in the team can fail or get attacked.

In this problem, each robot needs to take other robots' actions into consideration while making its decision. That is because neighboring robots' selected actions may have an impact on local robot's action selection. In other words, different action selections result in different performances.

6.2 A Consistent Algorithm for Distributed Resilient Submodular Maximization

We present a distributed resilient algorithm (Algorithm 4) for solving Problem 1. At a high level, Algorithm 4 contains two main procedures GenerateRemovals (Algorithm 5) and GenerateComplements (Algorithm 6). In the following, we present and analyze these procedures from robot i's perspective since other robots will follow the same procedures. In

Algorithm 4 Distributed Resilient Selection for Robot i

Input:

- Action set $\mathcal{V}_i$; number of anticipated attacks K;
- Communication graph $\mathcal{G}$; objective function $f(\cdot)$.

Output: Set $\mathcal{S}$.

1: $\mathcal{S}_1^i \leftarrow \emptyset, \mathcal{S}_2^i \leftarrow \emptyset, \alpha_1^i \leftarrow 0, \alpha_2^i \leftarrow 0$;
2: $\mathcal{S}_1^i \leftarrow$GenerateRemovals$(\mathcal{S}_1^i, \alpha_1^i)$;
3: $\mathcal{S}_2^i \leftarrow$GenerateComplements$(\mathcal{S}_1^i, \mathcal{S}_2^i, \alpha_2^i)$;
4: $\mathcal{S} \leftarrow \mathcal{S}_1^i \cup \mathcal{S}_2^i$.

general, robot i will use these two procedures to approximate the following two sets:

- $\mathcal{S}_1^i$: the set that *approximates* the optimal worst-case removal set. Since computing the optimal worst-case removal set is intractable, we use $\mathcal{S}_1^i$ as an approximation. We denote by $\mathcal{A}$ the indices of the robots used by $\mathcal{S}_1^i$. This is the phase I.
- $\mathcal{S}_2^i$: the set that *approximates* the optimal set that maximizes the objective function using $\mathcal{V} \setminus \mathcal{V}_i, \forall\, i \in \mathcal{A}$. Again, this is an approximation since computing the optimal set is intractable. This is phase II.

These two procedures will be executed sequentially. Robot i will use α_1^i and α_2^i as two counters for different phases to decide whether to stop the corresponding procedure or not. Upon the stopping of Algorithm 4, both $\mathcal{S}_1^i$ and $\mathcal{S}_2^i$ converge. The final solution of Problem 1 will then be $\mathcal{S}_1^i \cup \mathcal{S}_2^i$.

In each phase, robot i will approximate and update $\mathcal{S}_1^i$ and $\mathcal{S}_2^i$ through the following processes:

1. *Initialization*, which is used to make the first approximation.
2. *Inter-robot communication*, which is used to combine its local approximation with neighbors' approximations.

Algorithm 5 (Phase I) Generate Approximated Removal Set for Each Robot i

procedure GenerateRemovals($\mathcal{S}_1^i, \alpha_1^i$)
  while $\alpha_1^i < 2d(\mathcal{G})$ **do**
    if $\mathcal{S}_1^i = \emptyset$ **then** ▷ *1) Initialization*
      $\mathcal{S}_1^i \leftarrow \arg\max_{v \in \mathcal{V}_i} f(v)$;
      $f(s) \leftarrow \max_{v \in \mathcal{V}_i} f(v)$;
    end if

    $\mathcal{S}_1^i \leftarrow \mathcal{S}_1^i \cup \mathcal{S}_1^j, \forall\, j \in \mathcal{N}_i$; ▷ *2) Communication*
    $M = \min(K, |\mathcal{S}_1^i|)$; ▷ *3) Local computation*
    $\mathcal{S}_1^i \leftarrow$ top M actions [2] in $\mathcal{S}_1^i$;
    send $(\mathcal{S}_1^i, \{f(s)\}), \forall\, s \in \mathcal{S}_1^i$ to all $j \in \mathcal{N}_i$;
    update α_1^i.
  end while
end procedure

3. *Local computation*, which is used to update local approximation.

6.2.1 Phase I: Generate Approximated Removals

The procedure for generating approximated removals is called GenerateRemovals (Algorithm 5). This procedure aims to approximate K action removals $\mathcal{S}_1$ ($|\mathcal{S}_1| = K$) through the below processes.

1) Initialization: In Algorithm 5, robot i first selects an action that contributes the most to the objective function f regardless of other robots' selections. The selected action is $s \in \arg\max_{v \in \mathcal{V}_i} f(v)$. Following the constraint $|\mathcal{S} \cap \mathcal{V}_i| = 1, \forall\, i \in \mathcal{R}$, robot i is only allowed to select one action from its candidate action set $\mathcal{V}_i$ to update its action set $\mathcal{S}_1^i$. Meanwhile, $f(s)$ is also recorded.

2) Inter-robot communication: To update i's local approximation set $\mathcal{S}_1^i$, robot i needs to combine j's approximation $\mathcal{S}_1^j, \forall\, j \in \mathcal{N}_i$. Since our task in this phase is to approximate K action removals, we can merge j's approximation as $\mathcal{S}_1^i \leftarrow \mathcal{S}_1^i \cup \mathcal{S}_1^j$.

3) Local computation: Once receiving neighbor j's action set $\mathcal{S}_1^j$, robot i updates its action set $\mathcal{S}_1^i$ based on $\mathcal{S}_1^j$. We first form a new candidate set $\mathcal{S}_1^i \leftarrow \mathcal{S}_1^i \cup \mathcal{S}_1^j$ to update $\mathcal{S}_1^i$. Then, we need to select the top K actions for robot i. There are two cases:

- If $|\mathcal{S}_1^i| \leq K$, there is no need to update $\mathcal{S}_1^i$.
- If $|\mathcal{S}_1^i| > K$, robot i selects the top K actions from $\mathcal{S}_1^i$.

In Algorithm 5 Lines 9 to 10, we combine these two cases as a single operation. That is, robot i needs to select top $m := \min\left(K, |\mathcal{S}_1^i|\right)$ actions from $\mathcal{S}_1^i$. Finally, robot i shares $\mathcal{S}_1^i$ and the corresponding action values $f(s), \forall\, s \in \mathcal{S}_1^i$ with all its neighbors $j \in \mathcal{N}_i$.

4) Stopping condition: After one cycle of local computation and inter-robot communication, the local counter α_1^i will be incremented by 1. Finally, when α_1^i reaches $2d(\mathcal{G})$, robot i stops and all robots have an agreement on $\mathcal{S}_1$.

6.2.2 Phase II: Generate Approximated Complements

The procedure for generating the approximated complements is GenerateComplements shown in Algorithm 6. This procedure aims to approximate $N - K$ greedy action selections $\mathcal{S}_2$ $\left(|\mathcal{S}_2| = |\mathcal{R} \setminus \mathcal{A}| = N - K\right)$ for the remaining robots $\mathcal{R} \setminus \mathcal{A}$ through inter-robot communication and local computation with $\mathcal{A}$ denoting the robots that select actions in phase I. Depending on whether robot i is used as removals or not, robot i in phase II will have two different functionalities:

- If $i \in \mathcal{A}$, then robot i acts as a conveyor only to merge the approximation $\mathcal{S}_2^j$ from $j, \forall\, j \in \mathcal{N}_i$ and broadcast the merged/updated $\mathcal{S}_2^i$ to $j, \forall\, j \in \mathcal{N}_i$. So, robot i only participated in *inter-robot communication*.

Algorithm 6 (Phase II) Generate Complements for Robot i

procedure GenerateComplements($\mathcal{S}_1^i, \mathcal{S}_2^i, \alpha_2^i$)
  while $\alpha_2^i < 2d(\mathcal{G})$ **do**
    if $\mathcal{S}_2^i = \emptyset$ **then** ▷ *1) initialization*
      $s \in \arg\max_{v \in \mathcal{V}_i} f_\emptyset(v)$;
      $\mathcal{S}_2^i \leftarrow \{s\}$;
    end if

    for $j \in \mathcal{N}_i$ **do** ▷ *2) inter communication*
      $\mathcal{S}_2^{i+} \leftarrow \mathsf{sort}(\{\mathcal{S}_2^i, \mathcal{S}_2^j\}, \text{'descend'})$;
      $\mathcal{S}_2^{i+} \leftarrow$ remove redundant actions in $\mathcal{S}_2^{i+}$;
      $\mathcal{S}_2^{i+} \leftarrow$ remove order changed actions in $\mathcal{S}_2^{i+}$;
      $\mathcal{S}_2^i \leftarrow \mathcal{S}_2^{i+}$;
    end for

    $\mathcal{X} \leftarrow \emptyset$; ▷ *3) local computation*
    for $s_n \in \mathcal{S}_2^i, n = 1, \ldots, |\mathcal{S}_2^i|$ **do**
      $g \leftarrow f_{\{s_1,\ldots,s_{n-1}\}}(s_n)$;
      if $f_{\mathcal{X}}(\mathcal{X} \cup v) \geq g, \forall\, v \in \mathcal{V}_i$ **then**
        $s \in \arg\max_{v \in \mathcal{V}_i} f_{\mathcal{X}}(\mathcal{X} \cup v)$;
        $\mathcal{X} \leftarrow \mathcal{X} \cup \{s\}$;
        break;
      end if
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{s_n\}$.
    end for
    $\mathcal{S}_2^i \leftarrow \mathcal{X}$;

    send $\mathcal{S}_2^i$ and marginal gains of $s \in \mathcal{S}_2^i$ to $\mathcal{N}_i$;
    update α_2^i.
  end while
end procedure

- If $i \in \mathcal{R} \backslash \mathcal{A}$, robot i also needs to update its approximation $\mathcal{S}_2^i$ using the *local computation* process.

In the following, we demonstrate phase II from robot i's perspective assuming $i \in \mathcal{R} \setminus \mathcal{A}$. If $i \in \mathcal{A}$, then the local computation process will be skipped for robot i.

1) Initialization: At the first iteration of phase II, $\mathcal{S}_2^i = \emptyset$ and thus robot i can directly

update $\mathcal{S}_2^i$ as the action with the maximum marginal gain based on the empty set. That is, $s \in \arg\max_{v \in \mathcal{V}_i} f_\emptyset(v)$. Then, $\mathcal{S}_2^i$ is updated as $\mathcal{S}_2^i \leftarrow s$. The corresponding marginal gain $f_\emptyset(s)$ is also recorded.

2) Inter-robot communication: Let us consider the case where $|\mathcal{S}_2^i| = n$ with $n \leq N - K$ at some point before the algorithm stops. We first consider $s \in \mathcal{S}_2^i$ in the descending order they are added through the local computation procedure. For example, if $|\mathcal{S}_2^i| = n$, we can write $\mathcal{S}_2^i$ as $\mathcal{S}_2^i = \{s_1, \ldots, s_n\}$, such that $f_\emptyset(s_1) \geq \ldots \geq f_{\{s_1,\ldots,s_{n-1}\}}(s_n)$. We also use $\gamma(\cdot)$ to denote the order of action $s \in \mathcal{S}_2^i$ as

$$\gamma(s_1) = 1, \ldots, \gamma(s_n) = n.$$

Similarly, we also apply this reordering procedure to $\mathcal{S}_2^j, \forall\, j \in \mathcal{N}_i$. Thus, there is also a marginal gain and an order associated with the action $s \in \mathcal{S}_2^j$. With the marginal gains and orders ready, we are ready to merge $\mathcal{S}_2^j$ with $\mathcal{S}_2^i$. For every $\mathcal{S}_2^j$, we augment $\mathcal{S}_2^i$ with $\mathcal{S}_2^j$ and apply an operation as

$$\mathcal{S}_2^{i+} \leftarrow \mathsf{sort}\left(\{\mathcal{S}_2^i, \mathcal{S}_2^j\}, \text{'descend'}\right).$$

This operation is read as "$s \in \{\mathcal{S}_2^i, \mathcal{S}_2^j\}$ are sorted in a descending order based on the associated marginal gains".

Remove redundant actions: The merged set $\mathcal{S}_2^{i+}$ may contain *redundant* actions. By redundant actions, we refer to the actions $s \in \mathcal{S}_2^{i+}$ having the following *redundant action* properties:

- s appears in $\mathcal{S}_2^i$ and $\mathcal{S}_2^j$. e.g., $s = s'$ where $s \in \mathcal{S}_2^i$ and $s' \in \mathcal{S}_2^j$;
- The associated marginal gains are the same.
- The orders in $\mathcal{S}_2^i$ and $\mathcal{S}_2^j$ are the same. e.g., $\gamma(s) = \gamma(s')$.

We can check these properties for each $v \in \mathcal{S}_2^{i+}$ against all other actions to remove the redundant actions.

Remove order changed actions: After the above process, we know that there is an order associated with $s, \forall\, s \in \mathcal{S}_2^{i+}$. Similarly, we also know that $s \in \mathcal{S}_2^i$ and $s \in \mathcal{S}_2^j$ also have their orders. If the order of any action is changed before and after the augmentation, this action and the actions having lower marginal gains than this action's marginal gain will be invalid. This rule is from the submodularity of $f(\cdot)$. We can then use the following properties to remove order changed actions. Specifically, we need to remove any $s \in \mathcal{S}_2^{i+}$ if s satisfies the following *order changed* properties:

- s appears in $\mathcal{S}_2^{i+}$ and $\mathcal{S}_2^i$ (or $\mathcal{S}_2^j$). e.g., $s = s'$ where $s \in \mathcal{S}_2^{i+}$ and $s' \in \mathcal{S}_2^i$ (or $s' \in \mathcal{S}_2^j$).

- The orders are not the same. e.g., $\gamma(s) \neq \gamma(s')$.

This operation can be illustrated by the following example. If the local approximation $\mathcal{S}_2^i$ is

$$\mathcal{S}_2^i = \{s_1, s_2\} \quad \text{and} \quad f_\emptyset(s_1) \geq f_{s_1}(s_2),$$

Then, the orders of $s_1, s_2 \in \mathcal{S}_2^i$ are as follows $\gamma(s_1) = 1, \gamma(s_2) = 2$. Also, if a neighbor j's approximation is

$$\mathcal{S}_2^j = \{s_2, s_3\} \quad \text{and} \quad f_\emptyset(s_2) \geq f_{s_2}(s_3).$$

Then, the orders of $s_2, s_3 \in \mathcal{S}_2^j$ are $\gamma(s_2) = 1, \gamma(s_3) = 2$. Now consider the case where the augmented set is $\mathcal{S}_2^{i+} = \{s_1, s_2, s_3\}$, and the marginal gains are such that

$$f_\emptyset(s_1) \geq \underbrace{f_{s_1}(s_2)}_{s_2 \in \mathcal{S}_2^i} = \underbrace{f_\emptyset(s_2)}_{s_2 \in \mathcal{S}_2^j} \geq f_{s_2}(s_3). \tag{6.2}$$

After applying the above redundancy removal procedure, there may exist a case where

$$f_{\{s_1,s_2\}}(v) > f_{\{s_1,s_2\}}(s_3),$$

where $v \in \mathcal{V} \setminus (\mathcal{S}_1^i \cup \{s_1, s_2\})$. In this case, s_3 is no longer a valid action in $\mathcal{S}_2^{i+}$ as action v has a higher marginal gain. To deal with this case, we can use action orders. From the marginal gains relations in (6.2), we have

$$\gamma(s_1) = 1, \underbrace{\gamma(s_2)}_{s_2 \in \mathcal{S}_2^i} = 2, \underbrace{\gamma(s_2)}_{s_2 \in \mathcal{S}_2^j} = 3, \gamma(s_3) = 4.$$

Also, we know that the original orders of $s_2, s_3 \in \mathcal{S}_2^j$ are

$$\gamma(s_2) = 1, \gamma(v_3) = 2, \quad \text{where} \quad s_2, s_3 \in \mathcal{S}_2^j.$$

So, the order of s_2 and s_3 where $s_2, s_3 \in \mathcal{S}_2^j$ are changed after merging. Therefore, we need to remove these two actions from $\mathcal{S}_2^{i+}$. Finally, the augmented approximation is assigned to $\mathcal{S}_2^i$ as $\mathcal{S}_2^i \leftarrow \mathcal{S}_2^{i+}$.

3) Local computation: After the inter-robot communication process, robot i may need to change its original action selection. That is because robot i made its selection before knowing its neighbors' approximations $(\mathcal{S}_2^j, \forall\, j \in \mathcal{N}_i)$. Once receiving $\mathcal{S}_2^j, \forall\, j \in \mathcal{N}_i$, robot i can update its own action selection, i.e., $v \in \mathcal{V}_i$.

We update robot i's action selection based on the marginal gain of $v \in \mathcal{V}_i$ for every possible combination of its neighbors' selections. The necessity of this operation is from the observation that the marginal gain of an action will be changed if the already selected action set is changed. For example, after the inter-robot communication, if we have $\mathcal{S}_2^i = \{s_1, \ldots, s_n\}$ and $v \notin \mathcal{S}_2^i, \forall\, v \in V_i$, we then need to check the marginal gain of $v \in \mathcal{V}_i$ when v has different

orders in $\mathcal{S}_2^i$. Since we already know the associated marginal gains of $s \in \mathcal{S}_2^i$, we can compare

$$\begin{aligned} \max_{v \in \mathcal{V}_i} f_{\emptyset}(v) \quad &\text{vs.} \quad f_{\emptyset}(s_1), \\ &\vdots \\ \max_{v \in \mathcal{V}_i} f_{\{s_1,\ldots,s_{n-1}\}}(v) \quad &\text{vs.} \quad f_{\{s_1,\ldots,s_{n-1}\}}(s_n). \end{aligned}$$

Whenever $\forall\, v \in \mathcal{V}_i$ generates a better marginal gain than the compared one, we replace the compared action with the action v and delete the actions selected after the compared one. This is because if the orders of the actions are changed, then the marginal gains are invalid. This operation is shown in Lines 15 to 25 (Algorithm 6). Meanwhile, the associated marginal gain is also updated. Finally, the updated $\mathcal{S}_2^i$ along with the marginal gains of $s \in \mathcal{S}_2^i$ are broadcasted to $j \in \mathcal{N}_i$.

4) Stopping condition: After one cycle of local computation and inter-robot communication, the local counter α_2^i will be incremented by 1 if there is no change of $\mathcal{S}_2^i$ before and after these two processes. Otherwise, the counter α_2^i is reset to 0. When α_2^i reaches $2d(\mathcal{G})$, where $d(\mathcal{G})$ is the diameter of $\mathcal{G}$, robot i stops operations. Meanwhile, all robots have an agreement on the approximation of $\mathcal{S}_2$.

6.3 Performance Analysis

Lemma 1. *The procedure GenerateRemovals (Algorithm 5) for finding the approximated removals has the following performance:*

1. Approximation performance: *The approximated removals for robot i is $\mathcal{S}_1^i = \mathcal{S}_1$, where $\mathcal{S}_1$ is the K-max consensus result.*

2. Convergence time: *The algorithm takes $d(\mathcal{G})$ steps to converge, where $d(\mathcal{G})$ is the diameter of $\mathcal{G}$.*

3. Computational complexity: *The computational complexity for every robot is at most* $\mathcal{O}\left(|\mathcal{V}_i|\right)$.

Proof. 1). *Approximation performance:* In the centralized scenario, we know that we need to find the top K actions to approximate the removal set. In the distributed scenario, $\mathcal{S}_1^i$ is updated as $\mathcal{S}_1^i \leftarrow \arg\max_{v \in \mathcal{V}_1(i)} f(v)$ at the beginning. Assume that i and j are different before communicating with each other. Upon receiving $\mathcal{S}_1^j$, robot i's approximation $\mathcal{S}_1^i$ is updated by using $\min\left(K, |\mathcal{S}_1^i \cup \mathcal{S}_1^j|\right)$ actions as shown in Line 9 (Algorithm 5). Similarly, this procedure is also applied to j. Thus, robot i and j will agree with each other on the top K actions after communication. Finally, when all robots $r \in \mathcal{R}$ receive other robots' approximation after $d(\mathcal{G})$ steps, they achieve a consensus on the top K actions.

2) *Convergence time:* In every execution of Algorithm 5, i needs to update $\mathcal{S}_1^i$ through the received $\mathcal{S}_1^j$ from $j \in \mathcal{N}_i$. Similarly, it takes $d(\mathcal{G})$ steps for i to receive $\mathcal{S}_1^r$ from r that has the longest communication distance. During this procedure, every robot can receive all other robots' approximation at least once. Thus, Algorithm 5 takes $d(\mathcal{G})$ steps to converge.

3) *Computational complexity:* Robot i needs $|\mathcal{V}_i|$ evaluations to find the largest contribution $s \in \arg\max_{v \in \mathcal{V}_i} f(v)$. After communications, if $s \in \mathcal{S}_1^i$, then s is among the top K actions. If $s \notin \mathcal{S}_1^i$, then s is replaced by other actions with larger contributions and we do not need to evaluate for robot i again. So, the number of evaluations for every robot is at most $\mathcal{O}\left(|\mathcal{V}_i|\right)$. □

Lemma 2. *The procedure GenerateComplements (Algorithm 6) for finding the complements has the following performance:*

1. Approximation performance: *The approximated complements for i is $\mathcal{S}_2^i = \mathcal{S}_2$, where $\mathcal{S}_2$ is the centralized greedy solution generated by using marginal gains.*

2. Convergence time: *In the worst-case, the algorithm converges in $2(N-K+1)d(\mathcal{G})$ steps, where $d(\mathcal{G})$ is the diameter of $\mathcal{G}$.*

3. Computational complexity: *The computational complexity for every robot is at most $\mathcal{O}\left((N-K)^2|\mathcal{V}_i|\right)$.*

Proof. 1). *Approximation performance:* When merging $\mathcal{S}_2^j$ with $\mathcal{S}_2^i$, we first use the $\mathsf{sort}(\cdot)$ procedure to maintain the orders of the actions $s \in \mathcal{S}_2^{i+}$ regardless of the redundancy and the orders of the actions in $\mathcal{S}_2^{i+}$ as shown in Line 9 of Algorithm 6. Then, through the operation described in Section 6.2.2 (also in Lines 10 to 11 of Algorithm 6), we resolve these two issues by removing any $s \in \mathcal{S}_2^{i+}$ that is either redundant or order changed. In local computation, when robot i updates its action set, the marginal gains of $s \in \mathcal{S}_2^i \setminus \mathcal{V}_i$ are used as oracles. In the local computation procedure, when any $v \in \mathcal{V}_i$ replaces the compared action, the actions having lower marginal gains in $\mathcal{S}_2^i$ are removed from $\mathcal{S}_2^i$. This procedure maintains the descending orders of $s \in \mathcal{S}_2^i$ while updating robot i's contribution. Therefore, all these procedures help to keep the descending order of $s \in \mathcal{S}_2^i$. When these procedures are applied to all robots in $\mathcal{R}$, the system converges to the same approximation $\mathcal{S}_2^i$ since every robot will have an agreement on at least one action after each communication. Also, since the descending orders of $v \in \mathcal{S}_2^i$ are kept during all communications, the final converged $\mathcal{S}_2^i$ is the same as the centralized solution. That is, $\mathcal{S}_2^i = \mathcal{S}_2$.

2) *Convergence time:* Through the above analysis, we know that if $\mathcal{S}_2^i \neq \mathcal{S}_2^j$, then this disagreement is resolved through communications. In an extreme case, we assume that the communication distance between robot i and robot r is $d(\mathcal{G})$. It then takes $2d(\mathcal{G})$ steps for i to agree with r on at least one action that is selected by i or r. Also, $\max_{i\in\mathcal{R}} |\mathcal{S}_2^i| = N - K$.

Therefore, it takes at most $2(N-K)d(\mathcal{G})$ steps to reach the final agreement. Meanwhile, robot i needs to take another $2d(\mathcal{G})$ steps to confirm the convergence.

3) *Computational complexity:* Algorithm 6 needs at most $|\mathcal{V}_i||\mathcal{S}_2^i \cup \mathcal{S}_2^j|$ evaluations during each local computation procedure since i checks its maximum contribution against every combination of the actions in the merged set $\mathcal{S}_2^i \cup \mathcal{S}_2^j$. Also, it holds that $\max_{i,j\in\mathcal{R}} |\mathcal{S}_2^i \cup \mathcal{S}_2^j| = N-K$. Therefore, the computational complexity for every robot is at most $\mathcal{O}\left((N-K)^2|\mathcal{V}_i|\right)$.

□

Theorem 11. *Algorithm 4 has the following performance:*

1. Performance: *The approximation ratio is*

$$f\left(\mathcal{S}\setminus\mathcal{F}^\star\right) \geq \max\left\{\frac{1-c_f}{1+c_f}, \frac{1}{1+K}, \frac{1}{|\mathcal{R}|-K}\right\} f\left(\mathcal{S}^\star\setminus\mathcal{F}^\star\right).$$

   where $\mathcal{S}^\star$ is an optimal solution and $\mathcal{F}^\star$ is an optimal removal set with respect to $\mathcal{S}^\star$.

2. Convergence time: *In the worst-case, the algorithm converges in $(2N-2K+3)d(\mathcal{G})$ steps, where $d(\mathcal{G})$ is the diameter of $\mathcal{G}$.*

3. Computational complexity: *The computational complexity for every robot is at most $\mathcal{O}\left((N-K)^2|\mathcal{V}_i|\right)$.*

Proof. 1). *Approximation performance:* From Lemma 1 and Lemma 2, we know that $\mathcal{S}_1^i = \mathcal{S}_1$ and $\mathcal{S}_2^i = \mathcal{S}_2$, where $\mathcal{S}_1$ and $\mathcal{S}_2$ are the corresponding centralized solutions. Then, the approximation performance of the distributed resilient algorithm (Algorithm 4) is the same as that of its centralized counterpart [75]. That is,

$$f\left(\mathcal{S}\setminus\mathcal{F}^\star\right) \geq \max\left\{\frac{1-c_f}{1+c_f}, \frac{1}{1+K}, \frac{1}{|\mathcal{R}|-K}\right\} f\left(\mathcal{S}^\star\setminus\mathcal{F}^\star\right).$$

where $\mathcal{S}^\star$ is an optimal solution and $\mathcal{F}^\star$ is an optimal removal set with respect to $\mathcal{S}^\star$.

2) *Convergence time:* Based on the results from Lemma 1 and Lemma 2, we know that the convergence time is $(2N - 2K + 3)d(\mathcal{G})$.

3) *Computational complexity:* Combining the results from Lemma 1 and Lemma 2, we have the computational complexity as $\mathcal{O}\left((N-K)^2|\mathcal{V}_i|\right)$. □

6.4 Numerical Evaluation

6.4.1 Simulation Setup

Environment settings: We verify the performance of Algorithm 4 by implementing it into a scenario where a distributed multi-robot system explores an environment modeled by a Gaussian mixture model (GMM). Specifically, the environment is generated as $z(x, y) = \sum_{\ell=1}^{B} r_\ell b_\ell(x, y) = \mathbf{r}^\top \mathbf{b}$, where (x, y) are 2D coordinates, $r_\ell : \mathbb{R} \mapsto \mathbb{R}$ are the weights for the basis functions $b_\ell : \mathbb{R}^2 \mapsto \mathbb{R}, \forall\, \ell = 1, \ldots, B$. Also, $\mathbf{r} = [r_1, \ldots, r_\ell]^\top$ and $\mathbf{b} = [b_1, \ldots, b_\ell]^\top$ are the stacked weights and basis functions respectively. The number of basis function and variances are selected randomly. In the simulation, we use a 200×200 field to represent the environment. There is an environmental importance associated with each location. The importance value of a location equals to the GMM value of that location.

Compared algorithms: We compare the performance of Algorithm 4, which is referred to as "*distributed-resilient*", with the performance of the following methods:

- An *optimal* method, where the solution is generated through a brute-force search.
- A *semi-dist* method [78], where the solution is generated by first partitioning robots into groups and then running a centralized resilient algorithm in each group.

- A *cent-greedy* method [11], where the solution is generated greedily based on marginal gains that maximize the objective function in a centralized manner.

- A *cent-rand* method, where the solution is generated randomly in a centralized manner.

Multi-robot system settings: We compare the performance of the system using two different settings: 1), In the first setting, we compare the performance of our distributed resilient method with the optimal, the semi-dist, the cent-greedy method, and the cent-rand methods. We set the number of robots as $N = 5$, and the maximum number of attacks as $K = 3$. Then, we run 200 trials to compare the performance. We generate random initial locations for the robots in each trial, and each robot has four actions (forward, backward, left, and right). 2), In the second setting, we compare the performance of the resilient method, the semi-dist, the cent-greedy method, and the cent-rand method. Specifically, we set the number of robots as $N \in \{30, 40, 50\}$. The corresponding number of attacks K is randomly generated from $[0.5N, 0.75N]$ and rounded to an integer. This setting means that at least 50% of the robots will be attacked, and at most 75% of the robots will be attacked. The robots' rewards are added with white Gaussian noise with a mean of 10% of the original rewards and a variance of 5% of the original rewards. We then run 50 trials for each setting. However, since finding the worst-cast attacks is intractable and we aim to test the proposed algorithm's scalability, we use greedy attacks in this case. That is, the attacker attacks the robots' sensors greedily using the standard greedy algorithm [11]. Common simulation parameters include: in each of the above settings, the sensing range of the robots is set to 10; the reward of an action is the environmental importance explored by this action. In each trial, the robots are randomly placed in a region with $x \in [50, 100], y \in [50, 100]$. Finally, we perform Monte Carlo simulations to test the performance of these four methods with the same simulation parameters.

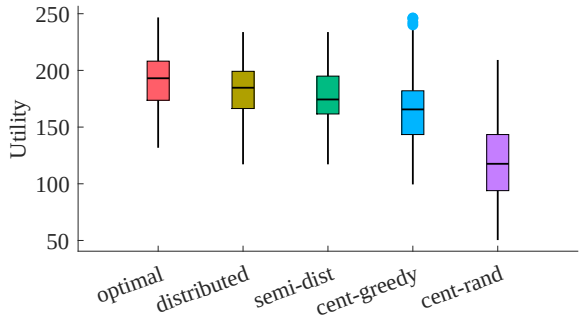


Fig. 6.3. The statistics of the utilities of the five different methods over 200 trials with the number of robots $N = 5$ and the number of attacks $K = 3$. The box-plot demonstrates the quartiles of different solutions.

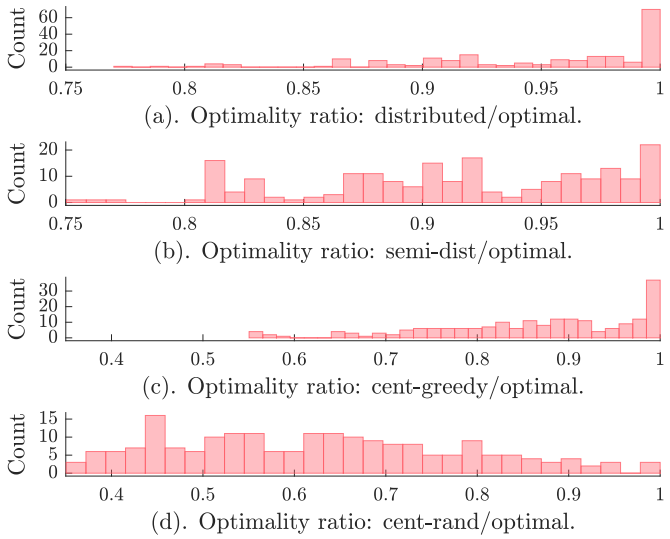
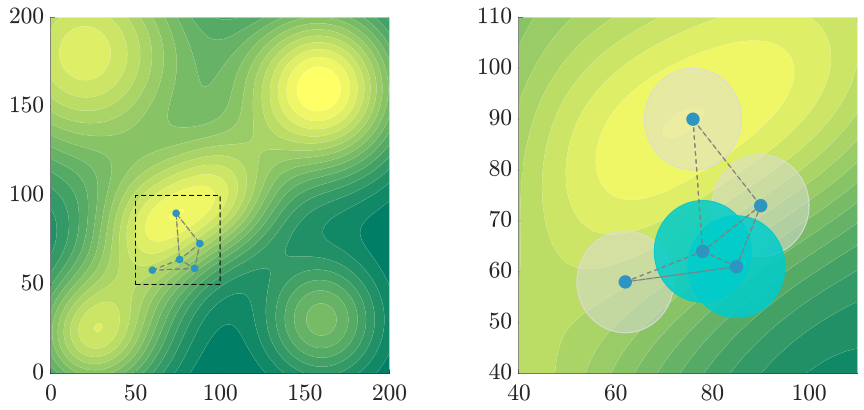


Fig. 6.4. The optimality ratios of different solutions with respect to their corresponding optimal solutions in each of the 200 trials.

6.4.2 Performance Comparison

Performance metric: The performance of different methods is captured by the sum of the importance in the explored area after worst-case attacks. Specifically, we first generate a solution for each method. Then, attackers attack and remove the contributions of attacked robots. Since finding worst-case attacks is intractable, we use a brute-force search to find the worst-case attacks for each generated solution.



(a) (b)

Fig. 6.5. (a). The robots ($N = 5$) are placed randomly in the environment, and a connected communication graph $\mathcal{G}$ is initialized randomly. (b). The resilient solution after the worst-case attacks ($K = 3$). The selections of worst-case attacked sensors are in gray. The selections of unattacked sensors are in cyan.

In the first setting, we compare the statistics of the utilities of different methods using 200 trials, as shown in Fig. 6.3. The utilities in the box-plot reflect the performance of different methods by using the quartiles of each solution. As suggested in the result, we observe that the median of the utilities generated by the proposed distributed resilient method shows better performance than that of the other three methods except for the optimal method.

In Fig. 6.4, we compare the optimality ratios of different solutions with respect to their corresponding optimal solutions in each setting. Specifically, the optimality ratio range of the proposed distributed-resilient is $[0.77, 1]$. The optimality ratio range of the semi-distri-resilient method is $[0.75, 1]$. The optimality ratio range of the centralized-greedy method is $[0.55, 1]$ The optimality ratio range of the centralized-random method is $[0.35, 1]$. This further illustrates that the proposed algorithm (Algorithm 4) is superior to the other methods since most of the cases have a close to optimal optimality ratio as shown in Fig. 6.4(a). In Fig. 6.5(a), we demonstrate the environment and the initial configuration of the robots of one instance. Then, the solution of the proposed distributed-resilient method using one instance

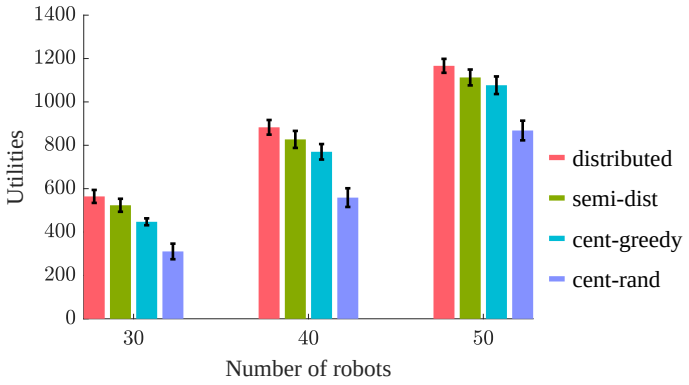


Fig. 6.6. The utilities of the four different methods with $N = 30, 40$, and 50 and with the number of attacks K (for each N) randomly generated from $[0.5N, 0.75N]$.

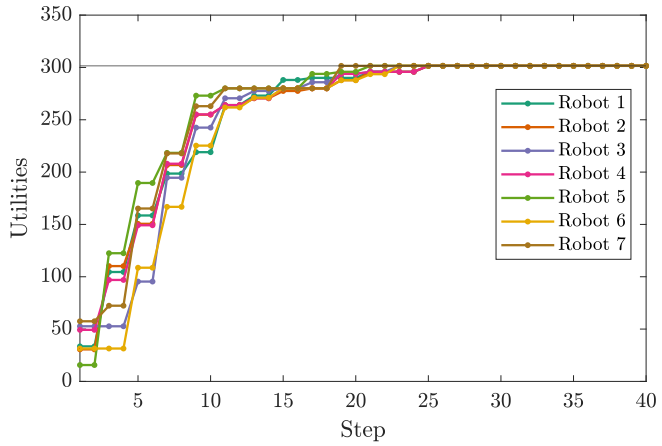


Fig. 6.7. The evolution of the utilities of the 7 robots that are not attacked when the number of robot $N = 15$ robots and the number of attacks $K = 8$.

is shown in Fig. 6.5(b).

In the second setting, we compare the mean of the utilities of different methods. Fig. 6.6 shows the utilities of four different approaches. The result demonstrates that Algorithm 4 yields superior results compared with the other methods. Finally, we plot in Fig. 6.7 the evolution of utilities for different robots when the number of robots is 15, and the number of attacks is 8, demonstrating the convergence of the proposed distributed resilient method.

Remark 4. *Under our problem formulation, imperfect motion and sensing impact our*

algorithm in two ways: they disturb how the robots evaluate the reward of their actions and the ability to execute high-reward actions as planned. These influences do not change the fundamental nature of the problem but would instead impact the collected reward. However, the proposed algorithm is still superior to the other three methods, as shown in Fig. 6.6.

6.5 Conclusions

In this chapter, we proposed a fully distributed algorithm for the problem of the resilient submodular action selection. We proved that the solution of the proposed algorithm converges to the corresponding centralized algorithm. We evaluated the algorithm's performance through extensive simulations.

Chapter 7

Asynchronous Distributed Decision-Making for Multi-robot Systems

In this chapter, we consider a general action selection problem for networked multi-robot systems under real-world conditions: asynchronous communication/computation, complex combinatorial and knapsack constraints, and fully distributed operation. The motivation for this work derives from ubiquitous multi-robot problems that involve action selection, e.g., target tracking, where asynchronous robot communication/computation can significantly impact system performance when solutions must be found in a completely distributed manner. In such contexts, an efficient algorithm is required to mitigate the impact of asynchronicity, while appropriately balancing communication and computational load. Thus, we first formulate a general action selection problem with complex action independence and budget constraints, which can be applied to a variety of multi-robot action selection problems. We then propose a distributed algorithm for solving this class of problems through local greedy selections and asynchronous communication/computation. Most importantly, our algorithm allows robots to operate in parallel under asynchronicity while avoiding conflicts and race conditions between local robot solutions. We prove that the local solution maintained by each robot eventually converges with theoretical guarantees. Furthermore, we demonstrate

that the result of the proposed distributed algorithm has the same performance as that of a centralized solution. Finally, Monte Carlo simulations are presented to corroborate our theoretical results.

Introduction

In networked multi-robot systems, action selection problems are of vital importance as they underlie a wide range of theoretical and practical applications. Examples can be found in formation control [83], task allocation [22], sensor networks [19], precision agriculture [7], target tracking [84], etc. In general, action selection problems are difficult combinatorial optimization problems, and in many cases are inapproximable [85]. A common way to model the objectives in such problems is to utilize the *submodularity* of the system reward, allowing greedy algorithms to produce solutions with bounded suboptimality. Submodularity is a diminishing returns property where the marginal gain/reward of the system is reduced when the solution set becomes larger [8]. In centralized systems, multiple methods have been proposed to tackle submodular optimization problems. A seminal work proposed in [23] introduced a greedy method for solving such problems under *matroid* constraints and proved bounds on solution quality. Since this early theoretical result, various robotic applications have been proposed based on submodular optimization, in both centralized and distributed settings (we summarize these works in the sequel). However, in realistic applications where *asynchronicity* may exist in robot communication/computation, the problem of selecting actions for the system while preserving performance guarantees is far more complicated. For example, in Fig. 7.1, a multi-robot system must select actions (trajectories) to fulfill a system objective, e.g., a coverage maximization problem. When the actions selected by one robot are sent to others, the other robots must update their own local solutions to exploit their neighbors' observations and subsequent actions. When this updating procedure is

performed asynchronously, action selections made locally may result in action conflicts or race conditions that negatively impact performance, or worse, safety. Fundamentally, there are two subproblems at the core of action selection problems in such systems: (1) determining the best action for each robot based on local observations and the system objective function; and (2) how to merge the contribution from each robot in a manner that yields a global solution without action conflicts. In this chapter, we aim to solve these problems in a distributed way when contributions from each robot are received *asynchronously*.

The study of distributed algorithms for coordinating multi-robot (-agent) systems has attracted tremendous attention over the last few decades. In [86], the authors proposed a seminal work on consensus where each robot in the system needs to move along the gradient of a disagreement function generated using the graph Laplacian of the system. Based on this work, extensions for dealing with various complications have been proposed. For example, constrained consensus [87], consensus with topology time-delays [86], consensus with switching topology [86]. etc. In general, the robots in such systems adjust their dynamics following the system's objective function gradient (in some sense the continuous analog of a greedy algorithm). The exchange of robot information happens when the system updates its objective function, and in early works this update was assumed synchronous. Alternatively, the gossip method [88] (and related variations) allow for asynchronicity, where each robot sends its information to a random neighbor to achieve a global objective. Examples can be found in distributed averaging problems [88], networked convex optimizations [89], rigidity evaluation [63], etc. In addition, multiple constrained versions have also been proposed for problems with budget constraints [90].

While we share a distributed and asynchronous nature, in contrast to the above works we instead propose an algorithm that solves a *combinatorial optimization problem* for multi-robot action selection. Specifically, we first present two motivating examples, a weapon-target

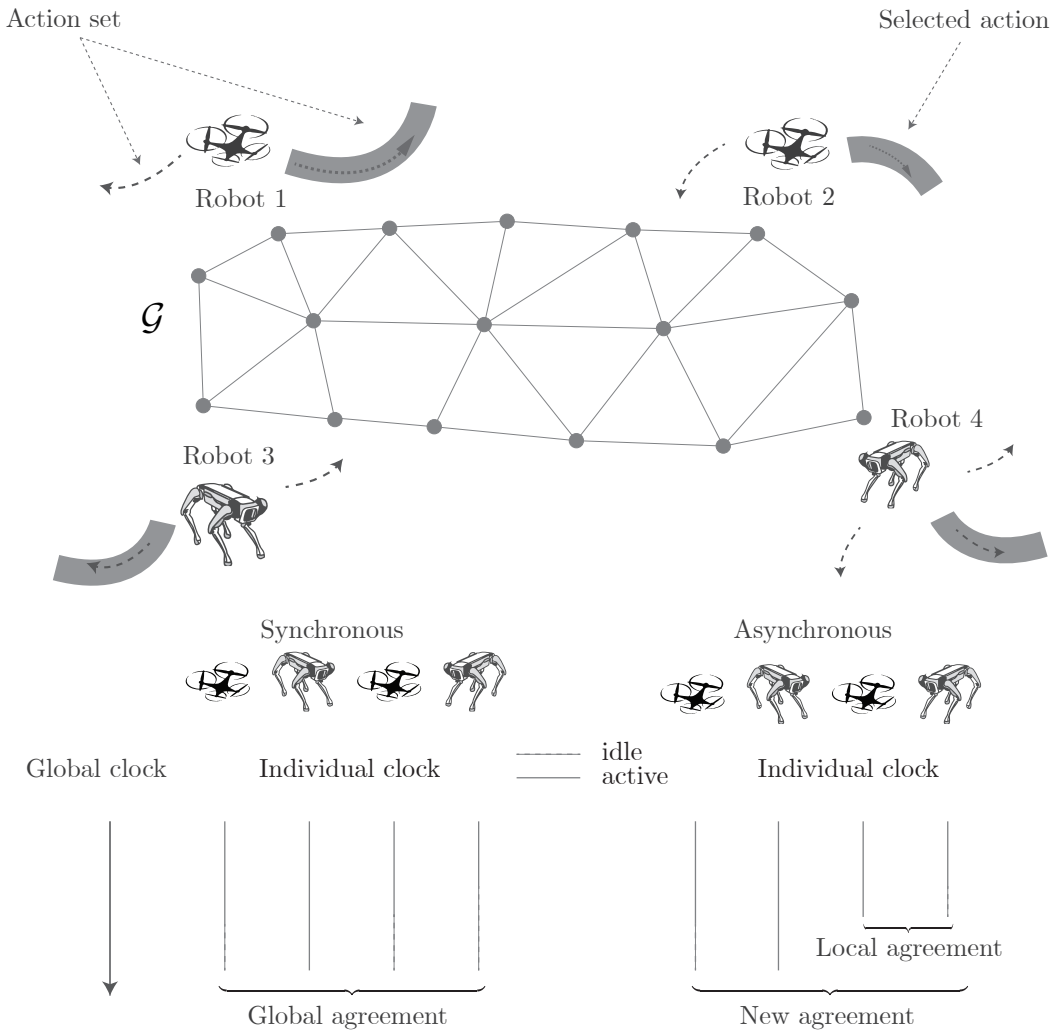


Fig. 7.1. Comparison between synchronous and asynchronous action selection. *(Top)* When maximizing a system objective, e.g., a coverage application with mobile robots, each robot needs to consider other robots' actions by exploiting communication over a graph ($\mathcal{G}$). *(Bottom left)* In a synchronous system, a global agreement between robots only occurs when all robots (eventually) communicate with each other. *(Bottom right)* In an asynchronous system, a local agreement can occur when any two (or more) robots communicate with each other, and thus a sequence of local agreements lead to a global agreement.

allocation problem and an intermittent deployment problem, to provide intuition of our problem formulation. In the first application, the robots communicate in an asynchronous way to maximize the system utility by selecting weapon-target pairs while respecting budgets. In the intermittent deployment problem, each robot needs to determine a monitoring/deployment

plan to maximize collected information while respecting the system's spatial and temporal constraints, again subject to asynchronicity. We propose a general problem that captures these applications (and many more general cases) in the form of a submodular maximization problem with matroid and knapsack constraints. Indeed, submodular objective functions and matroid constraints have wide applications, including sensor placement/planning [27, 91], task allocation [10, 79, 92], environmental monitoring [1], resilient coverage [6], network rigidity evaluation [63, 93], orienteering/routing problem [18], etc.

With our problem context defined, we now compare our efforts to the most closely related work. In [79], the authors proposed a synchronized consensus-based method for task (and task bundle) allocation problems. In each global tick, every robot needs to achieve a consensus. In [10], the authors proposed to use a matroid to model the task allocation constraint, however the proposed action conflict resolution method required synchronization in the robot network. In a similar application [92], the authors investigated a distributed primal-dual approach. In [21], the authors proposed a distributed way to address this problem, while again synchronicity of the system is used to resolve solution conflicts between robots. Similar efforts can also be found in [94]. Market-based approaches, such as [95–97], use the sequential auction method to solve combinatorial optimization problems with sub-optimality guarantees, again assuming synchronicity, which means it is hard to change/modify the already selected actions. It is worth noting that integer programming can be another alternative if the objective function is modular instead of submodular [98]. However, in general, integer programming cannot handle the diminishing returns property of the objective functions, which are commonly used in robotics objective functions. Another general method for handling distributed combinatorial optimization problems is MapReduce [99, 100], which splits the ground set into separate parts and transfers them to different robots. However, a central point of command is require to merge the individual solution from each robot in the final MapReduce stage.

Although a few studies have investigated distributed combinatorial optimization problems in multi-robot systems, we seek to establish a unified formulation for modeling these problems, while providing an asynchronous distributed algorithm that can efficiently produce solutions with performance guarantees.

Contributions

In summary, the contributions of this work are as follows:

- We propose an algorithm for solving the asynchronous distributed action selection problem.
- We prove the theoretical performance of the proposed distributed algorithm.
- We show that the proposed distributed solution converges to that of the centralized algorithm.

Organization of the chapter:

The remainder of the chapter is organized as follows. In Section 7.1, we introduce the problem formulation. In Section 7.2, we demonstrate the procedures of the proposed distributed framework, including local processing and communication. After that, we analyze and prove the result of the proposed algorithm in Section 7.3. In Section 7.4, we demonstrate the results of the proposed algorithm by using different applications through Monte Carlo simulations. Finally, we conclude the chapter in Section 7.5.

7.1 Problem Formulation

7.1.1 Robot and Network

Assumption 2 (Connectedness)**.** *We assume that the associated graph network $\mathcal{G}$ of the multi-robot system $\mathcal{A}$ is connected. There exists a sequence of distinct vertices starting with $i \in \mathcal{A}$ and ending with $j \in \mathcal{A}$ such that consecutive vertices are adjacent. Also, each robot $i \in \mathcal{A}$ is equipped with the ability to handle information from neighbors $j \in \mathcal{N}_i$ and send information to neighbors $j \in \mathcal{N}_i$ through edge $(i, j) \in \mathcal{E}$, where the message structure is defined by the senders.*

Assumption 3 (Bounded communication delay)**.** *Robot $i \in \mathcal{A}$ and $j \in \mathcal{N}_i$ can communicate at least once every $r(\mathcal{G})$ ticks. This guarantees that no robot $i \in \mathcal{A}$ has an infinite delay, which makes the communication between robots available under the asynchronous assumption.*

Assumption 4 (Asynchronicity)**.** *As the proposed method can operate in realistic networks, we assume the time model is asynchronous. Besides, we assume that each robot receives and processes messages from $j \in \mathcal{N}_i$ in a first-in-first-out (FIFO) manner.*

Under the asynchronicity assumption, each robot $i \in \mathcal{A}$ has an internal clock, e.g., a rate 1 Poisson process [63], which ticks according to a rate independent of other robots'.

Assumption 5 (Information sharing)**.** *Each robot $i \in \mathcal{A}$ can share its estimates with neighbors $j \in \mathcal{N}_i$, but cannot reveal its data with neighbors in the graph, including objective function, constraints, etc.*

In the above settings, robot $i \in \mathcal{A}$ is able to evaluate its objective function value and executes operations dictated by the proposed algorithm according to the ticks of its clock. Note that

robot i cannot evaluate the objective function of other robots. Otherwise, the system is centralized, and a single robot is sufficient for the system to make decisions.

Remark 5 (Network Structure)**.** *The graph $\mathcal{G}$ does not have to be static, but the bounded communication delay assumption (Assumption 3) needs to be satisfied to make the communications available. The proposed algorithms and analysis will be based on static graphs, but the procedures are the same for dynamic graphs.*

In this section, we first demonstrate two applications and then introduce a general distributed submodular maximization problem formulation to summarize those types of problems.

7.1.2 The Weapon-Target Allocation Problem

In a weapon-target task allocation application, the robot set $\mathcal{A}$ contains all the weapons, and the task set Ω contains all the targets. Both robots and tasks are randomly located in $[0,1]^3 \subseteq \mathbb{R}^3$. The robots can communicate asynchronously through the communication graph $\mathcal{G}$ defined in Section 7.1.1. The system ground set $\mathcal{V} = \mathcal{A} \times \Omega$ contains all the feasible robot-task combinations. The utility of every weapon-target pair is a function of the signal-to-noise ratio, a decreasing function of the distance between robots and targets. This objective function is approximated by the following,

$$f_1(\mathcal{X}) = \sum_{(a,\omega)\in\mathcal{X}} \sqrt{\frac{w_t}{\mathsf{dist}(a,\omega)^2}}, \quad (\mathcal{X} \subseteq \mathcal{V}), \tag{7.1}$$

where $a \in \mathcal{A}$ is an robot and $\omega \in \Omega$ is a target. The weight w_t is the reward of capturing target ω. The function $\mathsf{dist}(\cdot)$ is an Euclidean distance function in $\mathbb{R}^3$. We define $\mathcal{V}_a = a \times \Omega$ as a sub-ground set for robot a that contains all feasible tasks for robot $a \in \mathcal{A}$. Similarly, $\mathcal{V}_\omega = \omega \times \mathcal{A}$ is the sub-ground set for the target $\omega \in \Omega$, containing all feasible robots for

w. Since the resources of every robot is limited and may be different, the system has the following constraints.

- The number of targets that robot $a \in \mathcal{A}$ can handle cannot exceed λ_a. To satisfy this condition, consider a matroid constraint $\mathcal{M}_{11} = (\mathcal{V}, \mathcal{I}_{11})$, where

$$\mathcal{I}_{11} = \{\mathcal{X} \subseteq \mathcal{V} \mid |\mathcal{X} \cap \mathcal{V}_a| \leq \lambda_a, \forall\, a \in \mathcal{A}\}. \tag{7.2}$$

- Every target $\omega \in \Omega$ can only be selected by at most one robot. To satisfy this condition, consider a matroid constraint $\mathcal{M}_{12} = (\mathcal{V}, I_{12})$, where

$$\mathcal{I}_{12} = \{\mathcal{X} \subseteq \mathcal{V} \mid |\mathcal{X} \cap \mathcal{V}_\omega| \leq 1, \forall\, \omega \in \Omega\}. \tag{7.3}$$

  Note that both $\mathcal{I}_{11}$ and $\mathcal{I}_{12}$ are partition matroid constraints [14].

- The maximum number of robot-target pairs that the system can handle is λ. To satisfy this condition, consider a matroid constraint $\mathcal{M}_{13} = (\mathcal{V}, \mathcal{I}_{13})$, where

$$\mathcal{I}_{13} = \{\mathcal{X} \subseteq \mathcal{V} \mid |\mathcal{X} \cap \mathcal{V}| \leq \lambda\}, \tag{7.4}$$

  where λ is the system budget.

$\mathcal{M}_1$ is a separable matroid constraint, which means it can be checked locally by each $a \in \mathcal{A}$. While $\mathcal{M}_3$ is a non-separable constraint, meaning all robots need to cooperate to check the feasibility of solutions. As all the above constraints are matroid constraints, we can use a matroid intersection constraint to simplify those requirements. That is, $\mathcal{M}_1 = (\mathcal{V}, \mathcal{I}_1)$, where $\mathcal{I}_1 = \mathcal{I}_{11} \cap \mathcal{I}_{12} \cap \mathcal{I}_{13}$. The cardinality of this matroid intersection is $|\mathcal{M}_1| = 3$. Now, the problem is defined as follows.

Problem 4 (The weapon-target allocation problem)**.** *In reference to the previous framework, we consider a set of robots $\mathcal{A}$ that needs to be assigned to a set of tasks Ω. The robots communicate asynchronously through $\mathcal{G}$. The system aims to maximize the collected utility while respecting the constraint $\mathcal{M}_1 = (\mathcal{V}, \mathcal{I}_1)$. That is,*

$$\begin{aligned} \underset{\mathcal{X} \subseteq \mathcal{A} \times \mathcal{E}}{\textit{maximize}} \quad & \sum_{(a,\omega) \in \mathcal{X}} \sqrt{\frac{w_t}{\mathsf{dist}(a,\omega)^2}} \\ \textit{subject to} \quad & \mathcal{X} \in \mathcal{I}_1. \end{aligned}$$

This is a submodular maximization problem with a matroid intersection (three matroids) constraint. It is known that the submodular maximization problems with three or more matroids constraints are NP-hard [8].

7.1.3 The Intermittent Deployment Problem

Next, we consider an environmental monitoring problem where different robots need to communicate asynchronously to make deployment decisions. Consider a discretized 2D dynamic environment with size $u_x \times u_y$ evolving along the time horizon T. The associated location index set is $\mathcal{P}$, where $|\mathcal{P}| = u_x \times u_y$. The environment dynamics are modeled as a spatiotemporal process. Suppose that we are given the mean predictions μ associated with every location in the environment from $t = 1$ to $t = T$. Also, the associated covariance predictions is denoted by Σ. The environment will be monitored by a robot system $\mathcal{A}$. The goal of the monitoring is to maximize mutual information by selecting monitoring locations and times with waiting penalty considered while respecting system budgets. We formulate the decision element as

$$\mathbf{x}_\ell^a(t) = [x_\ell^a, y_\ell^a, t]^\top,$$

which can be read as "robot $a \in \mathcal{A}$ is deployed to the ℓth ($\ell \in \mathcal{P}$) location at time $t \in \mathcal{T}$ to collect samples". $\mathcal{T}$ is the time index set with $|\mathcal{T}| = T$. Therefore, the ground set $\mathcal{V}$ of our deployment problem is the Cartesian product of the robot set $\mathcal{A}$, the location set $\mathcal{P}$ and the time index set $\mathcal{T}$. That is $\mathcal{V} = \mathcal{A} \times \mathcal{P} \times \mathcal{T}$, which is equivalent to $\mathcal{V} = \{\mathbf{x}_\ell^a(t) \mid a \in \mathcal{A}, \ell \in \mathcal{P}, t \in \mathcal{T}\}$. In addition, we define $\mathcal{V}_t = \mathcal{A} \times \mathcal{P} \times t, \forall\, t \in \mathcal{T}$ as the sub-ground set at time t. Therefore, $\mathcal{V} = \bigcup_{t \in \mathcal{T}} \mathcal{V}_t$. We also define $\mathcal{E} = \mathcal{P} \times \mathcal{T}$ as the common decision set for every robot $i \in \mathcal{A}$.

Given the problem ground set $\mathcal{V}$, if the selected action set is $\mathcal{X}$, then the objective function $f_2(\mathcal{X})$ is modeled as

$$f_2(\mathcal{X}) = H(\mathcal{X}) - H(\mathcal{X} \mid \mathcal{V} \setminus \mathcal{X}) - \sum_{\forall\, t \in \mathbf{x}_\ell^a(t), \mathbf{x}_\ell^a(t) \in \mathcal{X}} w \cdot t, \tag{7.5}$$

where

$$H(\mathcal{X}) = \frac{1}{2} \log \det(2\pi e \Sigma(\mathcal{X}))$$

is the entropy and $\Sigma(\mathcal{X})$ is the covariance matrix of the selected action set $\mathcal{X}$, and w is the weight for the weight penalty associated with t since $\mathbf{x}_\ell^a(t) = [x_\ell^a, y_\ell^a, t]^\top$.

To make the deployment available under system budgets, we use the following deployment constraints:

- No more than λ_t robots can be deployed at time $t \in \mathcal{T}$. To satisfy this condition, consider a matroid constraint $\mathcal{M}_{21} = (\mathcal{V}, \mathcal{I}_{21})$, where $\mathcal{I}_{21}$ is

$$\mathcal{I}_{21} = \left\{\mathcal{X} \subseteq \mathcal{V} \mid |\mathcal{X} \cap \mathcal{V}_t| \leq \lambda_t\right\}. \tag{7.6}$$

- The number of times where there is at least one robot deployed is less than or equal to

λ_2. To satisfy this condition, consider a matroid constraint $\mathcal{M}_{22} = (\mathcal{V}, \mathcal{I}_{22})$, where $\mathcal{I}_{22}$ is

$$\mathcal{I}_{22} = \left\{ \mathcal{X} \subseteq \mathcal{V} \mid \sum_{t=1}^{T} \mathbb{1}(|\mathcal{X} \cap \mathcal{V}_t|) \leq \lambda_2 \right\}, \tag{7.7}$$

and $\mathbb{1}(\cdot)$ is an indicator function that takes the form

$$\mathbb{1}(|\mathcal{X} \cap \mathcal{V}_t|) = \begin{cases} 1 & \text{if } |\mathcal{X} \cap \mathcal{V}_t| \geq 1, \\ 0 & \text{if } |\mathcal{X} \cap \mathcal{V}_t| = 0. \end{cases}$$

- The number of deployments is less than or equal to λ_3. To satisfy this condition, consider a matroid constraint $\mathcal{M}_{23} = (\mathcal{V}, \mathcal{I}_{23})$, where $\mathcal{I}_{23}$ is

$$\mathcal{I}_{23} = \left\{ \mathcal{X} \subseteq \mathcal{V} \mid |\mathcal{X} \cap \mathcal{V}| < \lambda_3 \right\}. \tag{7.8}$$

- The total cost of the deployment is less than or equal to the budget λ. To satisfy this condition, consider a knapsack constraint.

$$\mathcal{K} = \left\{ \mathcal{X} \subseteq \mathcal{V} \mid c(\mathcal{X}) \leq \lambda_k \right\}, \tag{7.9}$$

where $c(\mathcal{X}) = \sum_{e \in \mathcal{X}} c(e)$ is the sum of the traveling cost $c(e)$ associated with $e \in \mathcal{X}$.

Similarly, we can also use a matroid intersection constraint to simplify those requirements. That is, $\mathcal{M}_2 = (\mathcal{V}, \mathcal{I}_2)$, where $\mathcal{I}_2 = \mathcal{I}_{21} \cap \mathcal{I}_{22} \cap \mathcal{I}_{23}$. The cardinality of this matroid intersection constraint is $|\mathcal{M}_2| = 3$. Now, the deployment problem is defined as follows.

Problem 5 (The intermittent deployment problem)**.** *In reference to the previous problem formulation framework, we consider a team of robots $\mathcal{A}$, which can communicate asynchronously through the associated communication graph $\mathcal{G}$. The system aims to maximize the mutual*

information by selecting a deployment set $\mathcal{X}$, including deployment locations and times while respecting the deployment constraint $\mathcal{M}_2$ and $\mathcal{K}$. That is,

$$\begin{aligned} \underset{\mathcal{X}\subseteq\mathcal{V}}{\textit{maximize}} \quad & f_2(\mathcal{X}) \\ \textit{subject to} \quad & \mathcal{X} \in \mathcal{I}_2, \mathcal{X} \subseteq \mathcal{K}. \end{aligned}$$

At a high level, this is a submodular maximization problem with matroid and knapsack constraints. It is worth pointing out that other task allocation problems [10,22,79,92] can also be formulated in the same way. The robot set in the task allocation problems corresponds to $\mathcal{A}$ in our problem setting. The task set corresponds to $\mathcal{E}$ in the above problem. We want to find a general way to formulate this type of problem.

7.1.4 The General Problem Formulation

Since the previous two applications share the same structure, we summarize the general form of the asynchronous distributed submodular maximization problem as follows.

Problem 6 (The asynchronous distributed action selection problem)**.** *Consider a multi-robot system $(\mathcal{A}, \mathcal{G})$ with a submodular objective function $f(\cdot)$, where $\mathcal{A}$ is the robot set and $\mathcal{G}$ is the associated communication graph. All robots communicate with each other asynchronously through $\mathcal{G}$. Meanwhile, the constraints contain a matroid intersection constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$, and a knapsack constraint $\mathcal{K}$. The general form of the asynchronous distributed submodular maximization problem can then be written as*

$$\begin{aligned} \underset{\mathcal{X}\subseteq\mathcal{V}}{\textit{maximize}} \quad & f(\mathcal{X}) \\ \textit{subject to} \quad & \mathcal{X} \in \mathcal{I}, \mathcal{X} \subseteq \mathcal{K}. \end{aligned} \tag{7.10}$$

This problem is a submodular maximization problem with matroids and knapsack constraints.

Proposition 1. *Finding an optimal solution for the asynchronous distributed action selection problem (Problem 6) is NP-hard.*

Proof. Submodular maximization without constraint is NP-hard [8], but adding more constraints is not guaranteed to make the problem computationally harder in general [101]. It is known that the submodular welfare problem is NP-hard [45]. We show the result by reducing the submodular welfare problem to the submodular maximization problem under matroid and knapsack constraints. The submodular welfare problem is to maximize a submodular welfare objective function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ under a partition matroid constraint. We reduce the submodular welfare problem to the intermittent deployment problem as follows: For $\mathcal{M}_{21}$, there is no need to reduce because it is a partition matroid, which is the same as in the submodular welfare problem. For $\mathcal{M}_2$, we set $\lambda = T$ (or $|\mathcal{T}|$). This change makes $\mathcal{M}_2$ accepts all $\mathcal{X}$'s. For the constraint $\mathcal{K}$, we set $\lambda = \sum_{e \in \mathcal{V}} c(e)$. Since this knapsack budget will never be exceeded, the constraint takes all $\mathcal{X}$'s. Under this reduction, a solution to the proposed intermittent deployment problem would be a solution to the submodular welfare problem. Therefore, finding an optimal solution to the above problem is NP-hard. □

7.2 An Asynchronous Distributed Submodular Optimization Algorithm

Without synchronicity in the system, as shown in Fig. 7.1, different robots do not need to synchronize the information across all robots a central point of command in the system. The proposed asynchronous distributed submodular maximization algorithm aims to select a set from the ground set $\mathcal{V}$ to maximize the objective function $f(\cdot)$ while respecting the constraint

$\mathcal{I}$ and $\mathcal{K}$. At a high level, each robot $i \in \mathcal{A}$ will select element from the ground set through local evaluation and communication. When the proposed algorithm stops, all robot will achieve an equilibrium, and each robot will hold the same solution that is the same as the centralized solution.

Since the distributed algorithms for solving this problem with different types of constraints will be slightly different, i.e., matroid constraint $\mathcal{I}$ vs. matroid and knapsack constraints $\mathcal{I}$ and $\mathcal{K}$, and the solutions share the common structures, we put these two different cases in the same framework as shown in Algorithm 7. If there is only a matroid intersection constraint $\mathcal{M}$, we call it a "*type* 1" constraint. If the constraint is a combination of a matroid intersection $\mathcal{M}$ and a knapsack $\mathcal{K}$ constraints, we call it a "*type* 2" constraint.

In general, each robot $i \in \mathcal{A}$ executes logic iteratively and communicates with neighbors $j \in \mathcal{N}_i$ asynchronously. During these operations, each robot $i \in \mathcal{A}$ computes an independent set and updates when robot i communicates with robot $j \in \mathcal{N}_i$. Finally, a stop checking is performed to decide whether to terminate the algorithm or not. Before proceeding, we first summarize the main notations.

- We denote by robot i the *local robot* as we will introduce and analyze the algorithm from the perspective of robot i. Also, denote by $\mathcal{S}(i)$ the *local independent set* maintained by robot i. This set contains the elements selected by (or assigned to) [1] i, which is the current solution maintained by i.

- The *local instance set* $\mathcal{L}(i) \subseteq \mathcal{S}(i)$ is such that

$$\mathcal{L}(i) = \left\{ e \mid e \in \mathcal{S}(i), \phi_e^{(i)} = i \right\}, \tag{7.11}$$

[1] When an element e is added to an independent set and $\phi_e^{(i)} = i$, we say e is selected by robot i or assigned to robot i.

TABLE 7.1
NOMENCLATURE

Notation	Interpretation
a	any robot
i	local robot
j	neighboring robot ($j \in \mathcal{N}_i$)
$\mathcal{I}$	matroid constraint
$\mathcal{K}$	knapsack constraint
$\mathcal{S}(i)$	local independent set
$\mathcal{L}(i)$	local instance set
$\phi_e^{(i)}$	e's corresponding robot index in $\mathcal{S}(i)$
$\Delta_e^{(i)}$	e's marginal gain in $\mathcal{S}(i)$

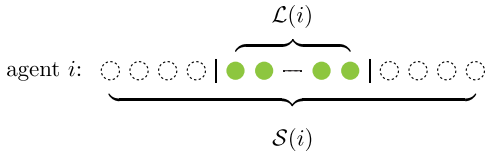


Fig. 7.2. Local independent set $\mathcal{S}(i)$ and local instance set $\mathcal{L}(i)$. The set $\mathcal{S}(i)$ contains all the elements of the independent set maintained by i, while $\mathcal{L}(i)$ only contains the elements selected by i. Thus, $\phi_e^{(i)} = i, \forall\, e \in \mathcal{L}(i)$, which means the corresponding robot index of $e \in \mathcal{L}(i)$ is i. Also, $\phi_e^{(i)} \neq i, \forall\, e \in \mathcal{S}(i) \setminus \mathcal{L}(i)$, which means the corresponding robot index of $e \in \mathcal{S}(i) \setminus \mathcal{L}(i)$ is not i.

where $\phi_e^{(i)}$ maps e to its corresponding robot index when $e \in \mathcal{S}(i)$. Unlike $\mathcal{S}(i)$, this local instance set $\mathcal{L}(i)$ only contains the elements selected by robot i. Therefore, $\phi_e^{(i)} \neq i, \forall\, e \in \mathcal{S}(i) \setminus \mathcal{L}(i)$.

- We denote by $\Delta_e^{(i)}$ the marginal gain of e with respect to (w.r.t.) $\mathcal{S}(i)$. The subscript e represents the element, while the superscript i represents the corresponding robot index. Note that once e is added to set $\mathcal{L}(i)$, then $\Delta_e^{(i)}$ is fixed. We only use this notation when e is selected by the algorithm.

Remark 6. *It is worth noting that $\mathcal{S}(i)$ contains not only the elements selected by i but also the elements selected by robots other than i. Every robot has an associated an independent set,*

Algorithm 7 Asynchronous Distributed Submodular Maximization for Robot i.

procedure RobotRun(i)
  ▷ Receiving information
  for $msg(j), \forall\, j \in \mathcal{N}_i$ **do**
    HandleMsg($msg(j)$);
  end for

  ▷ Local updating
  if *constraint type* = "type 1" **then**
    AlgMatroid(i);
  else if *constraint type* = "type 2" **then**
    AlgMatroidKnapsack(i);
  end if

  ▷ Sending information and updating
  if $\mathcal{S}(i)$ is unchanged **then**
    $counter(i) \leftarrow counter(i) + 1$;
  else
    $counter(i) \leftarrow 0$;
  end if
  if $counter(i) = r(\mathcal{G})d(\mathcal{G})$ **then**
    return $\mathcal{S}(i)$ & ***terminate***.
  end if
  $msg(i) \leftarrow \{\mathcal{S}(i), counter(i)\}$;
  send $msg(i)$ to $\forall\, j \in \mathcal{N}_i$;
end procedure

which will eventually converge the same one. In Fig. 7.2, we demonstrate the relationship between $\mathcal{S}(i)$ and $\mathcal{L}(i)$ from the perspective of robot i. To summarize the main notations, Table 7.1 shows the corresponding interpretations. We will also emphasize it whenever there is confusion.

Now, we focus on the algorithm's logic from the perspective of robot i. We call a full execution of RobotRun (Algorithm 7) of robot i as a *round*, which differentiates the *iterations* in while or for loops in the algorithm. We first give a high-level overview of the proposed distributed Algorithm 7.

1. *Receiving information:* Robot i receives information $msg(j)$ from neighbors $j \in \mathcal{N}_i$ through the message pipes asynchronously as shown in Fig. 7.3. If there is no information received, robot i skips this procedure.

2. *Local updating:* Under different constraint types, robot i executes different algorithms to update $\mathcal{S}(i)$. If the constraint is a "type 1" constraint, robot i executes AlgMatroid to update $\mathcal{S}(i)$. If the constraint is a "type 2" constraint, robot i then executes AlgMatroidKnapsack to update $\mathcal{S}(i)$.

3. *Sending information and stop checking:* Robot i holds a counter $counter(i)$ to calculate the number of times that $\mathcal{S}(i)$ has no change. This counter will then be used to decide whether to terminate the algorithm or not. If there is a change of $\mathcal{S}(i)$, then the counter updates as $counter(i) \leftarrow 0$. If there is no change, then the counter updates as $counter(i) \leftarrow counter(i) + 1$. Finally, when the counter reaches the threshold $r(\mathcal{G})d(\mathcal{G})$, then the algorithm achieves an equilibrium, i.e.,

$$\mathcal{S}(i) = \mathcal{S}(a), \forall\, a \in \mathcal{A} \setminus i,$$

and all procedures can terminate. If the stopping criterion does not satisfy, the next round starts.

7.2.1 Matroid Constraint

In the centralized scenario, when the constraint is a matroid constraint or a matroid intersection constraint, the algorithm greedily selects the best (maximum marginal gain) unassigned element from the ground set based on the local knowledge of the current independent set.

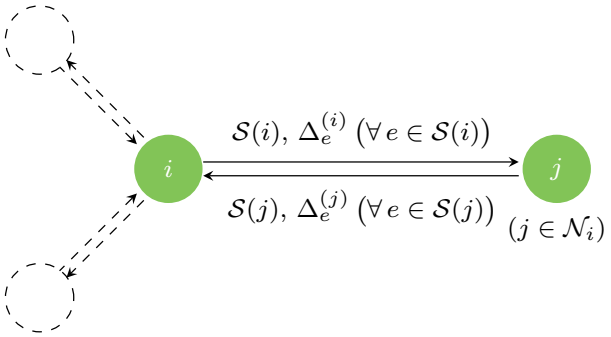


Fig. 7.3. The information flow between robot $i \in \mathcal{A}$ and one of its neighbors $j \in \mathcal{N}_i$. From i's perspective, i needs to share its independent set $\mathcal{S}(i)$ and the corresponding marginal gains $\Delta_e^{(i)}, \forall\, e \in \mathcal{S}(i)$ to its neighbors $j \in \mathcal{N}_i$.

Algorithm 8 Submodular Maximization with Matroid Constraint Handler for Robot i.

procedure AlgMatroid(i)
  $\mathcal{Y} \leftarrow \emptyset$;
  while true do
    $\mathcal{X} \leftarrow FindAllocatedSet(\mathcal{S}(i))$;
    $\mathcal{Z} \leftarrow \mathcal{V} \setminus (\mathcal{X} \cup \mathcal{Y})$;
    if $\mathcal{Z} = \emptyset$ **then**
      break;
    end if

    $e' \in \arg\max_{e \in \mathcal{Z}} f(e \mid \mathcal{S}(i))$;
    if $\mathcal{S}(i) \cup \{e'\} \in \mathcal{I}$ **then**
      $\mathcal{S}(i) \leftarrow \mathcal{S}(i) \cup \{e'\}$;
    else
      $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{e'\}$.
    end if
  end while
end procedure

That is,

$$\mathcal{S} \leftarrow \mathcal{S} \cup \left\{ \underset{e \notin \mathcal{S}: \mathcal{S} \cup \{e\} \in \mathcal{I}}{\arg\max} f(e \mid \mathcal{S}) \right\}, \tag{7.12}$$

where $\mathcal{S}$ is the independent set built iteratively.

In the distributed scenario, we will also follow the similar way to build the independent set for every robot. As the independent set is built greedily and asynchronously by all robots,

Algorithm 9 Find the Allocated Set Handler for Robot i.

procedure FindAllocatedSet(i)
  $\mathcal{X} \leftarrow \emptyset$;
  for $e \in \mathcal{S}(i)$ **do**
    if $\phi_e^{(i)} = i$ or $\mathcal{X} \cup \{e\} \notin \mathcal{I}$ (or $\mathcal{I}, \mathcal{K}$) **then**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$;
    end if

    if $\phi_e^{(i)} \neq i$ and $f(e \mid \mathcal{X}) < \Delta_e^{(i)}$ **then**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$;
    end if

    if $f(e \mid \mathcal{X}) = \Delta_e^{(i)}$ and $\phi_e^{(i)} < i$ **then**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$;
    end if
  end for
  return $\mathcal{X}$.
end procedure

the matroid constraint $\mathcal{I}$ is checked locally by each robot when a new element is considered. Note that robot i does not require the matroid constraints dictating the non-local selections. Also, we need extra steps to take care of the algorithm efficiency and conflicts, as shown in the procedure AlgMatroid depicted in Algorithm 8.

First, we use the procedure FindAllocatedSet depicted in Algorithm 9 to find the allocated set $\mathcal{X}$ from $\mathcal{S}(i)$. That is, $\mathcal{X} \subseteq \mathcal{S}(i)$. All elements $e \in \mathcal{X}$ will not participate in the following computations. Excluding $\mathcal{X}$ from the ground set $\mathcal{V}$ will help us to speed up the later computation procedures. In general, from the perspective of robot i, the set $\mathcal{X}$ contains the following two parts: a). all the elements that have been selected by robot i; b). all the elements that are not selected by i and will not be able to be selected by i later. For all $e \in \mathcal{S}(i)$, we use the following conditions to update $\mathcal{X}$ as $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$. These conditions are:

- If $\phi_e^{(i)} = i$, then $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$. Since robot i selects $e \in \mathcal{V}$ greedily and locally, we admit all $e \in \mathcal{S}(i)$ such that $\phi_e^{(i)} = i$. Equivalently, we admit all $e \in \mathcal{L}(i)$ since

$\phi_e^{(i)} = i, \forall\, e \in \mathcal{L}(i)$. This procedure corresponds to Lines 4 to 6 in Algorithm 9.

- If the constraint is "type 1", we use the condition $\mathcal{X} \cup \{e\} \notin \mathcal{I}$. If the constraints is "type 2", we use the condition: $\mathcal{X} \cup \{e\} \notin \mathcal{I}$ or $\mathcal{X} \cup \{e\} \nsubseteq \mathcal{K}$. Then, we update $\mathcal{X}$ as then $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$. Since getting the marginal gain is usually computational heavier than checking the constraint (or constraints) in practice, we use this condition to reduce the number of available candidates in $\mathcal{V}$ for updating $\mathcal{S}(i)$. This procedure also corresponds to Lines 4 to 6 in Algorithm 9.

- If $\phi_e^{(i)} \neq i$ and $f(e \mid \mathcal{X}) < \Delta_e^{(i)}$, then $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$. In this case, $e \notin \mathcal{L}(i)$ and $f(e \mid \mathcal{X}) < \Delta_e^{(i)}$. Therefore, e cannot be a candidate for adding to $\mathcal{S}(i)$ as the new marginal gain $f(e \mid \mathcal{X})$ is smaller than the old one $\Delta_e^{(i)}$. This procedure corresponds to Lines 8 to 10 in Algorithm 9.

- If $f(e \mid \mathcal{X}) = \Delta_e^{(i)}$ and $\phi_e^{(i)} < i$, then $\mathcal{X} \leftarrow \mathcal{X} \cup \{e\}$. When $f(e \mid \mathcal{X}) = \Delta_e^{(i)}$ happens, conflict occurs. We then choose the element with a lower robot index, i.e., $\phi_e^{(i)} < i$, as the potential element for $\mathcal{S}(i)$. This procedure corresponds to Lines 12 to 14 in Algorithm 9.

Remark 7 (Conflict resolution)**.** *Note that $\forall\, e \in \mathcal{X}$ will not participate in the following iterations for updating $\mathcal{S}(i)$. Also, when*

$$\phi_e^{(i)} \neq i \quad \textit{and} \quad f(e \mid \mathcal{X}) = \Delta_e^{(i)},$$

there are two robots, i.e., $\phi_e^{(i)}$ and i, having the same marginal gain. Here we assign the element to the robot with a lower robot index, i.e., $\phi_e^{(i)} < i$, as the conflict resolution criterion. However, other criteria are also feasible as long as consistency is maintained.

Next, we use $\mathcal{Y}$ to store the elements selected through the maximum operation but with the

constraint $\mathcal{I}$ unsatisfied. We then have $\mathcal{Z} \leftarrow \mathcal{V} \setminus (\mathcal{X} \cup \mathcal{Y})$ containing candidate elements for updating $\mathcal{S}(i)$. Through these operations, the number of elements that need to be checked is reduced, and the efficiency is increased. We then add the candidates to $\mathcal{L}(i)$ as follows.

$$\mathcal{S}(i) \leftarrow \mathcal{S}(i) \cup \left\{ \underset{e \in \mathcal{Z}: \mathcal{L}(i) \cup \{e\} \in \mathcal{I}}{\arg\max} f(e \mid \mathcal{S}(i)) \right\}. \tag{7.13}$$

This procedure corresponds to Lines 10 to 12 in Algorithm 8.

7.2.2 Matroid and Knapsack Constraints

When the constraint is "type 2", the procedure AlgMatroidKnapsack depicted in Algorithm 10 illustrates the subroutine to construct the independent set $\mathcal{S}(i)$ from the perspective of robot i. This is a distributed version of its corresponding centralized algorithm [46] with modification.

In the matroid only constraint case ("type 1"), we have one independent set $\mathcal{S}(i)$ for each robot $i \in \mathcal{A}$. In the matroid and knapsack constraint case ("type 2"), we need to prepare multiple candidate independent sets $\mathcal{S}_k(i)$, where $k \in [1, \lceil \log_{1+\epsilon} \frac{f(\mathcal{V})}{\max_{e \in \mathcal{V}} f(e)} \rceil]$ and $k \in \mathbf{Z}$, for every robot $i \in \mathcal{A}$. We read $\mathcal{S}_k(i)$ as "robot i's kth local independent set'. Also, there is a counter *counter*(i_k) associated with $\mathcal{S}_k(i)$, which is used to decide whether to terminate the algorithm or not. When the algorithm converges, the final independent set of robot i is the one with the highest utility. That is,

$$\mathcal{S}(i) \leftarrow \underset{\mathcal{X} \in \bigcup \mathcal{S}_k(i)}{\arg\max} f(\mathcal{X}). \tag{7.14}$$

At a high level, the Algorithm 10 first defines an upper bound $f(\mathcal{V})$ and a lower bound $\max_{e \in \mathcal{V}} f(e)$ for the possible marginal gain to cost ratio in Line 3. Then, this possible marginal value to cost value space is sliced into possibilities for the system to check. In each iteration,

Algorithm 10 Submodular Maximization with Matroid and Knapsack Constraints Handler for Robot i.

1: **procedure** AlgMatroidKnapsack(i)
2: $k \leftarrow 0, p \leftarrow |\mathcal{M}|, M \leftarrow \max_{e \in \mathcal{V}} f(e)$;
3: **for** $\rho = \frac{M}{p+1}; \rho \leq \frac{2f(\mathcal{V})}{p+1}; \rho \leftarrow (1+\epsilon)\rho$ **do**
4: $\mathcal{Y} \leftarrow \emptyset, k \leftarrow k+1$;
5: $\tau \leftarrow M_\rho \leftarrow \max\{f(e) \mid \frac{f(e)}{c(e)} \geq \rho\}$;
6:
7: **for** $\tau = M_\rho; \tau \geq \frac{\epsilon}{|\mathcal{V}|} M_\rho; \tau \leftarrow \frac{1}{1+\epsilon}\tau$ **do**
8: $\mathcal{X} \leftarrow FindAllocatedSet(\mathcal{S}_k(i))$;
9: $\mathcal{Z} \leftarrow \mathcal{V} \setminus (\mathcal{X} \cup \mathcal{Y})$;
10: **if** $\mathcal{Z} = \emptyset$ or $\tau < \frac{\epsilon M_\rho}{|\mathcal{V}|}$ **then**
11: **break**;
12: **end if**
13:
14: **for** $\forall\, e \in \mathcal{Z}$ **do**
15: $e' \in \arg\max_{e \in \mathcal{Z}} f(e \mid \mathcal{S}(i))$;
16: **if** $\mathcal{S}_k(i) \cup \{e'\} \in \mathcal{I}, \mathcal{K}, \frac{f(e' \mid \mathcal{S}_k(i))}{c(e')} \geq \rho, f(e' \mid \mathcal{S}_k(i)) \geq \tau$ **then**
17: $\mathcal{S}_k(i) \leftarrow \mathcal{S}_k(i) \cup \{e'\}$;
18: **else**
19: $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{e'\}$;
20: **end if**
21: **end for**
22: **end for**
23: **end for**
24: **end procedure**

the marginal gain to cost ratio upper bound ρ updates as $\rho \leftarrow (1+\epsilon)\rho$ until the system reaches its upper bound, where $\epsilon \in (0,1)$ is a tunable parameter.

Before updating $\mathcal{S}_k(i)$ using $e \in \mathcal{V}$, we also need to use the same procedure FindAllocatedSet depicted in Algorithm 9 to prepare the allocated set $\mathcal{X}$ to reduce the cardinality of the candidate set as shown in Line 8. Similarly, $\mathcal{Y}$ is used to keep track of all the elements that violate feasibility in the constraints, as shown in Lines 16 to 19 in Algorithm 10. Finally, the set $\mathcal{Z} \leftarrow \mathcal{V} \setminus (\mathcal{X} \cup \mathcal{Y})$ is prepared to reduce the number of available candidates before selecting, and a candidate can be added to $\mathcal{S}_k(i)$ when the followings requirements are satisfied. That

is,

$$\mathcal{S}_k(i) \cup \{e\} \in \mathcal{I}, \tag{7.15a}$$

$$\mathcal{S}_k(i) \cup \{e\} \in \mathcal{K}, \tag{7.15b}$$

$$f(e \mid \mathcal{S}_k(i)) \geq \tau, \tag{7.15c}$$

$$\frac{f(e \mid \mathcal{S}_k(i))}{c(e)} \geq \rho. \tag{7.15d}$$

Note that the requirements (7.15a) and (7.15b) are the basic problem constraints: matroid constraint $\mathcal{I}$ and knapsack $\mathcal{K}$. The requirements (7.15c) and (7.15d) are used to check if adding e to $\mathcal{S}_k(i)$ satisfies the requirements or not in the current iteration with respect to the parameter τ and ρ. Specifically, (7.15c) defines a marginal gain lower bound τ for adding $e \in \mathcal{Z}$ to $\mathcal{S}_k(i)$. This lower bound τ updates as $\tau \leftarrow \frac{1}{1+\tau}\tau$. While in (7.15d), ρ is used as a lower bound of marginal gain to cost ratio for any candidate e. Since we have different ρ's in different iterations (Line 3), therefore, we need to use different $\mathcal{S}_k(i)$ for different ρ. Thus, the local robot i holds $\lceil \log_{1+\epsilon} \frac{f(\mathcal{V})}{M} \rceil$ different $\mathcal{S}(i)$'s for getting the system's independent set since $\rho \in [\frac{M}{p+1}, \frac{2f(\mathcal{V})}{p+1}]$ and ρ update as $\rho \leftarrow (1+\epsilon)\rho$.

Remark 8. *Note that in Line 2 of Algorithm 10, we need to calculate* $M \leftarrow \max_{e \in \mathcal{V}} f(e)$*, which is the maximum element wise value of all elements. In the distributed scenario, an information spreading technique [102, 103] through the network* $\mathcal{G}$ *can be used to achieve a consensus on* M*.*

Broadly speaking, Algorithm 10 has two loops: the outer loop (Lines 3 to 23) and the inner loop (Lines 7 to 22). In each iteration of the outer loop, the algorithm picks a threshold ρ as a marginal-to-cost ratio, i.e., $\frac{m_e}{c(e)}, \forall\, e \in \mathcal{V}$. Again, $m_e = f(\{e\} \mid \mathcal{S}_k(i))$ is the marginal value of e under the current independent set $\mathcal{S}_k(i)$. The $\mathcal{S}_k(i)$ chosen by the outer loop will serve as one candidate for the final independent set $\mathcal{S}(i)$. The threshold ρ ensures we do not

Algorithm 11 Independent Set msg Handler for Robot i.

```
1: procedure HandleMsg(msg(j))
2:     {S(j), counter(j)} ← msg(j);
3:     if S(i) = S(j) then
4:         return;
5:     end if
6:
7:     S(i) ← LocalGreedy(S(i), S(j));
8:     S(j) ← LocalGreedy(S(j), S(i));
9:     S ← S(i) ∪ S(j);                                   ▷ merge
10:    if S ∈ I (or I, K) then
11:        S(i) ← S;
12:    else
13:        S(i) ← ∅;
14:        for e ∈ S do                                   ▷ greedy
15:            if S(i) ∪ {e} ∈ I (or I, K) then
16:                S(i) ← S(i) ∪ {e};
17:            end if
18:        end for
19:    end if
20:    return S(i).
21: end procedure
```

exceed the budget constraint by discretizing the space of the marginal-to-cost ratio. The inner loop (Lines 7 to 22) also has a threshold τ for the marginal value m_e and decreases in every iteration. This threshold τ acts as a lower bound for the marginal value of m_e in the current settings. Under different τ's, we can add different e's to the current independent set $\mathcal{S}_k(i)$ when constraints are satisfied. The final $\mathcal{S}$ forms one candidate for the final independent set. There might be a case that only the knapsack constraints are not satisfied. We then need to keep track of this case when deciding the final independent set $\mathcal{S}_(i)$.

Algorithm 12 Local Greedy Handler for Robot i.

procedure LocalGreedy($\mathcal{S}(i), \mathcal{S}(j)$)
  $\mathcal{X} \leftarrow \emptyset, \mathcal{Y} \leftarrow \emptyset$;
  for $e \in \mathcal{S}(i)$ **do**
    $\mathcal{X}' \leftarrow \{e' \mid \forall\, e \in \mathcal{S}(j) \setminus \mathcal{Y}, \phi_e^{(i)} = \phi_e^{(j)}\}$;
    if $e \in \mathcal{S}(j)$ and $\Delta_e^{(i)} < \Delta_e^{(j)}$ **then**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{X}'$;
    end if

    if $e \in \mathcal{S}(j)$ and $\Delta_e^{(i)} = \Delta_e^{(j)}$ and $\phi_e^{(i)} < \phi_e^{(j)}$ **then**
      $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{X}'$;
    end if
    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{e\}$;
  end for
  $\mathcal{S}(i) \leftarrow \mathcal{S}(i) \setminus \mathcal{X}$;
  return $\mathcal{S}(i)$.
end procedure

7.2.3 Interrobot Messaging

In general, when a neighboring (or non-local) independent set $\mathcal{S}(j), \forall\, j \in \mathcal{N}_i$ is received, robot i needs to update $\mathcal{S}(i)$ based on $\mathcal{S}(j)$, yielding a new $\mathcal{S}(i)$. The orderings of $e \in \mathcal{S}(i)$ and $e \in \mathcal{S}(j)$ dictate their marginal gains. When conflict occurs, we need to choose the element while respecting the fact that if the orderings change for $e \in \mathcal{S}(i) \setminus \mathcal{L}(i)$, we no longer know their marginal gains. We first provide the high-level idea of our conflict resolution as follows. Given $\mathcal{S}(i)$ and $\mathcal{S}(j)$ with their fixed marginal gains, we should select the elements greedily while respecting the constraint ("type 1" or "type 2"). From i's perspective, this process is performed as if $e \in \mathcal{S}(j)$ are of fixed marginal gains and were chosen in their given order by an oracle since i cannot make evaluations for $j \in \mathcal{N}_i$. We then correct $\mathcal{S}(i)$ by respecting the better selections made by robots $a \in \mathcal{A} \setminus i$ from $\mathcal{S}(j)$. Therefore, the greedy ordering of $\mathcal{S}(i) \setminus \mathcal{L}(i)$ and $\mathcal{S}(j) \setminus \mathcal{L}(i)$ are treated as oracle. We operate in this fashion because, in a distributed context, we can only modify the orderings of $\mathcal{L}(i)$, and must treat the received

orderings of $\mathcal{S}(i) \setminus \mathcal{L}(i)$ and $\mathcal{S}(j) \setminus \mathcal{L}(i)$ as fixed. When the same element e is selected by different robots in $\mathcal{S}(i)$ and $\mathcal{S}(j)$, we choose the element with a higher marginal value. If their marginal gains are the same, i.e., $\Delta_e^{(i)} = \Delta_e^{(j)}$, then conflict occurs.

The procedure that handles the received sets from neighbors is HandleMsg depicted in Algorithm 11. This procedure generally computes a new $\mathcal{S}(i)$ based on $\mathcal{S}(i)$ and $\mathcal{S}(j)$. To begin with, we first need to check if $\mathcal{S}(i) = \mathcal{S}(j)$. If so, robot i agrees with j and there is no conflict. If $\mathcal{S}(i) \neq \mathcal{S}(j)$, we need to use LocalGreedy depicted in Algorithm 12 to handle the potential conflict between $\mathcal{S}(i)$ and $\mathcal{S}(j)$. In general, from the i's perspective, we need to check the potential conflict for every $e \in \mathcal{S}(i)$ that also appears in $\mathcal{S}(j)$. Similarly, we also need to do this checking from j's perspective. Note that since both robot i and robot j have their own independent sets $\mathcal{S}(i)$ and $\mathcal{S}(j)$, there may be a case that the same element e is selected by different robots in i's and j's independent set. To simplify, we illustrate this procedure from i's perspective. If the relationship between $\Delta_e^{(i)}$ and $\Delta_e^{(j)}$ is as follows,

$$\Delta_e^{(i)} > \Delta_e^{(j)}, \forall\, e \in \{e' \mid e' \in \mathcal{S}(i), e' \in \mathcal{S}(j)\},$$

Then, e should be selected by $\phi_e^{(i)}$ as its marginal gain in $\mathcal{S}(i)$ is larger than that of its marginal gain in $\mathcal{S}(j)$. On the contrary, if the relationship between $\Delta_e^{(i)}$ and $\Delta_e^{(j)}$ is as follows,

$$\Delta_e^{(i)} \not> \Delta_e^{(j)}, \forall\, e \in \{e' \mid e' \in \mathcal{S}(i), e' \in \mathcal{S}(j)\},$$

Then, whether e is selected by $\phi_e^{(i)}$ or not depends on the conflict resolution mechanism. Here we use $\mathcal{X}$ to store all the elements currently in $\mathcal{S}(i)$, but will be removed from $\mathcal{S}(i)$. There are two cases in this scenario:

- If $\forall\, e \in \{e' \mid e' \in \mathcal{S}(i), e' \in \mathcal{S}(j)\}$ and $\Delta_e^{(i)} < \Delta_e^{(j)}$, we then update $\mathcal{X}$ as

$$\mathcal{X} \leftarrow \mathcal{X} \cup \{e \mid \forall\, e' \in \mathcal{S}(j) \setminus \mathcal{Y}, \phi_{e'}^{(i)} = \phi_e^{(i)}\},$$

  where $\mathcal{Y}$ contains all the elements $e \in \mathcal{S}(i)$ that have already been checked from i's perspective. In this case, the marginal gain $\Delta_e^{(i)}$ is smaller, which makes e to be removed by $\mathcal{S}(i)$. Meanwhile, if e is removed from $\mathcal{S}(i)$, all the elements that are selected by $\phi_e^{(i)}$ after selecting e will also be removed by $\mathcal{S}(i)$. These elements are in the set $\{e \mid \forall\, e' \in \mathcal{S}(j) \setminus \mathcal{Y}, \phi_{e'}^{(i)} = \phi_e^{(i)}\}$. Since this checking needs to be applied to $\forall\, e \in \mathcal{S}(i)$, then $\mathcal{X}$ will be updated multiple times. This operation follows the nature of the submodularity of $f(\cdot)$ that if the orderings of a selected set change, we no longer know their marginal gains.

- If $\forall\, e \in \{e' \mid e' \in \mathcal{S}(i), e' \in \mathcal{S}(j)\}$, $\Delta_e^{(i)} = \Delta_e^{(j)}$, and $\phi_e^i < \phi_e^j$, we then update $\mathcal{X}$ as

$$\mathcal{X} \leftarrow \mathcal{X} \cup \{e \mid \forall\, e' \in \mathcal{S}(j) \setminus \mathcal{Y}, \phi_{e'}^{(i)} = \phi_e^{(i)}\},$$

  where $\mathcal{Y}$ contains all the elements that have already been checked by the system. In this case, as there are two elements possess the same marginal gain, i.e., $\Delta_e^{(i)} = \Delta_e^{(j)}$, we then choose the element with a smaller robot index. This mechanism is consistent with Remark 7.

In above two cases, the set $\{e \mid \forall\, e' \in \mathcal{S}(j) \setminus \mathcal{Y}, \phi_{e'}^{(i)} = \phi_e^{(i)}\}$ contains all the elements that need to be removed from $\mathcal{S}(i)$. To make this process more clear, we give the following example. Suppose that

$$\mathcal{L}(i) = \{e_1, e_2, e_3, \ldots\},$$

and

$$\Delta_{e_1}^{(i)} \geq \Delta_{e_2}^{(i)} \geq \Delta_{e_3}^{(i)} \geq \dots.$$

We can also write this relationship as

$$f(e_1) \geq f(e_2 \mid \{e_1\}) \geq f(e_3 \mid \{e_1, e_2\}) \geq \dots,$$

from i's perspective. Then, if there is case that $\Delta_{e_1}^{(i)} < \Delta_{e_1}^{(j)}$, then this implies

$$\mathcal{L}(i) \leftarrow \mathcal{L}(i) \setminus \{e_1\} \text{ and } \mathcal{L}(j) \leftarrow \mathcal{L}(j) \cup \{e_1\}.$$

As a consequence,

$$f(e_2) \geq f(e_3 \mid \{e_2\}) \geq \dots,$$

does not hold anymore. So, this is reason why $\forall\, e \in \{e \mid \forall\, e' \in \mathcal{S}(j) \setminus \mathcal{Y}, \phi_{e'}^{(i)} = \phi_e^{(i)}\}$ should be removed.

Remark 9 (Conflict resolution consistency)**.** *The conflict resolution criterion can be arbitrary since different elements' contributions to the overall objective value are the same. However, it should be consistent as otherwise, there will be a case where an element is assigned to different robots back and forth.*

Remark 10 (Modular vs. submodular)**.** *If $f(\cdot)$ is modular instead of being submodular and there is a conflict, i.e., $e \in \mathcal{L}(i)$ and $e \in \mathcal{L}(j)$, we can simply update $\mathcal{S}(i)$ as $\mathcal{S}(i) \leftarrow \mathcal{S}(i) \setminus \{e\}$ if $\Delta_e^{(i)} < \Delta_e^{(j)}$, or $\Delta_e^{(i)} = \Delta_e^{(j)}$ and $\phi_e^{(i)} > \phi_e^{(j)}$ hold at the same time. A similar simplified operation can also be applied to $\mathcal{S}(j)$.*

Finally, $\mathcal{S}(i)$ is updated by removing $\mathcal{X}$ as

$$\mathcal{S}(i) \leftarrow \mathcal{S}(i) \setminus \mathcal{X}.$$

Those procedures should also be applied to $\mathcal{S}(j)$ to get a new $\mathcal{S}(j)$. Once an updated $\mathcal{S}(i)$ and $\mathcal{S}(j)$ is computed based on the procedure LocalGreedy and the conflict resolution mechanism, we can get

$$\mathcal{S} \leftarrow \mathcal{S}(i) \cup \mathcal{S}(j),$$

that contains all the potential elements for the final $\mathcal{S}(i)$.

Besides, as $a \in \mathcal{L}(a), \forall\, a \in \mathcal{A}$ were added in an decreasing order with respect to their marginal gains, we also reorder $e \in \mathcal{S}$ in decreasing order. Then, if $\mathcal{S} \in \mathcal{I}$ (or $\mathcal{I}, \mathcal{K}$), we can simply get the final $\mathcal{S}(i)$ as $\mathcal{S}(i) \leftarrow \mathcal{S}$. If not, we can build the final $\mathcal{S}(i)$ greedily. $\mathcal{S}(i)$ is first cleared out, and then updated as

$$\mathcal{S}(i) \leftarrow \mathcal{S}(i) \cup \{e\}, \forall\, e \in \mathcal{S},$$

if the constraint ("type 1" or "type 2") is satisfied until there is no $e \in \mathcal{S}$ available. Therefore, the procedure HandleMsg can process the received independent set $\mathcal{S}(j)$ while preserving the submodularity of $f(\cdot)$.

Finally, a flowchart showing all the procedures from the perspective of robot i is illustrated in Fig. 7.4.

7.3 Performance Analysis

This section first demonstrates the centralized algorithms for the proposed two types of problem constraints. We then analyze their relations with the proposed asynchronous distributed algorithms and show that the distributed algorithm achieves the same performance as their centralized version.

Matroid constraint ("type 1"): The centralized submodular maximization with matroid

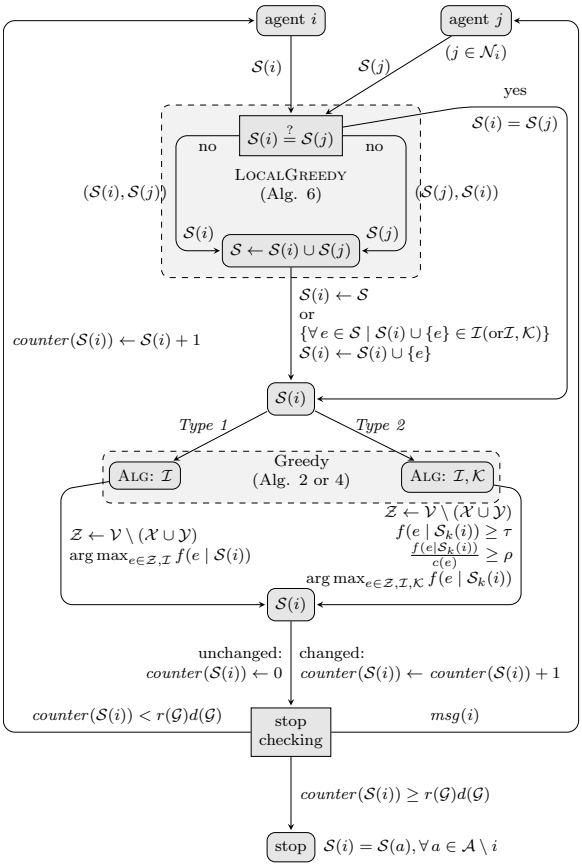


Fig. 7.4. The flowchart of the asynchronous distributed submodular maximization with matroid (and knapsack) constraints from the perspective of robot i.

constraint is shown in Algorithm 13. The performance is as follows.

Theorem 12 ([23])**.** *The centralized greedy algorithm for solving a submodular maximization problem with a matroid intersection yields a solution $\mathcal{S}$ with optimality bound:*

$$f(\mathcal{S}) \geq \frac{1}{|\mathcal{M}| + 1} f(O),$$

where $\mathcal{M}$ is the intersection of matroid constraints, $|\mathcal{M}|$ is the number of matroids in $\mathcal{M}$, and O is an optimal solution.

Matroid and knapsack constraint ("type 2"): The centralized greedy algorithm for solving

Algorithm 13 Centralized Submodular Maximization with Matroid Constraint.

Input: The inputs are as follows:

- matroid constraint $\mathcal{I}$;
- objective functions $f(\cdot)$.

Output: Set $\mathcal{S}$.

$\mathcal{S} \leftarrow \emptyset, \mathcal{Y} \leftarrow \emptyset, \mathcal{Z} \leftarrow \mathcal{V}$;
while $\mathcal{Z} \neq \emptyset$ **do**
  $e' \in \arg\max_{e \in \mathcal{Z}} f(e \mid \mathcal{S})$;
  if $\mathcal{S} \cup \{e'\} \in \mathcal{I}$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{e'\}$;
  end if
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{e'\}, \mathcal{Z} \leftarrow \mathcal{V} \setminus \mathcal{Y}$.
end while

the submodular maximization problem with matroid and knapsack constraints is shown in Algorithm 14. The performance is as follows.

Theorem 13 ([46])**.** *The centralized greedy algorithm for solving the submodular maximization problem with matroid and knapsack constraints yields a solution $\mathcal{S}$ with optimality bound:*

$$f(\mathcal{S}) \geq \frac{1}{(1+\epsilon)(|\mathcal{M}|+3)} f(O),$$

where $|\mathcal{M}|$ is the number of matroids in the matroid intersection, O is an optimal solution, $\epsilon \in (0,1)$ is a tunable parameter which is used to balance computation and accuracy as shown in Algorithm 14.

We are now ready to demonstrate that the distributed version of the centralized greedy algorithm will achieve the same theoretical guarantee as that of their centralized algorithms.

Lemma 3 (Set relations)**.** *If $\forall\, e \in \mathcal{V}$ is not selected by $i \in \mathcal{A}$ in the first round, i.e., $e \notin \mathcal{L}(i)$, then e will not be selected by i in the following rounds. In other words, $\mathcal{L}(i)$ in the last round*

Algorithm 14 Centralized Submodular Maximization with Matroid and Knapsack Constraints.

Input: The inputs are as follows:

- matroid constraint $\mathcal{I}$ and knapsack constraint $\mathcal{K}$;
- objective functions $f(\cdot)$.

Output: Set $\mathcal{S}$.

$k \leftarrow 0, p \leftarrow |\mathcal{M}|, M \leftarrow \max_{e \in \mathcal{V}} f(e)$;
for $(\rho = \frac{M}{p+1}; \rho \leq \frac{2f(\mathcal{V})}{p+1}; \rho \leftarrow (1+\epsilon)\rho)$ **do**
  $\mathcal{S}_k \leftarrow \emptyset, \mathcal{Y} \leftarrow \emptyset, \mathcal{Z} \leftarrow \mathcal{V}, k \leftarrow k+1$;
  $\tau \leftarrow M \leftarrow \max\{f(e) \mid \frac{f(e)}{c(e)} \geq \rho\}$;
  for $\tau = M_\rho; \tau \geq \frac{\epsilon}{|\mathcal{V}|} M_\rho; \tau \leftarrow \frac{1}{1+\epsilon}\tau$ **do**
    for $e \in \mathcal{Z}$ **do**
      if $\mathcal{S}_k \cup \{e\} \in \mathcal{I}, \mathcal{K}$ **then**
        if $f(e \mid \mathcal{S}_k) \geq \frac{\epsilon M_\rho}{|\mathcal{V}|} \& \frac{f(e|\mathcal{S}_k)}{c(e)} \geq \rho$ **then**
          $\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{e\}$;
        end if
      end if
      $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{e\}, \mathcal{Z} \leftarrow \mathcal{V} \setminus \mathcal{Y}$;
    end for
  end for
end for
$\mathcal{S} \leftarrow \arg\max_{\mathcal{X} \in \bigcup \mathcal{S}_k} f(\mathcal{X})$.

is a subset of $\mathcal{L}(i)$ in the first round.

Proof. The procedure LocalGreedy depicted in Algorithm 12 and the procedure HandleMsg depicted in Algorithm 11 are involved in the analysis. There exist two cases:

1. e is not selected by i and $j \in \mathcal{N}_i$ in the first round. That is,

$$e \notin \mathcal{L}(i) \quad \text{and} \quad e \notin \mathcal{L}(j), \forall\, j \in \mathcal{N}_i.$$

Then, we have

$$e \notin \mathcal{S}(i) \quad \text{and} \quad e \notin \mathcal{S}(j).$$

Since $\mathcal{S}(i)$ is updated as $\mathcal{S}(i) \leftarrow \mathcal{S}(i) \setminus \mathcal{X}$ through the LocalGreedy procedure, where $\mathcal{X}$ contains all the elements that may violate the submodularity of the problem, we have

$$e \notin \mathcal{S}(i) \setminus \mathcal{X}.$$

Then, it is true that $e \notin \mathcal{L}(i)$. This procedure also applies to j and results in $e \notin \mathcal{L}(j)$. Therefore, after i communicates with $j \in \mathcal{N}_i$ through the communication procedure HandleMsg depicted in Algorithm 11, we know that $e \notin \mathcal{L}(i)$ still holds. Therefore, $e \notin \mathcal{L}(i)$ holds after the LocalGreedy procedure and HandleMsg procedure.

2. e is not selected by i but is selected by $j \in \mathcal{N}_i$ in the first round. That is,

$$e \notin \mathcal{L}(i) \quad \text{and} \quad e \in \mathcal{L}(j), \forall\, j \in \mathcal{N}_i.$$

Using the same reasoning as above, we know that $e \notin \mathcal{S}(i)$ holds after the LocalGreedy procedure. However, there may be a case that $e \in \mathcal{S}(j)$ when $\mathcal{S}(j)$ is updated through this procedure. Then, after the HandleMsg procedure depicted in Algorithm 11, there may be a case that $e \in \mathcal{S}(i)$ since we have

$$e \in \mathcal{S}(j) \quad \text{and} \quad \mathcal{S}(i) \subseteq \mathcal{S}(i) \cup \mathcal{S}(j).$$

However, $e \notin \mathcal{L}(i)$ still holds as e is originally selected by j, which means $\phi_e^{(i)} = j$.

Thus, in both cases, if $e \notin \mathcal{L}(i)$ is true in the first round, then $e \notin \mathcal{L}(i)$ holds in the following rounds. □

Proposition 2. *The value $f(\mathcal{D})$, where $\mathcal{D} = \bigcup_{i\in\mathcal{A}} \mathcal{S}(i)$, obtained in the first round is an upper bound of $f(\mathcal{D})$ in the following rounds.*

Proof. This result is a direct consequence of Lemma 3. In the first round, since there is no interrobot messaging, we have the following. That is,

$$\mathcal{D} = \bigcup_{i\in\mathcal{A}} \mathcal{S}(i) = \bigcup_{i\in\mathcal{A}} \mathcal{L}(i).$$

From Lemma 3, we know that $\mathcal{L}(i)$ in the last round is a subset of $\mathcal{L}(i)$ in the first round. Besides, $f(\cdot)$ is non-decreasing. Therefore, $f(\mathcal{D})$ where $\mathcal{D} = \bigcup_{i\in\mathcal{A}} \mathcal{S}(i)$, is an upper bound among all the following rounds. □

Lemma 4 (Mutual exclusion)**.** *Consider the communication procedure HandlMsg and the local greedy procedures AlgMatroid and AlgMatroidKnapsack. It follows that upon termination, we have mutual exclusion $\mathcal{L}(i) \cap \mathcal{L}(a) = \emptyset, \forall\, i \neq a$, where $a \in \mathcal{A} \setminus i$.*

Proof. Since the conflicts are resolved through communications with neighbors, we consider the following case. Suppose that there is an element $e \in \mathcal{V}$ is selected by both i and $j \in \mathcal{N}_i$. That is,

$$e \in \mathcal{L}(i) \quad \text{and} \quad e \in \mathcal{L}(j).$$

In other words,

$$\phi_e^{(i)} = i \quad \text{and} \quad \phi_e^{(j)} = j.$$

So, there is a conflict between i and j with e. Then, the following scenarios must be considered.

1. $\Delta_e^{(i)} < \Delta_e^{(j)}$. After the LocalGreedy procedure, we have the following

$$\mathcal{L}(i) \leftarrow \mathcal{L}(i) \setminus \{e\}.$$

When i finishes message processing through HandleMsg procedure, since $\mathcal{S} \subset \mathcal{S}(i) \cup \mathcal{S}(j)$ and the final updated $\mathcal{S}(i)$ follows $\mathcal{S}(i) \subseteq \mathcal{S}$, we then have

$$e \in \mathcal{S}(i), \quad , e \notin \mathcal{L}(i), \quad \text{and} \quad \phi_e^{(i)} = j.$$

At the same time,

$$e \in \mathcal{S}(j), \quad , e \in \mathcal{L}(j), \quad \text{and} \quad \phi_e^{(j)} = j.$$

Through these procedures, i and j agree on e and $\mathcal{L}(i) \cap \mathcal{L}(j) = \emptyset$ holds.

2. $\Delta_e^{(i)} > \Delta_e^{(j)}$. In this case, it follows the same reasoning as above, except that $\mathcal{L}(i)$ keeps unchanged and $\mathcal{L}(j)$ is updated as

$$\mathcal{L}(j) \leftarrow \mathcal{L}(j) \setminus \{e\}.$$

Then, it follows that $\mathcal{L}(i) \cap \mathcal{L}(j) = \emptyset$ holds using the same reasoning as above.

3. $\Delta_e^{(i)} = \Delta_e^{(j)}$. In this case, if $\phi_e^{(i)} > \phi_e^{(j)}$, we should update $\mathcal{L}(i)$ as

$$\mathcal{L}(i) \leftarrow \mathcal{L}(i) \setminus \{e\}.$$

On the contrary, if $\phi_e^{(i)} < \phi_e^{(j)}$, we should update $\mathcal{L}(j)$ as

$$\mathcal{L}(j) \leftarrow \mathcal{L}(j) \setminus \{e\}.$$

After this process, it follows that $\mathcal{L}(i) \cap \mathcal{L}(j) = \emptyset$ holds using the same reasoning as above.

In all these scenarios, i and j agree on e after communication, and $\mathcal{L}(i) \cap \mathcal{L}(j) = \emptyset$. Therefore,

when i and j sends $\mathcal{S}(i)$ and $\mathcal{S}(j)$ to other neighbors $a \in \mathcal{A}\backslash(i \cup j)$, it implies that $a \in \mathcal{A}\backslash(i \cup j)$ will also agree with the selection of e. So, the above reasoning can be applied for $\forall\, a \in \mathcal{A}$. Thus, the mutual exclusion result $\mathcal{L}(i) \cap \mathcal{L}(a) = \emptyset, \forall\, i \neq a$, where $a \in \mathcal{A} \setminus i$ follows. □

Theorem 14 (Convergence and correctness)**.** *The proposed Algorithm 7 converges within at most $|\mathcal{V}|r(\mathcal{G})d(\mathcal{G})$ steps, and we have $\mathcal{S}(i) = \mathcal{S}(a), \forall\, a \in \mathcal{A} \setminus i$. Furthermore, the distributed solution achieves the same result as that of their centralized solution as shown in Algorithm 13 and Algorithm 14.*

Proof. From the procedure AlgMatorid and the procedure AlgMatroiKnapsack, we know that $e \in \mathcal{S}(a), \forall\, a \in \mathcal{A}$ are selected greedily based on their marginal gains while satisfying the constraints. At the second round, when every $a \in \mathcal{A}$ communicates with $a \in \mathcal{N}_a$, it follows that every robot $a \in \mathcal{A}$ will agree with e_1 within at most $r(\mathcal{G})d(\mathcal{G})$ steps, where

$$e_1 \in \arg\max_{e'} \{\Delta_{e'}^{(a)} \mid \forall\, e' \in \mathcal{L}(a), \forall\, a \in \mathcal{A}\}.$$

Then, after every robot's greedily selection, it again follows that every robot $a \in \mathcal{A}$ will agree with e_2 within at most $r(\mathcal{G})d(\mathcal{G})$ steps, where

$$e_2 \in \arg\max_{e'} \{\Delta_{e'}^{(a)} \mid \forall\, e' \in \mathcal{L}(a) \setminus \{e_1\}, \forall\, a \in \mathcal{A}\}.$$

Since $|\mathcal{V}|$ is fixed, there is an upper limit of the number of elements added to $\mathcal{S}(i)$. Therefore, the system will achieve an equilibrium, i.e., $\mathcal{S}(i) = \mathcal{S}(a), \forall\, a \in \mathcal{A}\backslash i$, within at most $|\mathcal{V}|r(\mathcal{G})d(\mathcal{G})$ steps. In the centralized greedy algorithms, Algorithm 13 and Algorithm 14 also select $e \in \mathcal{A}$ in decreasing order of the marginal gains. This implies that the orders of the distributed solution and the centralized solution are the same. This means Algorithm 7 will achieve the same performance as shown in Algorithm 13 and Algorithm 14 with the constraints are $\mathcal{I}$ and $\mathcal{I}, \mathcal{K}$. □

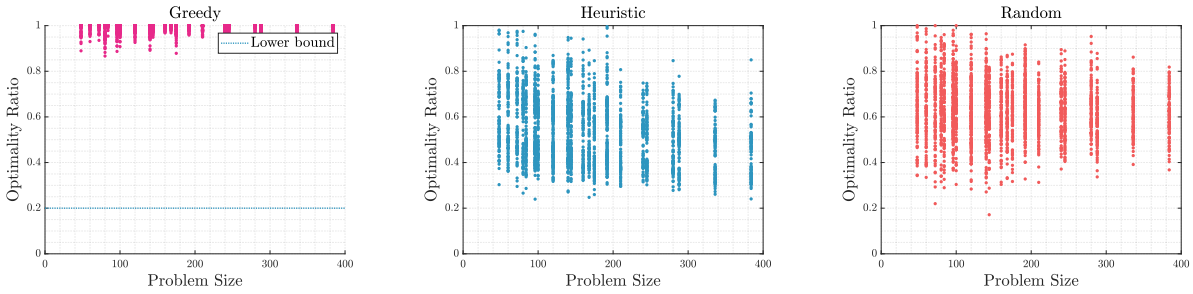
In reality, since each robot holds a different communication delay, the actual convergence steps are much smaller than this upper bound. Also, from the above analysis, we can see that every selected e will be agreed by every $a \in \mathcal{A}$ by at most $r(\mathcal{G})d(\mathcal{G})$ steps. Therefore, another alternative distributed strategy is to select one $e \in \mathcal{V}$ at every step and wait until all a's agree with this e. However, this is inefficient as the proposed framework can handle more than one agreement in $r(\mathcal{G})d(\mathcal{G})$ steps.

Due to the communication asynchronicity of $\mathcal{G}$, robot i may take different times to receive messages from $a \in \mathcal{A} \setminus i$ through neighbors $j \in \mathcal{N}_i$. Since we do not know if, with the new message, there will be another conflict, i.e., $\mathcal{L}(i) \cap \mathcal{L}(a) \neq \emptyset$, we need to decide how long i needs to wait before declaring the termination.

Theorem 15 (Termination)**.** *The Algorithm 7 stops when $\mathcal{S}(a), \forall\, a \in \mathcal{A}$ does not change for $2r(\mathcal{G})d(\mathcal{G})$ steps.*

Proof. It is easy to check whether $\mathcal{S}(a)$ changes. So, we focus on checking if there is a disagreement, i.e., $\mathcal{S}(i) \neq \mathcal{S}(a)$, in the message pipes. Robot i sends $\mathcal{S}(i)$ to other robots $a \in \mathcal{A}$ through neighbors $j \in \mathcal{N}_i$. Then, potential conflicts are resolved when i communicates with $j \in \mathcal{N}_i$. It takes $r(\mathcal{G})d(\mathcal{G})$ steps for $\mathcal{S}(i)$ to be broadcast to the farthest robot $a \in \mathcal{A} \setminus i$ in terms of communication distance. Again, $r(\mathcal{G})$ and $d(\mathcal{G})$ are the communication delay bound and the diameter. After that, it takes another $r(\mathcal{G})d(\mathcal{G})$ steps for i to get $\mathcal{S}(a)$ to confirm $\mathcal{S}(i) = \mathcal{S}(a)$. Therefore, i takes at most $2r(\mathcal{G})d(\mathcal{G})$ steps before i can declare the termination. On the contrary, if $\mathcal{S}(a)$ changes again during the $2r(\mathcal{G})d(\mathcal{G})$ steps communication, robot i's counter will be reset. Therefore, the result holds. □

Remark 11 (Dynamic Network)**.** *When the network structure $\mathcal{G}$ is dynamic while satisfying the bounded communication delay assumption and connectedness assumption in Section 7.1.1, the above results still hold. The procedures for getting the above results do not require the*



(a) Optimality ratio: greedy/optimal.

(b) Optimality ratio: heuristic/optimal.

(c) Optimality ratio: random/optimal.

Fig. 7.5. Weapon-target allocation: the optimality ratios of the proposed distributed greedy method, the heuristic method, and the random method against different problem sizes ($|\mathcal{A}| \times |\mathcal{E}| \times \lambda$). The horizontal line in (a) represents the 4-competitive lower bound of the distributed greedy algorithm.

structure to be fixed as long as the communications are available. However, we need to keep track of $d(\mathcal{G})$ *and update the stopping criterion* $2r(\mathcal{G})d(\mathcal{G})$ *once* $\mathcal{G}$ *is changed.*

7.4 Evaluations

In this section, we use multiple applications to demonstrate the effectiveness of the proposed algorithm through Monte Carlo simulations. The first application is a weapon-target task allocation problem. This application can be cast as a submodular maximization with a matroid intersection problem. In the second application, we extend the first problem by using an additional group constraint, where every robot has different budgets for different task groups. In the third application, we use the intermittent deployment problem to demonstrate the proposed method's performance when matroid and knapsack constraints are used to model different temporal and spatial budgets.

7.4.1 Weapon-Target Task Allocation

1) Comparisons between different methods

In this section, we use Monte Carlo simulations to test the performance of different methods using different problem instances.

Parameters settings: We run the problem with number of robots $|\mathcal{A}| \in \{4, \ldots, 8\}$, number of targets $|\mathcal{E}| \in \{4, \ldots, 8\}$, robot budget $\lambda_a \in \mathsf{rand}(1, |\mathcal{E}|)$, system budget $\lambda = \lfloor 0.8 \cdot |\mathcal{E}| \rfloor$, and target weight $w_e \in \mathsf{rand}(1, 10)$. The function $\mathsf{rand}(1, |\mathcal{E}|)$ is used to draw a sample randomly from 1 to $|\mathcal{E}|$, where $\mathcal{E}$ is the task set. To simulate the asynchronicity, each robot has a random communication delay that is sampled from $\mathsf{rand}(1, 3)$. We generate 25 different settings using those settings. Then, we generate different initial locations for robots and targets. For each of previous problem settings, we generate 100 different cases of randomly located robot-target pairs. Therefore, we finally run the simulations 2500 times using different problem instances.

Compared algorithms: In the experiments, the compared methods are as follows.

- The proposed distributed algorithm (Algorithm 7). We refer to this as "*greedy*".
- A brute-force method, which is used to generate an optimal solution. We refer to this algorithm by "*optimal*".
- A heuristic method, which is used to generate solution greedily for each robot from the first robot to the last while respecting system constraints. For example, robot 1 first generates its solution based on its local information. Then, robot 1's solution is delivered to robot 2 for making its decision and so on. Note that both individual-level and system-level constraints are respected by each robot. We refer to this by "*heuristic*".
- A random approach, which selects targets for different robots randomly while respecting system constraints. We refer to this by "*random*".

We compare the performance of different methods based on the collected utilities. We will also compare the proposed asynchronous distributed algorithm with the centralized algorithm and demonstrate the convergence of the system.

Performance: The optimality ratio comparisons between different methods and the distributed and centralized algorithm comparisons are as follows.

For each problem instance, we first collect the utilities of different methods. Then, we calculate the optimality ratios of different methods, defined as the ratio between different methods and the optimal method. The optimality ratios are reported in Fig. 7.5 under different problem instances. We also define each instance's problem size as: $|\mathcal{A}| \times |\mathcal{E}| \times \lambda$. Then, we draw the optimality ratios of different methods against different problem sizes, as shown in Fig. 7.5. Since the number of matroid constraints is $|\mathcal{M}| = 3$, we have the theoretical optimality bound of this problem as $f(\mathcal{S}) \geq \frac{1}{4} f(O)$ by referring Theorem 12. Again, $\mathcal{S}$ is the solution of the proposed distributed solution, and O is an optimal solution. We also plot the lower bound of the proposed distributed greedy method. From the results shown in Fig. 7.5, we have the following observations. The distributed greedy method performs much better than the other two methods in most cases as most of its optimality ratios are located on the top part of Fig. 7.5(a). The proposed algorithm also follows a 4-competitive bound. On the contrary, the other two methods can perform arbitrarily bad in some instances.

We then generate the statistics of the optimality ratios of different methods. In Fig. 7.6, the histograms of the optimality ratios of the compared methods are stacked vertically to illustrate the differences. In the distributed greedy approach, most cases have a 0.95 optimality ratio or better, and the average is 0.984. In the heuristic method, the average optimality ratio is 0.576. The random method has an average optimality ratio of 0.641. In Fig. 7.7, we demonstrate the box-plot and violin-plot using the generated utilities from the compared methods. This result shows that the average utilities of the distributed greedy approach and

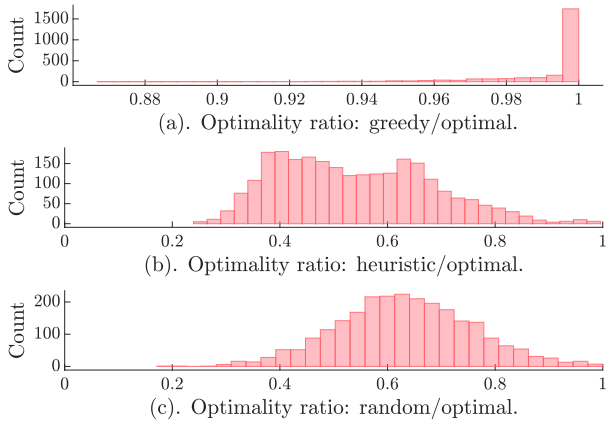


Fig. 7.6. Weapon-target allocation: the optimality ratio histogram comparisons between the distributed greedy method, the heuristic method, and the random method.

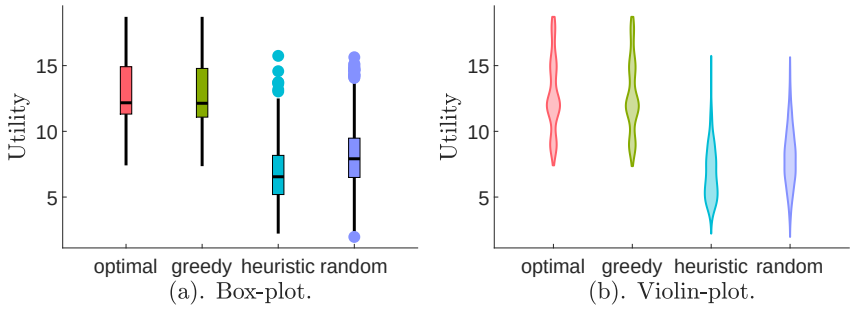


Fig. 7.7. Weapon-target allocation: the statistics of the utilities of the brute-force method, the proposed distributed greedy method, the heuristic method, and the random method. (a). Box-plot. (b). Violin-plot.

the optimal solution are close. Though the heuristic and random methods can generate good results in some instances, the average utilities of those two methods are much lower than the distributed greedy method.

2) Distributed vs. centralized

To demonstrate the convergence of the proposed distributed algorithm, we run the problem using different settings and then compare the performance against the corresponding centralized algorithm.

In the comparison, we run the problem with 20 robots and 25 targets. Each robot's budget λ_a is randomly selected from $\mathsf{rand}(1, 20)$. The system budget λ is set to 20. Since every robot $i \in \mathcal{A}$ has an independent rate 1 Poisson clock per Assumption 4, we then record the system's utility $f(\mathcal{S}(i))$ from each robot's perspective after every *round*. Note that we use a global clock the monitor this convergence process. As the independent set contains all the elements selected as far, we can also write the utility as $f(\mathcal{S}(i)) = \sum_{e \in \mathcal{S}(a)} \Delta_e^{(i)}$. Note that the round is defined as a run of RobotRun as shown in Algorithm 7. The converge result of different robots is shown in Fig. 7.8, containing 20 lines for 20 different robots. The x-axis represents the index of rounds, and the y-axis represents the utility of different robots at each round. Meanwhile, the objective function value of the centralized algorithm is plotted as the top reference line in Fig. 7.8. From this results, we see that the independent set $\mathcal{S}(i)$ for all $i \in \mathcal{A}$ converges after 31 rounds, and the system objective function value $\sum_{e \in \mathcal{S}(a)} \Delta_e^{(i)}, \forall\, a \in \mathcal{A}$ converges to the same result as that of the centralized solution.

7.4.2 Weapon-Target Task Allocation with A Group Budget

In this problem, we extend the first application by using an additional task group constraint while keeping the other settings and the comparison methods the same.

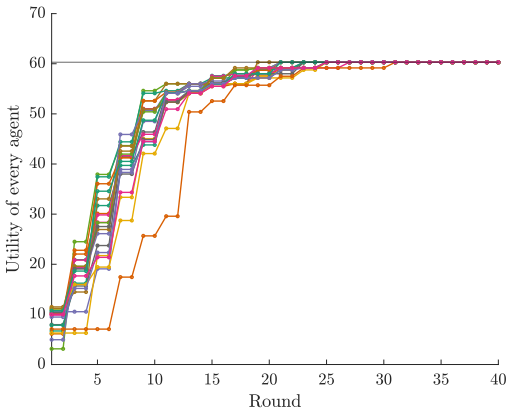
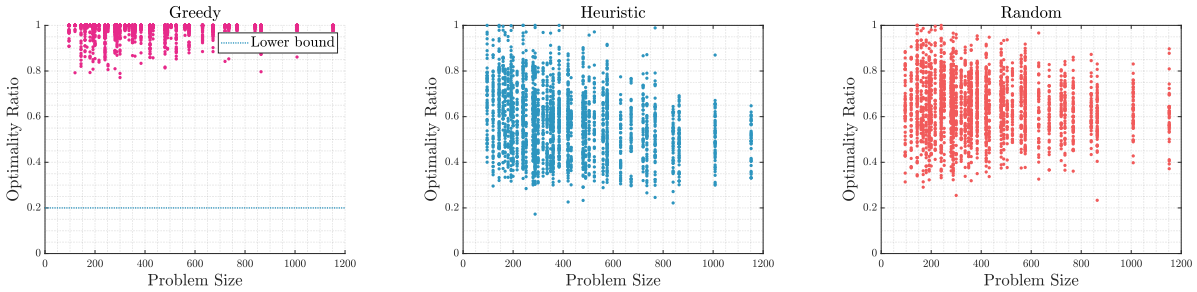


Fig. 7.8. Weapon-target allocation: the objective value calculated from each robot's perspective after each round. The number of robot is $|\mathcal{A}| = 20$, the number of tasks is $|\mathcal{E}| = 25$, and the number of the system budget is $\lambda = 20$. The upper reference line is the system objective function value of the centralized method.



(a) Optimality ratio: greedy/optimal.

(b) Optimality ratio: heuristic/optimal.

(c) Optimality ratio: random/optimal.

Fig. 7.9. Weapon-target allocation with a group constraint: the optimality ratios of the proposed distributed method, the brute-force method, a heuristic method, and a random method against different problem sizes ($|\mathcal{A}| \times |\mathcal{E}| \times |G| \times \lambda$). The reference line represents the 5-competitive bound of the distributed greedy algorithm.

Ground budget constraint: Due to the heterogeneity of targets, different targets may belong to different groups, e.g., ground target group, aerial target group, etc. Also, the system may have different processing abilities for different task groups. We, therefore, divide the ground set $\mathcal{V}$ into different groups according to the type of task associated with each element $e \in \mathcal{V}$. Then, we apply this setting to each problem instance for the simulations. We denote by $\mathcal{V}_i$ the sub-ground related to group g. Thus, $\mathcal{V}_i \subseteq \mathcal{V}$. In the evaluations, the system's processing budgets for different groups are set to $\mathsf{rand}(1, |\mathcal{V}_i|)$. For example, if the ground set is $\mathcal{V} = \{e_1, e_2, \ldots, e_5\}$, we can divide $\mathcal{V}$ into different groups related to different types of tasks, e.g., the group related to aerial target is set to $\{e_1, e_3, e_5\}$ and the group related to ground target is set to $\{e_2, e_4\}$. Note that every $e \in \mathcal{V}$ is an robot-task pair. The reader is referred to [22] for a more comprehensive overview. Here, the system group budget is formally defined as follows.

- This group budget constraint of the system is modeled as a matroid constraint. That is, $\mathcal{M}_{14} = (\mathcal{V}, \mathcal{I}_{14})$, where

$$\mathcal{I}_{14} = \{\mathcal{X} \subseteq \mathcal{V} \mid |\mathcal{X} \cap \mathcal{V}_i| < \lambda_g, \forall\, i = 1, 2, \ldots\}.$$

  where λ_i is the system budget for group i.

Performance: Next, we demonstrate the performance of optimality ratio comparisons and the comparison between the distributed and the centralized algorithm. The problem size is defined as $|\mathcal{A}| \times |\mathcal{E}| \times |G| \times \lambda$. We then run Monte Carlo simulations for 2500 times using the settings as stated in the first application. The optimality ratios of different methods against different problem sizes are shown in Fig. 7.9. This problem is also a submodular maximization problem with $|\mathcal{M}| = 4$ matroid constraints. We then have the optimality lower bound of this problem $f(\mathcal{S}) \geq \frac{1}{5} f(O)$ per Theorem 12. Again, $\mathcal{S}$ is the solution of the

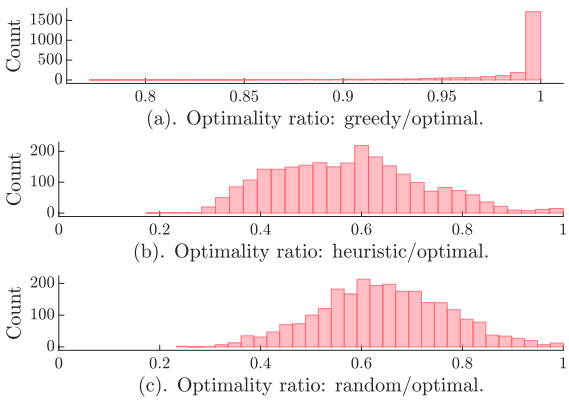


Fig. 7.10. Weapon-target allocation with a group constraint: the histogram of the optimality ratios of the greedy method, a heuristic method, and a random method.

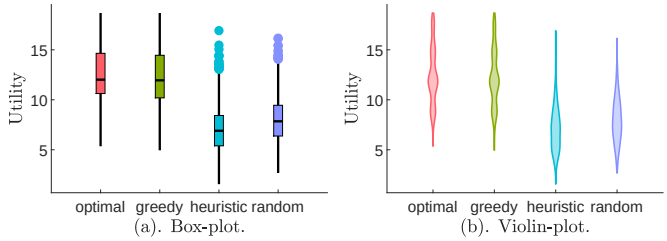


Fig. 7.11. Weapon-target allocation with a group constraint: statistics of the utilities of the proposed distributed method, the brute-force method, a heuristic method, and a random method. (a). Box-plot. (b). Violin-plot.

proposed distributed solution, and O is an optimal solution generated by using the brute-force method. In Fig. 7.10, we summarize the statistics of the utilities of different methods using histograms. The proposed distributed greedy method has close to optimal solutions for most cases compared to the other two methods. Similarly, the results shown in Fig. 7.11 also illustrate the same results when we use population statistics. Finally, in Fig. 7.12, we demonstrate the system's convergence performance by recording the utilities' change from every robot's perspective. In this result, the number of robots is $|\mathcal{A}| = 20$, the number of tasks is $|\mathcal{E}| = 25$, the number of system budget is $\lambda = 20$, and the number of task groups is 2.

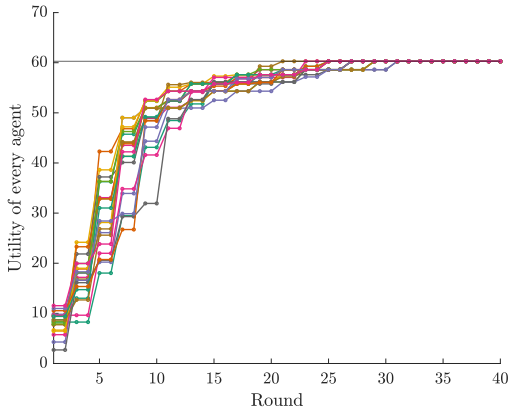


Fig. 7.12. Weapon-target allocation with a group constraint: The objective function value maintained by each robot $a \in \mathcal{A}$ after each round. In the problem, $|\mathcal{A}| = 20, |\mathcal{E}| = 25, \lambda = 20$, and the number of task groups is 2. The constraints are $\mathcal{M}'_1, \mathcal{M}'_2, \mathcal{M}'_3$, and $\mathcal{M}'_4$. The upper reference line is the overall objective value from the centralized solution.

7.4.3 The Intermittent Deployment Problem

Problem settings: In this problem, we use the proposed intermittent deployment problem to test the performance of the distributed algorithm. Since we are given the predicted mean and covariance in the problem setting, we first briefly demonstrate how to get those predictions. In general, we use a 100×100 Gaussian mixture model (GMM) [48] to generate the field that needs to be monitored. The GMM output is $\varphi(x, y, t) = \sum_{i=1}^{5} w_i(t) b_i(x, y) = \mathbf{w}(t)^\top \mathbf{b}(x, y)$, where (x, y) is a 2D location, t is the time index, $b_i(x, y) \in \mathbb{R}$ is the ith basis function, and $w_i(t) \in \mathbb{R}$ is the corresponding basis weight at time t. Here we use 5 bases, and the 2D locations of those bases are randomly selected from this 100×100 field. The corresponding initial weights are $\{5, 5, 3, 8, 4\}$. Also, $\mathbf{w}(t) \in \mathbb{R}^5$ is the stacked weights at time t and $\mathbf{b}(x, y) \in \mathbb{R}^{5\times 2}$ is the stacked bases for the location (x, y). The weight change of each basis function follows a 1D Gaussian process [104]. We then run the GMM model from $t = 1$ to $t = 10$, and then randomly pick 20 measurements from each time to form a training dataset $\mathcal{D}$. Next, we use a Gaussian process (GP) regression to model the environment and make predictions for the future using the training dataset. Note that we use squared

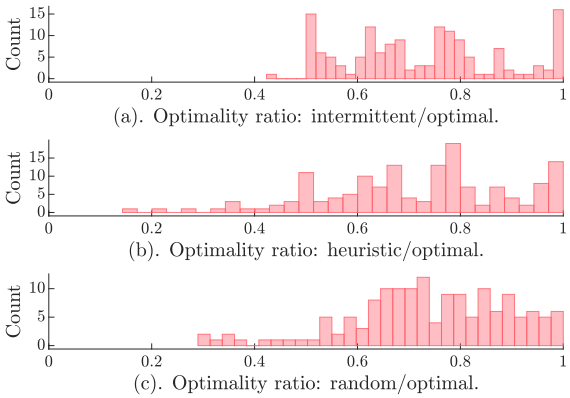


Fig. 7.13. The intermittent deployment problem: the histogram of the optimality ratios of the greedy method, a heuristic method, and a random method.

exponential kernel for the separable temporal and spatial properties. After that, we can use GP to make predictions from time $t = 11$ to $t \in \{15, \ldots, 20\}$. Meanwhile, we can get an associated covariance Σ for each time horizon. Next, we set up the system for the decision-making part. We randomly select the number of robots $|\mathcal{A}|$ from the set $\{5, \ldots, 8\}$. The traveling cost of every robot $a \in \mathcal{A}$ is the multiplication of the basic traveling cost and a weight factor $w_r \in \mathsf{rand}(1, 1.5)$ that models the importance of different robots. Finally, the communication delays of different robots are sampled from $\mathsf{rand}(1, 5)$, which are used to model the communication asynchronicity. The total number of deployments λ is selected from the set $\{5, \ldots, 10\}$. In the simulation, we first collect a training dataset from a dynamic environment. Here we use a 2D Gaussian mixture model (GMM) to generate the initial dataset. We collect samples from this environment for the first few times and then train a GP using these data. After that, we make predictions for the future outputs and compare the proposed method with other methods.

Performance: In this problem, we compare the proposed method "intermittent" with a "optimal", a "heuristic", and a "random" method. In the heuristic method, the robots are deployed with a fixed time interval and fixed locations. The optimality ratio comparisons and the comparison between the distributed and the centralized algorithm are as follows.

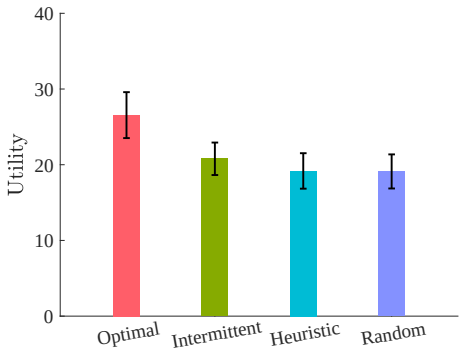


Fig. 7.14. The intermittent deployment problem: the utility comparison of different deployment policies.

The optimality ratio comparison of different methods is shown in Fig. 7.13. Also, the utility comparison between different methods is shown in Fig. 7.14. This problem is a submodular maximization with 3 matroid constraints and 1 knapsack constraint. We therefore have the optimality bound of this problem as $f(\mathcal{S}) \geq 15.2\% f(O)$ through Theorem 13 since $|\mathcal{M}| = 3$. Again, $\mathcal{S}$ is the solution of the proposed distributed solution, and O is an optimal solution from the brute-force method. From the optimality ratio comparisons, we observe that the intermittent deployment is able to maintain the theoretical guarantee while having a high average utility.

7.5 Conclusions

In this work, we proposed an asynchronous distributed algorithm for solving the general submodular action selection problems with matroid and knapsack constraints. The proposed algorithm can be used in distributed combinatorial robotics scenarios where robots can only share their estimates with neighbors and need to work together to fulfill system goals. In particular, we first introduced two robotics applications and then extracted a general problem formulation. We then demonstrated and proved that the proposed asynchronous

distributed algorithm converges to the same result as the centralized algorithm and has performance guarantees. Using Monte Carlo simulations, we demonstrated the performance of the distributed algorithm and also the convergence performance.

Chapter 8

Intermittent Deployment for Large-Scale Multi-Robot Forage Perception: Data Synthesis, Prediction, and Planning

Pasturelands are an integral part of agricultural production in the United States. To take full advantage of the forage resource and avoid environmental degradation, pastureland must be managed optimally. This chapter focuses on the question of how to deploy robot teams to sense and model physical processes over varying timescales. The goal of this work is to develop a new integrated pipeline for the long-term deployment of heterogeneous robot teams grounded in the problem of autonomous monitoring in precision grazing to improve land productivity. By using the proposed pipeline in grassland ecosystem management, we will have a better understanding of the physical environment while respecting energy budgets.

Introduction

Grasslands provide many ecosystem services such as livestock production, wildlife habitat, water infiltration, and carbon sequestration [105]. Consequently, monitoring the health

and vigor of grasslands is vital for informing management decisions to protect or optimize these ecosystem services [106]. Sward or canopy height data provide valuable insights into the productivity, habitat value, or maturity of grasslands. When sward height data are monitored over time, changes in sward height can indicate whether a grassland is deteriorating, maintaining its vigor, or becoming more productive. In agricultural systems, monitoring height data can inform decisions about the appropriate timing and intensity of grazing in order to meet economic and ecological goals.

Traditional methods of measuring aboveground height or aboveground biomass rely on labor-intensive methods [107]. For height, this necessitates measuring canopy height by hand using a meter stick or Robel pole [108]. Aboveground biomass measurements often consist of destructive harvest of forage. Samples are usually cut by hand from a quadrat or frame, bagged, then dried in a forced-air forage oven until reaching a constant weight, known as the dry matter. Depending on the size of the pasture or scope of the monitoring project, height and biomass sampling may involve dozens or hundreds of such samples to ensure that the collected data represents the entirety of the pasture or landscape being monitored.

Advancements in proximal sensing technologies can provide accurate measurements of height and biomass predictions faster and over larger areas than these labor-intensive traditional methods. However, regular field measurements from pastures through remote sensing methods such as Unmanned Aerial Vehicles (UAVs) are constrained by multiple factors such as limited spatial coverage and low frequency of UAV deployments. Moreover, adverse weather conditions and other resource constraints also contribute to limited deployment and consequently insufficient field measurements of the pasture required to plan grazing activities. In spite of these limitations, precision agriculture involving the use of UAVs to monitor the growth of crops is a promising approach to covering large areas in reasonable times. Analyzing point clouds obtained from the LiDAR attached to UAVs can help to determine the spatial

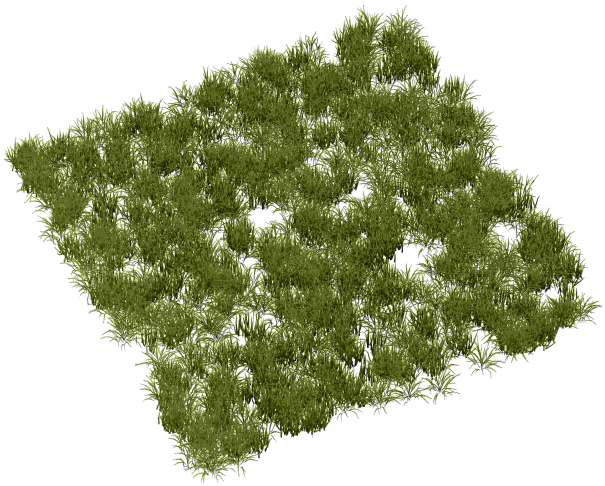


Fig. 8.1. A patch (2m ×2m) of the simulated pastureland with a density of 250 grass models per square meter.

distribution of plants and growth in different regions of the farm and consequently help the farmer focus their resources in regions where they are required the most. Developing prediction models for pastures that work within the limitation of UAV deployment schedules is a promising approach and helps alleviate resource-limited field measurements. With an intermittent deployment coupled with a tightly integrated prediction and planning model, this approach would generate maximum efficiency for livestock grazing and pasture recovery processes.

With the above motivation, this chapter explores the use of autonomous robots to facilitate environmental monitoring for improving land productivity. At a high level, our proposed pipeline works as follows. We first use historical data to synthesize a 3D dynamic field to simulate the spatiotemporal environmental process of the site that needs to be monitored (Section 8.3). For illustration, a small patch (2m ×2m) of the generated pasture is shown in Fig. 8.1, where the density is 250 grass models per square meter. We then use this data to train neural networks to learn the dynamics of this field (Section 8.4). Then, an intermittent deployment policy is designed for multi-robot teams (UAVs in our case) using the future

pasture height predictions while respecting system budgets (Section 8.5). After that, we simulate a pastureland environment in ROS Gazebo [109] to test the performance of the synthesized data (Section 8.6). Finally, we evaluate the performance of each aspect of our solution and compare it to competing methods (Section 8.7).

In order to synthesize appropriate training data for high-quality predictions, historical data were generated using the expert-informed Agricultural Production Systems sIMulator (APSIM) Next Generation in Section 8.3. APSIM is designed to model long-term agricultural production in a variety of systems [110]. Using historical meteorological data from three sites in Iowa, APSIM simulated 30 years of tall fescue (Schedonorus arundinacea) pasture dry matter production for each site. Simulated pasture yield was then used to generate average pasture height data based on the equation reported by Schaefer and Lamb [111] describing the relationship between pasture green dry matter and LiDAR-measured pasture height. Based on the historical average pasture height data, we then use a Gaussian mixture model (GMM) to simulate the dynamics of this field over the desired monitoring horizon in Section 8.3.

In this chapter, we aim to propose an entire pipeline for a precision agriculture application with the help of robots. The pipeline includes multiple parts: prediction, estimation, planning, evaluation, etc. Therefore, before conducting a real-world validation, we need to validate the effectiveness of the integration of different parts, as performing a real-world verification is a long process. Therefore, this chapter will use a high-fidelity world that simulates the growth of a pastureland environment with the help of high-fidelity grass models and historical data.

Even with a sufficient training data set based on expert-informed historical data, developing an effective prediction model for estimating pasture growth is challenging due to changes incurred by the climate and the spatiotemporal characteristics of the growth. Previous studies for forecasting spatiotemporal dependencies have been based on conventional approaches. These methods require a complex and meticulous simulation of the physical environment for a

particular application. Instead, learning-based models provide increased flexibility in tackling the difficult spatiotemporal sequence prediction problem that large-scale forage monitoring poses. To address this problem from a deep learning perspective, we use our proposed long short-term memory (LSTM) and convolutional neural network (CNN) based architecture [112] to model the pasture growth forecasting problem in Section 8.4.

Finally, with high-quality pasture growth predictions, a multi-robot deployment policy can be designed to guide future field measurements, as detailed in Section 8.5. Our proposed deployment policy aims to maximize the sum of collected information (uncertainty) from the environment while considering the system's waiting penalties and energy constraints. In general, we formulate this deployment problem as a submodular maximization problem with matroid constraints [8]. The energy constraints will be formulated as matroid constraints, while the collected information and the waiting penalties will be part of our objective function. Additionally, as the deployment policies gather field measurements, the prediction model can be updated iteratively to generate more accurate estimates of future pasture growth. Through an optimized prediction and deployment model, we show that UAV deployments for pasture monitoring can be scheduled based on the prediction model instead of at regular intervals, effectively reducing the required number of deployments. This *need-based intelligent* deployment policy substantially reduces the cost of gathering field measurements and effectively allows resource-constrained enterprises to manage pasture grazing more effectively.

Beyond the forage monitoring problem, the proposed pipeline can be mapped to other large-scale monitoring problems for sufficiently slow spatiotemporal processes where intermittent deployment is reasonable. That is, the data synthesis – neural network training/prediction – intermittent multi-robot deployment pipeline is quite general, given access to expert-informed data as a starting point. For example, in ocean monitoring applications [31, 113], we can

use the proposed pipeline to generate and refine deployment policies given the availability of high-quality models of aquatic processes.

Contributions

This is a collaboration work, and the contributions of this thesis are as follows:

- We propose an integrated pipeline[1] to simulate and solve an important problem in the precision agricultural space, forage monitoring, including data synthesis, prediction, planning, and perception (Section 8.2).
- We demonstrate how to exploit historical agricultural data to synthesize a pastureland environment in a manner that accommodates both neural network training and rapid prototyping in a simulation environment (Section 8.3).
- We introduce a novel intermittent deployment policy that integrates neural network-based predictions while considering budgets to deploy robots for autonomous environmental monitoring (Section 8.5).

A preliminary portion of this work appeared in [4, 112]. In this work, we focus on building a general pipeline for precision agriculture applications, which first focuses on utilizing the proposed deep learning networks [112] to make predictions and then on how to utilize those predictions to make robotic deployments with different budget constraints. We will also be introducing the general idea and settings of the scalable spatiotemporal learning part (Section 8.4) and the high-fidelity pastureland construction and perception part (Section 8.6) for the sake of the completeness of this work.

[1] https://github.com/precision-grazing/project

8.1 Related work

Predictive modeling of agricultural systems can provide helpful information on their long-term resilience and productivity. Predictive models have been developed for many crops and regions, tested extensively, and refined for increased predictive accuracy for commodities such as corn, soybeans, wheat, and rice, to name a few. We select APSIM as the optimal modeling program for our research, as the program has been used previously for pasture modeling in a variety of contexts [114, 115];

The problem of predicting evolving pastureland over time can be tackled with conventional methods such as Gaussian processes (GPs) [43], which focus on stochastic Gaussian processes to model the regression for different pasture heights or observations. This method is also known as Kriging [43]. GPs are non-parametric methods by defining the internal relations between observations [116]. Similarly, Gaussian Markov Random Fields (GMRFs) are often used to model spatial environmental fields [43]. These conventional methods are suitable for problems with significant prior knowledge of the environment that needs to be monitored [117, 118]. The historical data used in this chapter can give us a general trend of the average height change of the field. However, as we must make predictions on large scales, this prior is insufficient for us to generate a reasonable conventional model for the entire pastureland environment (not to mention the issues with computational scaling). Moreover, as the growth patterns may differ across the field, conventional models are not as flexible for modeling the heterogeneity of growth patterns spatially and temporally. We, therefore, sought to apply a neural network-based method to tackle this prediction problem when we have a large dataset for modeling the heterogeneity of the environment.

Specifically, we implement a framework that integrates a neural network-based encoder-decoder architecture to learn the historical data's underlying patterns over time for future

predictions. The problem of predicting future pasture heights is analogous to video frame prediction [119], with the key challenge in our case lying in predicting the growth of pasture surfaces. We utilize the recent developments in ConvLSTM applications and propose a novel prediction architecture that is particularly effective in predicting the rapidly changing dynamics of the pasture over long horizons. We achieve this through the use of recurrent encoder-decoder networks based on ConvLSTMs over different resolutions of the pasture to effectively capture different features and dependencies of the pasture dynamics. Moreover, by using appropriate processing described in Section 8.4 over raw pasture data, our pasture terrain forecasting technique can scale to any terrain size.

Standard deep learning-based models do not capture model uncertainty. Unlike regression problems, where a model can output a predictive probability, we need to extrapolate prediction uncertainty from training data as we deal with a sequence-to-sequence prediction problem. Bayesian probability theory has been used in deep learning to deal with uncertainty [120–125], where the weights of the neural network are defined as distributions. Bayesian neural networks (BNN) [126, 127] are more robust to over-fitting. However, they add significant computation complexity. Sampling-based and stochastic variational inference methods have been used to approximate Bayesian neural networks [128–130]. Similar to BNN, these methods incur high computational costs without additional benefits to improving accuracy. An alternative approach [131] with minimal computational and model complexity in deep learning models was proposed through the use of Dropouts [132]. The authors show that any neural network with arbitrary depth and non-linearities, modeled with dropouts behind every layer, is equivalent to the approximation of a probabilistic deep Gaussian process [133]. In our previous work [112], we integrate this concept of uncertainty estimation from dropout over our architecture to provide the necessary uncertainty maps for multi-robot monitoring from a typical machine learning perspective. Whereas in this work, our focus

is on integrating a combinatorial optimization-based planner with proposed deep learning-based predictions [112] and performing a deep evaluation of the entire pipeline with realistic simulations. From the multi-robot deployment perspective, to estimate the evolving processes of pastureland environments more efficiently, we cannot deploy robots (UAVs) to collect observations frequently because it is not energy efficient. Those types of observations can be point clouds, heightmaps, sonar data, etc. Meanwhile, the deployment strategy should be generated based on environmental information instead of being artificially defined. This intermittent idea can also be found in other robotics applications. In [134], the robots in a team are designed to communicate intermittently while working together to explore an environment. In [35], the authors use time windows to model the availability of robots at different times in task allocation applications. Therefore, robots are not required to work continuously. The idea of intermittence is contrary to that of persistence, where robots are required to work continuously to fulfill different tasks [135]. In [136], the authors proposed a deep learning-based method for combining environmental prediction and path planning, where spatial path planning is the primary concern in the chapter. In [137], the authors investigated a sampling-based path planning method for variance reduction. Similarly, in [138], the author used an adaptive sampling strategy for reducing entropy generated from GP modeling. In [139], a GMRF-based method was proposed for spatial predictions.

Since the monitored environment evolves spatially and temporally, we need to utilize the predicted environmental information to make a spatiotemporal deployment plan. In [3], we use a partially observable Markov decision process (POMDP) to model the dynamics of an environmental process. Then, a submodular objective function is applied to model the false alarm and delay cost. This method works only when the process models are known and can be modeled as POMDPs. In [1, 4], we use non-parametric Gaussian processes [43] to model the dynamics of a monitored environmental process and then use mutual information

as a metric to guide our deployments. Meanwhile, matroids are used to model the budget constants. Generally speaking, matroids [8, 14] can be used to model the independence in constraints, which can be found in many robotics applications, e.g., task allocation [22], deployment planning [2, 5], probabilistic security in multi-robot systems [140], etc. We will also use this tool to model the independence of different constraints in the deployment strategy in this work. In general, the environmental modeling methods used in our previous works are conventional GP-based methods, where a prior of the entire environment is needed to utilize historical data. While in this chapter, the environment modeling method is a data-driven approach, which builds the dynamics of the environment through a historical dataset where an exact process model is not required. Moreover, the environment modeling and the deployment policy generation are highly connected in this chapter. Finally, as the deployment policies gather more measurements, the proposed data-driven prediction model can be updated accordingly for better future predictions.

8.2 Problem Formulation

Consider a spatiotemporal forage process evolving in a 2D pasture environment (which we model rigorously in the sequel). In simple terms, we are interested in determining forage height for any location in the pasture environment over a long time horizon by deploying multi-robot teams. To fulfill this goal, we divide our efforts into two tasks: forage process prediction and multi-robot planning. In the following, we outline the fundamentals of these tasks and conclude with a concrete problem statement.

Prediction Assume that we are given historical 2D heightmaps $\mathbf{X}_t \in \mathbb{R}^{M \times N}$ of a pasture field at different times, where t is the associated time index, and M, N are the width and

the length of the pasture heightmap, which correspond to the discretization resolution. This resolution will be used for both prediction and planning problems. Denote by $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid t \in \mathcal{T}_x\}$ the historical dataset containing all the historical measurements, where $\mathcal{T}_x$ contains all the time indexes associated with each $\mathbf{X}_t \in \mathcal{X}$. The extracted height of the pasture in location (x, y) with respect to $\mathbf{X}_t$ is denoted by $\mathbf{X}_t(x, y)$. Our first goal is to train a neural network to predict future pasture heights $\bar{\mathbf{Y}}_t$ for the prediction horizon $\mathcal{T}_y$ using the historical dataset $\mathcal{X}$. This prediction process is modeled as

$$\left(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2\right) \leftarrow \Theta\left(\mathcal{X}, \mathbf{W}\right), \quad \forall\, t \in \mathcal{T}_y, \tag{8.1}$$

where Θ is our neural network model, $\mathbf{W}$ is the set of parameters of the model, $\bar{\mathbf{Y}}_t \in \mathbb{R}^{M\times N}$ is the predicted heightmap at t, and $\bar{\sigma}_t^2 \in \mathbb{R}^{M\times N}$ is the corresponding variance at t. Also, we denote by $\bar{\mathcal{Y}} = \left\{\bar{\mathbf{Y}}_t \in \mathbb{R}^{M\times N} \mid \forall\, t \in \mathcal{T}_y\right\}$ the prediction set that contains all of the predicted heightmaps for the prediction horizon $\mathcal{T}_y$, and denote by $\bar{\Sigma} = \left\{\bar{\sigma}_t^2 \in \mathbb{R}^{M\times N} \mid \forall\, t \in \mathcal{T}_y\right\}$ the corresponding variance set. The details will be specified in Section 8.4.

Planning Based on the predicted heightmaps $\bar{\mathcal{Y}}$ and variance maps $\bar{\Sigma}$, we seek a multi-robot deployment strategy to collect data, reinforce our predictions, and ultimately make better pastureland management decisions. To determine a deployment strategy, we first need to build the ground set $\mathcal{V}$, which contains all possible deployment decisions for robots over space and time. Consider a team of robots available to deploy over the time horizon $\mathcal{T}_y$, the prediction horizon. The family of all available locations is denoted by $\mathcal{P}$. Thus, we have $(x, y) \in \mathcal{P}$, and $|\mathcal{P}| = M \cdot N$ is the cardinality of the deployable locations. We also denote by $\mathcal{R}$ the set of indices of all robots. Each robot $r \in \mathcal{R}$ may have a different sensing ability, i.e., sensing noise. Therefore, the ground set $\mathcal{V}$ at time t, containing all available deployment

choices, is as follows:

$$\mathcal{V}_t = \left\{(x, y, r, t) : \forall (x, y) \in \mathcal{P}, r \in \mathcal{R}\right\}.$$

We interpret (x, y, r, t) as "location (x, y) is sensed by robot r at time t". To simplify the notation, we will use a 4-tuple $v = (x, y, r, t)$ to denote the deployment decision factor in the following. Finally, the ground set of the deployment problem is $\mathcal{V} = \bigcup_{t \in \mathcal{T}_y} \mathcal{V}_t$. The cardinality of the ground set is $|\mathcal{V}| = |\mathcal{P}| \cdot |\mathcal{R}| \cdot |\mathcal{T}_y|$. The associated predicted variance with respect to v at t is represented by $\bar{\sigma}_t^2(v), \forall\, v \in \mathcal{V}$. Since $\sigma_t^2(v)$ is the predicted variance map at time t for v, we can use $\bar{\sigma}_t^2(x, y) \in \mathbb{R}$ to denote the predicted variance that corresponds to location $(x, y) \in \mathcal{P}$ at time $t \in \mathcal{T}_y$. We want to determine a deployment policy set $\mathcal{S} \in \mathcal{V}$ to maximize the information we can get from the environment while respecting the system budgets. To this end, we denote by $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ the objective function and denote by $\mathcal{M}$ a set of sets defining the system's admissible deployment policies. The details will be specified in Section 8.5.

Problem Statement With the basics of prediction and planning outlined, we now formalize the problem we solve in this chapter.

Problem 7 (Intermittent Deployment)**.** *Consider a historical set* $\mathcal{X} = \left\{\mathbf{X}_t \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_x\right\}$, *containing heightmaps of a discretized* $M \times N$ *pasture over a time horizon* $\mathcal{T}_x$*. Let* $\mathcal{V}$ *be the ground set containing all possible multi-robot deployment factors* $v = (x, y, r, t)$ *over a time horizon* $\mathcal{T}_y$*. Assume further the existence of a predicted variance set* $\bar{\Sigma} = \left\{\bar{\sigma}_t^2 \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_y\right\}$*. The intermittent deployment problem is to maximize a set function* $f(\cdot)$ *by selecting appropriate deployment factors while respecting budget constraints. That is,*

$$\begin{aligned} \underset{\mathcal{S} \subseteq \mathcal{V}}{\textit{maximize}} \quad & f(\mathcal{S}) \\ \textit{subject to} \quad & \mathcal{S} \in \mathcal{M}. \end{aligned}$$

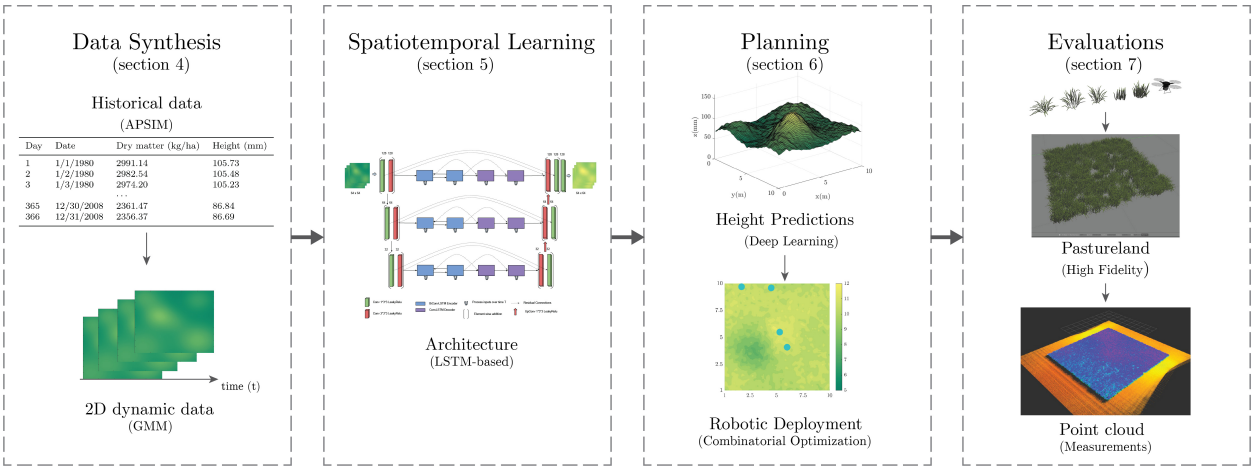


Fig. 8.2. A diagram of our overarching solution. Data synthesis is first used to simulate historical pasture data. Then spatiotemporal learning is used for neural network training and prediction based on historical data. Next, the planning aspect is used to make multi-robot deployment decisions for collecting new data. Finally, on the far right, we illustrate the high-fidelity simulation we have built for evaluating our pipeline.

By selecting the set $\mathcal{S}$ that maximizes the objective function, we can determine all deployment factors containing deployment locations and times to conduct deployments while maintaining efficiency.

Remark 12. *This work focuses on predictions and planning perspectives, especially in spatiotemporal deployment location selection under limited energy conditions. We should note that a path planning problem is a subsequent problem of the proposed framework, and any of them can be integrated into the framework. Therefore, we focus on prediction and planning to set up a cornerstone for the subsequent problems.*

The diagram of our proposed solution to the above problem is shown in Fig. 8.2. Our pipeline can be divided into the following aspects, which we detail in the sequel: data synthesis (Section 8.3), deep learning prediction (Section 8.4), planning (Section 8.5), and evaluations (Section 8.6).

8.3 Large-Scale Pasture Environment Synthesis

In this section, we focus on pasture environment synthesis. This process will be divided into two parts. First, we illustrate how to synthesize historical average pasture height data (Section 8.3.1). Based on this data, we then introduce how to utilize this data to create a dynamic pasture environment (Section 8.3.2).

8.3.1 Historical Data Preparation

'Historic' pasture data were generated using APSIM Next Generation's publicly available meteorological, soils, and pasture species modules. We selected three sites in Iowa due to the availability of meteorological data for each in APSIM's Met module as a result of prior research [141]. Meteorological data spanned 1979 to 2013 and included solar radiation (MJ/m^2), rain and snowfall, minimum and maximum temperature, atmospheric pressure, and day length. Soils selected in the module were fine-loamy, mixed, superactive, mesic Hapludolls common in Iowa, also available in APSIM's modules [141]. APSIM's tall fescue AgPasture module was used for the forage species [142]. Tall fescue was set to 1m rooting depth and initial belowground and aboveground biomass of 1000 kg/ha and 3000 kg/ha, respectively. The SoilOM module, which simulates soil organic matter processes, was set to 1000 kg/ha initial surface residue. Fertilizer application was simulated at 84 kg N/ha on January 1 and another 84 kg N/ha on August 15 each year in the form of nitrate (NO_3-N). The resulting simulated pasture yield was then used to generate average pasture height data using an equation reported by Schaefer and Lamb [111] describing the relationship between LiDAR-measured pasture height and pasture green dry matter, i.e., green aboveground biomass. In this chapter, we denote by $\mathbf{h} \in \mathbb{R}^{T_x}$ the historical data, where $T_x = |\mathcal{T}_x|$ is the length of the historical dataset horizon $\mathcal{T}_x$, which is also defined in the problem formulation

in Section 8.2. We also denote by $\mathbf{h}_t \in \mathbb{R}$ the average pasture height data at time t in the historical dataset. Further information on the above APSIM modules' functions and processes may be found in the work of Li and colleagues [142].

8.3.2 Pasture Environment Data Synthesis

The generated average pasture height data is temporal data. When making predictions for different locations and developing spatiotemporal deployment strategies, we need both spatial and temporal historical data to know the growth patterns in different places of the pasture field. Therefore, this section focuses on synthesizing spatiotemporal data from the historical average pasture height data. In general, this process fits a spatiotemporal process to the temporal aspects of the historical data.

In this work, we use a dynamic Gaussian mixture model (GMM), a combination of Gaussian distributions, to simulate the pasture evolution using historical data. The GMM used in this work is a discrete model as we need to make predictions for different days and make corresponding deployment decisions for different times and locations. We should note that there are many other ways to simulate a spatiotemporal process by using temporal data only, and we select GMM for the following reasons. GMM is a common tractable process representation, which allows us to adjust the model to match the actual field-evolving process. From expert inputs or manual field measurements, we can choose reasonable spatial parameters for the GMM to complement the temporal parameters coming from historical data, so the spatiotemporal model represents the pasture evolving process.

In general, we first use B components to build a dynamic GMM and then use the historical data $\mathbf{h}$ to adjust the generated model to match the historical information. The dynamic

GMM is modeled as

$$\mathbf{G}_t(x,y) = \sum_{i=1}^{B} w_i(t) b_i(x,y) = \mathbf{w}(t)^\top \mathbf{b}(x,y), \tag{8.2}$$

where $(x,y) \in \mathbb{R}^2$ is a 2D location from the location set $\mathcal{P}$, basis $b_i(x,y) \in \mathbb{R}$ is the output of ith basis function in the location (x,y), B is the number of basis functions, and $\mathbf{G}_t(x,y)$ is the output of the GMM at time t and location (x,y). The weight $w_i(t) \in \mathbb{R}$ is associated with $b_i(x,y)$, where $t \in \mathcal{T}_x$. The ith basis function $b_i(x,y)$ is a Gaussian kernel function. That is,

$$b_i(x,y) = \exp[-\frac{\big[(x,y) - (b_{xi}, b_{yi})\big]^2}{2c_i^2}], \tag{8.3}$$

where $(b_{xi}, b_{yi}) \in \mathbb{R}^2$ is the 2D position of ith basis function $b_i(x,y)$, and c_i is the corresponding length-scale. Also, $\mathbf{w}(t) \in \mathbb{R}^B$ is the stacked weights at time t and $\mathbf{b}(x,y) \in \mathbb{R}^B$ is the stacked basis functions for the location (x,y). One way to generate a dynamic 3D smooth surface to simulate pasture is to change the weights of different basis functions smoothly at different times. To achieve this goal, we use a 1D Gaussian process [104] to model the change of each weight at different times for the entire time horizon $\mathcal{T}_x$. After this step, a dynamic GMM is fully built.

Next, we need to tune the offset between the generated surface and the historical data to make them match with each other. Specifically, we need to adjust the generated GMM surfaces at different times using the historical average height data. That is:

$$\mathbf{X}_t(x,y) = \mathbf{G}_t(x,y) + \mathbf{h}_t - \bar{\mathbf{G}}_t,$$

where $\mathbf{h}_t \in \mathbb{R}$ is the historical average height at time t, $\mathbf{G}_t(x,y)$ is the generated GMM data for location (x,y) at time t, and $\bar{\mathbf{G}}_t \in \mathbb{R}$ is the simulated average height of the field at

that time, which is calculated by using $\bar{\mathbf{G}}_t = |\mathcal{P}|^{-1} \sum_{(x,y)\in\mathcal{P}} \mathbf{G}_t(x, y)$. The data $\mathbf{G}_t(x, y)$ is generated without incorporating our historical temporal data as we only focus on generating the dynamics of the field. To calculate the difference between our generated data and the actual historical temporal data, we can calculate the difference as $\mathbf{h}_t - \bar{\mathbf{G}}_t$, where $\mathbf{h}_t$ is the historical data at time t and $\bar{\mathbf{G}}_t$ is the average height of the generated 2D field. This difference is the offset between the average height of the generated surface and the historical average height. Then, based on this difference, we can tune the generated surface using this offset. After this series of operations, the spatiotemporal dataset $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid t \in \mathcal{T}_x\}$ is fully constructed and can be used in the later learning and predictions.

To summarize the data synthesis process, we first synthesize the historical average data $\mathbf{h}$. Then, we build a dynamic GMM denoted by $\mathbf{G}$, and adjust this model to make the temporal property of $\mathbf{G}$ match the historical average data $\mathbf{h}$. Finally, the adjusted dataset is denoted by $\mathcal{X}$ that will be used in learning and predictions (below).

8.4 Spatiotemporal Learning and Prediction

This learning and prediction part is a collaboration work. We will be introducing the general idea and settings of spatiotemporal Learning and Prediction for the sake of the completeness of this work. In this section, we will be first focusing on learning the dynamics of the generated field using the generated dataset $\mathcal{X} = \{\mathbf{X}_t \in \mathbb{R}^{M\times N} \mid t \in \mathcal{T}_x\}$. Note that pre-processing will be applied before feeding $\mathcal{X}$ into our training networks. Then, we will use the learned dynamics to make predictions of the future field, which will serve as inputs for the deployment strategy planning (Section 8.5).

In general, our spatiotemporal learning network is modeled as

$$\left(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2\right) \leftarrow \Theta\left(\mathcal{X}, \mathbf{W}\right), \quad \forall\, t \in \mathcal{T}_y,$$

where Θ is our neural network model, $\mathbf{W}$ is the set of parameters of the model, $\bar{\mathbf{Y}}_t \in \mathbb{R}^{M \times N}$ is the predicted heightmap at t, and $\bar{\sigma}_t^2 \in \mathbb{R}^{M \times N}$ is the corresponding variance at t. Next, we give the details of our mean and variance predictions.

Since the prediction model will take as input a sequence of heightmaps to predict multi-step long-horizon future pasture heights, we need to reorganize $\mathbf{X}_t \in \mathcal{X}, \forall\, t \in \mathcal{T}_x$ to satisfy this requirement. Meanwhile, we need pre-processing to enable our proposed model to tackle different pasture sizes. We aim to reorganize all $\mathbf{X}_t \in \mathcal{X}, \forall\, t \in \mathcal{T}_x$ to form multiple training sequences. We define the ith training sequence as $\mathcal{X}_i = \left\{\mathbf{X}_t \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_i\right\}$ with $\mathcal{T}_i = \{i, i+\delta, \ldots, i+\alpha\delta\}$, where δ is the number of intervals before each input observation in the training sequence, α is the number of observations in $\mathcal{X}_i$, and $\mathcal{T}_i$ contains all the time index associated with every $\mathbf{X}_t \in \mathcal{X}_i$. Note that $\mathcal{T}_i \subseteq \mathcal{T}_x$. The complementary training label (ground truth) for an input sequence $\mathcal{X}_i$ is then defined as $\mathcal{Y}_i = \left\{\mathbf{Y}_t \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_y\right\}$, where $\mathcal{T}_y = \{i + (\alpha+1)\delta, i + (\alpha+2)\delta, \ldots, i + (2\alpha+1)\delta\}$ contains the expected prediction step at δ intervals. Therefore, $\alpha = |\mathcal{T}_i| = |\mathcal{T}_y|$. The effective length (horizon) of the input and output sequences are then calculated as $L = \delta \cdot \alpha$. Varying the number of strides δ in the sequences allows our prediction model to work with both short-term and long-term horizons. For example, given an input sequence of $\alpha = 15$ and a stride of $\delta = 4$, we have an effective observation length of 60 days. Finally, the neural network outputs a prediction sequence$\bar{\mathcal{Y}}_i = \{\bar{\mathbf{Y}}_t \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_y\}$ for each input sequence $\mathcal{X}_i$, where $\bar{\mathbf{Y}}_t$ is the prediction with respect to the ground truth $\mathbf{Y}_t$ at t.

Remark 13. *For the scope of this chapter, we train our networks using the same input and*

output observation interval δ and the same number of observation/measurements α. That is, $\alpha = |\mathcal{T}_x| = |\mathcal{T}_y|$. However, it is to be noted that the proposed network can be trained on varying sequence lengths without changing the architectural design. This is due to the inherent flexibility of an encoder-decoder design.

The high-level idea of the mean predictions is as follows. For each time t, we will use the averaged prediction $\bar{\mathbf{Y}}_t$ from several Monte Carlo (MC) predictions $\hat{\mathbf{Y}}_t^{(k)} \in \mathbb{R}^{M \times N}$ as the final predicted heightmap at t, where $\hat{\mathbf{Y}}_t^{(k)}$ is the kth MC prediction at time t. That is, $\hat{\mathbf{Y}}_t^{(k)} \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \forall\, t \in \mathcal{T}_y$. The averaged mean prediction is given by

$$\bar{\mathbf{Y}}_t = \mathbb{E}_{\Pr(\hat{\mathbf{Y}}_t|\mathcal{X})}(\hat{\mathbf{Y}}_t) \approx \frac{1}{K} \sum_{i=1}^{K} \hat{\mathbf{Y}}_t^{(k)}, \quad \forall\, t \in \mathcal{T}_y. \tag{8.4}$$

Each MC prediction is a realization of the proposed network by using a different setting, which will be specified later. The number of sampled predictions is K for each heightmap prediction $\bar{\mathbf{Y}}_t, \forall\, t \in \mathcal{T}_y$.

In this work, we adapt the ubiquitous U-Net Convolution Neural Network framework [143] with ConvLSTM cells as an encoder-decoder framework for making sequence predictions. The details of this network appears in [7] and [112].

Formally, we define the model uncertainty through dropouts in the neural network by sampling K different sets of parameters $\mathbf{W}_\bullet$. That is,

$$\bar{\sigma}_t^2 = \mathrm{Var}\left(\hat{\mathbf{Y}}_t^{(k)}\right), \quad \forall\, t \in \mathcal{T}_y.$$

This operation is equivalent to performing K stochastic forward passes with the dropout of weights enabled during inference and then averaging the results. In this work, we simulate the MC dropout sampling using the $\Pr = 0.4$ probability of dropping each weight set for

stochastic inference during prediction. Through the use of dropouts between each layer in our network, both during training and testing time, we enable our encoder-decoder model to estimate the prediction set $\bar{\mathcal{Y}} = \left\{\bar{\mathbf{Y}}_t \in \mathbb{R}^{M\times N} \mid \forall\, t \in \mathcal{T}_y\right\}$ and the corresponding variance set $\bar{\Sigma} = \left\{\bar{\sigma}_t^2 \in \mathbb{R}^{M\times N} \mid \forall\, t \in \mathcal{T}_y\right\}$. The MC dropout method allows our model to generate the requisite prediction estimates and the uncertainty of its prediction in a computationally efficient process. Therefore, a more detailed network model showing our intermediate outputs, i.e., $\hat{\mathbf{Y}}_t$ and $\hat{\sigma}_t^2$, can be summarized as

$$\left(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2\right) \leftarrow \left(\hat{\mathbf{Y}}_t, \hat{\sigma}_t^2\right) \leftarrow \Theta\left(\mathcal{X}, \mathbf{W}\right), \quad \forall\, t \in \mathcal{T}_y.$$

We also refer the reader to [112] for more details about our mean and variance predictions.

After the spatiotemporal learning of the pasture, we are ready to make predictions for the horizon $\mathcal{T}_y$.

8.5 Multi-Robot Intermittent Deployment

The goal of the deployment is to maximize the information we can get from the environment while respecting the budget. To this end, we model the objective function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$ as follows:

$$f(\mathcal{S}) = \sum_{s\in\mathcal{S}} \left(\bar{\sigma}_t^2(s) \sum_{s'\in\mathcal{S}\setminus s} \frac{d(s,s')}{|\mathcal{S}\setminus s|}\right) - w_1(t - t_1), \tag{8.5}$$

where $\bar{\sigma}_t^2(s)$ is the prediction variance of the decision factor s at time t. Note that t_1 is the starting time index of the prediction horizon $\mathcal{T}_y$. The distance function $d(s, s')$ is the weighted Euclidean distance and time difference between s and s'. That is,

$$d(s,s') = w_2 \log(||(x,y) - (x',y')||) + w_3||t - t'||, \tag{8.6}$$

where $s = (x, y, r, t)$ and $s' = (x', y', r', t')$ are two different decision factors. And w_1 is the waiting penalty weight, w_2 is the weight for the physical distance, and w_3 is the weight for the time difference between s and $s' \in \mathcal{S} \setminus s$.

The objective function is a weighted sum of prediction variances. The nominator $\bar{\sigma}_t^2(s)$ is the prediction variance of the decision factor s at time t. The denominator is the sum of the weighted distances between s and $s' \in \mathcal{S} \setminus s$ for all $s \in \mathcal{S}$. Meanwhile, w_1 is the weight to penalize the waiting time. We can deploy robots using a deployment set $\mathcal{S}$ that maximizes the objective function $f(\cdot)$ to reduce the prediction uncertainty.

Since the deployment should be energy efficient, we have two constraints to model the deployment budgets. The first constraint is,

$$|\mathcal{S} \cap \mathcal{V}_t| \leq \ell_t, \forall\, t \in \mathcal{T}_y. \tag{8.7}$$

This constraint (per-day budget) indicates that the number of deployments cannot be larger than ℓ_t at time t, where $\ell_t \in \mathbb{R}$. The second constraint is,

$$\sum_{t \in \mathcal{T}_y} \mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) \leq \ell, \tag{8.8}$$

where $\mathbb{1}(\cdot)$ is an indicator function as

$$\mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) = \begin{cases} 1, & \text{if } |\mathcal{S} \cap \mathcal{V}_t| \geq 1, \\ 0, & \text{if } |\mathcal{S} \cap \mathcal{V}_t| = 0. \end{cases}$$

This constraint (total budget) suggests that the total number of deployable days cannot be

Algorithm 15 The Algorithm for the Long-Term Pasture Prediction and Sensing Problem.

Input: The inputs are as follows:

- The historical dataset $\mathcal{X} = \left\{\mathbf{X}_t \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_x\right\}$;
- The neural network $\Theta(\cdot)$;
- The deployment ground set $\mathcal{V}$;
- The objective function $f : 2^{\mathcal{V}} \mapsto \mathbb{R}$;
- The matroid intersection constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$.

Output: The deployment strategy set $\mathcal{S}$.

for $t \in \mathcal{T}_y = \{\tau, \ldots, T\}$ **do**
    $\hat{\mathbf{Y}}_t^{(k)} \leftarrow \Theta(\mathcal{X}, \mathbf{W}), \forall\, k = 1, \ldots, K$;
    $\bar{\mathbf{Y}}_t \leftarrow \frac{1}{K} \sum_{k=1}^{K} \hat{\mathbf{Y}}_t^{(k)}$; ▷ Mean
    $\bar{\sigma}_t^2 \leftarrow \mathrm{Var}(\hat{\mathbf{Y}}_t^{(k)})$; ▷ Variance
    $\bar{\Sigma} \leftarrow \bar{\Sigma} \cup \{\bar{\sigma}_t^2\}$;
end for

$\mathcal{S} \leftarrow \emptyset, \mathcal{Z} \leftarrow \mathcal{V}$;
while $\mathcal{Z} \neq \emptyset$ **do**
    $s \in \arg\max_{v \in \mathcal{V} \setminus \mathcal{Z}} f(\{v\} \mid \mathcal{S})$;
    if $\mathcal{S} \cup \{s\} \in \mathcal{I}$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$;
    end if
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{s\}$;
end while
$\mathcal{S} \leftarrow$ deployment strategy;

larger than ℓ, where $\ell \in \mathbb{R}$. Therefore, the problem formulation is

$$\begin{aligned} \underset{\mathcal{S} \subseteq \mathcal{V}}{\text{maximize}} \quad & f(\mathcal{S}, \bar{\Sigma}) \\ \text{subject to} \quad & |\mathcal{S} \cap \mathcal{V}_t| \leq \ell_t, \forall\, t \in \mathcal{T}_y, \\ & \sum_{t \in \mathcal{T}_y} \mathbb{1}\left(|\mathcal{S} \cap \mathcal{V}_t|\right) \leq \ell. \end{aligned}$$

It has been shown that both constraints are matroidal [4], and we will use $\mathcal{M}_1 = (\mathcal{V}, \mathcal{I}_1)$ and $\mathcal{M}_2 = (\mathcal{V}, \mathcal{I}_2)$ to denote those two, where $\mathcal{M}_1, \mathcal{M}_2$ are matroids and $\mathcal{I}_1, \mathcal{I}_2$ are independent sets. To simplify the notation, we use $\mathcal{M} = (\mathcal{V}, \mathcal{I})$, where $\mathcal{I} = \mathcal{I}_1 \cap \mathcal{I}_2$, to denote the intersection of two constraints. Thus, $\mathcal{M}$ is a matroid intersection constraint and the cardinality is $|\mathcal{M}| = 2$.

Using the proposed architecture with ConvLSTM and residual connections, we have the predicted variance set $\bar{\Sigma} = \left\{\bar{\sigma}_t^2 \in \mathbb{R}^{M \times N} \mid \forall\, t \in \mathcal{T}_y\right\}$. Given the intermittent deployment problem (Problem 7) and the deployment ground set $\mathcal{V}$, we propose to solve the problem using Algorithm 15. From Line 1 to Line 6, we first use the proposed architecture to predict the variance set $\bar{\Sigma}$ for the prediction time set $\mathcal{T}_y$. In the second part, the algorithm greedily selects all the available decision factors $v \in \mathcal{V} \setminus \mathcal{Z}$ based on the marginal gain $f(\{v\} \mid \mathcal{S})$, where $\mathcal{Z}$ is used to store all checked decision factors. The set $\mathcal{S}$ is the current solution of the deployment strategy and will be updated iteratively. Specifically, we initialize a set $\mathcal{Z}$ as $\mathcal{V}$. Then, in Line 10, we select one of the decision factors v that maximizes the marginal gain of the objective function $f(\{v\} \mid \mathcal{S})$, where $\mathcal{S}$ is the current solution of the problem and will be expanded as more decision factors are checked. In Line 11 to Line 13, we need to check if v satisfies the matroidal deployment constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$. If so, v is added to the solution set $\mathcal{S}$ as $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$. Otherwise, the next round will be started. Meanwhile, $\mathcal{Z}$ is updated to store the checked decision factor v as $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{s\}$. The iteration will be finished when every decision factor in $\mathcal{V}$ is checked against the constraint $\mathcal{M} = (\mathcal{V}, \mathcal{I})$.

If we define the optimal deployment policy of the intermittent deployment problem as $\mathcal{S}^\star$ with respect to the predicted variance set $\bar{\Sigma}$, we then have the following result.

Theorem 16 ([12])**.** *The optimality ratio of the greedy solution $\mathcal{S}$ generated by Algorithm 15*

has the following performance:

$$f(\mathcal{S}) \geq \frac{1}{|\mathcal{M}| + c_f} f\left(\mathcal{S}^\star\right) = \frac{1}{2 + c_f} f\left(\mathcal{S}^\star\right),$$

where c_f is the curvature of $f(\cdot)$, $|\mathcal{M}| = 2$ is the cardinality of the matroid intersection constraint, and $\mathcal{S}^\star$ is an optimal solution.

The above result gives our problem a lower bound of algorithm Algorithm 15. Note that the curvature of the objective function $f(\cdot)$ can be evaluated by checking the contribution of every decision factor v from the ground set $\mathcal{V}$. Therefore, by using the proposed Algorithm 15, we can get an intermittent deployment policy using the proposed method while having a performance guarantee, as shown above.

Remark 14. *The proposed pipeline can also be implemented in a receding horizon manner. That is, based on the proposed deployment policy $\mathcal{S}$, the new robot measurements from a series of deployments can be integrated into the historical dataset $\mathcal{X}$ to refine the learned network $\Theta(\cdot)$. Therefore, we can produce better predictions $(\bar{\mathbf{Y}}_t, \bar{\sigma}_t^2)$ and thus improved plans $\mathcal{S}$ for the future.*

8.6 Pasture Construction and Perception

The pasture construction and perception part is a collaboration work, we will be introducing the general idea and settings for the sake of the completeness of this work.

When we have a synthesized pasture, we need a representation of what aerial robots with LiDAR would measure. Thus, based on our synthesized data, we simulate a realistic pasture environment and LiDAR measurements in Gazebo. This high-fidelity pastureland simulation environment will help us to understand the effectiveness of the simulated process. In this

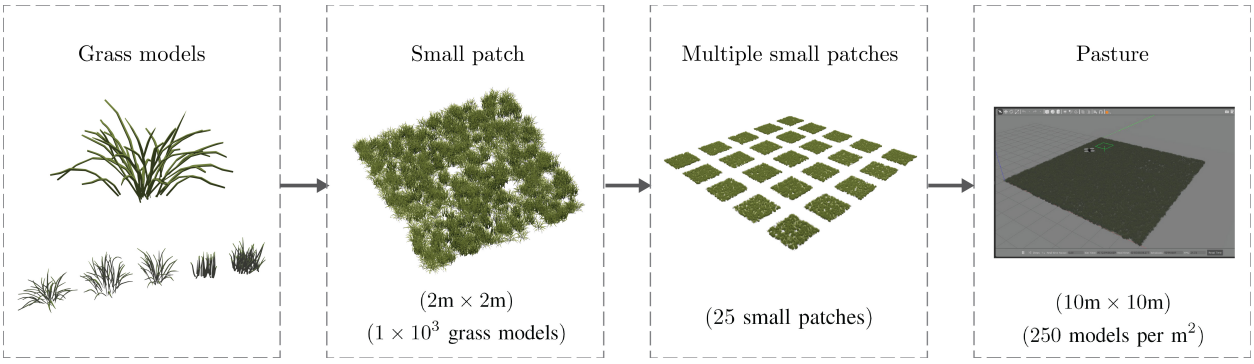


Fig. 8.3. A diagram of our pasture construction process.

section, we will first focus on constructing pasture environments from the simulated data and then on the height estimation using LiDAR measurements.

Pasture construction: In this work, we simulate a 10m×10m pasture using 2.5×10^4 grass models. We set the size of the simulated pasture and the density like this when considering the computational complexity. First, we randomly pick 2.5×10^4 locations from this pastureland environment. We then assign a 3D grass model to each sampled location. The heights of grass models correspond to the heights at the same locations in the smooth 3D surfaces. To accelerate the simulation speed and lower the computational requirement, we divide the pasture into 25 small patches (2m×2m per patch). In each patch, the density is 250 grass models per square meter with a total of 1×10^3 models per small patch. In this small patch, we use five species of plants to simulate different growth patterns as shown in Fig. 8.3. This selection will be validated later in the experimental section as the actual measurements look similar to our simulated environment. Each plant is spawned at the same randomly chosen coordinates throughout our simulation. We rescale a grass model in each dimension for each randomly selected location according to the desired height on the 3D surface. The flowchart of our pasture construction is shown in Fig. 8.3.

Pasture perception: To estimate the height of the pasture, we use a UAV equipped with a LiDAR to collect point clouds over the pasture. Meanwhile, we need post-processing to

remove noise after getting the point clouds. Those extra points are not part of the simulated field and need to be removed. To achieve this, we use crop box filters to remove the extra points and retain the points of the 10m×10m pasture as well as the points in the perimeter around it.

The height of a point cloud includes two parts: the height of the pasture and the height of the ground plane. To get the estimated height of the field, we use the mowed-down perimeter to estimate the ground plane. First, we have the following assumption of the ground of the environment.

Assumption 6 (Ground Plane)**.** *We assume that the ground of the pastureland is a plane in this chapter. However, we will also introduce some methods that can be used to tackle other cases in the followings.*

In the simulation, we use this assumption to facilitate the ground height estimation. However, other types of ground can also be integrated into our simulation framework, where we can use more sophisticated methods for ground surface regression.

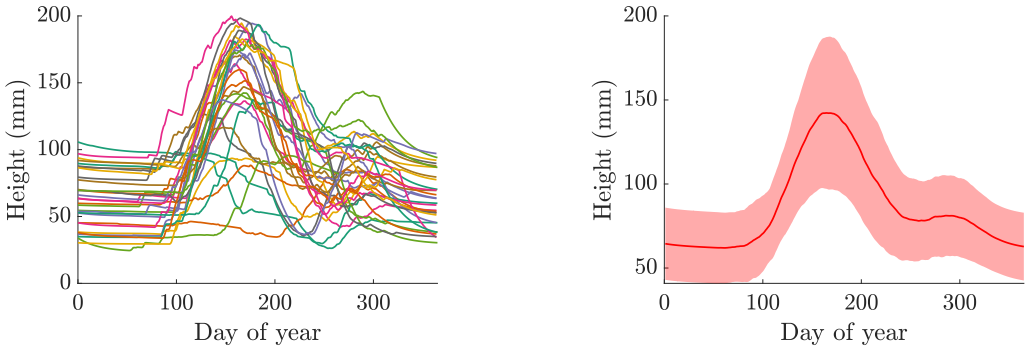
In this work, we use the least squares method to compute the height of the ground by using perimeter points. The perimeter is all the points surrounding the target plot area. These perimeter points will be used to estimate the ground plane that is used for height estimation. We direct the reader to [7] for this least squares method.

8.7 Evaluation

In this section, we demonstrate the results of each component in our pipeline. The historical data is generated using Matlab. The pasture is simulated by using Blender. The neural network training was conducted with a PyTorch backend on 2x AMD Epyc 7742 CPUs and a

TABLE 8.1
THE LOCATIONS, LENGTH-SCALES, AND INITIAL WEIGHTS OF DIFFERENT BASIS FUNCTIONS.

i	Location (x, y)	Length-scale c_i	Initial weight w_i
1	(5.0, 5.0)	0.13	4.17
2	(3.0, 4.0)	0.13	4.17
3	(2.0, 1.5)	0.15	2.50
4	(8.0, 8.0)	0.18	6.67
5	(8.0, 1.5)	0.13	3.33
6	(1.0, 1.0)	0.13	3.33
7	(1.0, 9.0)	0.25	4.17



(a) Average height.

(b) Mean & standard deviation.

Fig. 8.4. (a). The average height of the pastureland environment in 30 years (represented by 30 lines). The x-axis represents the day of the year. The y-axis represents the corresponding average height. (b). The mean and the standard deviation of each day's height are calculated using historical data.

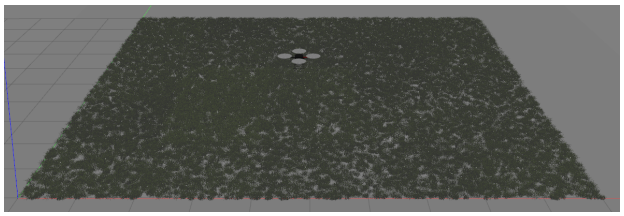


Fig. 8.5. A simulated pastureland 10m × 10m environment using 2.5×10^4 grass models

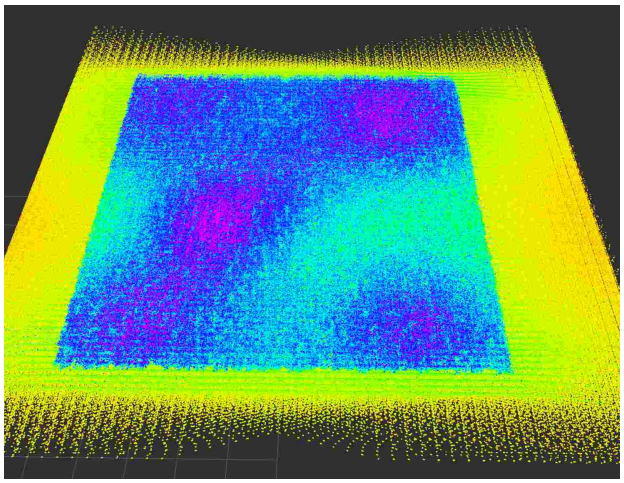


Fig. 8.6. The corresponding point cloud of the pasture (10m × 10m) shown in Fig. 8.5.

multi-GPU training regime with 8x Nvidia RTX 6000 GPUs.

8.7.1 Training Data Preparation

In this section, we focus on preparing training data. Based on historical meteorological data from a site in Iowa, we simulate 30 years of tall fescue pasture dry matter production. Specifically, we will use the first 28 years of data for training and reserve the last year's data for testing purposes. In Fig. 8.4, we demonstrate the statistics of the historical data. Fig. 8.4(a) shows the average height change of the simulated pastureland environment, where each line denotes the height change for different days in a year. We also calculate each day's mean and standard deviation using the historical data as shown in Fig. 8.4(b).

The simulated pasture yield was then used to generate average pasture height data based

on the model in [111] describing the relationship between pasture green dry matter and LiDAR-measured pasture height for each day. Following the pastureland generation procedure described in Section 8.3.2, we generate a 2D pastureland environment over 28 years using a dynamic GMM. In the simulation, we use $B = 7$ basis functions. The random initial settings of the basis function location (x, y), the length-scale c_i, and the initial weight w_i of each basis function $b_i(x, y)$ are shown in Table 8.1. Later on, in our experimental section (Section 8.7.5), we can see that those parameter settings can help us to simulate pastureland environments that are very close to the real-world scenario.

8.7.2 Point Cloud Testing Data Preparation

In this section, we focus on preparing the testing data. To test the performance of proposed pipeline, we simulate an *actual* 10m ×10m pasture (Fig. 8.5) in Gazebo using the method proposed as illustrate in Fig. 8.3. The selected pasture size, i.e., 10m ×10m, is a result of careful consideration of the computational speed and complexity of the entire process. We will then collect data from this simulated *actual* environment to test the prediction and deployment performance.

To further exploit parallelization computation to improve the simulation speed, we divide the pasture into 25 small 2m×2m patches. For each patch, we use 1×10^3 grass models (250 models per square meter). Finally, 25 small patches are attached to form one large 10m ×10m pastureland environment.

We randomly assign grass models and extrude the model in three dimensions to match the predefined height for the model in that location. As the swards of the grass grow with a curvature, we need to adjust the location of the base of each grass model, i.e., the location of each grass model at the ground plane to match the location of its sward at

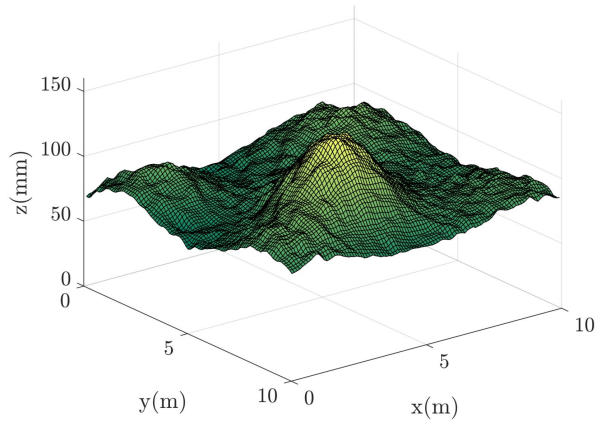
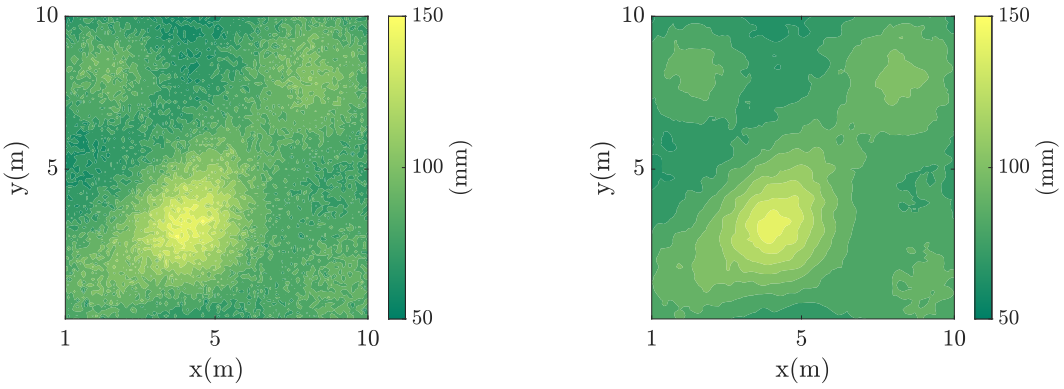


Fig. 8.7. The downsampled surface of the point cloud shown in Fig. 8.6.



(a) Before processing.

(b) After processing.

Fig. 8.8. The downsampled heightmaps of the point cloud shown in Fig. 8.6.

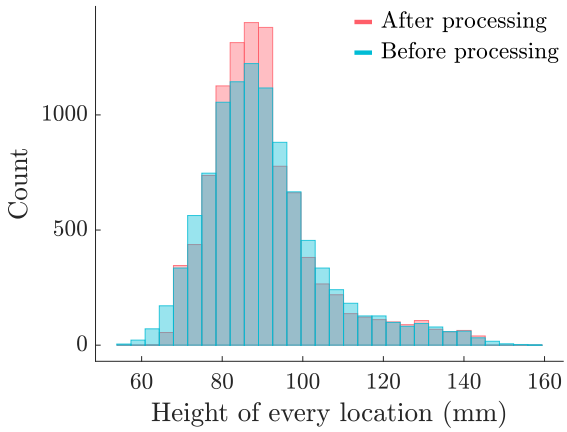
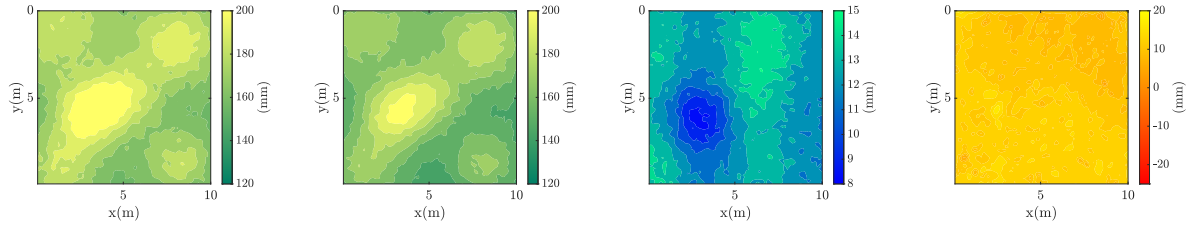


Fig. 8.9. The histogram comparison of the two heightmaps shown in Fig. 8.8.

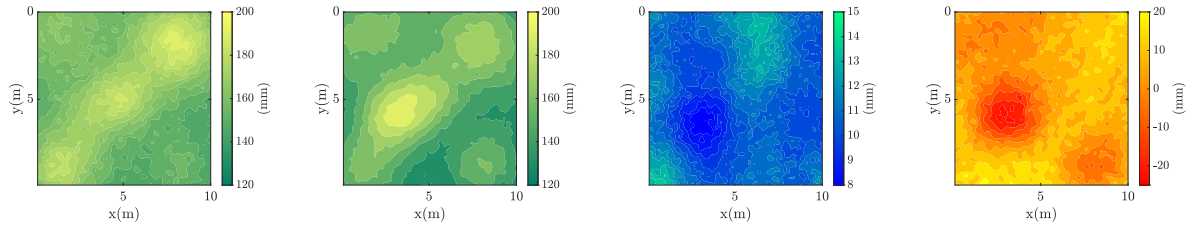
its highest point. This offset allows us to calculate the vertical height of each individual plant directly. Specifically, denote by $\gamma \in \mathbb{R}$ the scaling factor of a grass model, which is calculated by comparing the desired height of a grass model and the original height. We denote $\mathbf{m} = [m_x, m_y]^\top \in \mathbb{R}^2$ the original 2D location of the topmost point, and $\xi = [\xi_x, \xi_y]^\top \in \mathbb{R}^2$ the offset of the topmost point, where $\xi_x, \xi_y \in \mathbb{R}$ are the offset in x and y direction. We apply a shifting process as $\mathbf{m} \leftarrow \mathbf{m} - \gamma \cdot \xi$ to ensure that the topmost point is at the predefined location. After this adjustment, the 2D location of the topmost point of a grass model is transformed to the predefined 2D location.

Following the steps described in Section 8.6, we generate 15 pastureland environments in Gazebo using the historical data with a day interval of $\delta = 4$. An example illustrating the simulated pastureland environment (10m $\times$ 10m) is shown in Fig. 8.5. In this simulated pastureland environment, there are 2.5×10^4 grass models.

Next, we use a robot to collect a point cloud for each simulated environment. The simulated LiDAR has an inherent standard deviation of 4mm in its readings. We also assume the ground of the field is flat. Nevertheless, adding different terrains to simulate different types of ground is easy. During the simulations, the plant locations, pose, and species for a particular



(a) 5th ground truth. (b) 5th predicted mean. (c) 5th predicted std. (d) 5th predicted error.



(e) 15th ground truth. (f) 15th predicted mean. (g) 15th predicted std. (h) 15th predicted error.

Fig. 8.10. The comparison between deep learning predictions and the corresponding ground truth. The first row is for $\alpha = 5$ prediction (the effective horizon is 40 days), and the second row is for $\alpha = 15$ prediction (the effective horizon is 60 days). The comparisons include predicted mean, predicted standard deviation, and the prediction error for every location.

location in the pasture are fixed throughout the years to accelerate those processes. Finally, all the collected point clouds will be used as testing inputs.

8.7.3 Neural Network Training and Prediction

In this section, we focus on the following two different parts of our neural network.

- We first give details of our training settings based on training data generated from GMM, as stated in Section 8.7.1.
- We then test the performance of the networks based on two different types of testing inputs.

Training: Based on the generated GMM data from Section 8.7.1, the training sequences $\mathcal{X}_i$'s are generated by setting the number of measurements in each sequence as $\alpha = 15$. Also, the

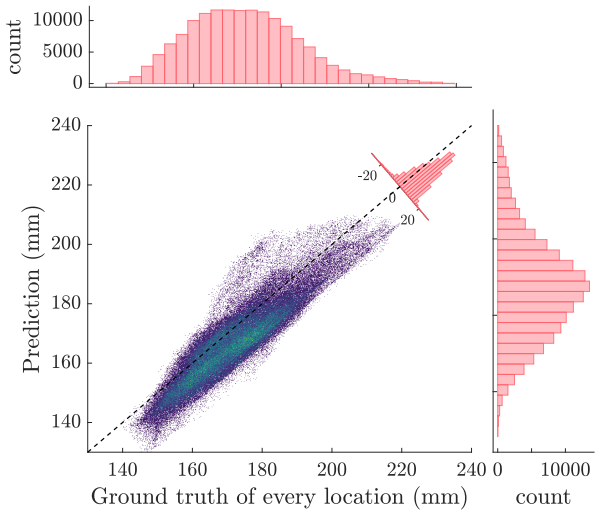


Fig. 8.11. The relationship between the ground truth and the corresponding prediction by using all the predictions and the ground truth. The x-axis represents the ground truth of every point, and the y-axis represents the corresponding prediction result. The marginal histograms show the statistics of the ground truth and the prediction independently.

number of intervals is $\delta = 4$. We use early stopping, where training is stopped if validation loss does not improve for ten epochs, usually resulting in 30 training epochs.

Testing: We evaluate the performance of the prediction network in two different cases:

1. In the first testing case, we use one group of point cloud measurements as testing inputs and another group of point cloud measurements as testing outputs. This testing case can help us understand the theoretical prediction performance.

2. In the second testing case, we use two groups of GMM data as testing inputs and outputs. This testing case can help us understand the practical prediction performance.

1) Testing by using point cloud measurements: We note that due to the computationally intensive process of constructing the aforementioned pastures and their subsequent point cloud measurements, we only generate a single test case consisting of 30 days, where the data is split into 15 days as an input sequence to the prediction network, and the remaining

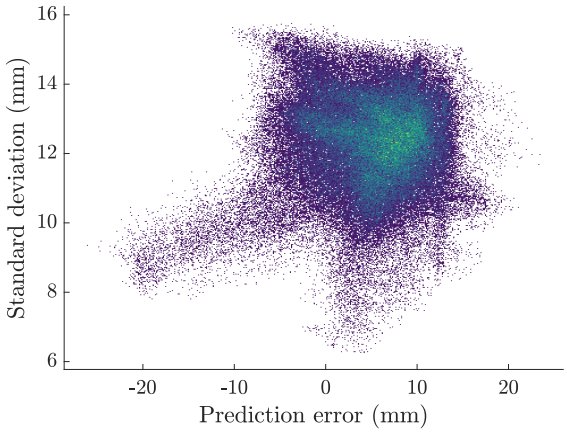


Fig. 8.12. The relationship between the prediction error and the corresponding prediction standard deviation of every location by using all the predictions.

15 days are used to evaluate the prediction performance. The collected point clouds will be first converted to heightmaps of size 100×100 as the original point clouds are too large to be used as network prediction inputs. Meanwhile, we need to process those heightmaps since the measurements are noisy. In Fig. 8.7, we demonstrate a downsampled surface of the point cloud shown in Fig. 8.6. Specifically, after downsampling, the processing includes two steps: (a). a size of 3×3 median filtering; (b). a size of 3×3 flat convolution filtering. After processing, we see that the growth pattern of the pastureland is visible as shown in Fig. 8.8(b) when compared with the raw map as shown in Fig. 8.8(a). Those processes make the neural networks' prediction of pastureland dynamics more efficient. Meanwhile, this processing only has a mild change on the original data, as can be seen from the histogram comparison in Fig. 8.9.

To test the prediction performance, we predict the heightmaps for another 15 days using a stride length of $\delta = \{1, 4\}$. The longest effective horizon for those two are $L = 15$ and $L = 60$ respectively. Then, we generate the corresponding point cloud measurement to check the testing performance. To estimate the uncertainties, we set the number of MC dropout samples as $K = 500$. In general, our network performance on average performs within a 12%

TABLE 8.2
ACCURACY METRICS (IN MM) FOR DIFFERENT METHODS WITH AND WITHOUT UNCERTAINTY ESTIMATES (UE), WHERE THE STRIDE LENGTH IS $\delta = \{1, 4\}$ AND THE CARDINALITY OF HORIZONS SET IS $\alpha = 15^*$.

	(GMM Data)				(Point Cloud Data)			
Model	RMSE	MAE	MPAE	ASTD	RMSE	MAE	MPAE	ASTD
$\delta = 4$ + UE	19.25	13.42	11.91	9.27	**7.41**	**6.46**	**3.672**	12.39
$\delta = 1$ + UE	**6.91**	**5.12**	**4.65**	**8.31**	–	–	–	–
$\delta = 4$	24.65	18.79	15.62	–	19.31	18.07	10.58	–
$\delta = 1$	18.52	14.38	13.13	–	–	–	–	–

*(Since the tests without UE do not implement MC Dropout methods, no ASTD values are available. Additionally, due to high computation requirements for point cloud data, results for $\delta = 1$ are unavailable.)

error rate in the worst case when $\delta = 4$ and $L = 60$, and within a 5% error rate when data is available more frequently when $\delta = 1$ and $L = 15$. To further demonstrate the details of our long-time horizon prediction results, we show the 5th and 15th prediction results with their ground truth in Fig. 8.10. Note that the effective time horizons for those two predictions are $L = 40$ and 60 days. The uncertainty estimations in terms of standard deviations of the predictions run over $K = 500$ samples. We observe that the network has higher confidence at the highest pasture heights since it is easier to learn feature mappings due to more significant correlations within its neighborhood. The prediction error is lowest at the highest points in the pasture since the network learns to estimate the peak points with higher confidence.

In Fig. 8.11, we compare the ground truth and the corresponding predictions using all the predictions across the output sequence. From the result, we can see that most prediction-ground truth pairs lie in the 45 degree line, where we can observe that most predictions are close to the ground truth. The prediction uncertainty of the model increases as the prediction error increases, as shown in Fig. 8.12, showing a clear correlation between the network's confidence against the prediction errors. This reiterates the suitability of the computationally efficient Bayesian approximation for uncertainty estimates, allowing a quick and efficient

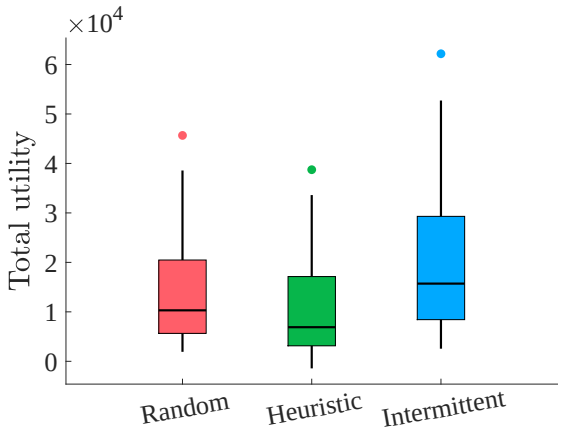


Fig. 8.13. The utility statistics of the three methods after running 50 trials. The proposed intermittent deployment method has the highest average utility.

turnaround in prediction. This methodology reduces the need for expensive hardware for training and inference of the deep learning-based prediction model and allows even small industries or farms to tune and integrate the prediction systems into their workflow.

Remark 15. *The network makes predictions from point cloud measurements in the above tests, and the original training dataset of pasture height maps generated from GMM is not used. That indicates the ability of our network to generalize beyond the simulated training data to real-world applications by using LIDAR-based measurements.*

2) Testing by using GMM data: Next, we use Monte Carlo simulations to test the prediction performance using different GMM inputs. Note that the training model is unchanged in this case. We set $|\mathcal{T}_y| = \alpha = 15$. That is, we use a length of 15 heightmaps to predict another group of 15 heightmaps. Since we will test the performance using two years' data and the first 15 heightmaps can only be used as inputs, the number of distinct output prediction sequences is $H = (365 - |\mathcal{T}_y|) \cdot 2$. We should also note that K in (8.4) is the number of samples used for calculating one prediction $\bar{\mathbf{Y}}_t, \forall\, t \in \mathcal{T}_y$. We evaluate the performance of the neural network predictions with the following metrics: Root Mean Square Error (RMSE), Mean Absolute

Error (MAE), Mean Absolute Percentage Error (MAPE), and Averaged Standard Deviation (ASTD).

$$\text{RMSE} = \sqrt{\frac{1}{H \cdot \left|\mathcal{T}_y\right|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left(\mathbf{Y}_t^{(h)} - \bar{\mathbf{Y}}_t^{(h)}\right)^2},$$

$$\text{MAE} = \frac{1}{H \cdot \left|\mathcal{T}_y\right|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left|\mathbf{Y}_t^{(h)} - \bar{\mathbf{Y}}_t^{(h)}\right|,$$

$$\text{MAPE} = \frac{1}{H \cdot \left|\mathcal{T}_y\right|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left|\frac{\mathbf{Y}_t^{(h)} - \bar{\mathbf{Y}}_t^{(h)}}{\mathbf{Y}_t^{(h)}}\right|,$$

$$\text{ASTD} = \sqrt{\frac{1}{H \cdot \left|\mathcal{T}_y\right|} \sum_{h=1}^{H} \sum_{t \in \mathcal{T}_y} \left(\bar{\sigma}_t^{(h)}\right)^2},$$

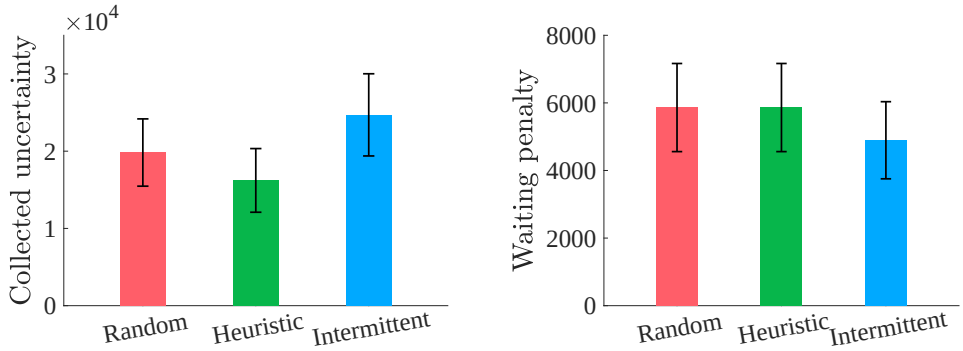
where H is the batch size of the output prediction. For simplicity, we omit the superscript h on $\mathcal{T}_y$ and note that the discrete time intervals t for the output predictions can differ for each batch sample.

The results of these metrics are reported in Table 8.2, containing the results from the above two testings. These metrics quantify the performance of the prediction algorithm by aggregating the errors and uncertainty over the discretized values of the pasture.

8.7.4 Robotic Deployments Results

In this section, we use Monte Carlo simulations to test the performance of the proposed integrated pipeline. Specifically, we have two goals for the testing:

1. Deployment strategy comparison: we want to test the effectiveness of the proposed pipeline using the same neural network prediction results but with different deployment strategies.



(a) Collected uncertainty. (b) Waiting penalty.

Fig. 8.14. (a). The comparison of the collected uncertainties of three different methods, which is the first part of the objective function. (b). The waiting penalty of three different methods, which is the second part of the objective function.

2. Pipeline comparison: We want to test the effectiveness of the proposed pipeline, i.e., neural network prediction and intermittent deployment, with another often used pipeline.

In the followings, we will first specify the details of the comparison settings and then demonstrate the comparison results for each testing scenario.

1) Deployment strategies comparison: Based on the same deep learning prediction result, we compare the performance using the following deployment policies:

- Intermittent: the intermittent deployment policy, where the policy is generated through the method proposed in Section 8.5. We refer to this deployment policy as "Intermittent".

- Heuristic: a heuristic deployment policy, which is used to mimic how humans manually monitor pasturelands on spatial and temporal scales. On the temporal scale, the robots are deployed within a fixed time interval. For example, if the planning horizon is 9 days and the robot team can only be deployed 3 times, then those robots will be deployed at 1st day, 5th day, and 9th day. On the spatial scale, the deployment locations are evenly

distributed across the pastureland. This spatial strategy simulates the case where each robot has the same coverage ability, and they are deployed to the locations where the overall coverage is maximized. We refer to this deployment policy as "Heuristic".

- Random: a random deployment policy, where both deployment times and locations are randomly selected from the ground set $\mathcal{V}$. We refer to this deployment policy as "Random".

We first evaluate the performance using the collected reward (the objective function value). Then, we collect samples based on the generated deployment policies. After that, we use the collected information from different methods to make 10 more predictions. We finally compare the prediction performance of those methods. Thus, those steps will help verify the proposed pipeline's effectiveness.

Based on the $\alpha = 15$ deep learning prediction results, we run 50 Monte Carlo simulations to generate different deployments under different settings. The settings are as follows. The maximum number of deployable days (total budget) ℓ is randomly sampled from the set $\{5, 6, \ldots, 12\}$. The number of maximum sampling points ℓ_t is sampled from the set $\{2^2, 3^2, \ldots, 8^2\}$. The cardinality of the planning horizon is $|\mathcal{T}_y| = 15$. We also assign each robot a random weight for the same traveling cost to simulate the heterogeneity. In each run, we generate one instance and then use the above three methods to compare the performance. Also, the number of robots is set to $\ell_t \cdot \ell$ for each instance. The weights for distance and time are $w_2 = 0.1, w_3 = 1$. And the weight for time penalty is $w_1 = 5$. The definitions of those parameters can be found in our problem formulation in Section 8.2.

Then, we first compare the results of different deployment policies by comparing the collected reward of each method after running 50 trials. The result is shown in Fig. 8.13, where the proposed intermittent policy has the highest averaged utility. Then, in Fig. 8.14, we

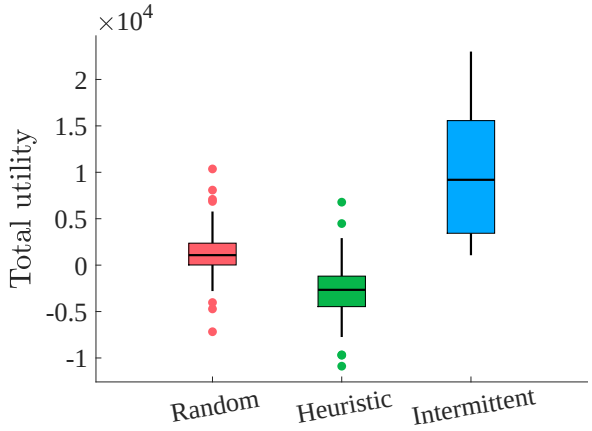
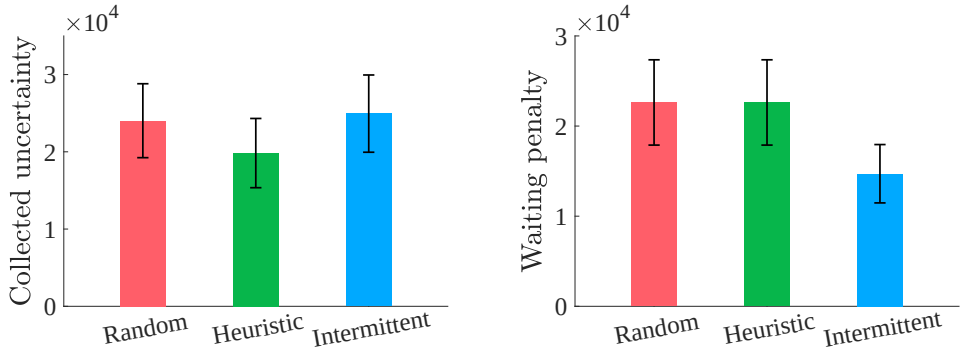
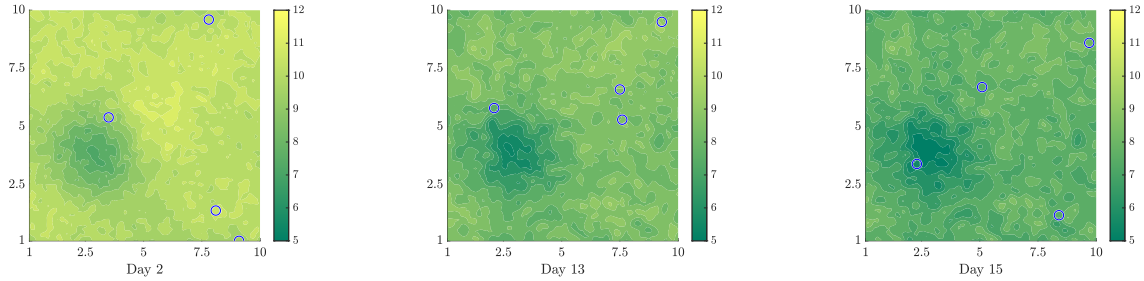


Fig. 8.15. The utility statistics of the three methods after running 50 trials when the waiting penalty weight w_1 is high.

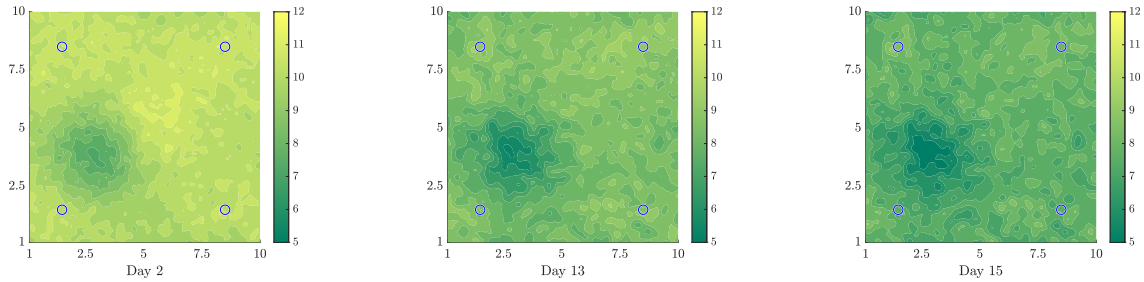


(a) Collected uncertainty. (b) Waiting penalty.

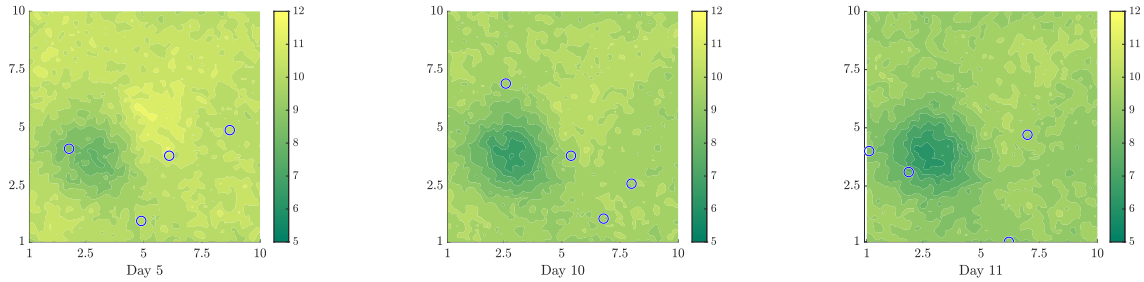
Fig. 8.16. The utility/cost of different parts of our objective function when the waiting penalty weight w_1 is high.



(a) Random 1st deployment (b) Random 2nd deployment (c) Random 3rd deployment



(d) Heuristic 1st deployment (e) Heuristic 2nd deployment (f) Heuristic 3rd deployment



(g) Intermittent 1st deployment (h) Intermittent 2nd deployment (i) Intermittent 3rd deployment

Fig. 8.17. An example of deployment policies generated by different methods. (a)-(c). random, (d)-(f). heuristic, (g)-(i). intermittent. Those are predicted variance maps.

demonstrate the two different parts of the objective function. In Fig. 8.14(a), we show the collected uncertainty of three different deployment methods, which is the first part of the objective function. In Fig. 8.14(b), we compare the waiting penalty part in the objective function. Then, in Fig. 8.15, we set a high waiting penalty as $w_1 = 10$ while keeping w_2, w_3 the same as before. Again, we run other 50 trials to compare the performance. In Fig. 8.15, the values of different parts in the objective function prove that our proposed method can

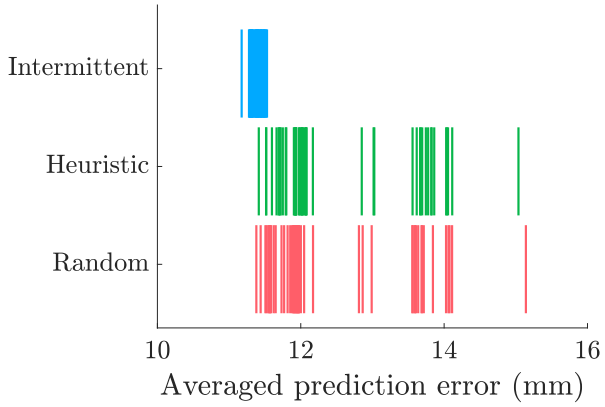
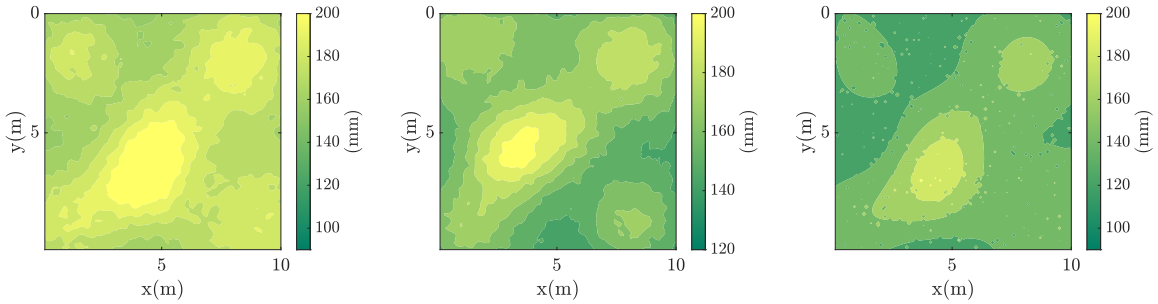


Fig. 8.18. The comparison of the averaged prediction error by calculating the mean of the errors at each location using 50 trials.

collect more rewards while having less waiting penalty. Similarly, in Fig. 8.16, we compare the result from each part of the objective function using three different deployment policies. This finishes up the first part of this comparison. In Fig. 8.17, we demonstrate a deployment result using one setting instance. In this result, we set the parameters as follows. The maximum number of deployments for each day is set to $\ell_t = 4$, and the maximum number of deployable days is set to $\ell = 3$.

Next, we use robots to collect data based on the generated deployment policies. The collected height information will be used to update our knowledge of the environment. Finally, we compare the predictions from each method using this updated information. Note that white noise is also added to simulate measurement noise. Specifically, we use the collected information to make another 10 more predictions, i.e., from $\alpha = 1$ to $\alpha = 10$, with the size of interval of $\delta = 2$. For each prediction, we compare it with the ground truth. Then, the final averaged prediction error is shown in Fig. 8.18. The averaged prediction error is calculated by averaging the prediction errors from all locations. Since we run the simulation 50 trials, there are 50 different averaged prediction errors for each method shown in Fig. 8.18. After using the updated information from different methods, we see that the proposed pipeline



(a) 5th ground truth. (b) 5th predicted using DL. (c) 5th prediction using GP.

(d) 15th ground truth. (e) 15th predicted using DL. (f) 15th prediction using GP.

Fig. 8.19. The comparison between the ground truth and the predictions by using deep learning (DL) based method and Gaussian process (GP) based method.

TABLE 8.3
The averaged prediction comparisons (in mm) between the deep learning (DL) based method and the Gaussian process (GP) for $\alpha = 5$ and $\alpha = 15$.

		(DL prediction)		(GP prediction)	
Time	Truth	Prediction	Error	Prediction	Error
$\alpha = 5$	181.53	169.82	**11.71**	149.87	31.66
$\alpha = 15$	164.42	158.83	**5.60**	117.08	47.34

has a lower average prediction error than the other two methods. That demonstrates the effectiveness of the proposed method.

2) Pipelines comparison: In the chapter, the deployment strategies are made based on the predicted variance map. However, as the deployment problem itself is an NP-hard combinatorial optimization problem, there is no simple way to get an optimal deployment strategy based on any prediction result. In the chapter, we adopt a commonly used mutual information-based pipeline in the second comparison category and compare it with our proposed pipeline. Specifically, we will use a 2D Gaussian process (GP) to model the environment and make predictions. We then use the proposed intermittent deployment idea to make deployment strategies based on the different prediction results. Again, we finally make another 10 future predictions after deployments to test the performance.

In this comparison, we first use GP to make predictions. The settings of this GP are as follows. The training dataset is from the point clouds generated in Gazebo as described in Section 8.6. We also need to convert those point clouds into heightmaps. We denote the training inputs of the GP by $\mathbf{x}_t = [x, y, t] \in \mathbb{R}^3$. The training outputs are corresponding grass height of location (x, y) at time t. From the above 15 heightmaps, we randomly pick 1×10^3 points across all those heightmaps to form the training set. Since GP is a non-parametric method, we select kernels as follows. The mean kernel is defined using the historical mean data as shown in Fig. 8.4. The covariance kernel is a composite of a 2D Gaussian covariance

kernel and a 1D linear covariance kernel with noise term. The Gaussian covariance kernel is similar to the one shown in (8.3). Then, those two covariance kernels are summed up to form the final covariance kernel. In Fig. 8.19, we show two prediction results when $\alpha = 5$ and $\alpha = 15$. We notice that the proposed deep learning-based method can maintain more details than the GP-based predictions. We also include the comparison details of those two comparisons in Table 8.3. Note that the comparison is based on the averaged prediction errors.

Next, we use the proposed intermittent deployment method to select measurement locations based on the mutual information while respecting the proposed constraints (8.7) and (8.8). Note that the mutual information matrix is built on all the available locations at different times. The parameters and the settings are the same as described in the first comparison. We then run 50 trials using those settings to generate different deployment policies. Since the mutual information-based method requires a full covariance matrix for the deployment ground set $\mathcal{V}$, which is extremely computationally expensive, we choose to reduce the original 100×100 deployable locations to 10×10 deployable locations. Based on the deployment result of each trial, the training set is updated using the new measurements from the deployments. After that, we make predictions for the next 10 steps. Therefore, the performance comparison between the mutual information and GP-based pipeline and the proposed DP-based pipeline is based on the newly updated prediction results. In Fig. 8.20 and Fig. 8.21, we demonstrate the averaged prediction error of each pipeline. Also, the statistics of this comparison are shown in Table 8.4.

TABLE 8.4
THE STATISTICS OF THE AVERAGED PREDICTION ERRORS OF THE PROPOSED DEEP LEARNING (DL) BASED PIPELINE AND THE GAUSSIAN PROCESS (GP) BASED PIPELINE USING 50 TRIALS.

	Mean (mm)	Std. (mm)
DL-based pipeline	**11.39**	**0.08**
GP-based pipeline	36.09	0.21

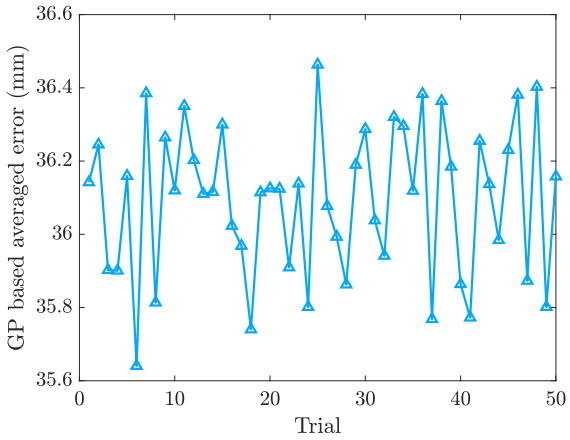


Fig. 8.20. The averaged prediction errors using Gaussian process (GP) and mutual information-based pipeline after 50 trials.

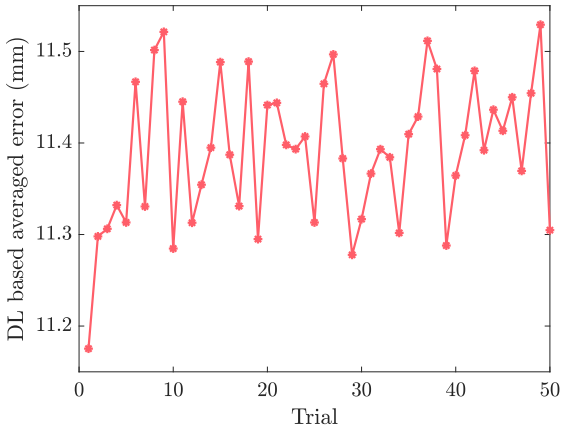


Fig. 8.21. The averaged prediction errors using the proposed deep learning (DL) based pipeline after 50 trials.

(a) (b)

Fig. 8.22. A UAV (DJI M600) mounted with a LiDAR (Velodyne VLP-16) and an onboard system (Nvidia Jetson TX2).

8.7.5 Perception Results

In this section, we demonstrate the result from a field experiment to illustrate the effectiveness of our simulated pastureland environment. This part is a collaboration work, and we demonstrate the result for the sake of the completeness of this thesis.

We experimented at Virginia Tech's Turfgrass Research Center to test our growth analysis algorithm. A 10m $\times$10m region of the Turfgrass center was reserved, and the grass was grown. It was untouched over time, but the perimeter around it was consistently mowed down. A DJI M600 UAV with a bottom-facing Velodyne VLP-16 3D-LiDAR, as shown in Fig. 8.22, was flown over the region. The localization information and LiDAR scans during flights were collected using an onboard NVIDIA Jetson TX2. This data was used to build a point cloud map of the region using a similar technique to the one used in the point cloud generation for the simulation discussed previously. These flights were conducted weekly to get temporal point cloud maps of the region. Manual measurements of the region were also collected. Nine spots throughout the region were used as the manual measurement points. These were averaged to get the manual measurements shown in Table 8.5.

TABLE 8.5
Turfgrass experiment heights.

	Week 1	Week 2	Week 3	Week 4
Hand Measurement	48.00 cm	68.17 cm	74.11 cm	69.04 cm
50th Percentile	3.97 cm	10.67 cm	12.17 cm	9.60 cm
75th Percentile	6.31 cm	16.01 cm	18.41 cm	14.60 cm
90th Percentile	8.73 cm	22.08 cm	26.11 cm	19.93 cm
95th Percentile	11.03 cm	27.48 cm	31.53 cm	24.49 cm
97.5th Percentile	13.43 cm	32.22 cm	36.64 cm	30.43 cm
99th Percentile	15.63 cm	36.43 cm	41.65 cm	36.17 cm
99.5th Percentile	16.70 cm	39.23 cm	44.29 cm	39.42 cm

TABLE 8.6
Turfgrass experiment growth.

	Week 1-2	Week 2-3	Week 3-4
Hand Measurement	20.17 cm	5.94 cm	-5.07 cm
50th Percentile	6.72 cm	1.49 cm	-2.58 cm
75th Percentile	9.70 cm	2.40 cm	-3.81 cm
90th Percentile	13.34 cm	4.03 cm	-6.18 cm
95th Percentile	16.45 cm	4.05 cm	-7.05 cm
97.5th Percentile	18.79 cm	4.40 cm	-6.21 cm
99th Percentile	20.80 cm	5.22 cm	-5.48 cm
99.5th Percentile	22.54 cm	5.05 cm	-4.87 cm

The distance between every point in the region's point cloud and the ground plane was computed to estimate the height of these points. This was done using the normal vector determined using the method described Section 8.6. Because the vast majority of LiDAR points were not at the very top of the grass, the height estimations using this method were under-estimations regardless of which percentile we looked at. This is shown in Table 8.5, and the result is shown in Fig. 8.23. However, since growth is what we are looking into, we can look at the relative differences in the estimations between each flight. These differences are shown in Table 8.6 with the first row showing the manually measured growth between sampling weeks. With perfect estimations, the percent difference between the manual measurements and estimated height would be 0%. However, our best results were found using the 99th percentile of growth estimates which had an average percent difference of 7.9% compared to manual measurements.

Those real-world experiments shown above validate our pasture simulation regime and thus our evaluation results. Meanwhile, the point cloud results from the real-world experiments are close to the results in our simulated world. All those results demonstrate the effectiveness of our proposed pipeline.

Finally, we mention some methods that can be used to tackle the case when the ground is not a plane.

A ground model can be created during the very first flight of the farm when it has had no growth. Flying the UAV over the entire farm, we could create a 3D point cloud map of the environment. Then, we could downsample the environment map (using a voxel grid filter to average all points within a voxel) and create a Delaunay triangulation of this downsampled map. This triangulation gives us localized ground planes used for height estimations.

If there are parts of the environment that we know will always be trimmed and each part

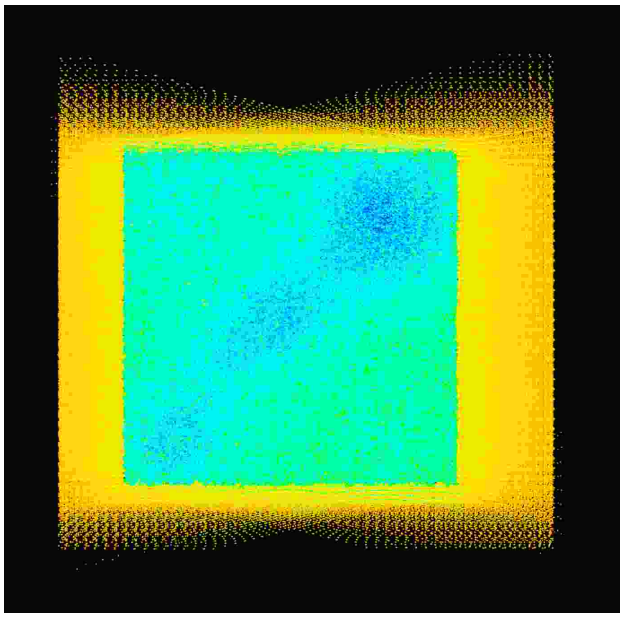


Fig. 8.23. Unprocessed point cloud of 10×10 meter Turfgrass plot with the perimeter. The UAV flew over the generated plot in the simulation. The data collected from the 3D LiDAR is shown in this figure. The yellow points are the perimeter around the plot, and the blue/green points are the points in the plot. With the plot being taller, the points registered were closer. The bluer the point, the closer it is.

can be approximated as flat ground, we could use the proposed approach in Section 8.6 to estimate both the height of crops and the height of the ground. In this case, the environment map is broken down into a grid where each grid cell covers a section of the trimmed area. Then, a localized ground plane model is found using the trimmed area to estimate the plane. With this, height estimates of non-ground points falling within the grid cell are measured using the localized ground plane.

Also, since we focus on growth (change in height), as long as the ground model is the same for each flight, we only need to focus on the relative changes. The relative growth of a "point" to a ground model will always be the same regardless of the ground model as long as that ground model does not change. Therefore, the measured growth will still be accurate even without a ground height estimation.

8.8 Conclusions

In this work, we proposed an integrated pipeline that can be used for long-term, large-scale forage perception applications. From the perspective of simulation, we demonstrated how to simulate large pastureland environments reasonably fast using parallel processing. From the perspective of pasture prediction, we proposed a new deep learning architecture that can be used for long-term pasture predictions. From the perspective of perception, we demonstrated how to get accurate pasture height estimation through regression. From the perspective of autonomy, we combined predictions and an intermittent deployment policy to deploy robots with high accuracy while at a low cost.

This work resulted in novel approaches from the initial data-generating to the final deployment testing. The proposed pipeline offers a promising and cost-effective alternative to real-life experiments and can be used as a platform for other testings.

Chapter 9

Conclusions and Future Work

9.1 Thesis Summary

Submodular optimization is a discrete optimization that utilizes the properties of submodular functions. At the same time, another topic that often comes with submodular optimization is matroid constraints. Matroids are special constraints that need to satisfy the definition. The combination of submodular objective functions and matroid constraints enables the finding of efficient solutions for such optimization problems. More interestingly, the above combination is very common in multi-robot decision-making problems, as has been shown in this thesis. Based on this motivation, we will summarize the contributions of this thesis in this manner.

The Intermittent Deployment Problem

In the first part of this thesis, we formulated an intermittent deployment problem in Chapter 3 for multi-robot systems in the context of a spatiotemporal environmental monitoring problem. As this is a constrained decision-making problem for multi-robot systems, we demonstrated how to use matroid and knapsack constraints to model different instances of the proposed problem. We then illustrated how to utilize a general submodular maximization method to solve the problem and proved the solution's effectiveness. Next, we move forward to a more complicated case where the predefined settings of our proposed intermittent planning problem can be influenced by other conditions, e.g., the environment prediction results. Thus, we

formulated a coupled intermittent problem in Chapter 4, where the first problem's outcome changes the second problem's initial settings on a temporal scale. Based on this setting, we also demonstrated how to solve the proposed coupled problem by formulating the coupled problem as a task allocation problem. Similarly, an environmental monitoring problem is also given to show how this problem is formulated and solved in real-world scenarios. Therefore, from the constraint perspective, we illustrated how to use matroid constraints to model the combined constraints. From the objective function perspective, we demonstrated under which conditions the objective function is submodular or modular, indicating an effective solution's existence.

The Resilient and Decentralized Submodular Optimization

Following the above-proposed problems, we extend our research to the cases where the robots in our decision-making problems are vulnerable to attacks. In this context, we can divide our contributions into two categories. In the first category, we mainly consider our problem in a centralized case, where a central point of command is available to all robots. Specifically, in Chapter 5, we considered a problem where different robots in a multi-robot system are tasked with selecting a common action set from the system ground set according to individualized objective functions, and the system performance is quantified by the minimum contribution from its robots. Also, attackers attempt to disrupt the system by removing a robot's contribution after knowing the system solution and thus can attack perfectly. Therefore, how to design decentralized algorithms for our multi-robot coordination problems and how to increase the system resilience are the main focus of this part. Through the proposed algorithm, we were able to solve the problem efficiently while having theoretical guarantees.

Then, we switch our focus from the centralized scenario to a decentralized scenario. In centralized scenarios, all robots can communicate with each other through a central commander.

In decentralized scenarios, robots can only communicate with neighbors, which significantly changes how centralized solutions work. Moreover, decentralized systems are more vulnerable to attacks or failures. In Chapter 6, we proposed a fully distributed algorithm for the problem of the resilient submodular action selection, where the system's performance can be guaranteed in a worst-case scenario. We evaluated the algorithm's performance through extensive simulations. In Chapter 7, we extended our problem in an asynchronous distributed scenario. In particular, we formulated our problem as a general decentralized task allocation problem, using submodular and matroid properties. We then proposed and demonstrated the effectiveness of a fully asynchronized distributed algorithm.

The Integrated Pipeline for Long-term Forage Perception Application

In the last part of this thesis, we demonstrated an integrated pipeline in Chapter 8 that can be used for long-term, large-scale forage perception applications using multi-robot systems. We first demonstrated how to model the environmental prediction problem using raw agricultural data. We then formulate a long-term environmental monitoring problem based on the predictions, which can also be viewed as a task allocation problem. Finally, extensive experiments are performed to demonstrate the effectiveness of the proposed pipeline. We can summarize the contribution of this integrated pipeline as follows. From the perspective of simulation, we demonstrated how to simulate large pastureland environments reasonably fast using parallel processing. From the perspective of pasture prediction, we demonstrated how to use a new deep-learning architecture for long-term pasture predictions. From the perspective of perception, we demonstrated how to get accurate pasture height estimation through regression. From the perspective of autonomy, we showed how to combine predictions and an intermittent deployment policy to deploy robots while satisfying the requirements.

9.2 Future Work

In this thesis, we explored multiple topics in multi-robot task allocation problems. We also explored robust and resilient versions of the proposed problems. In the cases when there is no central point of command available, we also presented a decentralized algorithm. Based on our contributions and framework, we can extend our investigation in several directions.

Submodular Task Allocation in Probabilistic Communication Networks

In our decentralized submodular optimization chapters, i.e., Chapters 6 and 7, we studied the problem based on the assumption that the network communications are asynchronous and deterministic. Also, the robots in this network can communicate with their neighbors within a finite time interval, and the intervals for different robots are fixed. However, in real-world scenarios, there are cases where the communications between different agents are stochastic. In those scenarios, agents can communicate with their neighbors based on a predefined probability distribution. Therefore, one possible direction of future work would be exploring the submodular action selection problems under stochastic communication assumptions.

Distributed Decision-making in Dynamic Networks

Our proposed decentralized decision-making method works for scenarios where the connections in communication graphs can be changed as long as they are connected. However, a more complicated scenario would be to consider a case where the robots in a system can come and go as they want. In this case, the number of robots and the communication graphs may be changed. This is a more challenging scenario since the rest of the robots in the system may need to change their selections/tasks accordingly. Therefore, another direction for our future work is to investigate this problem and develop an efficient algorithm to deal with this dynamic case.

Resilient Action Selection for Communication Failures

In Chapter 6, we considered the case where the agents in a team could be attacked. In more complicated real-world scenarios, the communications between different robots can also be attacked, which results in communication failures. In some cases, the communication graph can even be partitioned into several groups due to those failures. Therefore, we could also investigate how the performance is influenced by those failures and how to develop a strategy to solve the problem with performance guarantees. For example, if we know the probability of each communication link that can be attacked, which strategy should we use to ensure the system's performance in the worst-case scenario that we used in the thesis?

Improving the Fundamental Submodular Optimization Performance

In this thesis, we utilized the properties of the submodularity of objective functions to prove the theoretical guarantees of different proposed algorithms. This property is also called a diminishing return property, where the marginal gain becomes larger when the set becomes smaller. Built on this property, we were able to generalize the problem and explore the performance of centralized, decentralized, and resilient versions of the problem. However, the most common property that we exploited in all those proposed algorithms is that they are greedy. That is because the greedy method is the most convenient way to exploit the properties of submodular functions, as the values of objective functions are non-decreasing. Therefore, one open question would be whether it is possible to improve the fundamental performance of submodular optimizations with decreasing performance. Though many sophisticated methods have been proposed, an easy-to-implement (low time complexity) with a better performance is still yet to come. Thus, this would be another direction for this thesis's future work.

Bibliography

[1] J. Liu and R. K. Williams, "Monitoring over the long term: Intermittent deployment and sensing strategies for multi-robot teams," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2020, pp. 7733–7739.

[2] J. Liu and R. K. Williams, "Coupled temporal and spatial environment monitoring for multi-agent teams in precision farming," in *Proc. of the IEEE Conference on Control Technology and Applications*, 2020, pp. 273–278.

[3] J. Liu and R. K. Williams, "Optimal intermittent deployment and sensor selection for environmental sensing with multi-robot teams," in *Proc. of the International Conference on Robotics and Automation*, 2018, pp. 1078–1083.

[4] J. Liu and R. K. Williams, "Submodular optimization for coupled task allocation and intermittent deployment problems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3169–3176, 2019.

[5] J. Liu and R. K. Williams, "A fast algorithm for robust action selection in multi-agent systems," *arXiv preprint arXiv:2206.11824*, pp. 1–6, 2022.

[6] J. Liu, L. Zhou, P. Tokekar, and R. K. Williams, "Distributed resilient submodular action selection in adversarial environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5832–5839, 2021.

[7] J. Liu, M. Rangwala, K. S. Ahluwalia, S. Ghajar, H. S. Dhami, P. Tokekar, B. F. Tracy, and R. K. Williams, "Intermittent deployment for large-scale multi-robot forage per-

ception: Data synthesis, prediction, and planning," *IEEE Transactions on Automation Science and Engineering*, pp. 1–21, 2022.

[8] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Berlin, Germany: Springer, 2003, vol. 24.

[9] A. Krause and D. Golovin, "Submodular function maximization," in *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2014, vol. 3, pp. 71–104.

[10] G. Qu, D. Brown, and N. Li, "Distributed greedy algorithm for multi-agent task assignment problem with submodular utility functions," *Automatica*, vol. 105, pp. 206–215, 2019.

[11] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Math. Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[12] M. Conforti and G. Cornuéjols, "Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem," *Discrete Appl. Math.*, vol. 7, no. 3, pp. 251–274, 1984.

[13] M. Sviridenko, J. Vondrák, and J. Ward, "Optimal approximation for submodular and supermodular optimization with bounded curvature," *Math. Oper. Res.*, vol. 42, no. 4, pp. 1197–1218, 2017.

[14] J. G. Oxley, *Matroid theory*. New York, NY, USA: Oxford University Press, 2006, vol. 3.

[15] K. Cesare, R. Skeele, S.-H. Yoo, Y. Zhang, and G. Hollinger, "Multi-uav exploration with limited communication and battery," in *Proc. of the IEEE Int. Conf. Robot. Autom.*, 2015, pp. 2230–2235.

[16] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent communication control of mobile robot networks under time-critical dynamic tasks," in *Proc. of the IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–9.

[17] W. Dong, "Tracking control of multiple-wheeled mobile robots with limited information of a desired trajectory," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 262–268, 2012.

[18] S. Jorgensen, R. H. Chen, M. B. Milam, and M. Pavone, "The matroid team surviving orienteers problem: Constrained routing of heterogeneous teams with risky traversal," in *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 5622–5629.

[19] B. Zhang, J. Liu, and H. Chen, "AMCL based map fusion for multi-robot slam with heterogenous sensors," in *Proc. of the IEEE Int. Conf. Inf. Autom.*, 2013, pp. 822–827.

[20] J. Liu, H. Chen, and B. Zhang, "Square root unscented kalman filter based ceiling vision slam," in *Proc. of the IEEE Int. Conf. Robot. Biomimetics*, 2013, pp. 1635–1640.

[21] M. Corah and N. Michael, "Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice," *Auton. Robot.*, vol. 43, no. 2, pp. 485–501, Feb. 2019.

[22] R. K. Williams, A. Gasparri, and G. Ulivi, "Decentralized matroid optimization for topology constraints in multi-robot allocation problems," in *Proc. of the IEEE IEEE International Conference on Robotics and Automation*, 2017, pp. 293–300.

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

[24] S. L. Smith, M. Schwager, and D. Rus, "Persistent robotic tasks: Monitoring and sweeping in changing environments," *IEEE Trans. Robot.*, vol. 28, no. 2, pp. 410–426, Apr. 2012.

[25] X. Lan and M. Schwager, "Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields," in *Proc. of the IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2415–2420.

[26] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to persistent monitoring tasks," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1106–1118, Oct. 2016.

[27] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *J. Mach. Learn. Res.*, vol. 9, no. Feb, pp. 235–284, Feb. 2008.

[28] S. Joshi and S. Boyd, "Sensor selection via convex optimization," *IEEE Trans. Signal Process.*, vol. 57, no. 2, pp. 451–462, 2008.

[29] M. S. Corson, C. A. Rotz, and R. H. Skinner, "Evaluating warm-season grass production in temperate-region pastures: a simulation approach," *Agricultural Systems*, vol. 93, no. 1-3, pp. 252–268, 2007.

[30] T. Zhai, R. H. Mohtar, A. R. Gillespie, G. R. von Kiparski, K. D. Johnson, and M. Neary, "Modeling forage growth in a midwest usa silvopastoral system," *Agroforestry systems*, vol. 67, no. 3, pp. 243–257, 2006.

[31] R. N. Smith, M. Schwager, S. L. Smith, B. H. Jones, D. Rus, and G. S. Sukhatme, "Persistent ocean monitoring with underwater gliders: Adapting sampling resolution," *J Field Robotics.*, vol. 28, no. 5, pp. 714–741, 2011.

[32] M. Egorov, M. J. Kochenderfer, and J. J. Uudmae, "Target surveillance in adversarial environments using POMDPs," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2473–2476.

[33] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. I. Jordan, and S. S. Sastry, "Kalman filtering with intermittent observations," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1453–1464, Sep. 2004.

[34] G. Hollinger and S. Singh, "Multi-robot coordination with periodic connectivity," in *Proc. of the IEEE Int. Conf. Robot. Autom.*, 2010, pp. 4457–4462.

[35] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 4863–4869.

[36] J. Melvin, P. Keskinocak, S. Koenig, C. Tovey, and B. Y. Ozka, "Multi-robot routing with rewards and disjoint time windows," in *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 2332–2337.

[37] M. W. Savelsbergh, "Local search in routing problems with time windows," *Ann. Oper. Res.*, vol. 4, no. 1, pp. 285–305, 1985.

[38] M. McIntire, E. Nunes, and M. Gini, "Iterated multi-robot auctions for precedence-constrained task scheduling," in *Proc. Int. Conf. Auton. Agents Multiagent Syst.*, 2016, pp. 1078–1086.

[39] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. Hokeben, NJ, USA: John Wiley & Sons, 2014.

[40] A. N. Shiryaev, "On optimum methods in quickest detection problems," *Theory of Probability & Its Applications*, vol. 8, no. 1, pp. 22–46, 1963.

[41] W. S. Lovejoy, "Some monotonicity results for partially observed markov decision processes," *Oper. Res.*, vol. 35, no. 5, pp. 736–743, 1987.

[42] V. Krishnamurthy, "How to schedule measurements of a noisy markov chain in decision making?" *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4440–4461, 2013.

[43] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[44] C. Chekuri, J. Vondrák, and R. Zenklusen, "Submodular function maximization via the multilinear relaxation and contention resolution schemes," *SIAM J. Comput.*, vol. 43, no. 6, pp. 1831–1879, Nov. 2014.

[45] J. Vondrák, "Optimal approximation for the submodular welfare problem in the value oracle model," in *Proc. ACM Symp. Theory Comput.* ACM, 2008, pp. 67–74.

[46] A. Badanidiyuru and J. Vondrák, "Fast algorithms for maximizing submodular functions," in *Proc. ACM-SIAM Symp. Discrete Algorithms*. SIAM, 2014, pp. 1497–1514.

[47] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[48] R. K. Williams, A. Gasparri, G. Ulivi, and G. S. Sukhatme, "Generalized topology control for nonholonomic teams with discontinuous interactions," *IEEE Transactions on Robotics*, vol. 33, no. 4, pp. 994–1001, 2017.

[49] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 939–954, 2004.

[50] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *Int. J. Robot. Res.*, vol. 32, no. 12, pp. 1495–1512, Oct. 2013.

[51] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Trans. Autom. Control*, vol. 62, no. 7, pp. 3109–3121, Jul. 2017.

[52] S. T. Jawaid and S. L. Smith, "Submodularity and greedy algorithms in sensor scheduling for linear dynamical systems," *Automatica*, vol. 61, pp. 282–288, Nov. 2015.

[53] M. Saha and P. Isto, "Multi-robot motion planning by incremental coordination," in *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2006, pp. 5960–5963.

[54] N. Kalra, D. Ferguson, and A. Stentz, "A generalized framework for solving tightly-coupled multirobot planning problems," in *Proc. of the IEEE Int. Conf. Robot. Autom.*, 2007, pp. 3359–3364.

[55] M. Gienger, M. Toussaint, and C. Goerick, "Task maps in humanoid robot manipulation," in *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 2758–2764.

[56] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[57] V. Tzoumas, A. Jadbabaie, and G. J. Pappas, "Resilient non-submodular maximization over matroid constraints," *arXiv:1804.01013*, 2018.

[58] R. Santiago and F. B. Shepherd, "Multi-agent and multivariate submodular optimization," *arXiv:1612.05222*, 2016.

[59] A. Krause and D. Golovin, *Submodular Function Maximization*. Cambridge, UK: Cambridge University Press, 2014.

[60] P. R. Goundan and A. S. Schulz, "Revisiting the greedy approach to submodular set function maximization," *Optim. online*, pp. 1–25, 2007.

[61] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *J. Artif. Intell. Res.*, vol. 34, pp. 707–755, 2009.

[62] L. Heintzman, A. Hashimoto, N. Abaid, and R. K. Williams, "Anticipatory planning and dynamic lost person models for human-robot search and rescue," in *IEEE Int. Conf. Robot. Autom.*, 2021, pp. 8252–8258.

[63] R. K. Williams, A. Gasparri, A. Priolo, and G. S. Sukhatme, "Evaluating network rigidity in realistic systems: Decentralization, asynchronicity, and parallelization," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 950–965, 2014.

[64] L. Heintzman and R. K. Williams, "Multi-agent intermittent interaction planning via sequential greedy selections over position samples," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 534–541, 2020.

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

[66] A. Krause, B. McMahan, C. Guestrin, and A. Gupta, "Selecting observations against adversarial objectives," *Adv. Neural Inf. Process. Syst.*, vol. 20, pp. 1–8, 2007.

[67] Q. Hou and A. Clark, "Robust maximization of correlated submodular functions under cardinality and matroid constraints," *IEEE Trans. Autom. Control*, vol. 66, no. 12, pp. 6148–6155, 2021.

[68] T. Powers, J. Bilmes, S. Wisdom, D. W. Krout, and L. Atlas, "Constrained robust submodular optimization," in *NIPS OPT2016 workshop*, 2016.

[69] T. Fujito, "Approximation algorithms for submodular set cover with applications," *IEICE Trans. Information and Systems*, vol. 83, no. 3, pp. 480–487, 2000.

[70] K. Saulnier, D. Saldana, A. Prorok, G. J. Pappas, and V. Kumar, "Resilient flocking for mobile robot teams," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1039–1046, 2017.

[71] S. Gil, S. Kumar, M. Mazumder, D. Katabi, and D. Rus, "Guaranteeing spoof-resilient multi-robot networks," *Auton. Robot.*, vol. 41, no. 6, pp. 1383–1400, 2017.

[72] A. Mitra, J. A. Richards, S. Bagchi, and S. Sundaram, "Resilient distributed state estimation with mobile agents: Overcoming Byzantine adversaries, communication losses, and intermittent measurements," *Auton. Robot.*, vol. 43, no. 3, pp. 743–768, 2019.

[73] K. Wardega, R. Tron, and W. Li, "Masquerade attack detection through observation planning for multi-robot systems," in *Proc. of the Int. Conf. Auton. Agents Multi. Syst.*, 2019, pp. 2262–2264.

[74] D. R. Raymond and S. F. Midkiff, "Denial-of-service in wireless sensor networks: Attacks and defenses," *IEEE Pervasive Comput.*, vol. 7, no. 1, pp. 74–81, 2008.

[75] V. Tzoumas, K. Gatsis, A. Jadbabaie, and G. J. Pappas, "Resilient monotone submodular function maximization," in *Proc. of the IEEE Conf. Decis. Control*, 2017, pp. 1362–1367.

[76] G. Shi, L. Zhou, and P. Tokekar, "Robust multiple-path orienteering problem: Securing against adversarial attacks," in *Proc. Robot.: Sci. Syst.*, 2020.

[77] L. Heintzman and R. K. Williams, "A predictive autonomous decision aid for calibrating human-autonomy reliance in multi-agent task assignment," *arXiv preprint arXiv:2112.10252*, 2021.

[78] L. Zhou, V. Tzoumas, G. J. Pappas, and P. Tokekar, "Distributed attack-robust submodular maximization for multi-robot planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 2479–2485.

[79] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, 2009.

[80] D. Grimsman, M. S. Ali, J. P. Hespanha, and J. R. Marden, "The impact of information in distributed submodular maximization," *IEEE Trans. Control. Netw. Syst.*, vol. 6, no. 4, pp. 1334–1343, 2018.

[81] E. Mackin and S. Patterson, "Submodular optimization for consensus networks with noise-corrupted leaders," *IEEE Trans. Autom. Control*, vol. 64, no. 7, pp. 3054–3059, 2018.

[82] A. Clark, L. Bushnell, and R. Poovendran, "A supermodular optimization framework for leader selection under link noise in linear multi-agent systems," *IEEE Trans. Autom. Control*, vol. 59, no. 2, pp. 283–296, 2013.

[83] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[84] S. S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, 2004.

[85] V. V. Vazirani, *Approximation algorithms*. Springer, 2001.

[86] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[87] A. Nedic, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Trans. Autom. Control*, vol. 55, no. 4, pp. 922–938, 2010.

[88] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[89] A. Nedic, "Distributed gradient methods for convex machine learning problems in networks: Distributed optimization," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 92–101, 2020.

[90] S. Lee and A. Nedić, "Asynchronous gossip-based random projection algorithms over networks," *IEEE Trans. Autom. Control*, vol. 61, no. 4, pp. 953–968, Apr. 2015.

[91] M. Shamaiah, S. Banerjee, and H. Vikalo, "Greedy sensor selection: Leveraging submodularity," in *cdc*, 2010, pp. 2572–2577.

[92] L. Luo, N. Chakraborty, and K. Sycara, "Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks," *IEEE Trans. Robot.*, vol. 31, no. 1, pp. 19–30, 2014.

[93] A. Gasparri, R. K. Williams, A. Priolo, and G. S. Sukhatme, "Decentralized and parallel constructionsfor optimally rigid graphs in $\mathbb{R}^2$," *IEEE Transactions on Mobile Computing*, vol. 14, no. 11, pp. 2216–2228, 2015.

[94] P. Segui-Gasco, H.-S. Shin, A. Tsourdos, and V. Segui, "Decentralised submodular multi-robot task allocation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 2829–2834.

[95] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proc. IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[96] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Ann. Oper. Res.*, vol. 14, no. 1, pp. 105–123, 1988.

[97] M. M. Zavlanos, L. Spesivtsev, and G. J. Pappas, "A distributed auction algorithm for the assignment problem," in *Proc. of the IEEE Conf. Decis. Control*, 2008, pp. 1212–1217.

[98] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[99] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, "Distributed submodular maximization: Identifying representative elements in massive data," in *Proc. Adv. Neural. Inf. Process. Syst.*, 2013, pp. 2049–2057.

[100] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in mapreduce and streaming," *ACM Trans. Parallel Comput.*, vol. 2, no. 3, pp. 1–22, 2015.

[101] Z. Á. Mann, "The top eight misconceptions about NP-hardness," *Computer*, vol. 50, no. 5, pp. 72–79, 2017.

[102] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. IEEE Symp. Found. Comput. Sci.*, 2003, pp. 482–491.

[103] A. G. Dimakis, S. Kar, J. M. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proc. IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[104] P. Hennig, "Animating samples from gaussian distributions," Technical Report 8. Spemannstraße, 72076 Tübingen, Germany: Max Planck, Tech. Rep., 2013.

[105] J. Bengtsson, J. M. Bullock, B. Egoh, C. Everson, T. Everson, T. O'Connor, P. J. O'Farrell, H. G. Smith, and R. Lindborg, "Grasslands—more important for ecosystem services than you might think," *Ecosphere*, vol. 10, no. 2, pp. 1–20, 2019.

[106] R. L. Kallenbach, "Describing the dynamic: Measuring and assessing the value of plants in the pasture," *Crop Science*, vol. 55, no. 6, p. 2531–2539, 2015.

[107] J. Y. L. Tay, A. Erfmeier, and J. M. Kalwij, "Reaching new heights: can drones replace current methods to study plant population dynamics?" *Plant Ecology*, vol. 219, no. 10, p. 1139–1150, Oct 2018.

[108] K. R. Harmoney, K. J. Moore, J. R. George, E. C. Brummer, and J. R. Russell, "Determination of pasture biomass using four indirect methods," *Agronomy Journal*, vol. 89, no. 4, p. 665–672, Jul 1997.

[109] "Gazebo simulator," http://gazebosim.org.

[110] D. P. Holzworth, N. I. Huth, P. G. deVoil, E. J. Zurcher, N. I. Herrmann, G. McLean, K. Chenu, E. J. van Oosterom, V. Snow, and C. Murphy, "APSIM–evolution towards a new generation of agricultural systems simulation," *Environmental Modelling & Software*, vol. 62, p. 327–350, 2014.

[111] M. T. Schaefer and D. W. Lamb, "A combination of plant NDVI and LiDAR measurements improve the estimation of pasture biomass in tall fescue (festuca arundinacea var. fletcher)," *Remote Sensing*, vol. 8, no. 22, p. 109, Feb 2016.

[112] M. Rangwala, J. Liu, K. S. Ahluwalia, S. Ghajar, H. S. Dhami, B. F. Tracy, P. Tokekar, and R. K. Williams, "DeepPaSTL: Spatio-temporal deep learning methods for predicting long-term pasture terrains using synthetic datasets," *Agronomy*, vol. 11, no. 11, p. 2245, 2021.

[113] S. McCammon, G. Marcon dos Santos, M. Frantz, T. Welch, G. Best, R. K. Shearman, J. D. Nash, J. A. Barth, J. A. Adams, and G. A. Hollinger, "Ocean front detection and tracking using a team of heterogeneous marine vehicles," *J. Field Robot.*, 2021.

[114] M. Ahmed, D. Parsons, J. Morel, K. Uttam, B. Sandström, M. Lanna, and J. Wallsten, "APSIM next generation to model red clover under nordic climate," in *Proc. of the Int. Crop Modelling Symposium*, 2020.

[115] C. Bosi, P. C. Sentelhas, N. I. Huth, J. R. M. Pezzopane, M. P. Andreucci, and P. M. Santos, "APSIM-tropical pasture: A model for simulating perennial tropical grass

growth and its parameterisation for palisade grass (brachiaria brizantha)," *Agricultural Systems*, vol. 184, p. 102917, Sep 2020.

[116] J. Liu and R. K. Williams, "Data-driven models with expert influence: A hybrid approach to spatiotemporal process estimation," in *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 2467–2473.

[117] L. Heintzman and R. K. Williams, "Nonlinear observability of unicycle multi-robot teams subject to nonuniform environmental disturbances," *Autonomous Robots*, vol. 44, no. 7, pp. 1149–1166, 2020.

[118] L. Heintzman and R. K. Williams, "Nonlinear observability of unicycle multi-agent teams subject to sensor bias and environmental disturbance," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 6186–6191.

[119] M. Oliu, J. Selva, and S. Escalera, "Folded recurrent neural networks for future video prediction," in *Proc. of the European Conf. on Computer Vision*, 2018, pp. 716–731.

[120] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[121] D. J. MacKay, "A practical bayesian framework for backpropagation networks," *Neural computation*, vol. 4, no. 3, pp. 448–472, 1992.

[122] P. Xu, J.-H. Cho, and A. Salado, "Expert opinion fusion framework using subjective logic for fault diagnosis," *IEEE Trans. Cybern.*, pp. 1–12, 2020.

[123] P. Xu, A. Salado, and G. Xie, "A reinforcement learning approach to design verification strategies of engineered systems," in *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics*, 2020, pp. 3543–3550.

[124] J. Liu and A. Kumar, "Detecting sensor level spoof attacks using joint encoding of temporal and spatial features," in *Proc. of the IEEE Int. Conf. Imaging for Crime Detection and Prevention*, 2016, pp. 1–6.

[125] J. Liu and A. Kumar, "Detecting presentation attacks from 3d face masks under multispectral imaging," in *Proc. of the IEEE Conf. Computer Vision and Pattern Recognition Workshop on Biometrics*, 2018, pp. 47–52.

[126] P. Xu, A. Salado, and X. Deng, "A parallel tempering approach for efficient exploration of the verification tradespace in engineered systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2022.

[127] P. Xu and A. Salado, "Modeling correction activities in the context of verification strategies," *Systems Engineering*, vol. 25, no. 2, pp. 173–188, 2022.

[128] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[129] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proc. of the Int. Conf. on Mach. Learn.*, 2014, pp. 1278–1286.

[130] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference." *J. Mach. Learn. Res.*, vol. 14, no. 5, 2013.

[131] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proc. of the Int. Conf. on Mach. Learn.*, 2016, pp. 1050–1059.

[132] P. Baldi and P. J. Sadowski, "Understanding dropout," in *Proc. of the Adv. Neural Inf. Process. Syst.*, vol. 26, 2013, pp. 2814–2822.

[133] A. Damianou and N. D. Lawrence, "Deep gaussian processes," in *Artificial intelligence and statistics*, 2013, pp. 207–215.

[134] G. A. Hollinger and S. Singh, "Multirobot coordination with periodic connectivity: Theory and experiments," *IEEE Trans. Robot.*, vol. 28, no. 4, pp. 967–973, 2012.

[135] P. Tokekar and V. Kumar, "Visibility-based persistent monitoring with robot teams," in *Proc. of the IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 3387–3394.

[136] T. Li, C. Wang, M. Q.-H. Meng, and C. W. de Silva, "Attention-driven active sensing with hybrid neural network for environmental field mapping," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–18, 2021.

[137] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *Int. J. Robot. Res.*, vol. 33, no. 9, pp. 1271–1287, 2014.

[138] L. V. Nguyen, S. Kodagoda, R. Ranasinghe, and G. Dissanayake, "Information-driven adaptive sampling strategy for mobile robotic wireless sensor network," *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 1, pp. 372–379, 2015.

[139] Y. Xu, J. Choi, S. Dass, and T. Maiti, "Efficient bayesian spatial prediction with mobile sensor networks using gaussian markov random fields," *Automatica*, vol. 49, no. 12, pp. 3520–3530, 2013.

[140] R. Wehbe and R. K. Williams, "Optimizing topologies for probabilistically secure multi-robot systems," in *Proc. of the IEEE Int. Conf. Robot. Autom.*, 2020, pp. 6640–6646.

[141] S. V. Archontoulis, F. E. Miguez, and K. J. Moore, "Evaluating APSIM maize, soil water, soil nitrogen, manure, and soil temperature modules in the midwestern united states," *Agronomy Journal*, vol. 106, no. 3, p. 1025–1040, 2014.

[142] F. Li, P. Newton, and M. Lieffering, "Testing simulations of intra- and inter-annual variation in the plant production response to elevated co2 against measurements from an 11-year face experiment on grazed pasture," *Global change biology*, vol. 20, Aug 2013.

[143] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. of the Int. Conf. Med. Image. Comput. Comput. Assist. Interv.* Springer, 2015, pp. 234–241.