# Robust Multiple-Path Orienteering Problem: Securing Against Adversarial Attacks

Guangyao Shi[†], Lifeng Zhou[‡], Pratap Tokekar[††]

*Abstract*—**The multiple-path orienteering problem asks for paths for a team of robots that maximize the total reward collected while satisfying budget constraints on the path length. This problem models many multi-robot routing tasks such as exploring unknown environments and information gathering for environmental monitoring. In this paper, we focus on how to make the robot team robust to failures when operating in adversarial environments. We introduce the Robust Multiple-path Orienteering Problem (RMOP) where we seek worst-case guarantees against an adversary that is capable of attacking at most $\alpha$ robots. We consider two versions of this problem: RMOP offline and RMOP online. In the offline version, there is no communication or replanning when robots execute their plans and our main contribution is a general approximation scheme with a bounded approximation guarantee that depends on $\alpha$ and the approximation factor for single robot orienteering. In particular, we show that the algorithm yields a (i) constant-factor approximation when the cost function is modular; (ii) $\log$ factor approximation when the cost function is submodular; and (iii) constant-factor approximation when the cost function is submodular but the robots are allowed to exceed their path budgets by a bounded amount. In the online version, RMOP is modeled as a two-player sequential game and solved adaptively in a receding horizon fashion based on Monte Carlo Tree Search (MCTS). In addition to theoretical analysis, we perform simulation studies for ocean monitoring and tunnel information-gathering applications to demonstrate the efficacy of our approach.**

*Index Terms*—**Multi-robot system, Orienteering, robust planning, adversarial attacks.**

## I. Introduction

The Orienteering Problem (OP) is that of determining a path, whose length is less than a given budget, from a given starting vertex that maximizes the total reward collected along the path [1]. The reward depends on the vertices visited along the path. The OP[1] naturally models informative-path planning: a robot is tasked to gather as much information from the environment as possible within a given time or energy budget. For example, in [2]–[4], ocean monitoring, opportunistic surveillance, and 3D reconstruction tasks are formulated as the OP or its variants. In general, the OP is NP-hard but there are constant-factor approximation algorithms for many variants [5]. This includes the Multiple-path Orienteering Problem (MOP) [5] where the goal is to design paths for $N$ robots such that the sum of the rewards collected by all the robots is maximized. In this paper, we introduce the robust variant of OP. Specifically, we introduce the Robust Multiple-Path Orienteering Problem (RMOP) motivated by scenarios where robots operate in adversarial or failure-prone environments.

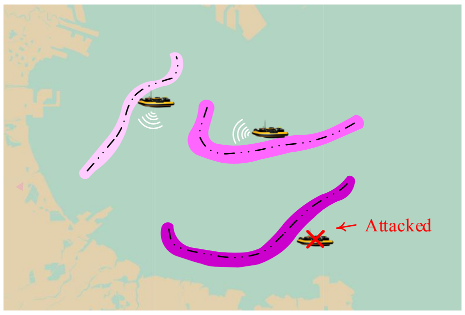


Fig. 1. Case study of monitoring a marine environment with aquatic robots. The robots are tasked with finding informative paths to gather data. The darker the color of the path is, the more valuable path since it gathers information from a more important region. We investigate the question of how the robots should plan their paths if we expect some of the robots to fail due to adversarial elements or natural causes.

Figure 1 shows a motivating scenario where a team of underwater robots is tasked with gathering data in an ocean. However, some robots in the team may fail to complete their paths either due to adversarial attacks [6] or hardware malfunction [7]. If a robot fails, then the data gathered by it is lost. Our goal is to provide efficient planning and coordination algorithms that are resilient to such failures.

Building robot teams that are robust to adversarial attacks is emerging as an important research area [8]–[11]. Our approach differs from classical fault-tolerant frameworks [12]–[14] that focus on making individual robots robust to failures. Instead, we focus on the question of how should the team coordinate their actions to improve redundancy in their plans such that even if some robots fail, the overall performance of the team will not drop significantly. As such, our work is completely different from the work on making individual robots robust.

In this paper, we focus on the RMOP to make progress toward the aforementioned broader goal. The RMOP seeks plans for a team of $N$ robots that guard against worst-case failures. Of course, in the worst-case, all $N$ robots may fail. To make it more meaningful we study the case where at most a given number $\alpha < N$ robots may fail. What we seek is to

This work is supported by the National Science Foundation under Grant No. 1943368, and the Office of Naval Research under Grant No. N000141812829.

[†]Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742 USA email:gyshi@terpmail.umd.edu.

[‡]Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104, USA email:lz457@drexel.edu.

[††]Department of Computer Science, University of Maryland, College Park, MD 20742 USA email: tokekar@umd.edu

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

understand how the performance of the team will be affected as a function of $\alpha$. We consider two types of RMOP, in both of which we seek to find a path consisting of multiple steps for each robot. In the offline RMOP in which robots cannot communicate with each other or the base station during tasks, our main contribution is an algorithmic scheme that uses a single robot OP solution as a subroutine. Choosing an appropriate subroutine allows us to investigate three variants of the original problem. In the general version, the reward collected by an individual robot is a submodular function of the vertices along the path. Submodularity is the property of diminishing returns [15]. Many information gathering measures such as mutual information [16] and coverage area [17] are known to be submodular. We also study special cases where the reward function is strictly modular (i.e., additive) and where the budget constraint for each robot can be relaxed by a bounded amount. In the online RMOP, we model the problem as a sequential two-player game and propose an adaptive strategy based on MCTS, and the problem is solved in a receding horizon fashion with the history of the observed attack taken into account.

## A. Related work

The orienteering problem has been researched extensively by both theoretical computer science and operations research communities. The review by Vansteenwegen [1] summarizes various algorithms for OP and its variants. We highlight the results most closely related to our work. Blum et al. [5] presented a polynomial-time 4–approximation for OP when the objective function is modular. This result is then extended to yield a 5–approximation for the MOP assuming all robots start at different vertices. If the reward function is submodular, Chekuri and Pal [18] present a recursive greedy algorithm for a single robot that yields a $O(\log(OPT))$ approximation algorithm, where $OPT$ is the reward collected by the optimal algorithm. The algorithm runs in quasi-polynomial time.

Singh et al. [19] showed how to use OP and MOP for active information gathering to learn a spatial model of the environment represented by Gaussian Processes. Their algorithms sequentially find paths for each robot using the single-robot algorithms by Blum et al. [5] and Chekuri and Pal [18] as subroutines. Atanasov et al. [20] recently presented a decentralized version for multi-robot information gathering along similar lines as [19]. They use a submodular objective function but solve a finite horizon planning problem as opposed to OP. However, none of these works account for potential failures of the robots, as we do in RMOP.

Recently, Jorgensen et al. introduced the Matroids Team Surviving Orienteers Problem (MTSO) [2] which does account for individual robot failures. They assume that there is some given probability of failure associated with every edge in the environment. The goal is to maximize the expected rewards while ensuring each path satisfies some survival chance constraints. MTSO is appropriate when the failures of robots are random and follow a known distribution. The version we study, the RMOP, accounts for worst-case failures which makes it better suited when operating in adversarial conditions or in stochastic conditions when worst-case guarantees are sought due to unknown probability distributions.

Our work builds on recent work on robust submodular maximization [21]–[26] which selects sets that are robust to worst-case removal of some subset of items. The challenge in this framework is to solve the trade-off between too much overlap, thereby not enough coverage (i.e., reward) and too little overlap, thereby not enough redundancy. The conceptual idea in these papers is similar — the final solution consists of two subsets, one that has enough redundancy to ensure robustness against worst-case removal and the other that has enough coverage to get good overall performance. Orlin et al. [21] term the former as “copies” whereas it is called “baits” in [25]. The robust submodular maximization formulation has been applied for multi-robot, multi-target tracking in centralized [25] and decentralized settings [26] as well as for active information gathering with multi-robot teams [24].

We seek similar robustness guarantees as in the works mentioned in the previous paragraph. The key technical advancement we make is that these prior work solve a single-step selection problem whereas we solve a multi-step planning problem. As a result, the single robot problem in the prior work can be trivially solved optimally (amounts to selecting the best amongst a finite set of options), whereas in the RMOP the single robot problem (OP) itself is NP-Hard. While both [25] and [24] use their results for planning over a finite horizon, they make key assumptions that are limiting. Schlotfeldt [24] considers the continuous counterpart of the combinatorial problem considered in this paper and they formulate the problem under the optimal control framework with one key assumption that the single robot as well as multi-robot information gathering problem (without attacks) can be solved optimally (c.f. Proposition 1). Zhou et al. [25] repeatedly solve the one-step problem at each time step. Instead, we show how to use an approximate solution to the OP to yield a bounded approximation solution to the combinatorial problem RMOP.

## B. Contributions

The main contributions of this paper are as follows. We introduce the Robust Multiple-Path Orienteering Problem and consider two specific types of this problem. For the offline case, We present a general approximation scheme to solve RMOP. We analyze the running time and the performance of the algorithm. In particular, we show that the approximation ratio is a constant of the approximation factor for single robot OP. We show how to employ three single robot algorithms for modular and submodular OP as subroutines in our algorithm and analyze their performance. For the online case, we model the problem as a sequential two-player game and propose an MCTS-based method to solve the problem. We evaluate the performance of our algorithm using simulations involving a case study of information gathering in a tunnel with a team of robots. In addition, we give an alternative and more complete proof of the bound of the sequential algorithm [19] which is of independent interest.

A preliminary version of this paper was presented in RSS 2020 [27]. Compared to our previous work which includes only the offline RMOP, we also consider the online RMOP in this paper and propose an MCTS-based strategy to solve the problem. New simulation results are also provided for the online RMOP.

The rest of the paper is organized as follows. We provide the necessary background on submodular functions and formally introduce the offline RMOP in Sec. II. Next, the approximation algorithm is proposed in Sec. III. The online RMOP is introduced in Sec. IV and the corresponding algorithm is explained in Sec. V. Detailed analysis of the proposed algorithm is given in Sec. VI. Finally, simulation results are given in Sec. VII and VIII.

## II. Problem Description

In this section, we formally describe the Robust Multiple Orienteering Problem. We start by introducing the notations and conventions used in the paper.

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

We now define two useful properties of set functions.

**Definition 1** (Normalized Monotonicity)**.** For a set $\mathcal{V}$, a function $f: 2^{\mathcal{V}} \mapsto \mathbb{R}$ is called as normalized, monotonically non-decreasing if and only if for any $\mathcal{A} \subseteq \mathcal{A}' \subseteq \mathcal{V}$, $f(\mathcal{A}) \leq f(\mathcal{A}')$ and $f(\mathcal{A}) = 0$ if and only if $A = \emptyset$.

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

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

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

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

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

$$f(\mathcal{S}') = g\left(\bigcup_{\mathcal{P}_i \in \mathcal{S}'} \mathcal{P}_i\right). \tag{1}$$

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

We are now ready to formally define our problem.

**Problem 1** (Offline RMOP)**.** *Given a metric graph $G(\mathcal{V},\mathcal{E})$, $N$ robots with starting positions $\{v_{s_1}, v_{s_2}, \dots, v_{s_N}\}$, budget constraint $B$, a robot reward function $g(\mathcal{P}) : \mathcal{T} \to \mathbb{R}_+$, and a team reward function $f$ as defined in Equation 1, the offline Robust Multiple-Path Orienteering Problem seeks to find a collection of $N$ paths, $\mathcal{S} = \{\mathcal{P}_1, \dots, \mathcal{P}_N\}$ that are robust to the worst-case failure of $\alpha$ robots:*

$$\begin{aligned} &\max_{\mathcal{S}\subseteq\mathcal{T}} \min_{\mathcal{A}\subseteq\mathcal{S}} f(\mathcal{S}\setminus\mathcal{A}) \\ \textit{s.t.}\quad & |\mathcal{A}| \leq \alpha, 0 < \alpha < N \\ & |\mathcal{S}| = N \\ & C(\mathcal{P}_j) \leq B. \end{aligned} \tag{2}$$

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

The offline RMOP is suited to model the scenarios where we need to plan paths for all the robots before they are deployed and they cannot communicate with the base station to transmit collected rewards or with each other to replan during execution. Therefore, once a robot fails during the task, the reward of the whole path of that robot will be lost, which corresponds to the set removal of paths, and the team cannot adapt to the failures of robots. One practical example for the offline RMOP is the naval mine countermeasure mission [28], in which a team of robots is deployed to detect undersea mine information. In such a case, reliable communication is usually not available and the robot may fail due to mines or other adversaries. Mathematically, the offline RMOP can be interpreted as two-stage perfect information sequential one-step game, where the first player (i.e., the team of robots) chooses a set $\mathcal{S}$, and the second player (i.e., the adversary), knowing $\mathcal{S}$, chooses a subset $\mathcal{A}$ to remove from $\mathcal{S}$. We seek worst-case guarantees — in practice, the adversary may not know the paths for each robot. By playing against this stronger adversary, we guarantee that the performance against a weaker one will be even better. We evaluate this empirically by considering attack models other than the worst one.

The adversarial model considered in this paper is the same as that in prior work on robust submodular optimization [22], [23], [25], [29]. However, the offline RMOP is even harder since even at the single robot level, the optimization problem we need to solve (i.e., OP) is NP-Hard. Nevertheless, we present a constant-factor approximation algorithm for this problem next.

## III. Algorithm for offline RMOP

In this section, we present the general algorithm to solve the offline RMOP (Algorithm 1). The algorithm uses a generic subroutine for solving OP. In the next section, we show examples of three subroutines that can be used and show how they affect the performance of the algorithm.

Our algorithm builds on those in [22], [29]. The key idea in these algorithms is to construct two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \cup \mathcal{S}_2$ is a feasible approximation solution to the corresponding problem and $f(s_1) \geq f(s_2), \forall s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2$. The main difference in our work lies in how these sets are computed. The problem in [22] considers only choosing elements from a given set, with a cardinality constraint. The algorithm in [22] finds $\mathcal{S}_1$ by sorting the elements and finds $\mathcal{S}_2$ by greedily adding elements with the largest marginal gain. A more general problem, with matroid constraints instead of a cardinality constraint, in considered in [29]. To find $\mathcal{S}_1$ from a given ground set $\mathcal{V}$, the algorithm loops over all elements in $\mathcal{V}$ and adds elements to $\mathcal{S}_1$ by considering the value of a single element ($f(y)$ in [29] line 3) and checking the matroid constraint (lines 4-5 in [29]). To find $\mathcal{S}_2$, the algorithm in [29] loops over $\mathcal{V} \setminus \mathcal{S}_1$ and adds elements to $\mathcal{S}_2$ incrementally by considering the marginal gain and matroid constraints (lines 10-12 in [29]). In contrast, in our problem the ground set $\mathcal{V}$ itself (the set of all paths) is not readily available. Enumerating this ground set is infeasible. Instead, we can use the set of all vertices in the graph as the ground set. However, then finding one path that maximizes the reward function is an NP-Hard problem. Thus, the simple selection step that can be solved just by looping over all elements (e.g., line 3 [29]) requires solving an NP-Hard problem. We first let all robots to solve OP individually with a bounded approximation algorithm (lines 2-5). Then, we find a candidate set for $\mathcal{S}_1$ by sorting (lines 8-9). Next, we use a Sequential Greedy Assignment (SGA) paradigm to construct a candidate set for $\mathcal{S}_2$. To guarantee $f(s_1) \geq f(s_2), \forall s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2$, we use the while loop to improve the candidate sets for $\mathcal{S}_1$ and $\mathcal{S}_2$.

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

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

As described earlier, there is a tradeoff between redundancy and coverage in RMOP. The two sets of paths are constructed so that $\mathcal{S}_1$ adds redundancy and $\mathcal{S}_2$ adds coverage, together yields a provably good solution for RMOP. We explain each step in Algorithm 1 next.

*Constructing $\mathcal{S}_1$:* Each of the $\alpha$ paths in $\mathcal{S}_1$ are better than those in $\mathcal{S}_2$. The paths in $\mathcal{S}_1$ may overlap with each other and also overlap with those in $\mathcal{S}_2$. Thus, these paths serve to add redundancy to the team. Constructing the best $\alpha$ paths with respect to $f$ itself is NP-hard. Therefore, Algorithm 1firstly solves the orienteering problem for each robot independently and stores in $\mathcal{M}$ the (approximately optimal) paths for individual robots (lines 2–5). Then Algorithm 1sorts the paths in $\mathcal{M}$ based on their collected rewards (line 8) and chooses the $\alpha$ best paths to be $\mathcal{S}_1$ (line 9).

**Algorithm 1:** Algorithm for Problem 1

```
Input  : Per problem 1 requires following inputs:
  • set of robots $\mathcal{R} = \{1, \ldots, N\}$
  • metric graph $G$
  • starting vertices $\{v_{s_1}, v_{s_2}, \ldots, v_{s_N}\}$
  • number of maximum potential attacks $\alpha$ and budget $B$
Output: Set $\mathcal{S}$ of paths for each robot
1  $\mathcal{S}_1 \leftarrow \emptyset, \mathcal{S}_2 \leftarrow \emptyset, \mathcal{M} \leftarrow \emptyset$
2  for $i \leftarrow 1$ to $N$ do
3      $\mathcal{P}_i \leftarrow OP(G, v_{s_i}, B)$
4      $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathcal{P}_i\}$
5  end
6  flag ← True
7  while flag do
8      Sort elements in $\mathcal{M}$ such that
         $\tilde{\mathcal{M}} = \{\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_N\}$ and
         $f(\{\mathcal{P}'_1\}) \geq f(\{\mathcal{P}'_2\}) \geq \ldots \geq f(\{\mathcal{P}'_N\})$
9      $\mathcal{S}_1 \leftarrow \{\mathcal{P}'_1, \mathcal{P}'_2, \ldots, \mathcal{P}'_\alpha\}$
10     //extract starting positions for the rest of robots
         $\tilde{v}_s \leftarrow \{v_{s_j} | \forall j \in \mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)\}$
11     //Sequential greedy assignment
         $\mathcal{S}_2 \leftarrow SGA(G, \tilde{v}_s, B)$
12     //while loop control
13     if $f(\mathcal{P}_i) \geq f(\mathcal{P}_j), \forall \mathcal{P}_i \in \mathcal{S}_1, \mathcal{P}_j \in \mathcal{S}_2$ then
14         $flag \leftarrow$ False
15     else
16         Find all robots $j \in \mathcal{R}(\mathcal{S}_2)$ such that
17         $\exists i \in \mathcal{R}(\mathcal{S}_1), f(\{\mathcal{P}_j \in \mathcal{S}_2\}) > f(\{\mathcal{P}_i \in \mathcal{S}_1\})$
18         Replace the path stored in $\mathcal{M}$ corresponding to
             robot $j$ with the better path found when
             constructing $\mathcal{S}_2$
19     end
20 end
21 $\mathcal{S} \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$
```

*Constructing $\mathcal{S}_2$:* After finding $\mathcal{S}_1$, Algorithm 1 needs to find the best paths for the rest of robots $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$. Unlike $\mathcal{S}_1$, here the algorithm explicitly considers overlap when finding the paths. Thus, $\mathcal{S}_2$ serves to add coverage to the solution.

However, selecting optimal paths for $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$ is a multiple-path orienteering problem and is also NP-hard. Therefore, Algorithm 1 approximates the solution by employing the sequential greedy algorithm (line 11). For completeness, we present the pseudocode for SGA in Algorithm 2.

Specifically, for robots in $\mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)$, Algorithm 2 finds a path using an approximation algorithm for OP (line 4). Then, Algorithm 2 sets the reward for the vertices visited by that robot to be zero (lines 6–8). This process repeats until we find a path for all robots. Here we implicitly assume that there is at least one path for each robot satisfying the budget constraints.

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

*Invariant:* Our analysis requires the paths in $\mathcal{S}_1$ and $\mathcal{S}_2$ to have the following property: $f(\{\mathcal{P}_i\}) \geq f(\{\mathcal{P}_j\}), \forall \mathcal{P}_i \in \mathcal{S}_1, \mathcal{P}_j \in \mathcal{S}_2$. This condition is trivially met if the single robot problem has to just choose the best amongst a fixed set of trajectories as in the prior work [22], [25]. However, when solving RMOP, we employ a subroutine for solving OP which gives us the paths in $\mathcal{S}_1$ and $\mathcal{S}_2$. Since the subroutine uses an approximation algorithm for OP instead of an exact optimal one, we cannot guarantee that this invariant holds. For example, if the subroutine uses randomness, then running the same algorithm twice may give different results. In any case, all we can guarantee is that the paths found by the subroutine will be no more than a constant from the optimal.

We fix this problem by utilizing a **while** loop (lines 7–20). When the condition of the **while** loop holds (lines 13–15), the loop flag is set to be false and the while loop terminates. Otherwise (lines 16–19), Algorithm 1 will find those robots that violate the above inequality and update their corresponding paths in the set $\mathcal{M}$. Recall that $\mathcal{M}$ is used to store the best path corresponding to each robot. Then, while loop will restart to construct $\mathcal{S}_1$ using the updated $\mathcal{M}$ and $\mathcal{S}_2$ for the remaining robots, again. We show that this loop will eventually terminate.

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

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

*Remark* 1. In practice, the loop in Algorithm 1 typically terminates after just one iteration. Paths in $\mathcal{S}_1$ are found without considering overlap. On the other hand, when solving SGA the robots find their paths by taking into account overlap with the previously found paths. The conditions in the latter are a subset of the former. Furthermore, none of the three subroutines that we employ for OP include any randomness. Therefore, it is unlikely that the paths in $\mathcal{S}_2$ will be better than that in $\mathcal{S}_1$. As such, it is less likely that the **while** loop will take more than one iteration. Nevertheless, we give the full algorithm for completeness.

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

**Algorithm 2:** Sequential Greedy Assignment

```
1  Function SGA($G, v_s, B$):
       Input :
         • A graph $G$ representing the environment
         • Budget $B$ for each robot
         • Starting positions $v_s$
       Output: a collection $\mathcal{A}$ of paths
2      $\mathcal{A} \leftarrow \emptyset, G' \leftarrow G, N \leftarrow length(v_s)$
3      for $j \leftarrow 1$ to $N$ do
4          $\mathcal{P}_j \leftarrow OP(G', v_{s_j}, B)$;
5          $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathcal{P}_j\}$
6          foreach $v \in \mathcal{P}_j$ do
7              Set the reward of $v \in G'$ to be zero
8          end
9      end
10     return $\mathcal{A}$
11 end
```

## IV. Online RMOP

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

**Definition 3** (Attacker behavior set)**.** An attack behavior set $\mathcal{A} = \{(t_1, r_1), \ldots, (t_\beta, r_\beta)\}$ is a set of tuples, each of which consists of two elements: the first element indicates when to attack and the second element indicates which robot to attack.

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

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

*that are robust to the worst-case failure of $\alpha$ robots:*

$$\begin{aligned}&\max_{\mathcal{S}\subseteq\mathcal{T}}\ \min_{\mathcal{A}=\{(t_j,r_j)\}}\ g\Big(\bigcup_{i=1}^{N}\mathcal{P}_i\setminus\bigcup_{j=1}^{|\mathcal{A}|}\mathcal{P}_{r_j}[t_j+1:end]\Big)\\ &s.t.\quad |\mathcal{A}|\le\alpha,0<\alpha<N\\ &\qquad\ \ |\mathcal{S}|=N\\ &\qquad\ \ C(\mathcal{P}_j)\le B,\end{aligned}\tag{3}$$

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

Intuitively, in Problem 2, we want to find paths for robots while $\alpha$ failures in total can happen at any nodes along the paths. If a robot fails at a particular node $\mathcal{P}(t)$, then it cannot collect reward after that node anymore. We model this problem as a discrete, sequential, two-player zero-sum game between the attackers and the robots. Since robots can communicate with each other, we aim to find one adaptive planning strategy that can adapt to the attacks.

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

*Proof.* One example is given in Fig. 2. ∎

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

## V. MCTS for online RMOP

MCTS is an approach for finding optimal actions by randomly selecting samples from search space and incrementally building the search tree [30] and is widely applied in robotics

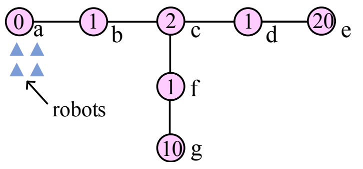


Fig. 2. One example to show that rational attackers will not always launch attacks at the first step. Four robots start from node $a$ and each has a budget of 4. There are $\alpha=2$ attacks in the environment. If attackers launch all attacks at the first step, the robots can collect 35 rewards in total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

applications, including scouting [31], active parameter estimation [32], environment monitoring [33], and multi-robot active perception [34]. As shown in Fig. 3, there are four basic steps in each iteration of an MCTS process: selection, expansion, simulation, and backpropagation [35].

In the paper, we model the Problem 2 as a sequential, discrete two-player game and we adopt the MCTS algorithm to solve the game in an online fashion. At each step, attackers use their strategies to take one action first, and then robots take one action. Intuitively, it means that attackers observe the states of robots to decide to attack or not, and then robots respond to that. To choose one action, the robots will incrementally grow the search tree with some computational budget and then select one action to take from the root of the search tree based on the average reward of each action. Such a process continues until all robots run all of the budgets.

When our algorithm grows the search tree, the state of each robot is a tuple $s=(v,I)$ where $v\in\mathcal{V}$ represent the current position of the robot and $I$ is an indicator on whether the robot has been attacked ($I=1$) or not ($I=0$). The joint state of the team is the product of the individual states of robots and is stored in each node of the search tree. Robots and attackers alternate turns in growing the tree. When it's robots' turn, they will decide the transition of the positions while the attackers can decide the value of the indicator state in attackers' turn. Once one indicator state is set to be one, which means a robot is attacked, it will remain to be one for the rest of the game and that robot cannot move anymore i.e, the only action that robot can take in the game is to stay there. Similarly, if a robot has run out of budget, the only action available is to stay at the current position. It should be noted that we present Algorithm 3 from the perspective of robots but attackers can also use similar strategies. In the following, we refer to two-player MCTS (if there are two alternating turns in the search) as MCTS with adversaries and one-player MCTS without considering opponents as naive MCTS. Details of the Algorithm 3 are given below.

1) *Selection* (Line 4 in Algorithm 3; Line 1-11 in Algorithm 4): Starting from the root node, a selection procedure is recursively applied until some leaf node is reached. In each recursion, a child node is selected based on Upper Confidence Bound for Trees (UCT) Kocsis and Szepesvári [36]. There are two parts to the UCT value: exploitation and exploration. The exploitation part corresponds to the average rollout reward obtained and the

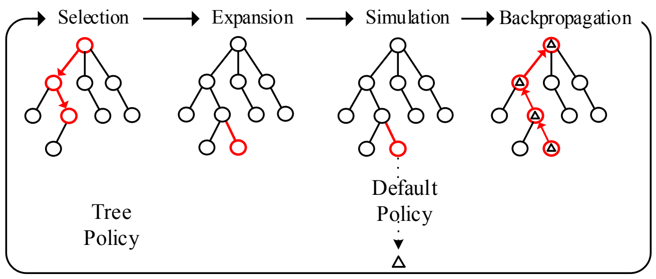


Fig. 3. One iteration of the general MCTS approach [30].

exploration part is decided by the number of times that the node has been visited ($n(v')$) and the number of times that the current node's parent has been visited ($n_p$). If a node is less visited, the exploration value will increase which encourages the selection of that node. It should be noted that if it's robots' turn robots will select the node with the highest UCT value (Line 6) while the attacker will select the node with the lowest UCT value if it's the attacker's turn (Line 8).

2) *Expansion* (Line 5 in Algorithm 3; Line 12-19 in Algorithm 4): One (or more) child nodes are added to the tree based on the available actions. If the node has reached the terminal level, e.g., run out of budget, the current node will be returned (Line 13-15). Otherwise, add all children of the current node to the tree and return the first child node (Line 15-18).
3) *Simulation* (Line 6 in Algorithm 3; Line 20-29 in Algorithm 4): A rollout is conducted from the chosen node using the default policy until some terminal condition is met. The obtained reward will be returned.
4) *Backpropagation* (Line 7 in Algorithm 3; Line 30-34 in Algorithm 4): The simulation result is propagated back to the root and update node information along the propagation path.

**Algorithm 3:** Monte Carlo Tree Search

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

**Algorithm 4:** Monte Carlo Tree Search Subroutines

```
1  Function Selection($tree$, $v$):
2      if $level(v)$ = $TERMINAL$ then
3          return $v$
4      end
5      if $turn(v)$ = $ROBOT$ then
           // $n_p$ is the number of times that the parent of
             $v$ has been visited
6          $$v \leftarrow \underset{v' \in \text{children}(v)}{\text{argmax}} \frac{Q(v')}{n(v')} + c\sqrt{\frac{2\ln n_p}{n(v')}}$$
7      else
8          $$v \leftarrow \underset{v' \in \text{children}(v)}{\text{argmin}} \frac{Q(v')}{n(v')} - c\sqrt{\frac{2\ln n_p}{n(v')}}$$
9      end
10     return Selection($tree$, $v$)
11 end

12 Function Expansion($tree$, $v$):
13     if $level(v)$ = $TERMINAL$ then
14         return $v$
15     else
16         Add all child nodes of $v$ to $Tree$
17         return the first child node
18     end
19 end

20 Function Simulation($tree$, $v$):
21     while $level(v) \neq TERMINAL$ do
22         if $turn(v)$=$ROBOT$ then
23             $v \leftarrow$ RobotDefaultPolicy($v$)
24         else
25             $v \leftarrow$ AttackerDefaultPolicy($v$)
26         end
27     end
28     return CollectReward
29 end

30 Function Backpropagation($tree$, $Reward$, $v$):
31     while $v \neq NULL$ do
           // update total reward value
             $tree.v.Q \leftarrow tree.v.Q + Reward$
32         $tree.v.n \leftarrow tree.v.n + 1$
33     end
34 end
```

## VI. Performance Analysis

In this section, we quantify the performance of the proposed Algorithm 1. We first present a new analysis for the Sequential Greedy Assignment (SGA) and then show the performance bound for our algorithm. The performance is based on the notion of curvature of the set functions.

**Definition 4** (Total Curvature)**.** Consider a finite ground set $\mathcal{V}$ and a monotone submodular set function $h : 2^{\mathcal{V}} \mapsto \mathbb{R}$. The curvature of $h$ is defined as

$$k_h = 1 - \min_{v \in \mathcal{V}^\dagger} \frac{h(\mathcal{V}) - h(\mathcal{V} \setminus v)}{h(v)}, \tag{4}$$

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

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

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

## A. Sequential Greedy Assignment

SGA was first proposed in [19] to solve the MOP. Note that the MOP is the same as RMOP if we consider $\alpha = 0$. SGA solves the problem by finding the path for the $i^{th}$ robot in the $i^{th}$ iteration, by considering the paths found in the previous $i - 1$ iterations. As part of our analysis, we also find a more general approximation bound for SGA, given in Theorem 1, generalizing the one in [19] using the curvature of the submodular function.

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

**Theorem 1.** *Algorithm 2 (SGA) gives a $k_f + \eta$ approximation for MOP, where $\eta$ is the approximation factor for OP and $k_f \in [0, 1]$ is the total curvature for the reward function $f(\cdot)$.*

The proof of Theorem 1 is given in the appendix. Note that this bound $k_f + \eta$ generalizes the $1 + \eta$ bound in [19]. We now use this result to prove our main result.

## B. Analysis for Algorithm 1

**Theorem 2.** *Algorithm 1 returns a set $\mathcal{S}$ such that*

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

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

The omitted proofs can be found in the appendix. Our proof builds on the ones in [22], [29]. The main difference is that the proof in [22], [29] is suitable for picking a subset of elements directly from a given ground set, while our proof (and algorithm) deals with finding set of paths, each of which consists of several elements (vertices). All the proofs relies on the property that for any $s_1 \in \mathcal{S}_1$ (*bait set*), $s_2 \in \mathcal{S}_2$ (*reward set*), the inequality $f(s_1) \geq f(s_2)$ is always satisfied. Besides, we need to consider the fact that the single robot submodular OP, which is NP-hard, can only be solved approximately whereas this step is trivial in the earlier work.

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

**Corollary 2.** *If recursive greedy algorithm [18] is used as a subroutine to solve OP and additionally each robot has a predefined terminal vertex, then $\eta$ in Theorem 2 equals to $\log(OPT)$. Here $OPT$ is the reward collected by the optimal algorithm. The running time of the resulting algorithm is quasi-polynomial since the running time of recursive greedy is quasi-polynomial.*

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

**Corollary 3.** *If General Cost-Benefit (GCB) approximation algorithm [37] is used as subroutine for OP and we are allowed to relax given budget to $\frac{\psi(n)K_c}{\beta(1+\beta(K_c-1)(1-k_c))}B$, then $\eta$ in Theorem 2 equals to $2(1-e^{-1})^{-1}$. Here, $\psi(n), \beta, K_c, k_c$ as defined in [37]. The GCB algorithm runs in polynomial time.*

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

**Corollary 4.** *If the reward function $g$ is modular, then using the approximation algorithm for OP [5] yields an $\eta = 4$ in Theorem 2. The running time of the algorithm [5] is polynomial.*

## C. Running Time

MCTS is an anytime algorithm and can converge to the optimal solution as computational time increases. In applications, the running time is decided by the available computational budget. Next, we will mainly focus on the running time analysis of the proposed Algorithm 1.

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

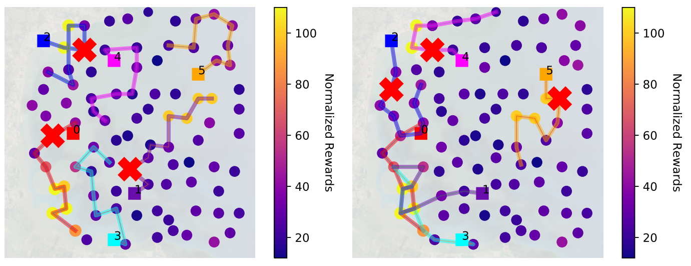
$$n_w \leq \sum_{i=1}^{N} n_i \leq n_p N.$$

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

## VII. Numerical Simulations for RMOP without Communication

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

### A. Simulation Setup



(a) SGA paths (b) RMOP paths

Fig. 4. Case study of monitoring macroalgal blooms using $N = 6$ robots assuming $\alpha = 3$ failures. Colored dots indicate locations to be monitored along with their importance (i.e., rewards). Red crosses indicate worst-case attacks found using brute force. Paths returned by the proposed algorithm manages to cover one of the three important areas (lower left corner) while SGA loses all three. The background map is part of the Yellow Sea, where green tides prevail every summer since 2007.

*Application case study:* We use the application of monitoring a marine environment for mapping oil leaks, macroalgal blooms, or pH values. Specifically, as explained in [38], such tasks usually calls for collaboration of multiple sensors including satellites, which can provide coarse prior information on the concentration of the phenomenon of interest, and mobile robotic sensors, which can use the prior information for targeted data collection. Using this as motivation, we consider a scenario where prior information from satellites (for example) can be used to define an importance map over the environment to be monitoring. Fig. 4 shows the setup which consists of 96 vertices placed in the environment. The color of the vertex reflects the importance of that vertex which gives the reward associated with visiting that vertex. Here, the single robot function, $g(\mathcal{P}_i)$, is a modular reward. The cost along the edges is the Euclidean distance between the vertices. Assuming unit speed of travel, the cost of a path reflects the travel time of the robot. In all the instances, each robot is given a budget $B = 60$ units.

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

For both SGA and the proposed algorithm, we use the GCB algorithm solving OP due to its efficiency and ease of implementation. Specifically, we implement GCB using details provided in [37]. When running GCB, we simply set the relaxed budget itself to be $B$.

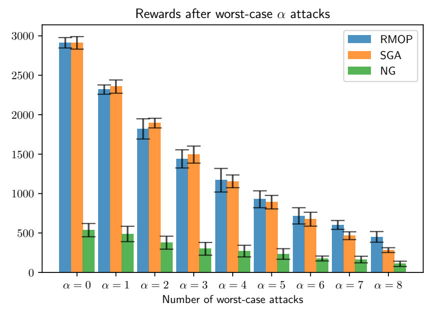
*Attack models:* Our algorithm is designed to give performance guarantees against worst-case attacks. However, in practice, we would like for any algorithm to be robust to not just the worst-case attacks but also other attacks. Therefore, we evaluate two other types of attacks besides worst-case attacks.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

### B. Results

Fig. 4 shows a qualitative example comparing our proposed algorithm and SGA with $N = 6$ and $\alpha = 3$. Not surprisingly, the six paths found by SGA do not have any overlap but the ones found by our algorithm do. As a result, the worst-case attack takes away all three robots covering the important regions in SGA, whereas one of the three regions is still covered with our algorithm. The worst-case attacks were computed using brute force.

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

Fig. 5a shows the comparison between our algorithm for RMOP with SGA and NG as $\alpha$ increases with $N = 10$. The bars show the rewards collected by the robots after attacks. Our algorithm returns paths that are slightly worse than SGA when $\alpha$ is small. This is not surprising since our algorithm will have overlapping paths whereas SGA will not. NG is the other extreme since each robot plans for itself which can lead to a high degree of overlap. Though our algorithm has only comparable performance to SGA when $\alpha$ is relatively small, the reward gap is small. As $\alpha$ increases, our algorithm gradually significantly outperforms SGA. For example, when $\alpha = 8$ our algorithm yields a reward of 451 whereas SGA only yields 283 on average. As a result, in general, the overall performance of the Algorithm 1 can be trusted especially for the large $\alpha$. Moreover, in practice, since we cannot decide the


(a)

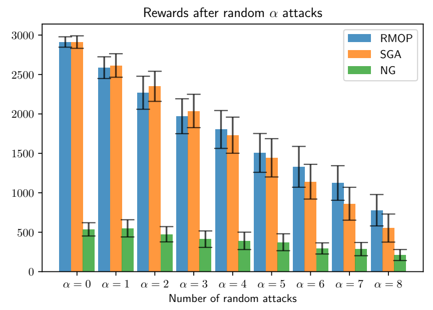
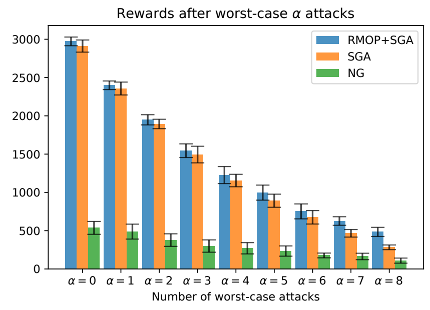

(b)

Fig. 5. Results for Algorithm 1 and the baseline algorithms. (a) Rewards after worst-case attack with increasing $\alpha$ and $N = 10$. (b) Rewards after random $\alpha$ attacks and $N = 10$.


(a)

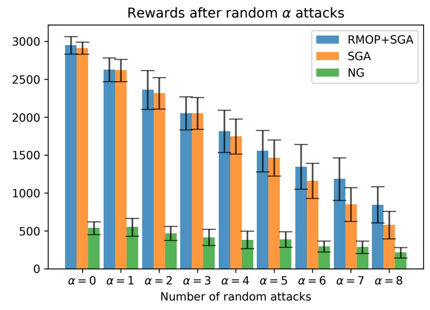

(b)

Fig. 6. Results for the mixed strategy and the baseline algorithms. (a) Rewards after worst-case attack with increasing $\alpha$ and $N = 10$. (b) Rewards after random $\alpha$ attacks and $N = 10$.

threshold above which Algorithm 1 significantly outperforms SGA, we can run SGA and Algorithm 1 in parallel for offline planning if the planning budget permits and select the one that returns higher rewards.

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

We conducted an experiment to compare SGA and the mixed SGA+Algorithm 1. The result is shown in Fig. 6. As shown in Fig. 6a, the mixed strategy collects more reward on average as compared to the SGA, across all $\alpha$ when the worst-case attack happens. We observe the same trend for the random attack case shown in Fig. 6b.

Next, we evaluate the performance of our algorithm when the attack model does not match the worst-case one assumed during planning. The goal is to verify the robustness of the algorithm to other attack models. Fig. 5b shows the comparison between our algorithm and the two baselines as $\alpha$ varies when the attacked robots are randomly chosen. Our algorithm still plans to assume worst-case attacks. We observe the same trend with random attacks as with the worst-case ones — as $\alpha$ increases, our algorithm outperforms SGA.

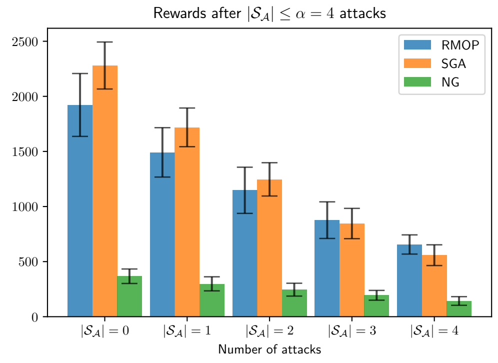
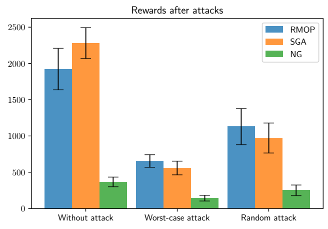


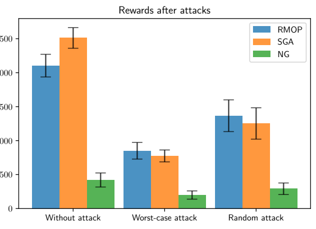
Fig. 7. Rewards after worst-case attacks of increasing sizes, $|\mathcal{S}_{\mathcal{A}}| \leq \alpha$. Here the planner uses $N = 7$ and $\alpha = 4$.

Fig. 7 shows results for the case where we construct paths assuming $\alpha = 4$ robots will be attacked but in practice only $|\mathcal{S}_{\mathcal{A}}| \leq \alpha$ robots suffer from worst-case attacks. SGA performs better than our algorithm when the number of robots actually attacked $|\mathcal{S}_{\mathcal{A}}|$ is far from the designed value of $\alpha$. As the actual number of robots attacked increases and $|\mathcal{S}_{\mathcal{A}}|$ approaches $\alpha$, our algorithm outperforms SGA. This suggests that we need to have an accurate estimate of the $\alpha$ before applying our algorithm, which is one limitation of this paper. However, in many robotic applications, it's possible to estimate $\alpha$ using historical data. Take the information gathering in the marine environment for example. We can use the number of robots that survived in the previous executions of the mission to estimate $\alpha$.

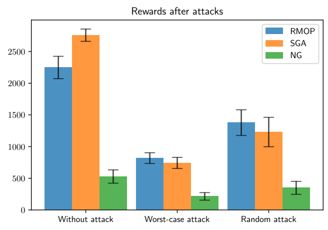
Fig. 8 shows the evaluation when there are (1) no attacks; (2) worst-case attacks; and (3) random attacks for four configurations of $N$ and $\alpha$. In all three cases, our algorithm still



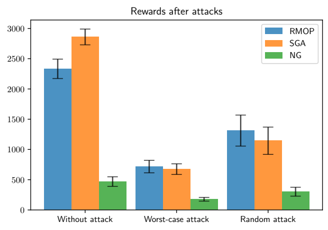
(a) 7 robots with $\alpha = 4$



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



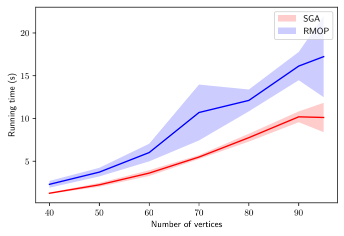
(c) 9 robots with $\alpha = 5$



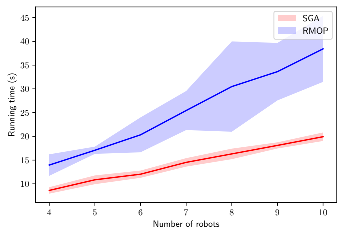
(d) 10 robots with $\alpha = 6$

Fig. 8. Rewards after: (1) no attacks; (2) $\alpha$ out of $N$ robots under worst-case attack; (3) $\alpha$ out of $N$ robots under random attack.

plans the paths assuming worst-case attacks for the given value of $\alpha$. When there are no attacks (first set of bars in each subfigure), SGA outperforms our algorithm as observed in previous charts. When worst-case attacks do happen (middle set of bars), the average rewards collected by the unattacked robots employing our algorithm are better than that of SGA. This is also the case when the $\alpha$ attacked robots are chosen randomly (third set of bars). This trend holds for various values of $N$ and $\alpha$ as shown.



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



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

Fig. 9. Running time of Algorithm 1 and SGA

The running time comparisons between our algorithm and SGA are shown in Fig. 9. We vary the number of robots as well as the size of the graph. Our algorithm takes longer than SGA which is expected since it uses SGA as a subroutine. Nevertheless, we observe similar trends in the runtime.

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

## VIII. Numerical Simulations for RMOP with Communication

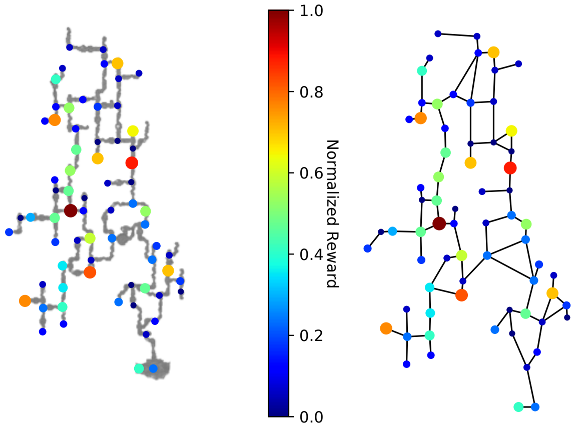
In this section, we validate the performance of Algorithm 3 for the online RMOP. In particular, we present a case study on information gathering in a tunnel and compare the performance of the team when robots and attackers adopt different strategies.

### A. Simulation Setup

*Application case study:* We use the application of information gathering in a tunnel in which we assume that some coarse prior information on the rewards of some locations are known and a team of robots is sent to gather detailed information but at most $\alpha$ attackers may attack them at any locations. We also assume that communication, though maybe degraded, is available. We use a tunnel map from Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge and use image skeletonization algorithm and corner detection algorithm [39] to identify several points as locations of interest. Fig. 10a shows the setup which consists of 69 vertices placed in the environment. The color and the size of the node reflect the importance of that vertex which gives the reward associated with visiting that vertex. The corresponding graph abstraction is shown in Fig. 10b in which we assume that it takes one unit of time to transit between two adjacent nodes. Even though adding some virtual nodes on the long edges will make this assumption better justified, for simplicity, we ignore these virtual nodes. Here, the single robot function, $g(\mathcal{P}_i)$, is a modular reward. The cost along the edge is the distance between two adjacent nodes which is defined in the image coordinate as the shortest distance in the skeleton image ($739 \times 520$ pixels). In this case study, there are four robots in the environment and $\alpha = 2$ attackers and each robot has a budget of $B = 500$ units.

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

*Strategies:* We consider two strategies for robots including MCTS with adversaries (Algorithm 3, two-player search, we refer it as MA) and naive MCTS (don't consider failures/attacks, one-player search, we refer it as M) and two strategies for attackers including MCTS with adversaries (Similar to Algorithm 3, two-player search, we refer it as MA, and RandomMove (randomly choose an available action each time, we refer it as R). Problem 2 is simulated as a two-player game in which at each step attackers first use their strategy to



(a) Tunnel map (b) Tunnel graph

Fig. 10. A tunnel map for a case study of information gathering. Colored dots with various sizes indicate locations to be visited along with their importance i.e., rewards. (a) A tunnel map with 69 locations of interest. (b) The graph abstraction of the tunnel map. The tunnel map is from Defense Advanced Research Projects Agency (DARPA) Subterranean Challenge.

choose one available action and then robots choose one action. Such a process continues until all robots run out of budget.

### B. Results

Fig. 10 shows the tunnel map from DARPA subterranean challenge. Fig. 10a is the original tunnel map with 69 locations of interest and Fig. 10b is the corresponding abstract graph representation of the tunnel map. Fig. 11 shows a qualitative example of how robots and attackers behave in a two-player sequential game when both of them use MCTS with adversaries. In each step, attackers will first grow the search tree based on what they observed so far and choose an action. Then, robots will grow the search tree and choose one action. Such a process continues until the budgets of robots are used up. As a result, robot 0 follows the path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 34 \rightarrow 31 \rightarrow 62 \rightarrow 12 \rightarrow 21 \rightarrow 10 \rightarrow 44 \rightarrow 49$; robot 1 follows a path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 51 \rightarrow 1 \rightarrow 40 \rightarrow 25$ and is attacked at node 25; robot 2 follows a path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 25$ and is attacked at node 25; robot 3 follows a path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 34 \rightarrow 31 \rightarrow 62 \rightarrow 12 \rightarrow 21 \rightarrow 10 \rightarrow 5 \rightarrow 68$. As shown in Fig. 11, attackers launch the first attack when they observe that robots 0 and 3 reach node 34 and robot 2 reaches node 25 because if they don't attack at that moment robot 2 will move downward to collect more rewards. Attackers launch another attack later when robot 2 reaches node 25 and may collect more rewards from the nodes below.

Fig. 12 shows the results when robots and attackers use different strategies in four graphs. Robots are randomly initialized in different vertices and for each initialization, the two-player game is conducted 20 times. The collected reward of the team is the sum of the rewards of nodes visited. As shown in Fig. 12, when attackers use MCTS with adversaries (first two bars blue and orange), robots can collect more reward on average if they also use the MCTS with adversaries (orange) compared to the case where they use a naive MCTS without considering attacks. If attackers use a random strategy (last two bars green and red), the MCTS with adversaries strategy (red) is also on average better than a naive MCTS without considering attacks (green). Moreover, robots can collect more rewards on average if attackers just select an action randomly (green and red compared to blue and orange).

We empirically evaluate how the number of iterations influences the performance of MCTS. As shown in Fig. 13, as the number of iterations increase, the average reward also improves. However, the rate of improvement shows diminishing returns. When the number of iterations increase from 500 to 10000, the average collected reward increases about 18% (from 229 to 271), suggesting that too many iterations may not be necessary in practice.

## IX. Conclusion

We introduced a new problem, termed Robust Multiple-Path Orienteering Problem, in which we seek to construct a set of paths for robots such that even if a subset of robots fails, the rest of the team still performs well. We consider two types of RMOP. In the offline RMOP in which robots cannot communicate with each other or the base station during the execution of tasks, we provided a general approximation framework for the offline RMOP, which builds on bounded approximation algorithms for OP and the sequential greedy assignment framework. We showed three variants of the general algorithm that use three different subroutines for OP and still yield a bounded approximation for RMOP. In addition to theoretical results, we presented empirical results showing that our algorithm is robust to attacks other than the worst-case ones. We also compare our performance with baseline algorithms and show that our algorithm yields better performance as more and more robots are attacked. In the online version, RMOP is modeled as a two-player sequential game and solved adaptively in a receding horizon fashion based on Monte Carlo Tree Search (MCTS). Simulation results show that MCTS with adversaries performs better on average than the MCTS without considering attacks/failures.

## References

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

[2] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5622–5629.

[3] D. Thakur, M. Likhachev, J. Keller, V. Kumar, V. Dobrokhodov, K. Jones, J. Wurz, and I. Kaminer, "Planning for opportunistic surveillance with multiple robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 5750–5757.

[4] A. Sadeghi, A. B. Asghar, and S. L. Smith, "On minimum time multi-robot planning with guarantees on the total collected reward," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2019, pp. 16–22.

[5] A. Blum, S. Chawla, D. R. Karger, T. Lane, A. Meyerson, and M. Minkoff, "Approximation algorithms for orienteering and discounted-reward tsp," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, Oct 2003, pp. 46–55.

[6] E. Sless, N. Agmon, and S. Kraus, "Multi-robot adversarial patrolling: facing coordinated attacks," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1093–1100.

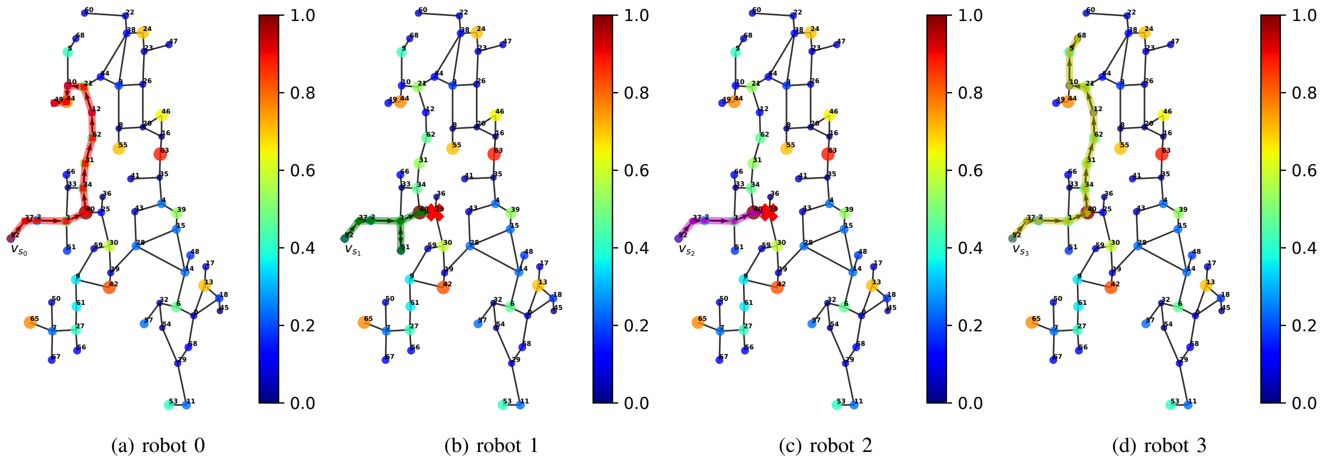


Fig. 11. A case study of information gathering in a tunnel with $N = 4$ robots assuming $\alpha = 2$ failures/attacks. Colored dots indicate locations to be visited along with their importance (i.e., rewards). Red across indicates the attacks found when robots and attackers play the two-player sequential game and both use MCTS with adversaries. $v_{s_i}$ represents the starting position of the robot $i$. Attackers launch the first attack when they observe that robot 0 and 3 reach node 34 and robot 2 reaches node 25 at the fifth step and launch another attack later when robot 1 reaches node 25 at the seventh step. (a) robot 0 follows the path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 34 \rightarrow 31 \rightarrow 62 \rightarrow 12 \rightarrow 21 \rightarrow 10 \rightarrow 44 \rightarrow 49$. (b) robot 1 follows the path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 51 \rightarrow 1 \rightarrow 40 \rightarrow 25$ and is attacked after two steps. (c) robot 2 follows the path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 25$ and is attacked after three steps. (d) robot 3 follows the path $52 \rightarrow 37 \rightarrow 2 \rightarrow 1 \rightarrow 40 \rightarrow 34 \rightarrow 31 \rightarrow 62 \rightarrow 12 \rightarrow 21 \rightarrow 10 \rightarrow 5 \rightarrow 68$.

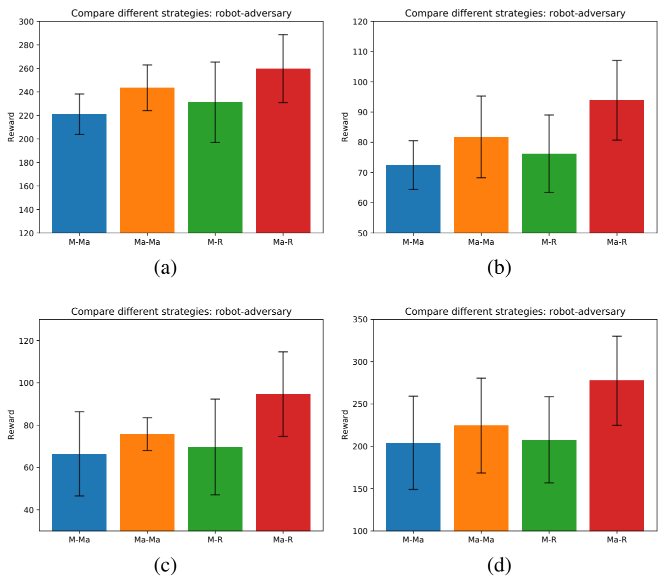


Fig. 12. Collected rewards for different strategies in four different maps: (1) M-MA: robots use one-player MCTS (M) and attackers use MCTS with adversaries (MA); (2) MA-MA: both robots and attackers use MCTS with adversaries; (3) M-R: robots use one-player MCTS (M) and attackers attacks randomly (R); (4) Ma-R: robots use MCTS with adversaries and attackers attacks randomly (R).

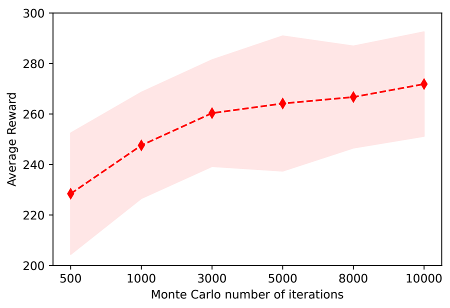


Fig. 13. The effect of increasing the number of iterations in MCTS with $N = 4$ robots and $\alpha = 2$ failures/attacks.

[7] J. Carlson, R. R. Murphy, and A. Nelson, "Follow-up analysis of mobile robot failures," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5. IEEE, 2004, pp. 4987–4994.

[8] A. Prorok, B. M. Sadler, M. Egerstedt, and V. Kumar, "Guest editorial: Special section on "foundations of resilience for networked robotic systems"," *Autonomous Robots*, vol. 43, no. 3, pp. 741–741, 2019.

[9] L. Guerrero-Bonilla, A. Prorok, and V. Kumar, "Formations for resilient robot teams," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 841–848, 2017.

[10] A. Prorok, "Redundant robot assignment on graphs with uncertain edge costs," in *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 313–327.

[11] K. Saulnier, D. Saldana, A. Prorok, G. J. Pappas, and V. Kumar, "Resilient flocking for mobile robot teams," *IEEE Robotics and Automation letters*, vol. 2, no. 2, pp. 1039–1046, 2017.

[12] M. W. Spong, "On the robust control of robot manipulators," *IEEE Transactions on automatic control*, vol. 37, no. 11, pp. 1782–1786, 1992.

[13] Y. Ting, S. Tosunoglu, and B. Fernandez, "Control algorithms for fault-tolerant robots," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 910–915.

[14] D. Crestani and K. Godary-Dejean, "Fault tolerance in control architectures for mobile robots: Fantasy or reality?" in *CAR: Control Architectures of Robots*, 2012.

[15] A. Clark, B. Alomair, L. Bushnell, and R. Poovendran, *Submodularity in dynamics and control of networked systems*. Springer, 2016.

[16] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, no. Feb, pp. 235–284, 2008.

[17] A. Krause and C. Guestrin, "Submodularity and its applications in optimized information gathering," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 4, pp. 1–20, 2011.

[18] Chandra Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, Oct 2005, pp. 245–253.

[19] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, vol. 34, pp. 707–755, 2009.

[20] N. Atanasov, J. Le Ny, K. Daniilidis, and G. J. Pappas, "Decentralized

active information acquisition: Theory and application to multi-robot slam," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4775–4782.

[21] J. B. Orlin, A. S. Schulz, and R. Udwani, "Robust monotone submodular function maximization," in *Proceedings of the 18th International Conference on Integer Programming and Combinatorial Optimization-Volume 9682*. Springer-Verlag, 2016, pp. 312–324.

[22] V. Tzoumas, K. Gatsis, A. Jadbabaie, and G. J. Pappas, "Resilient monotone submodular function maximization," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 1362–1367, arXiv version.

[23] V. Tzoumas, A. Jadbabaie, and G. J. Pappas, "Resilient monotone sequential maximization," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 7261–7268.

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

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

[26] ——, "Distributed attack-robust submodular maximization for multi-robot planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, to appear.

[27] G. Shi, L. Zhou, and P. Tokekar, "Robust multiple-path orienteering problem: Securing against adversarial attacks," in *Robotics: Science and Systems (RSS)*, 2020.

[28] S. Sariel, T. Balch, and N. Erdogan, "Naval mine countermeasure missions," *IEEE Robotics & Automation Magazine*, vol. 15, no. 1, pp. 45–52, 2008.

[29] V. Tzoumas, A. Jadbabaie, and G. J. Pappas, "Resilient non-submodular maximization over matroid constraints," *arXiv preprint arXiv:1804.01013*, 2018.

[30] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[31] Z. Zhang, J. M. Smereka, J. Lee, L. Zhou, Y. Sung, and P. Tokekar, "Game tree search for minimizing detectability and maximizing visibility," *Autonomous Robots*, pp. 1–15, 2021.

[32] P. Slade, P. Culbertson, Z. Sunberg, and M. Kochenderfer, "Simultaneous active parameter estimation and control using sampling-based bayesian reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 804–810.

[33] R. Marchant, F. Ramos, S. Sanner *et al.*, "Sequential bayesian optimisation for spatial-temporal monitoring." in *UAI*, 2014, pp. 553–562.

[34] G. Best, O. M. Cliff, T. Patten, R. R. Mettu, and R. Fitch, "Dec-mcts: Decentralized planning for multi-robot active perception," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 316–337, 2019.

[35] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[36] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[37] H. Zhang and Y. Vorobeychik, "Submodular optimization with routing constraints," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[38] Q. Xing, D. An, X. Zheng, Z. Wei, X. Wang, L. Li, L. Tian, and J. Chen, "Monitoring seaweed aquaculture in the yellow sea with multiple sensors for managing the disaster of macroalgal blooms," *Remote Sensing of Environment*, vol. 231, p. 111279, 2019.

[39] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.

## Appendix

## Supplementary Materials

### A. Proof of Theorem 1

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

$$k_0 = 1 - \min_{i:s_i \in \mathcal{S}^\dagger} \frac{f_{\mathcal{S}_{i-1}\cup\Omega}(s_i)}{f_{\mathcal{S}_{i-1}}(s_i)}, \tag{5}$$

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

Recall that the definition of the total curvature is

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

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

**Lemma 1.**

$$k_0 \le k_f. \tag{6}$$

*Proof.*

$$\begin{aligned}
k_0 &= \max_{i:s_i\in\mathcal{S}^\dagger} \frac{f_{\mathcal{S}_{i-1}}(s_i) - f_{\mathcal{S}_{i-1}\cup\Omega}(s_i)}{f_{\mathcal{S}_{i-1}}(s_i)} && (7)\\
&= \max_{i:s_i\in\mathcal{S}^\dagger} 1 - \frac{f_{\mathcal{S}_{i-1}\cup\Omega}(s_i)}{f_{\mathcal{S}_{i-1}}(s_i)} && (8)\\
&\le \max_{i:s_i\in\mathcal{S}^\dagger} 1 - \frac{f_{\mathcal{G}\setminus\{s_i\}}(s_i)}{f(s_i)} && (9)\\
&\le \max_{i:s_i\in\mathcal{G}^\dagger} 1 - \frac{f_{\mathcal{G}\setminus\{s_i\}}(s_i)}{f(s_i)} = k_f. && (10)
\end{aligned}$$

Eq. 9 follows Eq. 8 due to the monotonicity of the submodular function. Eq. 10 follows Eq. 9 since $\mathcal{S}^\dagger \subseteq \mathcal{G}^\dagger$. ■

**Lemma 2.**

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

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

**Lemma 3.**

$$\begin{aligned}
f(\Omega) \le k_0 \sum_{i:s_i\in\mathcal{S}\setminus\Omega} f_{\mathcal{S}_{i-1}}(s_i) &+ \sum_{i:s_i\in\Omega\cap\mathcal{S}} f_{\mathcal{S}_{i-1}}(s_i)\\
&+ \sum_{i:\omega_i\in\Omega\setminus\mathcal{S}} f_{\mathcal{S}}(\omega).
\end{aligned} \tag{12}$$

*Proof.* By definition and submodularity,

$$\begin{aligned}
f(\Omega \cup \mathcal{S}) &= f(\mathcal{S}) + f_{\mathcal{S}}(\Omega \setminus \mathcal{S}) && (13)\\
&\le f(\mathcal{S}) + \sum_{\omega\in\Omega\setminus\mathcal{S}} f_{\mathcal{S}}(\omega). && (14)
\end{aligned}$$

By Lemma 2 and the definition of $k_0$,

$$\begin{aligned}
f(\Omega \cup \mathcal{S}) &= f(\Omega) + \sum_{i:s_i\in\mathcal{S}\setminus\Omega} f_{\Omega\cup\mathcal{S}_{i-1}}(s_i) && (15)\\
&\ge f(\Omega) + (1-k_0) \sum_{i:s_i\in\mathcal{S}\setminus\Omega} f_{\mathcal{S}_{i-1}}(s_i). && (16)
\end{aligned}$$

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

Combining Eq. (14) and Eq. (16), we have

$$f(\Omega) \le k_0 \sum_{i:s_i\in\mathcal{S}\setminus\Omega} f_{\mathcal{S}_{i-1}}(s_i) \quad + f(\mathcal{S}) - \sum_{i:s_i\in\mathcal{S}\setminus\Omega} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{\omega\in\Omega\setminus\mathcal{S}} f_{\mathcal{S}}(\omega) \tag{17}$$

$$= k_0 \sum_{i:s_i\in\mathcal{S}\setminus\Omega} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{i:s_i\in\mathcal{S}\cap\Omega} f_{\mathcal{S}_{i-1}}(s_i) + \sum_{\omega\in\Omega\setminus\mathcal{S}} f_{\mathcal{S}}(\omega). \tag{18}$$

■

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

Let $\sigma_{\mathcal{S}}(s_i)$ denote the index of the element $s_i$ in the ordered set $\mathcal{S}$. For example, $\sigma_{\mathcal{A}}(\mathcal{P}_i) = i$. In the following proof, we will treat $\mathcal{O}$ as an ordered set (different orders of $\mathcal{O}$ will not change $f(\mathcal{O})$) and order the elements in $\mathcal{O}$ in such a way:

$$\text{if } \mathcal{O}_i^* \in \mathcal{A}\cap\mathcal{O}, \quad \sigma_{\mathcal{O}}(\mathcal{O}_i^*) = \sigma_A(\mathcal{O}_i^*), \tag{19}$$

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

$$\{i \mid o_i \in \mathcal{O}\cap\mathcal{A}\} = \{i \mid a_i \in \mathcal{A}\cap\mathcal{O}\}, \tag{20}$$

and

$$\begin{aligned}\{i \mid o_i \in \mathcal{O}\setminus\mathcal{A}\} &= \{1,2,\dots,N\} - \{i \mid o_i \in \mathcal{O}\cap\mathcal{A}\}\\ &= \{1,2,\dots,N\} - \{i \mid a_i \in \mathcal{A}\cap\mathcal{O}\}\\ &= \{i \mid a_i \in \mathcal{A}\setminus\mathcal{O}\}.\end{aligned} \tag{21}$$

Next we will start our proof of Theorem 1.

*Proof.* By Lemma 3,

$$f(\mathcal{O}) \le k_0 \sum_{i:a_i\in\mathcal{A}\setminus\mathcal{O}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:a_i\in\mathcal{O}\cap\mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:o_i\in\mathcal{O}\setminus\mathcal{A}} f_{\mathcal{A}}(o_i). \tag{22}$$

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

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

For the second and third term,

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

$$\le \sum_{i:a_i\in\mathcal{O}\cap\mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{o_i\in\mathcal{O}\setminus\mathcal{A}} f_{\mathcal{A}_{i-1}}(o_i) \tag{24}$$

$$\le \sum_{i:a_i\in\mathcal{O}\cap\mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:o_i\in\mathcal{O}\setminus\mathcal{A}} f_{\mathcal{A}_{i-1}}(\mathcal{P}_i^{*\text{local}}) \tag{25}$$

$$\le \sum_{i:a_i\in\mathcal{O}\cap\mathcal{A}} f_{\mathcal{A}_{i-1}}(a_i) + \sum_{i:a_i\in\mathcal{A}\setminus\mathcal{O}} \eta f_{\mathcal{A}_{i-1}}(a_i) \tag{26}$$

$$= \sum_{i:a_i\in\mathcal{O}\cap\mathcal{A}} \eta f_{\mathcal{A}_{i-1}}(a_i) + \sum_{a_i\in\mathcal{A}\setminus\mathcal{O}} \eta f_{\mathcal{A}_{i-1}}(a_i) \tag{27}$$

$$= \eta f(A), \tag{28}$$

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

Eq. (24) follows from Eq. (23) due to the submodularity. Eq. (25) follows from Eq. (24) due to the definition of $\mathcal{P}_i^{*\text{local}}$:

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

Eq. (26) follows from Eq. (25) based on Eq. (21) and the property of the approximation algorithm. Eq. (27) transits to Eq. (28) by definition.

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

$$f(\mathcal{O}) \le k_0 f(\mathcal{A}) + \eta f(\mathcal{A}) \tag{29}$$

$$\le (k_f + \eta) f(\mathcal{A}). \tag{30}$$

■

## *B. Proof of Theorem 2*

*Proof.* Our proof relies on the following three inequalities:

$$f(\mathcal{S}\setminus\mathcal{A}^*(S)) \ge (1 - k_f) f(\mathcal{S}_2) \tag{31}$$

$$f(\mathcal{S}\setminus\mathcal{A}^*(S)) \ge \frac{1}{\alpha+1} f(\mathcal{S}_2) \tag{32}$$

$$f(\mathcal{S}\setminus\mathcal{A}^*(S)) \ge \frac{1}{N-\alpha} f(\mathcal{S}_2) \tag{33}$$

The proof for inequality (31) is similar to the proof (Eq. 16-20) in [22] when combined with the invariant maintained by the **while** loop in Algorithm 1. Likewise, the proof for inequality (32) resembles that given in [22] (from Eq. 21 to Eq. 25) and inequality (33) can be proved in the same fashion as that in [29] (Eq. 57 to Eq. 58).

From Theorem 1 we have,

$$f(\mathcal{S}_2) \ge \frac{1}{k_f + \eta} f(\mathcal{Q}^*) \tag{34}$$

where $\mathcal{Q}^*$ is the optimal solution to the multi-path orienteering problem for robots $\mathcal{R}\setminus\mathcal{R}(\mathcal{S}_1)$. Similar to Lemma 9 in [29], we have another inequality:

$$f(\mathcal{Q}^*) \ge f^* = f(\mathcal{S}^*\setminus\mathcal{A}^*(\mathcal{S}^*)), \tag{35}$$

where $f^*$ is the optimal solution to RMOP. For completeness, we give the proof in the supplementary document. Combining inequalities (31) – (35), we get the statement for Theorem 2. ■

## C. Proof of Inequality in Equation (31)

We will prove the following,

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq (1 - k_f) f(\mathcal{S}_2) \tag{36}$$

where $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ is the solution returned by our algorithm; $\mathcal{A}^*(S)$ is the optimal removal of $\mathcal{S}$; $k_f$ is the curvature of function $f$. Next, we prove inequality 36 and we will use Lemma 1 from [22] without proof. Proof here is essentially the same as that in [22] but with different notations for better understanding.

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

$$f(\mathcal{A}) \geq (1 - k_f) \sum_{a \in \mathcal{A}} f(a) \tag{37}$$

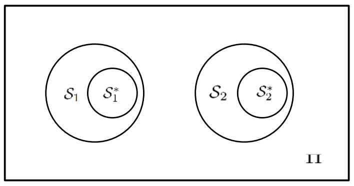


Fig. 14. Venn diagram, where $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_1^*, \mathcal{S}_2^*$ are defined as follows: Per run of proposed algorithm for RMOP, $\mathcal{S}_1$ and $\mathcal{S}_2$ are intermediate results such that $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, and $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. Let $\mathcal{S}^*_{\mathcal{A}}(\mathcal{S})$ be the optimal removal from $\mathcal{S}$. Then $\mathcal{S}_1^*, \mathcal{S}_2^*$ are defined such that $\mathcal{S}_1^* = \mathcal{A}^*(\mathcal{S}) \cap \mathcal{S}_1$ and $\mathcal{S}_2^* = \mathcal{A}^*(\mathcal{S}) \cap \mathcal{S}_2$. By definition, $\mathcal{S}_1^* \cap \mathcal{S}_2^* = \emptyset$ and $\mathcal{S}^*_{\mathcal{A}}(\mathcal{S}) = \mathcal{S}_1^* \cup \mathcal{S}_2^*$.

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

$$\begin{aligned}
f(\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})) &= f(\mathcal{S}_1^+ \cup \mathcal{S}_2^+) && (38)\\
&\geq (1 - k_f) \sum_{s \in \mathcal{S}_1^+ \cup \mathcal{S}_2^+} f(s) && (39)\\
&\geq (1 - k_f)\Big( \sum_{s \in \mathcal{S}_2 \setminus \mathcal{S}_2^+} f(s) + \sum_{s \in \mathcal{S}_2^+} f(s)\Big) && (40)\\
&\geq (1 - k_f)(f(\mathcal{S}_2 \setminus \mathcal{S}_2^+) + f(\mathcal{S}_2^+)) && (41)\\
&\geq (1 - k_f)((\mathcal{S}_2 \setminus \mathcal{S}_2^+) \cup \mathcal{S}_2^+) && (42)\\
&= (1 - k_f) f(\mathcal{S}_2) && (43)
\end{aligned}$$

where (38) holds by definition; (38) to (39) holds due to Lemma 4; (40) follows from (39) since all paths $s \in \mathcal{S}_1^+$ and all paths $s' \in \mathcal{S}_2 \setminus \mathcal{S}_2^+$, the inequality $f(s) \geq f(s')$ holds, i.e. paths in $\mathcal{S}_1$ have more rewards compared to that in $\mathcal{S}_2$ (note that by definitions of sets $\mathcal{S}_1^+$ and $\mathcal{S}_2^+$ it is $|\mathcal{S}_1^+| = |\mathcal{S}_2^*| = |\mathcal{S}_2 \setminus \mathcal{S}_2^+|$, i.e. the number of non-removed paths in $\mathcal{S}_1$ is equal to the number of removed paths in $\mathcal{S}_2$); from (40) to (41), it is due to submodularity; and (43) follows from (42) by definition.

## D. Proof of Inequality in Equation (32)

We will prove the following,

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{1}{\alpha + 1} f(\mathcal{S}_2) \tag{44}$$

We will use Lemma 2 from [22] without proof. Proof here is essentially the same as that in [22] but with notations consistent with our paper for better understanding.

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

$$f_{\mathcal{Y}}(\mathcal{P}) \leq |\mathcal{P}| f(\mathcal{Y}) \tag{45}$$

First we introduce one notation:

$$\xi = \frac{f_{\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*)}{f(\mathcal{S}_2)} \tag{46}$$

To prove (44), we still need to discuss two cases: $\mathcal{S}_2^* = \emptyset$ and $\mathcal{S}_2^* \neq \emptyset$. When $\mathcal{S}_2^* = \emptyset$, we have $f(\mathcal{S} \setminus \mathcal{A}^*(S)) = f(\mathcal{S}_2)$, and (44) holds. Next, we consider the case where $\mathcal{S}_2^* \neq \emptyset$ holds. The proof starts with one observation that

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \max\{f(\mathcal{S} \setminus \mathcal{A}^*(S)), f(\mathcal{S}_1^+)\}, \tag{47}$$

and then prove the following three inequalities:

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq (1 - \xi) f(\mathcal{S}_2) \tag{48}$$

$$f(\mathcal{S}_1^+) \geq \xi \frac{1}{\alpha} f(\mathcal{S}_2) \tag{49}$$

$$\max\{(1 - \xi, \xi \frac{1}{\alpha})\} \geq \frac{1}{\alpha + 1} \tag{50}$$

Next, if substitute (48), (49), and (50) to (47), then (44) is proved.

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

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

2) *proof of inequality 48*: The proof can be done in two steps. Firstly, it can be verified using $f_{\mathcal{A}}(\mathcal{B}) = f(\mathcal{A} \cup \mathcal{B}) - f(\mathcal{A})$ that

$$\begin{aligned}
f(\mathcal{S} \setminus \mathcal{A}^*(S)) = f(\mathcal{S}_2) &- f_{\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*)\\
&+ f_{\mathcal{S}_2}(\mathcal{S}_1) - f_{\mathcal{S} \setminus \mathcal{S}_1^*}(\mathcal{S}_1^*)
\end{aligned} \tag{51}$$

It should be noted that $f_{\mathcal{S}_2}(\mathcal{S}_1) - f_{\mathcal{S} \setminus \mathcal{S}_1^*}(\mathcal{S}_1^*) \geq 0$ for two following observations: i) $f_{\mathcal{S}_2}(\mathcal{S}_1) \geq f_{\mathcal{S}_2}(\mathcal{S}_1^*)$ since $f$ is monotone and $\mathcal{S}_1^* \subseteq \mathcal{S}_1$; ii) $f_{\mathcal{S}_2}(\mathcal{S}_1^*) \geq f_{\mathcal{S} \setminus \mathcal{S}_1^*}(\mathcal{S}_1^*)$ since $f$ is submodular and $\mathcal{S}_2 \subseteq \mathcal{S} \setminus \mathcal{S}_1^*$ (see also Fig. 14). Then we have

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

Inequality 48 proved.

3) *Proof of inequality 49*: Since it is $\mathcal{S}_2^* \neq \emptyset$ which suggests that $\mathcal{S}_1^+ \neq \emptyset$ and for all paths $a \in \mathcal{S}_1^+$ and all elements $b \in \mathcal{S}_2^*$ it is $f(a) \geq f(b)$, from Lemma 5, we have

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

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

$$\begin{aligned} f(\mathcal{S}_1^+) &\geq \frac{1}{\alpha} f_{\mathcal{S}_1^+}(\mathcal{S}_2^*) \\ &\geq \frac{1}{\alpha} f_{\mathcal{S}_1^+ \cup \mathcal{S}_1^+}(\mathcal{S}_2^*) \\ &= \frac{1}{\alpha} f_{\mathcal{S} \setminus \mathcal{A}^*(\mathcal{S})}(\mathcal{S}_2^*) \\ &= \xi \frac{1}{\alpha} f(\mathcal{S}_2) \end{aligned} \tag{52}$$

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

4) *Proof of inequality 50*: Let $b = \frac{1}{\alpha}$. We complete the proof first for the case where $(1-\xi) \geq \xi b$, and then for the case where $(1-\xi) < \xi b$: when $(1-\xi) \geq \xi b$, $\max\{(1-\xi), \xi b\} = 1 - \xi$ and $\xi \leq \frac{1}{1+b}$; due to the latter, $1 - \xi \geq \frac{b}{1+b} = \frac{1}{\alpha+1}$, which suggests inequality 50 holds; Finally, when $1 - \xi < \xi b$, $\max\{(1-\xi), \xi b\} = \xi b$ and $\xi > \frac{1}{1+b}$; due to the latter, $\xi b > \frac{b}{1+b} = \frac{1}{1+\alpha}$, which also suggests inequality (50) holds. Thus the inequality (44) is proved.

## *E. Proof of Inequality in Equation (33)*

We will prove the following,

$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq \frac{1}{N-\alpha} f(\mathcal{S}_2) \tag{53}$$

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

The following two cases are enough to explain why (53) holds.

- when $\mathcal{S}_2^* = \emptyset$, i.e. all paths in $\mathcal{S}_1$ are removed and correspondingly no paths are removed in $\mathcal{S}_2$. Then $f(\mathcal{S} \setminus \mathcal{A}^*(S)) = f(\mathcal{S}_2)$, and (53) holds.
- when $\mathcal{S}_2^* \neq \emptyset$, that is there is at least one path left in $\mathcal{S}_1$ and choose one $s$ from any of them, then
$$f(\mathcal{S} \setminus \mathcal{A}^*(S)) \geq f(s) \tag{54}$$
since $f$ is non-decreasing. Moreover,
$$f(\mathcal{S}_2) \leq \sum_{v \in \mathcal{S}_2} f(v) \leq (N-\alpha) f(s) \tag{55}$$
where the first inequality holds due to submodularity and the second holds because the proposed algorithm will construct $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $f(v') \geq f(v), \forall v' \in \mathcal{S}_1, v \in \mathcal{S}_2$.

Combine two cases, inequality (53) is proved.

## *F. Proof of Inequalities in Equation 35*

We will prove the following

$$f(\mathcal{Q}^*) \geq f^*, \tag{56}$$

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

We first prove the inequality 56. Let $\Pi_i$ be the set of all feasible paths for robot $i$, which is hard to compute but is assumed here somehow known for analysis purposes. Then the ground set is defined as

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

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

$$\mathcal{I} = \{I \subseteq \Pi \mid |I \cap \Pi_i| \leq 1, \forall i = 1, 2, \ldots, N\} \tag{57}$$

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

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

$$\begin{aligned} \mathcal{I}' = \{I \subseteq \Pi \mid |I \cap \Pi_i| \leq a,\ & a = 1, \forall i \in \mathcal{R}(\mathcal{S}_1); \\ & a = 0, \forall i \in \mathcal{R} \setminus \mathcal{R}(\mathcal{S}_1)\} \end{aligned} \tag{58}$$

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

In inequality (56), L.H.S, for any set $\mathcal{S}_1 \subseteq \Pi$ returned by algorithms with $|\mathcal{S}_1| = |\mathcal{R}(\mathcal{S}_1)| = \alpha$ such that $\mathcal{S}_1 \in \mathcal{I}$ and $\mathcal{S}_1 \in \mathcal{I}'$. By definition of $\mathcal{Q}^*$

$$\begin{aligned} f(\mathcal{Q}^*) &= \max_{\mathcal{S}_2 \subseteq \Pi, \mathcal{S}_2 \cup \mathcal{S}_1 \in \mathcal{I}} f(\mathcal{S}_2) && (59) \\ &= \max_{\mathcal{S}_2 \subseteq \Pi \setminus \mathcal{S}_1, \mathcal{S}_2 \cup \mathcal{S}_1 \in \mathcal{I}} f(\mathcal{S}_2) && (60) \\ &\geq \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} \max_{\mathcal{S}_2 \subseteq \Pi \setminus \tilde{\mathcal{S}}_1, \mathcal{S}_2 \cup \tilde{\mathcal{S}}_1 \in \mathcal{I}} f(\mathcal{S}_2) && (61) \\ &= \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \subseteq \in \mathcal{I}'} \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}, \tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) && (62) \\ &= \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \subseteq \in \mathcal{I}'} \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) && (63) \\ &\triangleq h && (64) \end{aligned}$$

(59) is the definition of $\mathcal{Q}^*$; (59) to (60) holds since we have a partition matroid with independent set defined as (57) and the intersection of $\mathcal{S}_1$ and $\mathcal{S}_2$ will be empty; In (60), $\mathcal{S}_1$ is a specific subset of $\Pi$ and in the independent set $\mathcal{I}'$. If we think $\mathcal{S}_1$ as an instantiation of a certain 'set variable' $\tilde{\mathcal{S}}_1$, we can find a minimal value (R.H.S of (61)) through optimizing over $\tilde{\mathcal{S}}_1$ and (60) should be greater than minimal value, i.e. (60) to (61) holds; next we use the trick of changing of variables: let $\tilde{\mathcal{S}} = \tilde{\mathcal{S}}_1 \cup \mathcal{S}_2$ and $\mathcal{S}_2 = \tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1$ due to the fact that $\tilde{\mathcal{S}}_1$ and $\mathcal{S}_2$ are disjoint. As a result, (61) to (62) holds; notice that in (62) the minimization operation over $\tilde{\mathcal{S}}_1$ can guarantee that the solution satisfies $\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}$ and we can remove the redundant constraint $\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}$ in maximization, i.e. (62) to (63) holds; and we define (63) as $h$.

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

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

which implies:

$$\begin{aligned} h &\geq \min_{\tilde{\mathcal{S}}_1 \subseteq \Pi, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} f(\mathcal{S} \setminus \tilde{\mathcal{S}}_1) \\ &= \min_{\tilde{\mathcal{S}}_1 \subseteq \mathcal{S}, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} f(\mathcal{S} \setminus \tilde{\mathcal{S}}_1) \end{aligned} \tag{66}$$

Notice that (66) holds for all $\mathcal{S} \in \mathcal{I}$. As a result,

$$h \geq \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} \min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \tag{67}$$

Next we consider the minimization operation of (67). For any $\tilde{\mathcal{S}} \in \mathcal{I}$,

$$\min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \geq \min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, |\tilde{\mathcal{S}}_1| \leq \alpha} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) \tag{68}$$

Reasons for (68) to hold: the L.H.S of (68) only allows remove $\tilde{\mathcal{S}}_1 \in \mathcal{I}'$ and $|\tilde{\mathcal{S}}_1|$ can go up to $|\mathcal{R}(\mathcal{S}_1)| = \alpha$ (refer to definition of $\mathcal{I}'$); by contrast, R.H.S of (68) is less constrained and can also remove up to $\alpha$ elements from $\tilde{\mathcal{S}}$. Thus, the R.H.S can get the result no greater than L.H.S. As a result,

$$\begin{aligned} h &\geq \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} \min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, \tilde{\mathcal{S}}_1 \in \mathcal{I}'} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) && (69) \\ &\geq \max_{\tilde{\mathcal{S}} \subseteq \Pi, \tilde{\mathcal{S}} \in \mathcal{I}} \min_{\tilde{\mathcal{S}}_1 \subseteq \tilde{\mathcal{S}}, |\tilde{\mathcal{S}}_1| \leq \alpha} f(\tilde{\mathcal{S}} \setminus \tilde{\mathcal{S}}_1) && (70) \\ &= f(\mathcal{S}^* \setminus \mathcal{A}^*(\mathcal{S}^*)) && (71) \\ &= f^* && (72) \end{aligned}$$

In sum,

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

**Guangyao Shi** received the B.Sc. and the M.Sc. degrees in in electrical and computer engineering from Harbin Institute of Technology, Harbin, China, in 2015 and 2017, respectively. He is currently working toward the Ph.D. degree with University of Maryland, MD, USA.

His research interest include multi-robot informative path planning, decision-making under uncertainty for robotic teams, and learning-augmented planning algorithms.

**Lifeng Zhou** is an Assistant Professor in the Department of Electrical and Computer Engineering at Drexel University. Previously, he was a Postdoctoral Researcher at the GRASP lab of the University of Pennsylvania. He obtained his Ph.D. in Electrical and Computer Engineering from Virginia Tech in 2020. Before that, he received his MS from Shanghai Jiao Tong University in 2016 and his BS from Huazhong University of Science and Technology in 2013. His academic expertise lies in multi-robot coordination, control algorithms, and graph neural networks and he designs resilient, risk-aware algorithms for security, trustworthiness, and long-term autonomy of robotics and autonomous systems. He regularly publishes in top-tier journals and conferences in robotics, including IEEE Transactions on Robotics (T-RO), IEEE Transactions on Automation Science and Engineering (T-ASE), IEEE Robotics and Automation Letters (RA-L), Autonomous Robots (AURO), Robotics: Science and Systems (RSS), IEEE International Conference on Robotics and Automation (ICRA), and IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). His research has been funded by the U.S. Army Research Laboratory (ARL)-Distributed and Collaborative Intelligent Systems and Technology (DCIST). He serves as an Associate Editor for the ICRA Conference Editorial Board and as Program Committee Member for the AAAI-22 Student Abstract and Poster Program and ACM Symposium on Applied Computing.

**Pratap Tokekar** is an Associate Professor in the Department of Computer Science and UMIACS at the University of Maryland. Between 2015 and 2019, he was an Assistant Professor at the Department of Electrical and Computer Engineering at Virginia Tech. Previously, he was a Postdoctoral Researcher at the GRASP lab of University of Pennsylvania. He obtained his Ph.D. in Computer Science from the University of Minnesota in 2014. He is a recipient of the NSF CAREER award (2020), Amazon Research Award (2022), and CISE Research Initiation Initiative award (2016). His research focuses on developing algorithms for multi-robot coordination and multi-agent reinforcement learning.