Multi-robot extension for safe planning under dynamic uncertainty

Master Thesis

Author(s):
Tihanyi, Dániel

Publication date:
2021-01-11

Permanent link:
https://doi.org/10.3929/ethz-b-000471229

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

IFA Automatic Control Laboratory

Master Thesis

Multi-robot extension for safe planning under dynamic uncertainty

Tihanyi Dániel
January 11, 2021

Advisors
Prof. Maryam Kamgarpour, Dr. Orçun Karaca, Yimeng Lu

Contents

1 Introduction **4**

2 Notations **6**

3 Preliminaries **7**

4 Two-stage multi-robot safe planning framework **10**

4.1 Single-robot safe planning problem 10

4.1.1 Map model 11

4.1.2 Robot dynamics 12

4.1.3 Hazard dynamics 12

4.1.4 Target execution 12

4.1.5 Combined state space 13

4.1.6 Controller synthesis via dynamic programming 14

4.2 Multi-robot task allocation problem 15

4.3 Full-fleet safe planning framework 16

5 Greedy approach for multi-robot task allocation under dynamic uncertainties **18**

5.1 Forward greedy algorithm 18

5.1.1 Algorithm formulation 18

5.1.2 Performance guarantee 20

5.1.3 Distributed algorithm formulation 20

5.2 Reverse greedy algorithm 22

5.2.1 Algorithm formulation 22

5.2.2 Performance guarantee 24

5.2.3 Distributed algorithm formulation 24

5.3 Comparison of performance guarantees for forward and reverse greedy approaches 26

6 Numerical case studies **29**

6.1 Performance comparison of task allocation algorithms 29

6.2 Counterexample for the usage of the multiplicative group success 32

6.3 Performance comparison of full-fleet safe planning and two-stage multi-robot safe planning 35

7 Conclusions **38**

8 Appendix **42**

8.1 Calculating the contamination risk p_H^k 42

8.2 Definition of the combined transition kernel τ_S^k 43

8.3 Proof of Equation (13) 45

8.4 Full-fleet safe planning problem 47
8.5 Mild conditions for multiplicative group success to be a lower bound on the probability of group success 49
8.6 Performance guarantee for the forward greedy algorithm 50
8.7 Performance guarantee for the reverse greedy algorithm 52
8.8 Example hazard model τ_Y 54

Abstract

Safe planning problem arises in many applications including autonomous driving and exploration scenarios. In this thesis, we focus on a particular case studied for emergency rescue missions. The main challenge of such problems is the computational complexity of handling a dynamic uncertainty, e.g., a spreading hazard. A multi-agent extension can potentially improve the safety of the mission. However, it further increases the computational complexity with the need to consider exponentially many possible task-robot combinations. To overcome these computational issues, we propose a two-stage framework splitting the multi-robot safe planning problem into a low-level single-agent safe planning problem and a high-level multi-robot task allocation problem. For single-agent safe planning, we utilize an efficient Monte-Carlo sampling-based approximation to handle the dynamic uncertainty. For the task allocation problem, we use forward and reverse greedy heuristics to obtain approximate solutions. These algorithms are equipped with provable performance bounds on the safety of the resulting approximate solutions. Finally, we present several case studies on example environments to compare the performance of these different algorithms.

1 Introduction

The usage of multi-agent systems is of increasing interest in robotics [1]. In many applications, a fleet of robots has to operate cooperatively to reach a common goal, such as in robotic-teamwork football games [2], automated guided vehicle systems in warehouses [3], surveillance or monitoring missions [4–6] and others. Systems using multiple agents enjoy trivial advantages over single-agent solutions. They can distribute tasks or workload among themselves and reduce execution time by working in parallel. This makes multi-agent solutions capable of handling problems with higher complexity, larger number of tasks, carrying out more deliveries at a given time span, or covering larger areas in surveillance missions than their single-agent counterparts. Multi-agent systems are also highly reliable and robust against failure, since the system can still continue working even in case of multiple agents failing. These advantages, however, come with an increased computational complexity which requires the usage of more sophisticated approaches and algorithms. The challenges and the tools can vary depending on the application. This thesis aims to extend the existing research in planning for safety-critical rescue missions to a multi-agent framework. This requires a combined study of two fields: safe planning and multi-agent task allocation.

The goal of safe planning is to maximize the probability of successfully executing a given set of tasks by deriving control policies in a potentially hostile environment. The applications include safe autonomous driving [7], exploration scenarios [8], or as in our case, emergency rescue missions with dangerously spreading fire or toxic contamination threatening the life of survivors [9]. This problem comes with several challenges to provide solutions in such applications. A major challenge originates from modelling the dynamic uncertainties of the environment caused by the evolving hazard. Many approaches either use restrictive Gaussian models [10,11] or more general yet computationally intractable Markov models [12]. We build on the previous work of [13] which provides a trade-off between both issues by using a Monte-Carlo sampling-based approximation to reduce the state space required for its Markov model. Another challenge originates from high-level decision making such as the one of ordering sequential execution of multiple tasks. To handle both these high-level decisions and the low-level point-to-point path planning, an extended state space definition is used in [9, 13] incorporating the execution of tasks into the problem definition. The optimal policies are derived using the well-known dynamic programming algorithm [14].

Multi-agent task allocation problems aim to decide which tasks should be executed by which agents in order to maximize a collective success measure. In their general form, such set partition problems are known to be NP-hard, hence the usage of heuristics as an approximate solution method is an attractive option for their scalability [15]. Variants of the greedy algorithm are widely-used in combinatorial optimization literature [4, 5, 16–24]. A valid allocation constituting a partition is known to be given by the base of a matroid where the ground set is all agent-task pairs. Our goal is then to maximize the set function mapping allocations to some collective objective. The greedy algorithm achieves this by iteratively adding the best

agent-task pair towards a valid allocation. Under a submodular objective function greedy enjoys 1/2 performance guarantee [4, 16–19, 21]. The probability of successful navigation in the emergency rescue missions is, however, non-submodular. To mitigate this issue, the concepts of submodularity ratio and curvature were introduced, which measure how far a function is from being submodular and supermodular, respectively. Previous works on non-submodular set function maximization used the notion of the submodularity ratio [22, 25], the curvature [24, 26] or both [20, 23] to provide performance guarantees for the greedy algorithm. We build on these studies and provided two novel performance guarantees for greedy algorithms in matroid optimization. The first is on the forward greedy algorithm, improving and generalizing [16] and [22] by the inclusion of both the submodularity ratio and curvature properties, respectively. The second is on the reverse greedy algorithm improving and generalizing [27] by removing the strong requirement of using the notion of total curvature, and [23] by reducing the requirement on the cardinality of the ground set.

It is important to note alternative approaches to solve set partition problems, coinciding with a multi-agent task allocation problem. The authors in [28] consider a set partitioning problem, where each subset of tasks is associated with a fixed cost for agents, and the goal is to find the optimal partitioning. This approach requires the cost of each subset to be evaluated in advance. In our case, this can only be obtained by solving the single-agent safe planning problem for all these subsets, which would be computationally demanding. Another approach would be to use a bipartite graph listing all agents and tasks as vertices where the edges, each associated with a cost, represent the assignment between them, see [29]. In our multi-agent safe planning problem, we cannot define such costs for each task-agent pair independently, since our objective is non-additive (in other words, non-modular). We can only define such costs for subsets of tasks allocated to an agent. In conclusion, none of the aforementioned approaches are applicable for our purposes.

Combining safe planning and multi-agent task allocation, our contribution is to provide a scalable framework for the multi-robot emergency rescue scenario. The usage of multiple agents could potentially improve success probability, the system would be able to handle more tasks, hence more survivors could be saved with higher probability. On the high-level, task allocation aims to allocate each survivor to a robot. According to the taxonomy in [15], this problem would be classified as a multi-task robot, single-robot task, instantaneous assignment[1] problem (MT-SR-IA). On the other hand, notice that each robot can handle multiple tasks. Task allocation is solved via greedy algorithms. We analyze two different algorithms, the forward and reverse greedy approaches, and compare them both in terms of their theoretical performances, experimental performances, and computational times. In doing that, we provide two novel performance guarantees for these greedy algorithms applied to general matroid optimization problems. On the low-level, we derive control policies for each agent for a given subset of tasks, while maximizing the probability of successful navigation. This low-level framework is an efficient implementation of the ones in [9, 13], where a Monte-Carlo sampling based algorithm is

[1]Since we allocate the tasks to the robots before execution, we have the so-called instantaneous assignment setting.

proposed to overcome the computational burdens of Markovian models of stochastically evolving hazard.

We organize this thesis report as follows. Preliminaries summarize the necessary mathematical background for this report in Section 3. In Section 4, we introduce the two-stage framework by formulating both the single-robot safe planning and the multi-robot task allocation problems. Next, we propose the two greedy approaches in Section 5, the forward and reverse greedy algorithms. Finally, we show three case studies comparing algorithm performance, see Section 6, and we then conclude the paper in Section 7.

2 Notations

Unless stated otherwise, we use the following conventions when naming variables throughout the thesis report. We denote finite sets by block letters, their elements by lower case letters, and families of sets by calligraphic block letters. Let X be a finite set. We use the definition of indicator function $\mathbb{1}_{\bar{x}} : X \to \{0, 1\}$ for an element $\bar{x} \in X$ and $\mathbb{1}_{\bar{X}} : X \to \{0, 1\}$ for a subset $\bar{X} \subset X$. They are defined the following way

$$\mathbb{1}_{\bar{x}}(x) = \begin{cases} 1 & \text{if } x = \bar{x}, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathbb{1}_{\bar{X}}(x) = \begin{cases} 1 & \text{if } x \in \bar{X}, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, let $|X|$ denote the size of a finite set X. We use the notation $\wedge$ for the logical 'AND' and $\vee$ for the logical 'OR' in mathematical statements. Let $p(x = \bar{x})$ denote the probability of a discrete random variable x taking the value of $\bar{x}$.

3 Preliminaries

In this section, we introduce well-studied notions from discrete mathematics literature [30–33]. These notions will be used for the derivations in the remainder of this report. Let W be a nonempty ground set and $f : 2^W \to \mathbb{R}$ a set function for the following definitions.

Definition 1 *(Monotonicity properties) The set function f is non-decreasing if for all $A \subseteq B \subseteq W$, $f(A) \leq f(B)$. We call $-f$ non-increasing. If the inequality is strict, then f is strictly increasing and $-f$ is strictly decreasing.*

Definition 2 *(Discrete derivative) For the set function f, $A \subseteq W$ and $e \in W$, the discrete derivative of f at A with respect to e is given by*

$$\rho_f(e|A) := f(A \cup \{e\}) - f(A).$$

We simply use the notation $\rho(e|A)$, if the function f is clear from the context. Moreover for any set $B \subset W$, we will generalize the definition above to denote $\rho(B|A) = f(A \cup B) - f(A)$.

Definition 3 *(Submodularity) A non-decreasing set function f is submodular if it holds that*

$$\rho(e|B) \leq \rho(e|A), \tag{1}$$

for all $A \subseteq B \subseteq W$, for all $e \in W \setminus B$.

Submodularity is a useful property commonly used in combinatorics. Equation (1) expresses that the marginal gains of f are decreasing when expanding the set A to B, which happens to be the case in many realistic examples, see [33–35]. Many set function optimizing algorithms take advantage of this notion, such as the greedy algorithm used later in this report. Unfortunately, the objective functions used in many problems, including ours, do not have the submodular property. Instead, these problems allow the use of *submodularity ratio* describing how far a non-submodular set function is from being submodular. This property was first introduced in [25].

Definition 4 *(Submodularity ratio) The submodularity ratio of a nondecreasing set function f is the largest scalar $\gamma \in \mathbb{R}_+$ such that*

$$\gamma \cdot \rho(e|B) \leq \rho(e|A), \tag{2}$$

for all $A \subseteq B \subseteq W$, for all $e \in W \setminus B$.

It can easily be verified that f is submodular if and only if $\gamma = 1$, and we also have $\gamma \in [0, 1]$. For derivations, kindly refer to [20]. Furthermore, there exist an alternative but

non-equivalent submodularity ratio notion [20,36]: the *cumulative submodularity ratio* of a non-decreasing set function f is the largest scalar $\gamma' \in \mathbb{R}_+$ such that

$$\gamma' \cdot \rho(B|A) \leq \sum_{e \in B \setminus A} \rho(e|A), \tag{3}$$

for all $A, B \subseteq W$. The submodularity ratio of Equation (2) satisfies the inequalities listed in Equation (3), but the reverse argument does not necessarily hold. Hence, $\gamma \leq \gamma'$ [23, Appendix B]. Later in Sections 5.1 and 5.2, we discuss the necessity of utilizing this notion as per Definition 4 for the guarantee we derive for the greedy algorithms.

Definition 5 *(Supermodularity) A non-decreasing set function f is supermodular if it holds that*

$$\rho(e|A) \leq \rho(e|B), \tag{4}$$

for all $A \subseteq B \subseteq W$, for all $e \in W \setminus B$.

Supermodularity is the property describing that the marginal gain of f is increasing when expanding the set A to B. Similar to the discussions provided for the submodularity ratio, whenever supermodularity is not found, it may instead be possible to use the notion of *curvature* to describe how far a non-supermodular function is from being supermodular.

Definition 6 *(Curvature) The curvature of a non-decreasing set function f is the smallest scalar $\alpha \in \mathbb{R}_+$ such that*

$$(1 - \alpha) \cdot \rho(e|A) \leq \rho(e|B), \tag{5}$$

for all $A \subseteq B \subseteq W$, for all $e \in W \setminus B$.

It can easily be verified that f is supermodular if and only if $\alpha = 0$, and we also have $\alpha \in [0, 1]$. For derivations, kindly refer to [20]. Finally, note that it is also possible to have cumulative definitions of the curvature, similar to that of Equation (3). However, for our derivations in Sections 5.1 and 5.2, we draw special attention on where we require the inequalities in Equation (5).

Complex constraints in many combinatorial optimization problems can be modelled by using notions from matroid theory introduced in the following. Reformulating this way generalizes the constraints and helps to provide performance guarantees.

Definition 7 *(Matroid). A matroid is a pair $\mathcal{M} = (W, \mathcal{I})$, such that $\mathcal{I} \subseteq 2^W$ is a collection of subsets of W called the independent set satisfying the two following properties*

(i) *$A \subseteq B \subseteq W$ and $B \in \mathcal{I}$ implies $A \in \mathcal{I}$*
(ii) *$A, B \in \mathcal{I}$ and $|B| > |A|$ implies $\exists e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$.*

The concept of a matroid is considered to be a generalization of linear-independence well known from linear algebra. We introduce a special type of matroid used for the problem formulations of this report, the *partition matroid*.

Definition 8 *(Partition matroid) The pair $\mathcal{M} = (W, \mathcal{I})$ is a partition matroid if a partition of W exists characterized by $\{B_i\}_{i=1,\dots,n}$, where $W = \bigcup_{i=1}^{n} B_i$ and $B_i \cap B_{i'} = \emptyset$ for all pairs $i, i' \in \{1, \dots, n\}$, furthermore there exist a set of positive integers $l_i \in Z_+$ for all $i = 1, \dots, n$, such that*

$$\mathcal{I} = \{A \subseteq W \,|\, |A \cap B_i| \leq l_i, \forall i = 1, \dots, n\}.$$

Furthermore, we also use the following matroid theory related properties.

Definition 9 *(Base of a matroid) Let $\mathcal{M} = (W, \mathcal{I})$ be a matroid. We call $B \in \mathcal{I}$ a base of matroid $\mathcal{M}$, if $|A| \leq |B|$ for all $A \in \mathcal{I}$. In other words, a base of a matroid is an inclusion-wise maximal independent set. Notice, that every base has the same cardinality.*

Definition 10 *(Dual of a matroid) For a matroid, $\mathcal{M} = (W, \mathcal{I})$, the dual matroid $\bar{\mathcal{M}} = (W, \bar{\mathcal{I}})$ is defined so defined so that the bases $\bar{B} \in \bar{\mathcal{I}}$ are exactly the complements of the bases $B \in \mathcal{I}$, that is, $\bar{B} = W \setminus B$.*

4 Two-stage multi-robot safe planning framework

In this section, we introduce the *multi-robot safe planning problem* and propose a framework that provides a computationally tractable solution. Consider a fleet of autonomous agents navigating through an environment with a stochastically evolving hazard, for example, a fire inside a building. The *mission* of the fleet is to visit a set of known targets and then get out safely (e.g. rescuing survivors). Each target is required to be visited once by any robot. The map of the environment, the initial position of the robots, the areas initial contaminated by the hazard, and the stochastic model of the dynamics of both the hazard and the robots are assumed to be known. The multi-robot safe planning problem aims to design *control policies* for the robots. These control policies maximize the *probability of success* – the probability of successfully finishing the mission. However, planning for multiple robots is computationally much more challenging than planning for a single robot. Hence our framework splits the problem into the following two hierarchical stages: high-level task allocation (dividing the targets between robots) and low-level path planning (optimizing control policies for each robot individually for a subset of assigned targets). We call this the *two-stage multi-robot safe planning framework*. The low-level stage introduced in Section 4.1 aims to obtain an optimal control policy for a single robot maximizing the probability of successfully visiting only a chosen subset of all targets. The high-level stage, which builds upon the low-level one, is a multi-robot task allocation problem aiming to divide the targets among robots and is described in Section 4.2. To justify the need for splitting the problem into stages as in our framework, we formulate a safe planning problem for the whole fleet combined in Section 4.3 and show its computational burdens. Throughout this section, we use an example multi-agent planning problem to illustrate our framework, see Figure 2.

4.1 Single-robot safe planning problem

This section introduces the low-level *singler-robot safe planning framework* for a single agent aiming to obtain an optimal control policy by maximizing the probability of successfully visiting a set of targets and leave the environment while avoiding the evolving hazard. The problem formulation is based on previous works in path planning under dynamic uncertainties, namely, [9] and [13], and presented as follows. We first concisely introduce this model and leave the details in the following subsections. We start by considering a single-robot system, describe the environment with a discretized map, then introduce the robot and hazard evolution dynamics. Next, we show how the agent keeps track of the high-level target execution. We then define a combined state space for path planning considering the robot dynamics, the target execution and hazard avoiding aspects. Finally, we define the controller synthesis problem and present a dynamic programming algorithm that solves this problem. To obtain a tractable version of this approach, we propose an approximation based on Monte-Carlo sampling to overcome the computational issues caused by the presence of dynamic uncertainties.

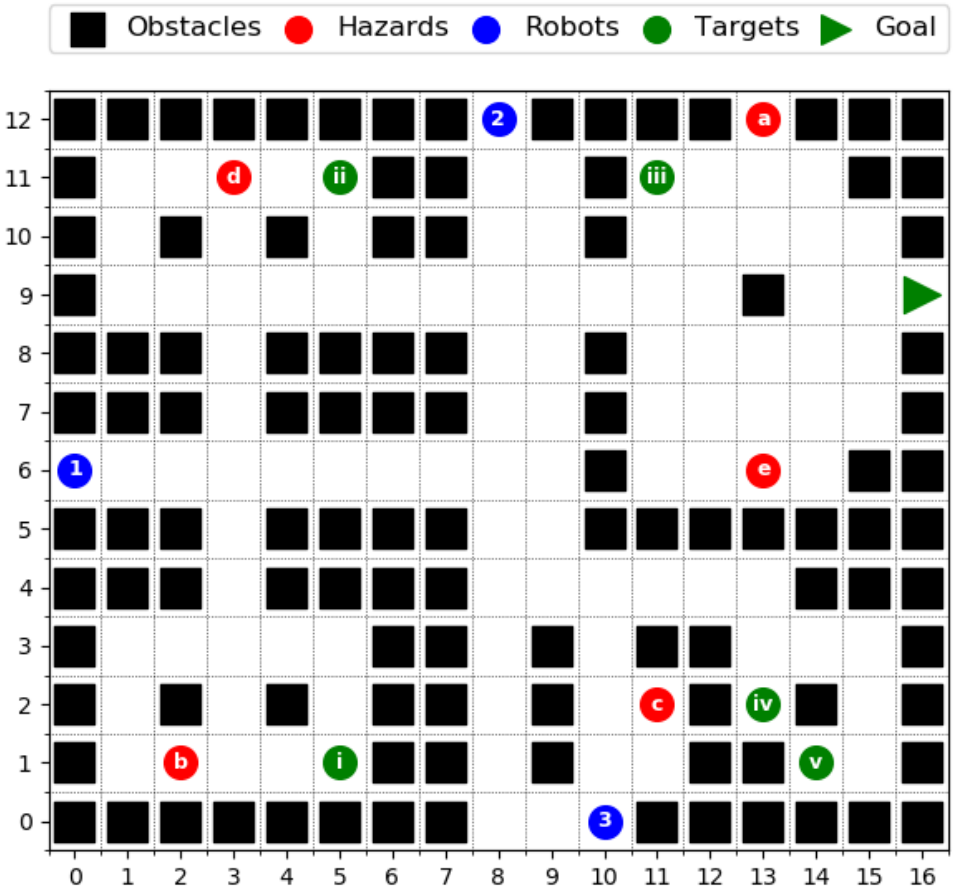


Figure 2: Example environment. The fleet of robots have to reach the goal position after cooperatively visiting the targets while avoiding the stochastically evolving hazards.

4.1.1 Map model

We define a discretized model of the map where the robot operates. Let

$$M_{m\times n} = \{(a,b) \,|\, a \in \{0,\dots,m-1\}, b \in \{0,\dots,n-1\}\}\,,$$

be an $m \times n$-sized grid-shaped map (the grid length equals to 1) and $O \subset M_{m\times n}$ be a set of obstacles (e.g., walls) untraversable for the robot. Then, $X = M_{m\times n} \setminus O$ is the set of all traversable positions. Furthermore, for all $x \in X$ let

$$N(x) = \left\{x' \in X \,|\, \|x' - x\|_2 = 1\right\},$$
$$D(x) = \left\{x' \in X \,|\, \|x' - x\|_2 = \sqrt{2}\right\},$$

be the neighboring and diagonally neighboring positions for x, respectively. We use the notation $\|x' - x\|_2$ for the Euclidean distance between points x and x'. The usage of a discredited map simplifies the formulation from a non-convex continuous optimization problem to an easier combinatorial optimization problem. It is a reasonable approximation of the environment also used in [9] and [13].

4.1.2 Robot dynamics

We introduce the dynamics of the motion of the robot. We define a set of possible inputs: $U = \{\text{Stay, North, East, South, West}\}$. Each input $u \in U$ is associated with a direction

$$d_{\text{Stay}} = (0,0), \qquad d_{\text{North}} = (0,1), \qquad d_{\text{East}} = (1,0),$$
$$d_{\text{South}} = (0,-1), \qquad d_{\text{West}} = (-1,0).$$

In each position $x \in X$, the set of

$$U(x) = \{u \in U \,|\, x + d_u \in X\} \subseteq U,$$

are the inputs available to the robot.

The motion of robot r is defined by a stochastic Markov process $x^{k+1} \sim \tau_X\left(\cdot \,|\, x^k, u^k\right)$, $k \in \{0,1,\dots\}$ with initial position $x_r^0 \in X$, where $\tau_X : X \times X \times U \to [0,1]$ denotes the transition kernel between $x^k \in X$ at time step k and $x^{k+1} \in N(x)$ at step $k+1$ under control input $u^k \in U(x^k)$. Note that different robots can be equipped with different dynamics. We say that the robot dynamics are deterministic, if for all $x^k \in X$ and $u^k \in U(x^k)$

$$\tau_X\left(x^{k+1} \,|\, x^k, u^k\right) = \mathbb{1}_{x^k + d_{u^k}}(x^{k+1}). \tag{6}$$

4.1.3 Hazard dynamics

We introduce the model of the hazard and how it spreads across the map. Let $Y = 2^X$ be the hazard state space. Each element $y \in Y$ denotes a set of contaminated cells being a subset of the reachable map X. The stochastic Markov process $y^{k+1} \sim \tau_Y\left(\cdot \,|\, y^k\right)$, $k \in \{0,1,\dots\}$ defines the hazard evolution dynamics with transition kernel $\tau_Y : Y \times Y \to [0,1]$ between states $y^k \in Y$ at time step k and $y^{k+1} \in Y$ at step $k+1$. At time 0, we assume the hazard state $y^0 \in Y$ to be known to the robot.

4.1.4 Target execution

During the execution of the mission, the agent needs to keep track of which target locations have already been visited. To this end, we introduce the following *target execution state*. First we define $T_r \subset X$ as the *target list* of robot r, the set of all target locations the agent has to visit. Then we define the set $Q = 2^{T_r}$ and the target execution state $q^k \subseteq T_r$ at time step k, where $q^k \in Q$ for all k. The transition at time step k from q^k to q^{k+1} given the robot is at position x^{k+1} at step $k+1$ is described by the following time homogeneous transition kernel

$$\tau_Q(q^{k+1} \,|\, q^k, x^{k+1}) = \begin{cases} 1 & \text{if } q^{k+1} = q^k \cup (x^{k+1} \cap T_r), \\ 0 & \text{otherwise,} \end{cases}$$

where $\tau_Q : Q \times Q \times X \to [0, 1]$. Every time one of the target positions $x^{k+1} \in T_r$ is visited, it is added to the list q^{k+1}. For any other non-target position $x^{k+1} \notin T_r$, we have $x^{k+1} \cap T_r = \emptyset$ and $q^{k+1} = q^k$ stays the same. If a target position $x^{k+1} \in T_r$ is visited more then one time, hence $x^{k+1} \in q^k$, then $q^{k+1} = q^k \cup \{x^{k+1}\} = q^k$.

4.1.5 Combined state space

In this section, we define the combined state space of the agent in order to model the complex mission of motion, target collection and hazard avoidance. At time step k the state should contain both the robot location $x^k \in X$ and the target execution state $q^k \in Q$. Some pairs of (q^k, x^k) are impossible to occur specifically, when $x^k \in T_r$ but $x^k \notin q^k$, hence we can reduce the size of the state space by removing these pairs. We further assume that the agent cannot observe the state of the hazard $y^k \in Y$, which is a reasonable assumption in most realistic scenarios, hence we do not include y^k in the state space [2]. However, a *contamination state* noted by s_H should be introduced to capture the contamination of the robot. The agent transmits into state s_H if it moves to a contaminated area $x^k \in y^k$, after which it cannot leave this state anymore. Reaching the contamination state indicates an *unsuccessful mission*. We can now write the combined state space as follows

$$S = \{s_H\} \cup (Q \times X) \setminus \{(q, x) \,|\, x \in T_r \wedge x \notin q\}. \tag{7}$$

We can further specify the goal location as $x_G \in X$ and the *goal state* denoted by $s_G = (T_r, x_G)$. The state s_G indicates a *successful mission*, where every target is visited and the robot has reached the safe goal location without getting contaminated. We also define the initial state for robot r as $s_r^0 = (\emptyset, x_r^0)$, where no targets are visited and the robot is at the initial position x_r^0. The state s_r^0 is certain and known to the agent, since x_r^0, the initial position of robot r is assumed to be given.

Although the agent cannot observe the state of the hazard $y^k \in Y$ at a certain time step k, it can still use the knowledge of the hazard dynamics τ_Y and the initial hazard state y^0. To this end, we define the function $p_H^k : X \times X \to [0, 1]$ describes the *contamination risk*, the risk the robot takes while moving to a new grid cell at a given time step k. The value of $p_H^k(x^{k+1}, x^k)$ for the pair x^{k+1} and x^k is equal to the probability of $x^{k+1} \in y^{k+1}$ getting contaminated at time step $k + 1$, given that $x^k \notin y^k$ is not contaminated at step k. For values $x^{k+1} = \bar{x}^{k+1}$ and $x^k = \bar{x}^k$

$$p_H^k(\bar{x}^{k+1}, \bar{x}^k) = P(x^{k+1} \in y^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k). \tag{8}$$

We provide the details of calculating p_H^k in Appendix 8.1. Due to the exponentially increasing size of $|Y| = 2^{|X|}$, the precise calculation of function p_H^k described in Appendix 8.1 is computationally intractable. In order to overcome this issue, a Monte-Carlo sampling based algorithm is proposed

[2] For further justification why y^k should not be included in the state space, let us study the complexity of X, Q and Y. The size of X depends on the size of the map, whereas the number of target execution states $|Q| = 2^{|T_r|}$ grows exponentially with the size of T_r. In practice, we have $|X| \gg |T_r|$. Thus, the size of the hazard state space $|Y| = 2^{|X|}$ is the main source of complexity. Including Y in the state space would make the problem intractable even for small maps.

in [13, Algorithm 1], which provides a tractable approximation of p_H^k. For the rest of the report we refer to p_H^k as the approximate value obtained by [13, Algorithm 1].

The evolution of the combined state of the agent can now be described by the following stochastic process defined by transition kernel $\tau_S^k : S \times S \times U \to [0, 1]$. Given that the robot is in state s^k at time step k, the probability of getting into state s^{k+1} at step $k+1$ by applying control input u^k can be written as follows (see Appendix 8.2)

$$\tau_S^k(s^{k+1} \,|\, s^k, u^k) = \begin{cases} 1 & \text{if } s^{k+1} = s^k \in \{s_G, s_H\}, \\ \sum\limits_{x^{k+1} \in X} p_H^k(x^{k+1}, x^k) & \\ \quad \times \tau_X(x^{k+1} \,|\, x^k, u^k) & \text{if } s^{k+1} = s_H \\ & \wedge\ s^k = (q^k, x^k) \notin \{s_G, s_H\}, \\ \big(1 - p_H^k(x^{k+1}, x^k)\big) & \\ \quad \times \tau_Q(q^{k+1} \,|\, q^k, x^{k+1}) & \\ \quad \times \tau_X(x^{k+1} \,|\, x^k, u^k) & \text{if } s^{k+1} = (q^{k+1}, x^{k+1}) \neq s_H \\ & \wedge\ s^k = (q^k, x^k) \notin \{s_G, s_H\}, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Both the goal s_G and hazard s_H states are defined to be *absorbing*, which means, once they are reached, the system state does not change anymore. In any other states $s^k = (q^k, x^k) \notin \{s_G, s_H\}$, the agent can either get contaminated and reach state s_H or move to another state following the dynamics defined by transition kernels τ_X and τ_Q.

4.1.6 Controller synthesis via dynamic programming

Based on the previously described combined state space and transition dynamics, we state the success probability maximizing optimization problem. We also propose a dynamic programming algorithm to solve this problem and obtain the optimal control policy. We assume that robot r, a set of target locations $T_r \subset X$, the initial state s_r^0 and a finite time horizon $N \in \mathbb{N}_{>0}$ is given. We aim to compute the optimal (that is success probability maximizing) closed-loop control policy $\pi_r(T_r) = \{\mu_r^0, \ldots, \mu_r^{N-1}\}$ as a function of T_r, where $\mu_r^k : S \to U$ refers to the control law at time step k, so that $u^k = \mu_r^k(s^k)$. We denote the optimal probability of success under the optimal control policy $\pi_r(T_r)$ by $f_r(T_r)$. Furthermore, the probability of success $f_r(\pi, T_r)$ under a generic control policy $\pi = \{\mu^0, \ldots, \mu^{N-1}\}$ can be described by reaching the goal state $s^k = s_G$ at any step within the given time horizon $k \leq N$ while avoiding the contamination state $s^k = s_H$ at all time steps $k = \{0, \ldots, N\}$. Since both s_H and s_G are absorbing, the condition $s^N = s_G$ is sufficient for a successful mission as shown below

$$f_r(\pi, T_r) = P\left(s^N = s_G \,|\, \pi\right). \tag{10}$$

Our goal is then to solve

$$\pi_r(T_r) = \arg\max_{\pi} f_r(\pi, T_r). \tag{11}$$

Problem (11) can be solved using the well known dynamic programming algorithm [14]: For $k = N$, let us define $V^N(s^N) = \mathbb{1}_{s_G}(s^N)$ as the *value function*, while for $1 \leq k \leq N$,

$$V^{k-1}(s^{k-1}) = \max_{u \in U(x^{k-1})} \left\{ \sum_{s^k \in S} \tau_S^{k-1}(s^k \mid s^{k-1}, u) \cdot V^k(s^k) \right\}. \tag{12}$$

Now $\mu_r^k(s^k)$ can be obtained as the optimal $u \in U(x^k)$ at step k. Furthermore it holds (see Appendix 8.3), that

$$f_r(T_r) = f_r(\pi_r, T_r) = \max_{\pi} f_r(\pi, T_r) = V^0(s_r^0). \tag{13}$$

4.2 Multi-robot task allocation problem

In this section, we formulate the high-level task allocation problem aiming to optimally assign the targets among the agents. Let T be the set of all targets and R the set of all robots. A valid task allocation assigns each task to exactly one agent by dividing set T into partitions $\{T_r\}_{r \in R}$, where $T_r \subset T$ for all $r \in R$, $T_r \cap T_{r'} = \emptyset$ for any pair $r, r' \in R$ where $r \neq r'$ and $\bigcup_{r \in R} T_r = T$. Each partition T_r represents the subset of targets assigned to robot r. We aim to find the optimal task allocation which maximizes the probability of successfully finishing the mission of visiting every target without getting any of the agents contaminated [3]. We use the *multiplicative group success* as the objective function defined by

$$F(\{T_r\}_{r \in R}) = \prod_{r \in R} f_r(T_r), \tag{14}$$

where the values of $f_r(T_r)$ are obtained by solving the single-robot path planning problem introduced in Section 4.1 for each $r \in R$ (see Equation (13)). Note that the multiplicative group success equals the product of single-agent success probabilities. Hence it assumes these success probabilities to be independent of each other. This assumption does not hold in general. However, in most cases, one of the robots succeeding makes it more probable for the others to succeed as well (see Appendix 8.5). Under this mild condition, the multiplicative group success can serve as a good and computationally tractable approximation. Now we can formulate the task allocation problem as follows

$$F^* = \max_{\{T_r\}_{r \in R}} \prod_{r \in R} f_r(T_r) \quad \text{s.t.} \quad T_r \cap T_{r'} = \emptyset, \forall r \neq r', \bigcup_{r \in R} T_r = T. \tag{15}$$

Every task is allocated to exactly one robot. Following this argument, each task can

[3] According to the taxonomy in [15], this problem can be classified as an NP-hard multi-task robot, single-robot task, instantaneous assignment task allocation problem (MT-SR-IA). Multi-task robot – because one robot can visit multiple targets, single-robot task – since it is enough for targets to be visited by only a single robot, and instantaneous assignment – since tasks are allocated only once before the run and not continuously during execution.

be allocated to any robot among $|R|$ different robots. Hence there are $|R|^{|T|}$ possible allocations in total. Therefore, the problem is exponential in the number of tasks, which motivates using polynomial-time heuristic algorithms to obtain approximate solutions. We propose such algorithms in Section 5 and provide detailed descriptions.

Furthermore, adding a task to the target list of a robot cannot increase its success probability. Because when one additional task is added to the task list of a robot, the updated path due to this additional task either becomes longer or deviate from the original path in most of the cases. Both of them decreases the probability of success. To capture this, we assume the individual functions f_r to be strictly and bounded decreasing, hence $\exists \underline{f}_r, \overline{f}_r \in \mathbb{R}$ such that

$$0 < \underline{f}_r \leq f_r(T_r) - f_r(T_r \cup t) \leq \overline{f}_r < 1, \tag{16}$$

for all $T_r \subsetneq T$, for all $t \in T \setminus T_r$ and every $r \in R$. We use this assumption in Section 5. This assumption might occasionally be violated if task t already lies on the path of robot r when executing task list T_r. In this rare case $f_r(T_r \cup t) = f_r(T_r)$.

4.3 Full-fleet safe planning framework

We generalize the single-robot planning formulation introduced in Section 4.1 and propose the *full-fleet safe planning framework* for $|R| \geq 1$ as an alternative approach for solving the multi-robot safe planning problem. We also show the computational burdens of this approach and compare it to the two-stage multi-robot safe planning framework (Section 4).

When describing robot locations, instead of considering the position of a single robot $x \in X$, we define the tuple $x_M = (x_1, \ldots, x_{|R|}) \in X_M$ as the combined position of the fleet, where $X_M = X^{|R|}$. As this is not the main focus of this study, we assume that the robots do not collide with each other. Hence multiple robots can occupy the same grid at the same time. We also define the combined input of the fleet as $u_M = (u_1, \ldots, u_{|R|}) \in U_M = U^n$. We extend the state space S introduced in Section 4.1.5 to be consistent with the definition of x_M the following way

$$S_M = \{s_{H,M}\} \cup (Q \times X_M) \setminus \{(q, x_M) \,|\, \exists r \in R \text{ st. } x_r \in T \wedge x_r \notin q\}, \tag{17}$$

where $s_{H,M}$ is the *combined contamination state* and $q \in Q$ is the task execution state analogous to s_H and Q defined in Section 4.1.5 and 4.1.4. The system transmits to $s_{H,M}$ if at least one robot becomes contaminated. Finally, we formulate the control synthesis of the full-fleet safe planning framework. We show the details of this formulation in Appendix 8.4. The solution to the full-fleet safe planning problem for a task list T is the *optimal group policy* $\pi_M(T) = \{\mu_M^0, \ldots, \mu_M^{N-1}\}$, where $\mu_M^k : S_M \to U_M$ is the *group control law* used at step k, and the *probability of group success* $F_M(T) \in [0, 1]$ using policy $\pi_M(T)$. These notations are defined analogous to the optimal policy $\pi_r(T_r)$ and the probability of success $f_r(T_r)$ introduced in Section 4.1.6 with the difference of considering the fleet as a whole for $|R| > 1$ instead of optimizing the path of a single-robot.

The state space of the full-fleet safe planning formulation grows exponentially in the number of robots, since $X_M = X^{|R|}$ (see Equation (17)). This phenomenon causes the solution

of the problem to be intractable even for a few number of robots and for small sized maps. The two-stage multi-robot safe planning framework overcomes this issue by using the following relaxations. First, it decouples the decisions of the agents. Instead of solving a joint problem for all robots at once and calculating the combined optimal group policy $\pi_M(T)$, we consider single-robot solutions defined by Section 4.1 obtaining policies $\{\pi_r(T_r)\}_{r\in R}$ for individual agents $r \in R$ independently from each other. Second, we consider the success of individual robots to be independent. Instead of considering the probability of group success $F_M(T)$, the probability of every robot succeeding simultaneously, we use the multiplicative group success defined by Equation (14), which is the product of individual robot success probabilities. This way we neglect the existing correlation between agents succeeding or failing, but obtain a computationally tractable optimization problem. We provide a detailed example to illustrate this correlation in Section 6.2. Under some mild conditions introduced in Appendix 8.5, the multiplicative group success $F(\{T_r\}_{r\in R})$ is a lower bound of the probability of group success $F_M(T)$. For the remainder, we restrict our attention to the multiplicative group success.

5 Greedy approach for multi-robot task allocation under dynamic uncertainties

We introduce the greedy approach which provides a computationally tractable approximation to the multi-robot task allocation problem of Section 4.2. We formulate the forward and reverse greedy algorithms in Section 5.1 and 5.2, respectively. The forward greedy approach is initialized with no tasks allocated to the robots and iteratively updates the allocation by adding the task-robot pair obtaining the best optimality gain until every task is allocated. The reverse greedy algorithm, however, allocates every task to every robot in the beginning and keeps removing the task-robot pairs. It converges when every task is allocated to exactly one robot. We also provide performance guarantees for both algorithms (see Section 5.1 and 5.2) and compare them in Section 5.3.

5.1 Forward greedy algorithm

This section introduces the forward greedy algorithm and provides a performance guarantee. First, we reformulate the allocation problem described by Equation (15) to a set function minimization problem over matroid constraints and propose the forward greedy algorithm. Then we state the performance guarantee comparing the approximate solution of the greedy algorithm to the optimal solution of the task allocation problem. Since the multiplicative group success (see Equation (14)) is a non-submodular objective function, we use the submodularity ratio (Equation (2)) and curvature (Equation (5)) properties to obtain the performance guarantee. Finally, we provide a distributed formulation of the forward greedy algorithm where robots make computations in parallel in order to increase calculation speed.

5.1.1 Algorithm formulation

In the following, we reformulate the task allocation problem described by Equation (15) as a set function minimization problem over matroid constraints and define the forward greedy algorithm. First of all, let the set of tasks be denoted by T and the set of robots by R. In Section 4.2, we described a valid task allocation by partitions $\{T_r\}_{r\in R}$, where $T_r \subset T$ denotes the set of tasks allocated to robot $r \in R$. In order to ensure that every task is allocated to exactly one robot, we introduced the constraints below, also used in Equation (15)

$$T_r \cap T_{r'} = \emptyset, \forall r, r' \in R, r \neq r', \bigcup_{r\in R} T_r = T. \tag{18}$$

Now, we define an alternative yet equivalent description for a valid task allocation. We define $P = T \times R$ as the ground set of all task-robot pairs and $P_t = \{(t, r)\}_{r\in R} \subset P$ as the task-robot pairs related to task $t \in T$. Note that the sets $\{P_t\}_{t\in T}$ define a partitioning of P, since $P_t \cap P_{t'} = \emptyset$ for all pairs $t, t' \in T$ if $t \neq t'$ and $\bigcup_{t\in T} P_t = P$. Now $A \subset P$ is a valid task allocation

expressed as a set of task-robot tuples, if

$$|A \cap P_t| = 1,\, \forall t \in T, \tag{19}$$

which is equivalent to the constraints described by Equation (18). Note that we can transform $\{T_r\}_{r\in R}$ to A the following way

$$A = \bigcup_{r\in R} \{(t, r)\}_{t\in T_r}. \tag{20}$$

Let us further define $K = |T|$, the set

$$\mathcal{I} = \{A \subset P \,|\, |A \cap P_t| \leq 1,\, \forall t \in T\},$$

the partition matroid $\mathcal{M} = (P, \mathcal{I})$ (see Definition 8) and the objective function $F_{\text{fg}} : P \to [0, 1]$

$$F_{\text{fg}}(A) = -\prod_{r\in R} f_r(T_r), \tag{21}$$

where the relationship between A and $\{T_r\}_{r\in R}$ is defined by Equation (20). Now Equation (15) can be reformulated the following way

$$A_* = \underset{A\in\mathcal{I}}{\arg\min}\, F_{\text{fg}}(A) \quad \text{s.t.} \quad |A| = K, \tag{22}$$

where $A \in \mathcal{I}$ together with $|A| = K$ ensures the conditions of Equation (19). Finally, we propose the forward greedy algorithm (see Algorithm 1) based on [23, Algorithm 1] which approximates the solution of the optimization problem defined by Equation (22).

Algorithm 1 Forward Greedy Algorithm over Matroid

Input: set function F_{fg}, ground set P, matroid $\mathcal{M} = (P, \mathcal{I})$, K cardinality constraint

Output: approximately optimal task allocation $A_{\text{fg}} = A^{|K|}$

begin
  initialization: $A^0 = \emptyset$, $U^0 = \emptyset$, $k = 1$ **while** $U^{k-1} \neq P \wedge |A^{k-1}| < K$ **do**
    $a^k \leftarrow \underset{a\in P\setminus U^{k-1}}{\arg\min}\, \rho_{F_{\text{fg}}}(a|A^{k-1})$
    if $A^{k-1} \cup \{a^k\} \notin \mathcal{I}$ **then**
      $U^{k-1} \leftarrow U^{k-1} \cup \{a^k\}$
    else
      $A^k \leftarrow A^{k-1} \cup \{a^k\}$
      $U^k \leftarrow U^{k-1} \cup \{a^k\}$
      $k \leftarrow k + 1$
    end
  end
end

Let us analyse Algorithm 1 step-by-step. We first define $A^k \in \mathcal{I}$ as the task allocation at algorithm step k, and U^k as a set keeping track of task-robot pairs which the algorithm already checked. In Line 2 we initialise with no tasks allocated and no task-robot pairs checked. We iterate the following steps until we run out of possible task-robot pairs or we already allocated all K tasks (Line 2). In each step, we choose the task-robot pair a^k from the available ones $P \setminus U^{k-1}$ which minimises the marginal gain $\rho_{F_{\text{fg}}}(a|A^{k-1})$ (Line 3). If adding a^k does not satisfy the constraints, hence $A^{k-1} \cup \{a^k\} \notin \mathcal{I}$, we add it to U^{k-1} (Line 5), otherwise we add it to the current allocation and to U^{k-1} (Lines 7–9).

5.1.2 Performance guarantee

We propose the following performance guarantee for Algorithm 1 defined by Theorem 1. We provide the proof in Appendix 8.6.

Theorem 1 *Let A_* denote the optimal allocation defined by Equation* (22) *and A_{fg} the forward greedy allocation obtained by Algorithm 1. Then, the following holds*

$$\frac{F_{fg}(A_{fg}) - F_{fg}(\emptyset)}{F_{fg}(A_*) - F_{fg}(\emptyset)} \leq \frac{1}{\gamma \cdot (1-\alpha)},$$

where α and γ are the curvature and submodularity ratio properties of the non-submodular objective function F_{fg} introduced by Equations (5) *and* (2)*, respectively.*

Note that because of the assumption of Equation (16) and the definition of F_{fg} in Equation (21), F_{fg} is strictly and bounded increasing. Combining this with the definitions of the submodularity ratio and curvature (see Equations (2) and (5), respectively), we have $0 < \gamma < 1$ and $0 < \alpha < 1$, hence $1 < \frac{1}{\gamma \cdot (1-\alpha)} < \infty$.

Calculating the values of γ and α for function F_{fg} is challenging. According to Definition 4 and 6, the calculations involve checking every possible combinations of $A \subseteq B \subseteq P$ and $e \in P \setminus B$, which is computationally intractable. To mitigate the issue, [20] uses the *greedy submodularity ratio* γ^G and *greedy curvature* α^G. Both values can be obtained without additional calculations during the execution of the greedy algorithm. However, the guarantee of Theorem 1 does not hold for γ^G and α^G, they can serve as computationally tractable approximations of γ and α, since $\gamma^G \geq \gamma$ and $\alpha^G \leq \alpha$ hold.

5.1.3 Distributed algorithm formulation

We also propose an equivalent distributed version (Algorithm 2) of the forward greedy algorithm (Algorithm 1) which approximates the solution of the task allocation problem defined by Equation (15). We take advantage of the fact, that the step in Line 3 of Algorithm 1 can be calculated in parallel by the robots.

Algorithm 2 Forward Distributed Greedy Algorithm

Input: set of robots R, set of tasks T, set functions $\{f_r\}_{r\in R}$

Output: approximately optimal task allocation $\{T_r^{\text{fg}} = T_r^{|T|}\}_{r\in R}$

1 **begin**

2 initialization: $T_r^0 = \emptyset$, $f_r^0 = f_r(\emptyset)$, $\forall r$, $J^0 = T$, $R^0 = R$ **for** $k = 1, \ldots, K = |T|$ **do**

3 **for** $r \in R^{k-1}$ **do**

4 $t_r^k \leftarrow \arg\min_{t\in J^{k-1}} -\rho_{f_r}(t|T_r^{k-1})$

5 $\delta_r^k \leftarrow -\rho_{f_r}(t_r^k|T_r^{k-1})$

6 **end**

7 $(t_r^k, \delta_r^k) \leftarrow (t_r^{k-1}, \delta_r^{k-1})\ \ \forall r \notin R^{k-1}$

8 $r^k \leftarrow \arg\min_{r\in R} \delta_r^k \cdot \prod_{r'\in R\setminus\{r\}} f_{r'}^{k-1}$

9 $f_r^k \leftarrow \begin{cases} f_r^k - \delta_r^k, \text{ if } r = r^k \\ f_r^{k-1}, \text{ otherwise} \end{cases}$

10 $T_r^k \leftarrow \begin{cases} T_r^k \cup t_r^k, \text{ if } r = r^k \\ T_r^{k-1}, \text{ otherwise} \end{cases}$

11 $R^k \leftarrow \{r \,|\, t_r^k = t_{r^k}^k\}$

12 $J^k \leftarrow J^{k-1} \setminus t_{r^k}^k$

13 **end**

14 **end**

Let us analyse Algorithm 2 step-by-step. We first define the following variables for each algorithm step k: $\{T_r^k\}_{r\in R}$ denotes the current task allocation while $\{f_r^k\}_{r\in R}$ refers to the evaluated function values for each robot r. The evaluation of $f_r^k(T_r^k)$ requires solving the single-robot safe planning problem of Section 4.1, which comes with a significant computational cost. Therefore once we evaluated the function for a specific target allocation, we save it in variable f_r^k. Furthermore, J^k is the set of tasks not yet allocated and R^k is the set of robots which need to update their bids in the next step. We initially assign no tasks to the robots, hence $T_r^0 = \emptyset$ and $f_r^0 = f_r(\emptyset)$ for all $r \in R$ and we evaluate and save values $\{f_r^0\}_{r\in R}$, J^0 and R^0 (see Line 2). Since in each step exactly one task is allocated, we need $K = |T|$ steps to complete the allocation of every task (Line 2). In each iteration k, all robots $r \in R$ submit a bid (see Line 3–7), which consists of the pair (t_r^k, δ_r^k). Each robot r chooses the task t_r^k from the list of unallocated tasks J^{k-1}, which obtains the best optimality gain δ_r^k with respect to the individual objective function of the robot, f_r. After collecting all bids, we choose the robot r^k which generates the best optimality gain with respect to the collective objective, the multiplicative group success F (Line 8). Between Lines 9–12, we simply set the values of f_r^k, T_r^k for all $r \in R$ and R^k, J^k according to our choice of task allocation at step k. Note that only the robots choosing the same task as r^k have to update their bids in the next iteration, hence we define the set R^k in

Line 11 and use it in Line 3. The rest of the robots simply submit their bids from the previous iteration (see Line 7). The variable R^k is initialized with $R^0 = R$, since in the first iteration all robots have to calculate their bids.

5.2 Reverse greedy algorithm

This section introduces the reverse greedy algorithm and provides a performance guarantee. First, we reformulate the allocation problem described by Equation (15) to a set function maximization problem over matroid constraints and propose the reverse greedy algorithm. Then we state the performance guarantee comparing the approximate solution of the greedy algorithm to the optimal solution of the task allocation problem. Since the multiplicative group success (see Equation (14)) is a non-submodular objective function, we use the submodularity ratio (Equation (2)) and curvature (Equation (5)) properties to obtain the performance guarantee. Finally, we provide a distributed formulation of the reverse greedy algorithm where robots make computations in parallel in order to increase calculation speed.

5.2.1 Algorithm formulation

In the following, we reformulate the task allocation problem described by Equation (15) as a set function maximization problem over matroid constraints and define the reverse greedy algorithm. First of all, let the set of tasks be denoted by T and the set of robots by R. In Section 4.2, we described a valid task allocation by partitions $\{T_r\}_{r\in R}$, where $T_r \subset T$ denotes the set of tasks allocated to robot $r \in R$. In order to ensure that every task is allocated to exactly one robot, we introduced the constraints below, also used in Equation (15)

$$T_r \cap T_{r'} = \emptyset, \forall r, r' \in R, r \neq r', \bigcup_{r\in R} T_r = T. \tag{23}$$

Now, we define an alternative yet equivalent description for a valid task allocation. We define $P = T \times R$ as the ground set of all task-robot pairs and $P_t = \{(t, r)\}_{r\in R} \subset P$ as the task-robot pairs related to task $t \in T$. Note that the sets $\{P_t\}_{t\in T}$ define a partitioning of P, since $P_t \cap P_{t'} = \emptyset$ for all pairs $t, t' \in T$ if $t \neq t$ and $\bigcup_{t\in T} P_t = P$. Now $\bar{A} \subset P$ defines a valid task allocation expressed as a set of task-robot tuples to be removed from P, where

$$A = P \setminus \bar{A}, \tag{24}$$

is the task allocation used for the forward greedy algorithm in Section 5.1. Note that every task should be removed from all the robots except for one, hence the following should hold for $\bar{A}$

$$|\bar{A} \cap P_t| = |R| - 1, \forall t \in T, \tag{25}$$

which is equivalent to the constraints described by Equation (23) and Equation (19). Note that we can transform $\{T_r\}_{r\in R}$ to $\bar{A}$ the following way

$$\bar{A} = P \setminus \bigcup_{r\in R} \{(t,r)\}_{t\in T_r}. \tag{26}$$

Let us further define $\bar{K} = |T| \cdot (|R| - 1)$, the set

$$\bar{\mathcal{I}} = \left\{ \bar{A} \subset P \,|\, |\bar{A} \cap P_t| \leq |R| - 1,\, \forall t \in T \right\},$$

the partition matroid $\bar{\mathcal{M}} = (P, \bar{\mathcal{I}})$ (see Definition 8) and the objective function $F_{\text{rg}} : P \to [0,1]$

$$F_{\text{rg}}(\bar{A}) = \prod_{r\in R} f_r(T_r) = -F_{\text{fg}}(P \setminus \bar{A}), \tag{27}$$

where the relationship between $\bar{A}$ and $\{T_r\}_{r\in R}$ is defined by Equation (26) and between $\bar{A}$ and A by Equation (24). Note that $\bar{\mathcal{M}}$ is the dual of matroid $\mathcal{M}$, see Definition 10. Now Equation (15) and Equation (22) can be reformulated the following way

$$\bar{A}_* = \arg\max_{\bar{A}\in\bar{\mathcal{I}}} F_{\text{rg}}(\bar{A}) \quad \text{s.t.} \quad |\bar{A}| = \bar{K}, \tag{28}$$

where $\bar{A} \in \bar{\mathcal{I}}$ together with $|\bar{A}| = \bar{K}$ ensures the conditions of Equation (25). Finally, we propose the reverse greedy algorithm (see Algorithm 3) based on [23, Algorithm 2] which approximates the solution of the optimization problem defined by Equation (28).

Algorithm 3 Reverse Greedy Algorithm over Matroid

Input: set function F_{rg}, ground set P, matroid $\bar{\mathcal{M}} = (P, \bar{\mathcal{I}})$, $\bar{K}$ cardinality constraint

Output: approxiamtely optimal exclusion set $\bar{A}_{\text{rg}} = \bar{A}^{|\bar{K}|}$

```
begin
    initialization Ā^0 = ∅, U^0 = ∅, k = 1 while U^{k-1} ≠ P ∧ |Ā^{k-1}| < K̄ do
        ā^k ← arg max_{ā∈P\U^{k-1}} ρ_{F_rg}(ā|Ā^{k-1})
        if Ā^{k-1} ∪ {ā^k} ∉ Ī then
            U^{k-1} ← U^{k-1} ∪ {ā^k}
        else
            Ā^k ← Ā^{k-1} ∪ {ā^k}
            U^k ← U^{k-1} ∪ {ā^k}
            k ← k + 1
        end
    end
end
```

Let us analyse Algorithm 3 step-by-step. We first define $\bar{A}^k \in \bar{\mathcal{I}}$ as the task allocation at algorithm step k, and U^k as a set keeping track of task-robot pairs which the algorithm already checked. In Line 2 we initialise with no tasks removed, hence every task allocated to every robot simultaneously and no task-robot pairs checked. We iterate the following steps until we run out of possible task-robot pairs or we already removed every task from all the robots except for one, hence we removed $\bar{K} = |T| \cdot (|R| - 1)$ task-robot pair (Line 2). In each step, we choose the task-robot pair $\bar{a}^k$ from the available ones $P \setminus U^{k-1}$ which maximises the marginal gain $\rho_{F_{\text{rg}}}(\bar{a}|\bar{A}^{k-1})$ (Line 3). If adding $\bar{a}^k$ does not satisfy the constraints, hence $\bar{A}^{k-1} \cup \{\bar{a}^k\} \notin \bar{\mathcal{I}}$, we add it to U^{k-1} (Line 5), otherwise we add it to the current allocation and to U^{k-1} (Lines 7–9).

5.2.2 Performance guarantee

We propose the following performance guarantee for Algorithm 3 defined by Theorem 2. We provide the proof in Appendix 8.7.

Theorem 2 *Let $\bar{A}_*$ denote the optimal allocation defined by Equation* (28) *and $\bar{A}_{rg}$ the reverse greedy allocation obtained by Algorithm 3. Then, the following holds*

$$\frac{\bar{\gamma}}{1 + \bar{\gamma} \cdot \bar{\alpha}} \leq \frac{F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)}{F_{rg}(\bar{A}_*) - F_{rg}(\emptyset)},$$

where $\bar{\alpha}$ and $\bar{\gamma}$ are the curvature and submodularity ratio properties of the non-submodular objective function F_{rg} introduced by Equations (5) *and* (2)*, respectively.*

Note that because of the assumption of Equation (16) and the definition of F_{rg} in Equation (27), F_{rg} is strictly and bounded decreasing. Combining this with the definitions of the submodularity ratio and curvature (see Equations (2) and (5), respectively), we have $0 < \bar{\gamma} < 1$ and $0 < \bar{\alpha} < 1$, hence $0 < \frac{\bar{\gamma}}{1+\bar{\gamma}\cdot\bar{\alpha}} < 1$.

Calculating the values of $\bar{\gamma}$ and $\bar{\alpha}$ for function F_{rg} is challenging. According to Definition 4 and 6, the calculations involve checking every possible combinations of $A \subseteq B \subseteq P$ and $e \in P \setminus B$, which is computationally intractable. To mitigate the issue, [20] uses the *greedy submodularity ratio* $\bar{\gamma}^G$ and *greedy curvature* $\bar{\alpha}^G$. Both values can be obtained without additional calculations during the execution of the greedy algorithm. However, the guarantee of Theorem 2 does not hold for $\bar{\gamma}^G$ and $\bar{\alpha}^G$, they can serve as computationally tractable approximations of $\bar{\gamma}$ and $\bar{\alpha}$, since $\bar{\gamma}^G \geq \bar{\gamma}$ and $\bar{\alpha}^G \leq \bar{\alpha}$ hold.

5.2.3 Distributed algorithm formulation

We also propose an equivalent distributed version (Algorithm 4) of the reverse greedy algorithm (Algorithm 3) which approximates the solution of the task allocation problem defined by Equation (15). We take advantage of the fact, that the step in Line 3 of Algorithm 3 can be calculated in parallel by the robots.

Algorithm 4 Reverse Distributed Greedy Algorithm

Input: set of robots R, set of tasks T, set functions $\{f_r\}_{r\in R}$

Output: approximately optimal task allocation $\{T_r^{\mathrm{rg}} = T_r^{|T|\cdot(|R|-1)}\}_{r\in R}$

1 **begin**

2 initialization: $T_r^0 = T$, $f_r^0 = f_r(T)$, $\forall r$, $J^0 = T$, $R^0 = R$ **for** $k = 1, \ldots, \bar{K} = |T| \cdot (|R| - 1)$ **do**

3 **for** $r \in R^{k-1}$ **do**

4 $t_r^k \leftarrow \underset{t \in J^{k-1} \cap T_r^{k-1}}{\arg\max} f_r(T_r^{k-1} \setminus t) - f_r(T_r^{k-1})$

5 $\delta_r^k \leftarrow f_r(T_r^{k-1} \setminus t_r^k) - f_r(T_r^{k-1})$

6 **end**

7 $(t_r^k, \delta_r^k) \leftarrow (t_r^{k-1}, \delta_r^{k-1}) \;\; \forall r \notin R^{k-1}$

8 $r^k \leftarrow \arg\max_{r\in R} \delta_r^k \cdot \prod_{r' \in R\setminus\{r\}} f_{r'}^{k-1}$

9 $f_r^k \leftarrow \begin{cases} f_r^k + \delta_r^k, \text{ if } r = r^k \\ f_r^{k-1}, \text{ otherwise} \end{cases}$

10 $T_r^k \leftarrow \begin{cases} T_r^k \setminus t_r^k, \text{ if } r = r^k \\ T_r^{k-1}, \text{ otherwise} \end{cases}$

11 $R^k \leftarrow \begin{cases} \left\{r \,|\, t_r^k = t_{r^k}^k\right\}, \text{ if } \left|\left\{r \,|\, t_{r^k}^k \in T_r^k\right\}\right| = 1 \\ \{r^k\}, \text{ otherwise} \end{cases}$

12 $J^k \leftarrow \begin{cases} J^{k-1} \setminus t_{r^k}^k, \text{ if } \left|\left\{r \,|\, t_{r^k}^k \in T_r^k\right\}\right| = 1 \\ J^{k-1}, \text{ otherwise} \end{cases}$

13 **end**

14 **end**

Let us analyse Algorithm 4 step-by-step. We first define the following variables for each algorithm step k: $\{T_r^k\}_{r\in R}$ denotes the current task allocation while $\{f_r^k\}_{r\in R}$ refers to the evaluated function values for each robot r. The evaluation of $f_r^k(T_r^k)$ requires solving the single-robot safe planning problem of Section 4.1, which comes with a significant computational cost. Therefore once we evaluated the function for a specific target allocation, we save it in variable f_r^k. Furthermore, J^k is the set of tasks not yet removed and R^k is the set of robots which need to update their bids in the next step. We initially assign all tasks to every robot simultaneously, hence $T_r^0 = T$ and $f_r^0 = f_r(T)$ for all $r \in R$ and we evaluate and save values $\{f_r^0\}_{r\in R}$, J^0 and R^0 (see Line 2). Since in each step exactly one task is removed, we need $\bar{K} = |T| \cdot (|R| - 1)$ steps (Line 2). In each iteration k, all robots $r \in R$ submit a bid (see Line 3–7), which consists of the pair (t_r^k, δ_r^k). Each robot r chooses the task t_r^k from the list of unremoved tasks J^{k-1}, which obtains the best optimality gain δ_r^k with respect to the individual objective function of the robot f_r. After collecting all bids, we choose the robot r^k which generates the best optimality gain

with respect to the collective objective, the multiplicative group success F (Line 8). Between Lines 9–12, we simply set the values of f_r^k, T_r^k for all $r \in R$ and R^k, J^k according to our choice of task allocation at step k. Note that a task $t_{r^k}^k$ is removed from J^k if it is only allocated to single robot, hence $|\{r|t_{r^k}^k \in T_r^k\}| = 1$. Also note that only the robots choosing the same task as r^k have to update their bids in the next iteration and only when $t_{r^k}^k$ just got removed from J^k, hence we define the set R^k in Line 11 and use it in Line 3. The rest of the robots simply submit their bids from the previous iteration (see Line 7). The variable R^k is initialized with $R^0 = R$, since in the first iteration all robots have to calculate their bids.

5.3 Comparison of performance guarantees for forward and reverse greedy approaches

We compare the forward and reverse greedy algorithms in terms of their performance guarantees. First, we build on previously derived relations between the two greedy formulations. Then we obtain equations suitable for comparison of the performance guarantees. Finally, we compare the algorithms in terms of their performance guarantees.

According to Equation (24), if $A = P \setminus \bar{A}$, then A and $\bar{A}$ describe the same task allocation. In this case, we can also write $F_{\text{rg}}(\bar{A}) = -F_{\text{fg}}(A)$ (Equation (27)). Furthermore, since the problems described by Equation (22) and (28) are equivalent, we have $A_* = P \setminus \bar{A}_*$ and $F_{\text{rg}}(\bar{A}_*) = -F_{\text{fg}}(A_*)$. Now we define the following variables

$$F^* = F_{\text{rg}}(\bar{A}_*) = -F_{\text{fg}}(A_*), \tag{29}$$

$$F^{\text{fg}} = -F_{\text{fg}}(A_{\text{fg}}), \tag{30}$$

$$F^{\text{rg}} = F_{\text{rg}}(\bar{A}_{\text{rg}}), \tag{31}$$

where F^*,F^{fg} and F^{rg} denote the probability of success obtained by the optimal, the forward greedy and the reverse greedy task allocations, respectively. Furthermore, we describe the relationship between the submodularity ratio γ, $\bar{\gamma}$ and curvature α, $\bar{\alpha}$ values of the functions F_{fg} and F_{rg}, respectively (see Theorem 1 and 2). According to [23, Proposition 2], we can write

$$\bar{\gamma} = 1 - \alpha, \tag{32}$$

$$\bar{\alpha} = 1 - \gamma. \tag{33}$$

Now based on Equations (24), (27), (29)–(31) and (32)–(33) we can rearrange the performance guarantees in Theorem 1 and 2 the following way

$$\frac{-F^{\text{fg}} - F_{\text{fg}}(\emptyset)}{-F^* - F_{\text{fg}}(\emptyset)} \leq \frac{1}{\gamma \cdot (1 - \alpha)}, \tag{34}$$

$$\frac{1 - \alpha}{1 + (1 - \alpha) \cdot (1 - \gamma)} \leq \frac{F^{\text{rg}} - F_{\text{rg}}(\emptyset)}{F^* - F_{\text{rg}}(\emptyset)}. \tag{35}$$

At this point we make assumptions about $F_{\text{fg}}(\emptyset)$ and $F_{\text{rg}}(\emptyset)$. In the former case $A = \emptyset$, no tasks are allocated to the robots. This means that each robot has a high probability of success

due to the lack of tasks, hence we can assume $(-F_{\text{fg}}(\emptyset)) \approx 1$. In the latter case $\bar{A} = \emptyset$ and $A = P \setminus \bar{A} = P$, all tasks are allocated to all robots. This means that each robot has a low probability of success due to being overwhelmed with tasks, hence we can assume $F_{\text{rg}}(\emptyset) \approx 0$. Using these assumptions we can rearrange Equation (34) and (35) the following way

$$F^* \cdot \frac{1}{\gamma \cdot (1-\alpha)} + \frac{\gamma \cdot (1-\alpha) - 1}{\gamma \cdot (1-\alpha)} = g^{\text{fg}}(\alpha, \gamma, F^*) \leq F^{\text{fg}}, \tag{36}$$

$$F^* \cdot \frac{1-\alpha}{1 + (1-\alpha) \cdot (1-\gamma)} = g^{\text{rg}}(\alpha, \gamma, F^*) \leq F^{\text{rg}}. \tag{37}$$

In Equation (36) and (37) the expressions $g^{\text{fg}}(\alpha, \gamma, F^*)$ and $g^{\text{rg}}(\alpha, \gamma, F^*)$ provide lower bounds on the probability of success obtained by the forward and reverse greedy solutions (F^{fg} and F^{rg}). The lower bounds are expressed as a function of the α,γ properties and the probability of success obtained by the optimal task allocation F^*.

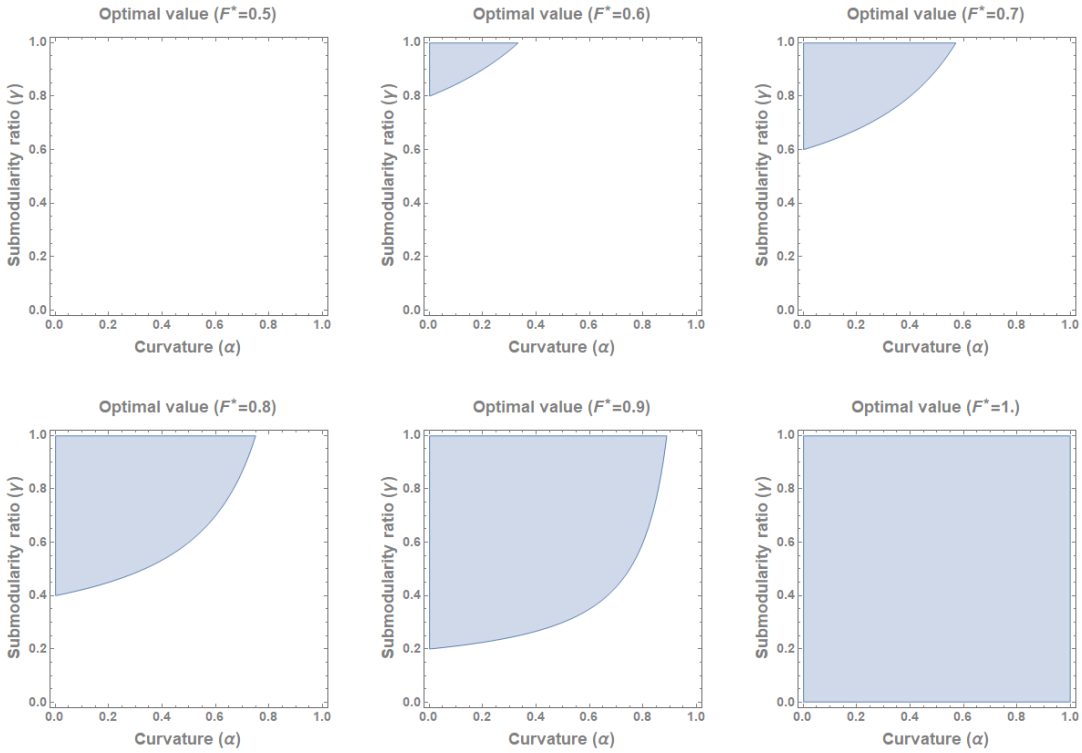


Figure 3: Comparison of forward and reverse greedy algorithms in terms of performance guarantees. For each optimal F^* value, the shaded region represents the (α, γ) pairs for which the forward greedy outperforms the reverse greedy. For $F^* \leq 0.5$, the reverse greedy outperforms the forward greedy for any (α, γ) pair.

In Figure 3, we shaded the (α, γ) pairs defined by $\{(\alpha, \gamma) \in [0,1] \times [0,1] \,|\, g^{\text{fg}}(\alpha, \gamma, F^*) \geq g^{\text{rg}}(\alpha, \gamma, F^*)\}$ for specific values of F^*. The shaded area for a certain F^* value represents the (α, γ) pairs, where the forward greedy algorithm has a higher performance guarantee than the reverse greedy, hence it is more reliable. If F^* is approximately 1, using the forward greedy algorithm is a better choice in terms of reliability regardless of the values of α and γ. However, if F^* is below 0.5, the reverse greedy approach enjoys better performance guarantees for all

(α, γ) pairs. By decreasing the value of F^* from 1 to 0.5, the area where the forward greedy algorithm is more reliable, shrinks. According to the figure, the performance guarantee of the reverse greedy approach becomes higher for combinations with high α and low γ values.

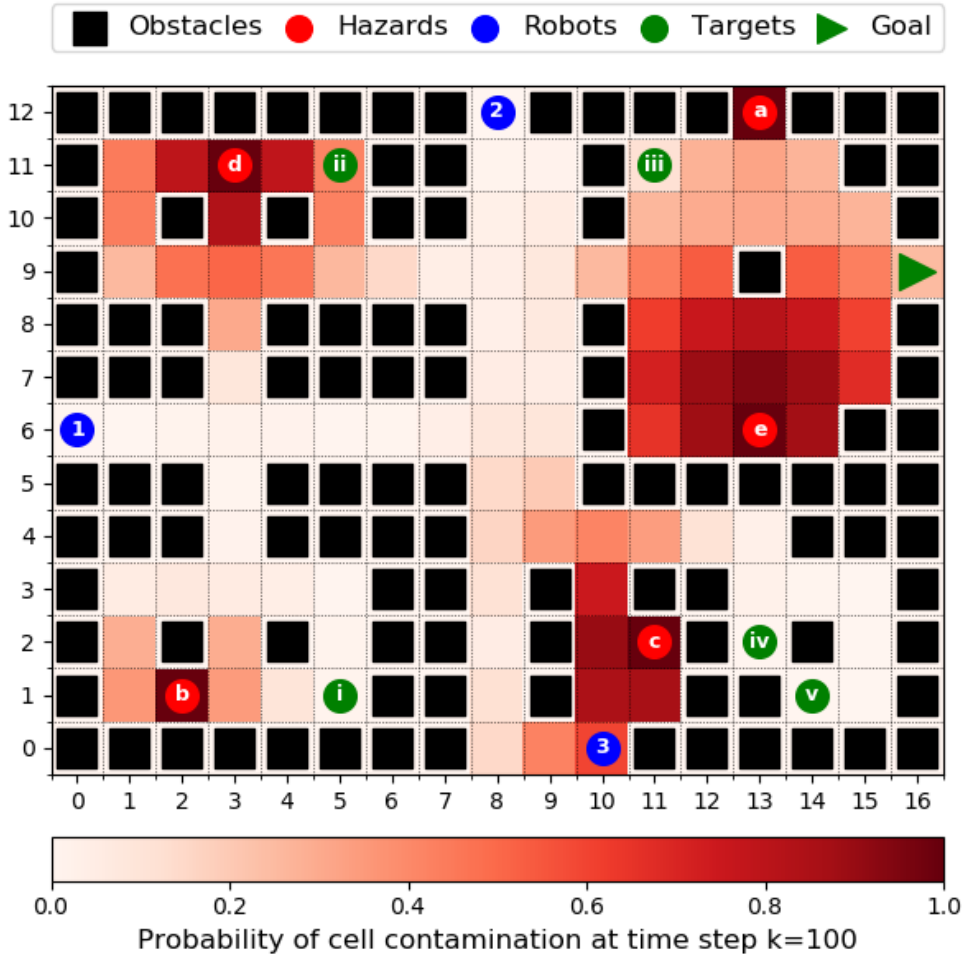
6 Numerical case studies

We illustrate our findings by presenting three case studies. In the first one, shown in Section 6.1, we solve an example via our two-stage multi-robot safe planning framework under different task allocation methods. We then compare the performance of these methods both in their success probabilities and computation time. We provide the second example in Section 6.2 to illustrate the rare case when using the multiplicative group success (see Equation (14)) as the objective function is inaccurate. Finally, the example shown in Section 6.3 highlights the computational benefits of the two-stage planning approach over the full-fleet safe planning framework.

6.1 Performance comparison of task allocation algorithms

We introduce an example solution of the two-stage multi-robot safe planning framework defined in Section 4. For implementing the high-level task allocation stage of the framework (see Section 4.2) we used the approximate forward and reverse greedy approaches of Section 5. We designed the example to be small enough in size so that we can calculate the brute force optimal allocation as well for comparison. The brute force solution is obtained by check each valid task allocation separately. This would not be possible for larger examples. We considered the 17-by-13 sized grid-shaped map and the initial state shown in Figure 4. Three robots enter the environment from different entries, they visit the five target positions while avoiding the evolving hazard and leave safely through the goal location within the given time horizon. We set the time horizon to $N = 100$ steps, which is long enough for each robot to be able to finish. For this example, we chose the robots to behave the same way according to the deterministic robot dynamics defined by Equation (6). The only difference between them is their initial positions. We planted five different hazard sources defined by their initial positions, parameters of their spreading speed θ (see the table attached to Figure 4) and evolution dynamics described by Appendix 8.8. To help visualize the nature of the hazard spread, we also added a heat map showing the probability of a cell getting contaminated within the given time horizon. The higher this probability is for a cell, the more dangerous the cell is for the robots.

Let us now analyze the results. We compared the solutions and computational performance of the three different task allocation methods in Table 1. The forward and reverse greedy algorithms are approximating the solution of the brute force allocation. In this case, both greedy approaches have found the optimal allocation, hence all three solutions match. However, the measured computation time results reveal the benefits of the greedy approximations. Both greedy algorithms outmatched the brute force solution in terms of computational complexity by orders of magnitude. It is clear, that for larger examples it would be intractable to use brute force. The greedy algorithms provide a tractable approximation close to optimality. Note that the reverse greedy algorithm takes significantly more time to converge, then the forward greedy solution. It can be explained with two arguments. By looking at Algorithm 2, we see, that the forward greedy approach takes $|T|$ steps, while the reverse greedy version (Algorithm 4)



Hazard source spread speed parameters		
$\theta_a = 0.002$	$\theta_b = 0.004$	$\theta_c = \theta_d = \theta_e = 0.012$

Figure 4: Example environment. The robots have to reach the goal position after visiting the targets while avoiding the evolving hazards. The hazard spread is illustrated by a heat map showing the probability of each cell getting contaminated within 100 time steps. The spread speed parameters of the hazard sources are shown in the table below the map.

needs $|T| \cdot (|R| - 1)$ steps, which is significantly larger. The computation time of evaluating the solution of the single-robot path planning problem (see Section 4.1) for a subset of tasks $T_r \in T$ depend greatly on the number of targets $|T_r|$ allocated to a robot. Section 4.1.4 explains, that $|Q| = 2^{|T_r|}$ grows exponentially with $|T_r|$, which has an effect on computational complexity and computation times. The forward greedy method starts with a smaller number of tasks allocated to the robots and possibly never evaluates the single-robot path planner with nearly all tasks assigned to a single agent. However, the reverse greedy approach starts with all tasks being allocated to every agent and improves by removing them. This requires the evaluation of the single-agent path planning framework multiple times with a high number of targets per agent, which explains the significantly increased computation times. We also examined what would happen if we allocated every task to a single agent, robot 3. The probability of success in this case would be 0.21. This shows the significance of using multiple agents since for this example,

Table 1: Comparison of task allocation algorithms. The meaning of each column: *Task allocation* – optimal allocation provided by the corresponding algorithm, *Computation time* – full algorithm run time [4], *Success probability* – probability of success under the optimal allocation provided by the corresponding algorithm.

Algorithm	Task allocation		Computation time	Success probability
Forward Greedy	Robot	Tasks	8 minutes 42 seconds	0.702
	1	{i}		
	2	{ii, iii}		
	3	{iv, v}		
Reverse Greedy	Robot	Tasks	35 minutes 52 seconds	0.702
	1	{i}		
	2	{ii, iii}		
	3	{iv, v}		
Brute Force	Robot	Tasks	6 hours 15 minutes 41 seconds	0.702
	1	{i}		
	2	{ii, iii}		
	3	{iv, v}		

it can increase the probability of success by almost 50%. It also reveals that a proper task allocation method is necessary and tasks should not be allocated arbitrarily or randomly.

We now analyze the common optimal allocation and robot paths shown in Figure 5 using our intuition. Since we used the deterministic robot dynamics for the example, we can show the optimal agent paths assuming a scenario where neither of the robots got contaminated. 'Task i' is taken by 'robot 1' due to its accessibility for the agent on its way to the exit. 'Task ii' could be taken by both 'robot 1' or 'robot 2' based on their distance. According to Figure 4, 'hazard d' has a high spread speed parameter, 0.012, hence 'task ii' is in great danger, it has to be urgently saved. We can also confirm this by looking at the heat map of Figure 5 representing the probability of each cell getting contaminated in the given time window. 'Robot 2' has the highest chance for saving 'task ii', since this agent is located to closest to its position initially. Note that visiting this task means greater danger for 'robot 2', since it has to take a longer route going in the opposite of the exit, but this decision is beneficial for the group objective. 'Task iii' is also taken by 'robot 2', since it arrives to this location the first among all three robots. 'Tasks iv' and 'task v' are allocated to 'robot 3' due to their close locations. By looking at the chosen paths of the individual robots, there are two interesting details. Each robot has to go through the room in the top right corner of the map before leaving the map through the goal location. They all have to navigate between hazard a and e and passing by an obstacle in the middle (location (13,9)). The choice of direction is not arbitrary. Since 'hazard e' spreads faster, they prefer to go closer to 'hazard a'. Also notice, that 'robot 3' chooses a slightly longer path through position (8,2) instead of directly passing next to 'hazard c' through (10,2) which would be more dangerous. The choices of tasks and optimal paths seem intuitively reasonable for all

[4]Measured on a computer equipped with Core i7 CPU running at 2.6GHz, with 8GB of RAM.

robots. It also shows the group and individual objective maximizing decisions made during the task allocation and path planning stages, respectively.

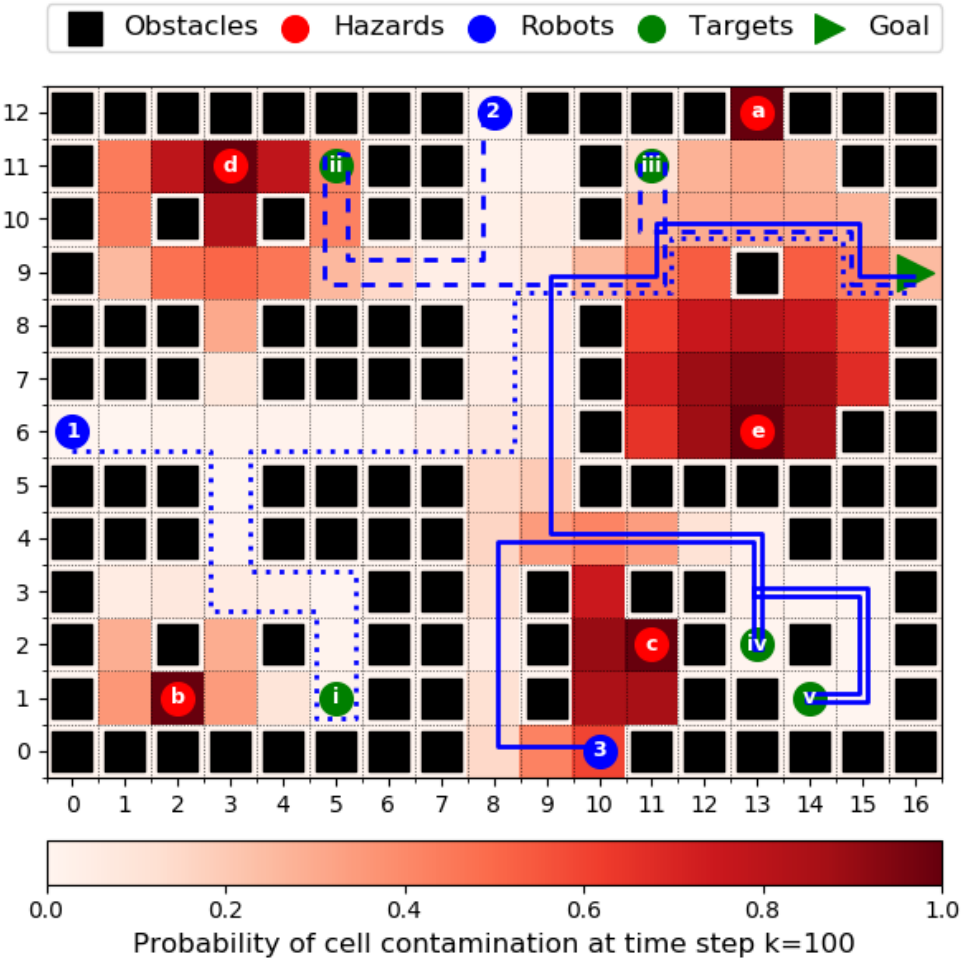


Figure 5: Optimal robot paths and task allocation obtained by the algorithms.

6.2 Counterexample for the usage of the multiplicative group success

In Section 4.3 we highlighted the computational advantages behind the usage of the multiplicative group success $F(\{T_r\}_{r\in R})$ (see Equation (14)) in comparison to the probability of group success $F_M(T)$ (see Appendix 8.4). We also mentioned that the multiplicative group success assumes independence between individual agents succeeding. In this example, we introduce a counterexample when this independence does not hold and the multiplicative group success is a misleading objective. We also show how under some mild conditions introduced in Appendix 8.5, the multiplicative group success can be used as a lower bound of the probability of group success, hence a computationally tractable approximation.

The example in Figure 6 shows two robots both of which can choose independently between two different paths in order to reach the goal position: 'path 1' or 'path 2'. We removed the tasks from this example, the mission of the fleet is simply to reach the exit safely. We consider deterministic robot dynamics for both agents (see Equation (6)). Let the evolution

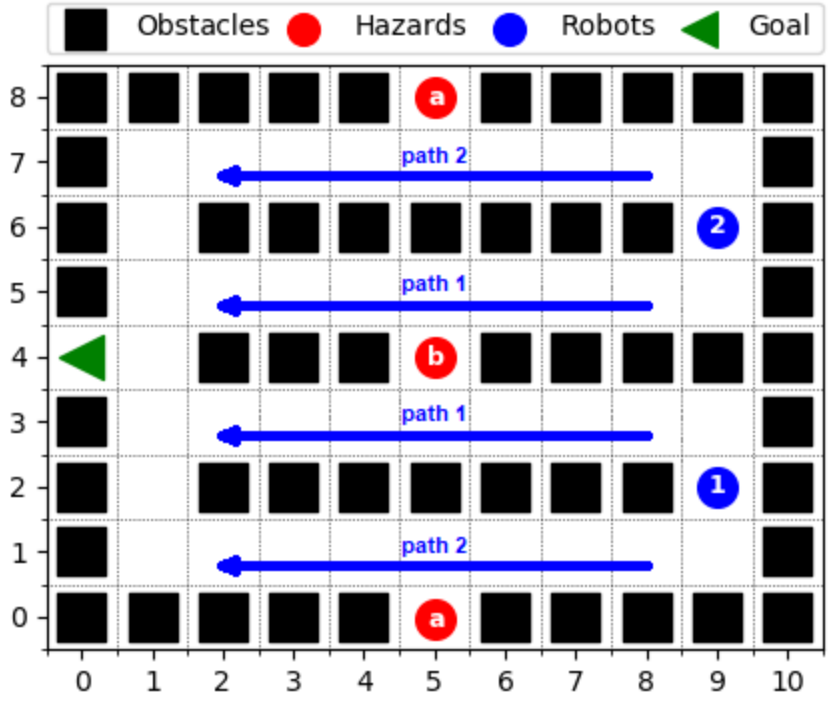


Figure 6: Counterexample for the usage of the multiplicative group success.

$\tau^k_{Y,b} : Y \times Y \to [0,1]$ of 'hazard b' at position $(5,4)$ be defined the following way: for $k=0$ let

$$\tau^0_{Y,b}(y^1|y^0) = \begin{cases} 0.5 & \text{if } y^0 = \{(5,4)\}, \\ & \wedge\ y^1 = \{(5,4),(5,5)\}, \\ 0.5 & \text{if } y^0 = \{(5,4)\}, \\ & \wedge\ y^1 = \{(5,4),(5,3)\}, \\ 0 & \text{otherwise,} \end{cases}$$

while for all $k \neq 0$ let $\tau^k_{Y,b}(y^{k+1}|y^k) = \mathbb{1}_{y^k}(y^{k+1})$. This way at step 1 either position $(5,5)$ or $(5,3)$ is contaminated with equal 0.5 probability and stay that way throughout the whole process. Let the evolution $\tau^k_{Y,a} : Y \times Y \to [0,1]$ of 'hazard a' at positions $(5,0)$ and $(5,8)$ be defined in a similar fashion: for $k=0$ let

$$\tau^0_{Y,a}(y^1|y^0) = \begin{cases} 0.7 & \text{if } y^0 = \{(5,0),(5,8)\}, \\ & \wedge\ y^1 = \{(5,0),(5,8),(5,1),(5,7)\}, \\ 0.3 & \text{if } y^0 = \{(5,0),(5,8)\}, \\ & \wedge\ y^1 = \{(5,0),(5,8)\}, \\ 0 & \text{otherwise,} \end{cases}$$

while for all $k \neq 0$ let $\tau^k_{Y,a}(y^{k+1}|y^k) = \mathbb{1}_{y^k}(y^{k+1})$. This way at step 1 either both $(5,1)$ and $(5,7)$ get contaminated with probability 0.7 or the hazard stops spreading with 0.3 chance for the rest

of the process. 'Hazard a' and 'hazard b' are considered to spread independently.

Now we compare the solution of the two approaches. First, let us analyze the two-stage multi-robot framework solution. Both robots decide their paths individually, they either choose 'path 1' passing next to 'hazard b', which provides a safe passage with probability $1-0.5=0.5$, or 'path 2' close to 'hazard a', where there is a $1-0.7=0.3$ chance of succeeding. This way both robots choose 'path 1', and the multiplicative objective is $F=0.5\cdot 0.5=0.25$. However, since 'hazard b' contaminated at least one of the robots due to the design of the example, the chance of success for both robots simultaneously is actually 0. The difference can be explained by the fact that the multiplicative group success handles the success of agents as independent random variables and neglects the existing correlation between them. In this case, the multiplicative group success is overly optimistic about the outcome. The full-fleet safe planning framework, however, considers path planning for both agents simultaneously. This way the choices are the following: sending both robots on 'path 1' – the probability of group success is 0, sending one of the robots on 'path 1' the other on 'path 2' – the probability of group success is $(1-0.5)\cdot(1-0.7)=0.5\cdot 0.3=0.15$, or sending both robots on 'path 2' – the probability of group success is $1-0.7=0.3$. The algorithm chooses the third option, which has a 0.3 group success probability in comparison to the 0 gained from using the multiplicative objective. This example clearly shows how misleading the multiplicative group success can be. Note that in this scenario if 'robot 1' succeeds, the probability of 'robot 2' also succeeding decreases to 0 and vice versa. Hence, the conditions of Appendix 8.5 do not hold.

Let us change the behavior of 'hazard b' the following way so that the conditions of Appendix 8.5 hold: for $k=0$ let

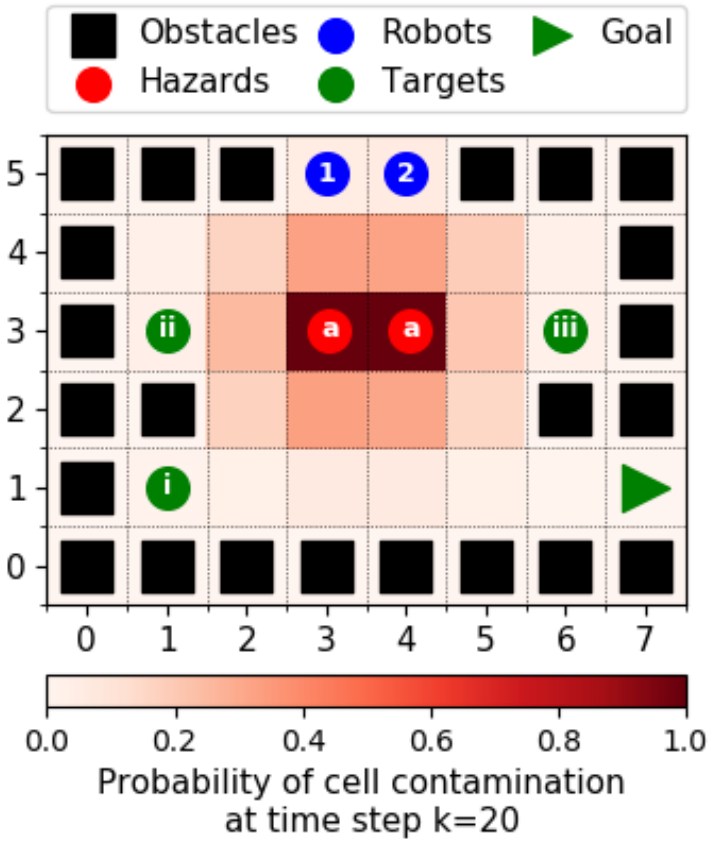
$$\tau^0_{Y,b}(y^1|y^0)=\begin{cases}0.5 & \text{if } y^0=\{(5,4)\},\\ & \wedge\ y^1=\{(5,4),(5,3),(5,5)\},\\ 0.5 & \text{if } y^0=\{(5,4)\},\\ & \wedge\ y^1=\{(5,4)\},\\ 0 & \text{otherwise,}\end{cases}$$

while for all $k\neq 0$ let $\tau^k_{Y,b}(y^{k+1}|y^k)=\mathbb{1}_{y^k}(y^{k+1})$. This way in step 1 either both $(5,3)$ and $(5,5)$ get contaminated simultaneously with 0.5 chance, or none of them become hazardous with probability 0.5 for the rest of the process. In this case, the individually generated paths for robots do not change, the agents both choose to go on 'path 1', and the multiplicative group success stays $0.5\cdot 0.5=0.25$. However, the probability of both robot succeeding simultaneously is now 0.5. The available choices for the full-fleet safe planning framework also changed the following way: sending both robots on 'path 1' – the probability of group success is 0.5, sending one of the robots on 'path 1' the other on 'path 2' – the probability of group success is $(1-0.5)\cdot(1-0.7)=0.5\cdot 0.3=0.15$, or sending both robots on 'path 2' – the probability of group success is $1-0.7=0.3$. In this case, the algorithm choices the first option, which has a 0.5 group success probability. The multiplicative group success became overly pessimistic when changing

the example for the conditions of Appendix 8.5 to hold. Note that the chosen paths for both approaches now match.

6.3 Performance comparison of full-fleet safe planning and two-stage multi-robot safe planning

In this section, we highlight the computational benefits of the two-stage multi-robot safe planning framework over the full-fleet safe planning approach through an example. Let us consider the environment shown in Figure 7, where two robots need to visit three targets before leaving without getting contaminated. Similarly to the example shows in Section 6.1, the agents differ in their initial positions only and we use the deterministic robot dynamics (see Equation (6)). We also use the hazard model defined in Appendix 8.8. The time horizon is set to $N = 20$ now due to the small size of the map.



Hazard source spread speed parameter
$\theta_a = 0.001$

Figure 7: Example environment. The robots have to reach the goal position after visiting the targets while avoiding the evolving hazards. The hazard spread is illustrated by a heat map showing the probability of each cell getting contaminated within 20 time steps. The spread speed parameter of the hazard source is shown in the table below the map.

In order to solve the proposed problem, we used four different algorithms: forward greedy, reverse greedy, brute force and full-fleet safe planning framework. Similarly to the case study shown in Section 6.1, the first three algorithms are based on the two-stage multi-robot framework. The difference between them is the implementation of the task allocations stage. For this example, all four algorithms generated the same task allocation, optimal paths and success probabilities as shown in Table 2 and Figure 8. However, their performance in terms of computation time differs greatly. As expected from the example presented in Section 6.1, the forward greedy is the fastest algorithm, followed by the reverse greedy and the brute force approaches, respectively. The full-fleet safe planning algorithm performed the worst in terms of computation time by orders of magnitude. This phenomenon can be explained by the fact that the state space used for this formulation is much larger than in the case of all three other algorithms (see Section 4.3). This example shows how computationally inefficient it is to use the full-fleet safe planning framework without any significant optimality gain. For larger problems, this approach becomes intractable leaving the two-stage multi-robot safe planning framework as the only option for generating the solution.

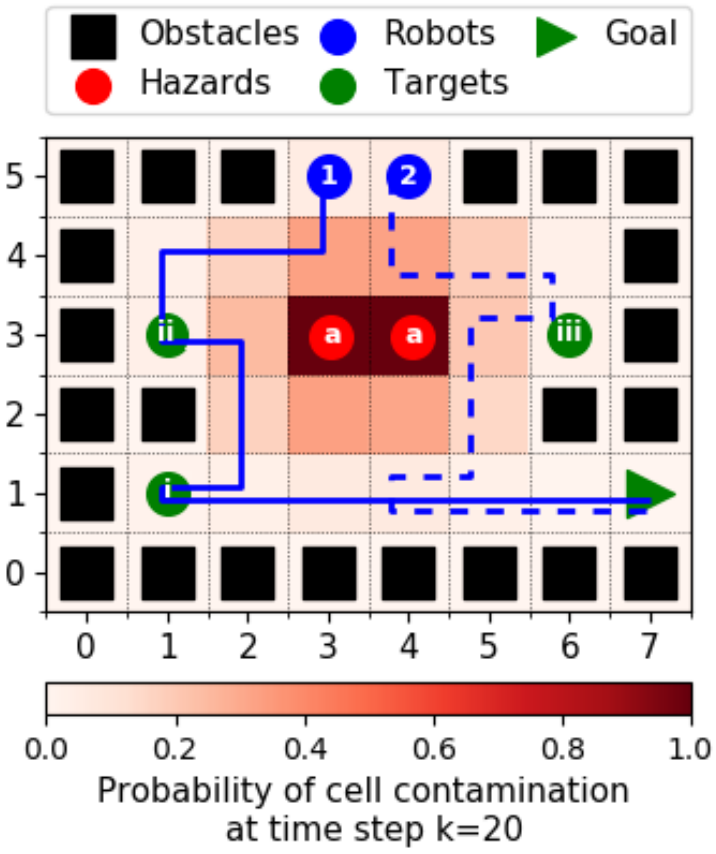


Figure 8: Optimal robot paths and task allocation obtained by the algorithms.

Table 2: Comparison of task allocation algorithms. The meaning of each column: *Task allocation* – optimal allocation provided by the corresponding algorithm, *Computation time* – full algorithm run time [5], *Success probability* – probability of success under the optimal allocation provided by the corresponding algorithm.

Algorithm	Task allocation		Computation time	Success probability
Forward Greedy	Robot	Tasks	0.45 seconds	0.626
	1	$\{i, ii\}$		
	2	$\{iii\}$		
Reverse Greedy	Robot	Tasks	0.66 seconds	0.626
	1	$\{i, ii\}$		
	2	$\{iii\}$		
Brute Force	Robot	Tasks	1.11 seconds	0.626
	1	$\{i, ii\}$		
	2	$\{iii\}$		
Multi Robot Dynamic Programming	Robot	Tasks	13 minutes 54 seconds	0.626
	1	$\{i, ii\}$		
	2	$\{iii\}$		

[5]Measured on a computer equipped with Core i7 CPU running at 2.6GHz, with 8GB of RAM.

7 Conclusions

In this thesis, we proposed a two-stage framework for solving a multi-robot safe planning problem in a computationally tractable way. We built on previous research in safe planning and extended the existing approaches to a multi-agent framework to increase system performance. We mitigated the issue of the increased computational complexity of using multiple agents by splitting the problem into a low-level single-robot path planning and a high-level multi-robot task allocation problem. We used greedy approximation algorithms to solve the NP-hard task allocation problem and provided performance guarantees. We also verified our framework by introducing case studies. We made comparisons between the approximate solution of our framework and the optimal one. We found that our algorithms perform well both in terms of computation time and optimality in the examined examples.

References

[1] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *Int. J. of Adv. Rob. Syst.*, vol. 10, no. 12, p. 399, 2013.

[2] P. Stone and M. Veloso, "Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork," *Artificial Intelligence*, vol. 110, no. 2, pp. 241–273, 1999.

[3] H. Fazlollahtabar and M. Saidi-Mehrabad, "Methodologies to optimize automated guided vehicle scheduling and routing problems: a review study," *Journal of Intelligent & Robotic Systems*, vol. 77, no. 3, pp. 525–545, 2015.

[4] 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 *IROS*. IEEE, 2017, pp. 5622–5629.

[5] T. Li, H.-S. Shin, and A. Tsourdos, "Threshold greedy based task allocation for multiple robot operations," *arXiv preprint arXiv:1909.01239*, 2019.

[6] X. Sun, C. G. Cassandras, and X. Meng, "Exploiting submodularity to quantify near-optimality in multi-agent coverage problems," *Aut.*, vol. 100, pp. 349–359, 2019.

[7] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *ASME Dyn. Syst. and Contr. Conf.*, 2017.

[8] Z. Zhu, E. Bıyık, and D. Sadigh, "Multi-agent safe planning with gaussian processes," *arXiv preprint arXiv:2008.04452*, 2020.

[9] T. A. Wood and M. Kamgarpour, "Automaton-based stochastic control for navigation of emergency rescuers in buildings," in *IEEE CCA*. IEEE, 2016, pp. 587–592.

[10] B. Luders, M. Kothari, and J. How, "Chance constrained rrt for probabilistic robustness to environmental uncertainty," in *AIAA GNCC*, 2010, p. 8160.

[11] B. D. Luders, G. S. Aoude, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe avoidance of dynamic obstacles with uncertain motion patterns," Tech. Rep., 2011.

[12] S. Brechtel, T. Gindele, and R. Dillmann, "Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps," in *17th ITSC*. IEEE, 2014, pp. 392–399.

[13] Y. Lu and M. Kamgarpour, "Safe mission planning under dynamical uncertainties," in *ICRA*. IEEE, 2020, pp. 2209–2215.

[14] D. P. Bertsekas, "Dynamic programming and optimal control 3rd edition, volume ii," *Belmont, MA: Athena Scientific*, 2011.

[15] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The Int. J. of Rob. Res.*, vol. 23, no. 9, pp. 939–954, 2004.

[16] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—i," *Math. Prog.*, vol. 14, no. 1, pp. 265–294, 1978.

[17] V. Il'ev and N. Linker, "Performance guarantees of a greedy algorithm for minimizing a supermodular set function on comatroid," *EJOR*, vol. 171, no. 2, pp. 648–660, 2006.

[18] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions—ii," in *Polyhedral Combinatorics*. Springer, 1978, pp. 73–87.

[19] G. Qu, D. Brown, and N. Li, "Distributed greedy algorithm for satellite assignment problem with submodular utility function," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 258–263, 2015.

[20] A. A. Bian, J. M. Buhmann, A. Krause, and S. Tschiatschek, "Guarantees for greedy maximization of non-submodular functions with applications," in *ICML*. PMLR, 2017, pp. 498–507.

[21] J. Liu and R. K. Williams, "Submodular optimization for coupled task allocation and intermittent deployment problems," *IEEE Rob. and Aut. Letters*, vol. 4, no. 4, pp. 3169–3176, 2019.

[22] B. Guo, O. Karaca, T. Summers, and M. Kamgarpour, "Actuator placement for optimizing network performance under controllability constraints," in *58th CDC*. IEEE, 2019, pp. 7140–7147.

[23] B. Guo, O. Karaca, T. H. Summers, and M. Kamgarpour, "Actuator placement under structural controllability using forward and reverse greedy algorithms," *IEEE Trans. on Aut. Contr.*, 2020.

[24] L. F. Chamon, A. Amice, and A. Ribeiro, "Approximately supermodular scheduling subject to matroid constraints," *arXiv preprint arXiv:2003.08841*, 2020.

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

[26] S. T. Jawaid and S. L. Smith, "Informative path planning as a maximum traveling salesman problem with submodular rewards," *Discrete Applied Mathematics*, vol. 186, pp. 112–127, 2015.

[27] M. Sviridenko, J. Vondrák, and J. Ward, "Optimal approximation for submodular and supermodular optimization with bounded curvature," *Math. of OR*, vol. 42, no. 4, pp. 1197–1218, 2017.

[28] A. Atamtürk, G. L. Nemhauser, and M. W. Savelsbergh, "A combined lagrangian, linear

programming, and implication heuristic for large-scale set partitioning problems," *J. of Heuristics*, vol. 1, no. 2, pp. 247–259, 1996.

[29] M. Khoo, T. A. Wood, C. Manzie, and I. Shames, "A distributed augmenting path approach for the bottleneck assignment problem," *arXiv preprint arXiv:2011.09606*, 2020.

[30] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer Science & Business Media, 2003, vol. 24.

[31] D. J. Welsh, *Matroid theory*. Courier Corporation, 2010.

[32] J. G. Oxley, *Matroid theory*. Oxford Univ. P., 2006, vol. 3.

[33] A. Krause and D. Golovin, "Submodular function maximization." *Tractability*, vol. 3, pp. 71–104, 2014.

[34] A. Krause and C. E. Guestrin, "Near-optimal nonmyopic value of information in graphical models," *arXiv preprint arXiv:1207.1394*, 2012.

[35] F. Bach, "Learning with submodular functions: A convex optimization perspective," *arXiv preprint arXiv:1111.6453*, 2011.

[36] A. Das and D. Kempe, "Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection," *arXiv preprint arXiv:1102.3975*, 2011.

8 Appendix

8.1 Calculating the contamination risk p_H^k

We provide the details for calculating the contamination risk $p_H^k : X \times X \to [0,1]$. The contamination risk is defined so that a robot can calculate the risk of moving to a contaminated area. This is challenging, since the robot cannot observe the hazard state. The calculations have to rely on the knowledge of the initial hazard state and hazard evolution model. The value $p_H^k(\bar{x}^{k+1}, \bar{x}^k)$ denotes the probability of robot position $x^{k+1} \in y^{k+1}$ being contaminated at time step $k+1$ given the robot moves from $x^k = \bar{x}^k$ to $x^{k+1} = \bar{x}^{k+1}$ and $x^k \notin y^k$ is not contaminated at step k. Based on Equation (8) we can write the following

$$\begin{aligned} p_H^k(\bar{x}^{k+1}, \bar{x}^k) &= P(x^{k+1} \in y^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k) \\ &\overset{(1)}{=} \frac{P(x^{k+1} \in y^{k+1}, x^k \notin y^k \,|\, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k)}{P(x^k \notin y^k \,|\, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k)} \\ &\overset{(2)}{=} \frac{\sum\limits_{\bar{y}^{k+1}: \bar{x}^{k+1} \in \bar{y}^{k+1}} \sum\limits_{\bar{y}^k: \bar{x}^k \notin \bar{y}^k} P(y^{k+1} = \bar{y}^{k+1}, y^k = \bar{y}^k)}{\sum\limits_{\bar{y}^k: \bar{x}^k \notin \bar{y}^k} P(y^k = \bar{y}^k)}, \end{aligned} \tag{38}$$

where we made the following steps:

(1) Using well known transformations from probability theory for conditional probabilities.
(2) Using conditions $x^{k+1} = \bar{x}^{k+1}$ and $x^k = \bar{x}^k$ and the addition rule of probabilities.

We can also write

$$\begin{aligned} P(y^{k+1} = \bar{y}^{k+1}, y^k = \bar{y}^k) &\overset{(1)}{=} P(y^{k+1} = \bar{y}^{k+1} \,|\, y^k = \bar{y}^k) \cdot P(y^k = \bar{y}^k) \\ &\overset{(2)}{=} \tau_Y(\bar{y}^{k+1} \,|\, \bar{y}^k) \cdot P(y^k = \bar{y}^k), \end{aligned} \tag{39}$$

due to steps:

(1) Using well known transformations from probability theory for conditional probabilities.
(2) By the definition of τ_Y in Section 4.1.3.

Furthermore, we can write

$$\begin{aligned} P(y^k = \bar{y}^k) &\overset{(1)}{=} \sum_{\bar{y}^{k-1} \in Y} P(y^k = \bar{y}^k, y^{k-1} = \bar{y}^{k-1}) \\ &\overset{(2)}{=} \sum_{\bar{y}^{k-1} \in Y} P(y^k = \bar{y}^k \,|\, y^{k-1} = \bar{y}^{k-1}) \cdot P(y^{k-1} = \bar{y}^{k-1}) \\ &\overset{(3)}{=} \sum_{\bar{y}^{k-1} \in Y} \tau_Y(\bar{y}^k \,|\, \bar{y}^{k-1}) \cdot P(y^{k-1} = \bar{y}^{k-1}) \end{aligned}$$

$$\overset{(4)}{=} \sum_{\bar{y}^{k-1}\in Y} \tau_Y(\bar{y}^k \mid \bar{y}^{k-1}) \cdot \sum_{\bar{y}^{k-2}\in Y} P(y^{k-1}=\bar{y}^{k-1}, y^{k-2}=\bar{y}^{k-2})$$
$$\overset{(5)}{=} \sum_{\bar{y}^{k-1}\in Y} \cdots \sum_{\bar{y}^{0}\in Y} \tau_Y(\bar{y}^k \mid \bar{y}^{k-1}) \cdots \tau_Y(\bar{y}^1 \mid \bar{y}^0) \cdot \mathbb{1}_{y^0}(\bar{y}^0), \tag{40}$$

where the following hold:

(1) Using marginalization of discrete random variables.
(2) Using the definition of conditional probabilities.
(3) By the definition of τ_Y in Section 4.1.3.
(4) Similarly to Step (1).
(5) Continue iterating Steps (1)-(3) and finally adding that the initial hazard state y^0 is known.

Combining Equations (39) and (40) with Equation (38), we finally get

$$p_H^k(\bar{x}^{k+1}, \bar{x}^k) = \frac{\sum_{\bar{y}^{k+1}:\bar{x}^{k+1}\in\bar{y}^{k+1}} \sum_{\bar{y}^k:\bar{x}^k\notin\bar{y}^k} \tau_Y(\bar{y}^{k+1} \mid \bar{y}^k) \cdot P(y^k=\bar{y}^k)}{\sum_{\bar{y}^k:\bar{x}^k\notin\bar{y}^k} P(y^k=\bar{y}^k)},$$

with

$$P(y^k=\bar{y}^k) = \sum_{\bar{y}^{k-1}\in Y} \cdots \sum_{\bar{y}^0\in Y} \tau_Y(\bar{y}^k \mid \bar{y}^{k-1}) \cdots \tau_Y(\bar{y}^1 \mid \bar{y}^0) \cdot \mathbb{1}_{y^0}(\bar{y}^0).$$

8.2 Definition of the combined transition kernel τ_S^k

In this section, we provide a detailed description of the combined state transition kernel $\tau_S^k : S \times S \times U \to [0,1]$ (see Equation (9)). We consider three cases of transitions: from $s^k \in \{s_G, s_H\}$ to $s^{k+1} \in S$, from $s^k \in S \setminus \{s_G, s_H\}$ to $s^{k+1} = s_H$, from $s^k \in S \setminus \{s_G, s_H\}$ to $s^{k+1} \in S \setminus \{s_H\}$.

- Transition from $s^k \in \{s_G, s_H\}$ to $s^{k+1} \in S$:
  Once the system transmits into the goal or contamination state, it stays in that state, hence if $\bar{s}^{k+1} = \bar{s}^k \in \{S_G, S_H\}$, then for any k and for all $\bar{u}^k \in U$

$$\tau_S^k(\bar{s}^{k+1} \mid \bar{s}^k, \bar{u}^k) = P(s^{k+1}=\bar{s}^{k+1} \mid s^k=\bar{s}^k, u^k=\bar{u}^k) = 1. \tag{41}$$

- Transition from $s^k \in S \setminus \{s_G, s_H\}$ to $s^{k+1} = s_H$:
  The transition to the contamination state $s^{k+1} = s_H$ from $s^k \neq s_H$ happens if and only if $x^{k+1} \in y^{k+1}$ and $x^k \notin y^k$. Hence if $\bar{s}^{k+1} = s_H$ and $\bar{s}^k = (\bar{q}^k, \bar{x}^k)$, then for any k and $\bar{u}^k \in U$

$$\begin{aligned}\tau_S^k(\bar{s}^{k+1} \mid \bar{s}^k, \bar{u}^k) &= P(s^{k+1}=\bar{s}^{k+1} \mid s^k=\bar{s}^k, u^k=\bar{u}^k) \\ &\overset{(1)}{=} P(x^{k+1}\in y^{k+1} \mid x^k\notin y^k, q^k=\bar{q}^k, x^k=\bar{x}^k, u^k=\bar{u}^k)\end{aligned}$$

$$
\begin{aligned}
&\overset{(2)}{=} P(x^{k+1} \in y^{k+1} \,|\, x^k \notin y^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(3)}{=} \sum_{\bar{x}^{k+1} \in X} P(x^{k+1} \in y^{k+1}, x^{k+1} = \bar{x}^{k+1} \,|\, x^k \notin y^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(4)}{=} \sum_{\bar{x}^{k+1} \in X} P(x^{k+1} \in y^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\times P(x^{k+1} = \bar{x}^{k+1} \,|\, x^k \notin y^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(5)}{=} \sum_{\bar{x}^{k+1} \in X} P(x^{k+1} \in y^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k) \\
&\times P(x^{k+1} = \bar{x}^{k+1} \,|\, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(6)}{=} \sum_{\bar{x}^{k+1} \in X} p_H^k(\bar{x}^{k+1}, \bar{x}^k) \cdot \tau_X(\bar{x}^{k+1} \,|\, \bar{x}^k, \bar{u}^k), \qquad (42)
\end{aligned}
$$

where we made steps:

(1) Due to the definition of $\bar{s}^{k+1}$ and $\bar{s}^k$.
(2) Since both x^{k+1} and y^{k+1} are independent of q^k.
(3) Using the addition rule of probabilities.
(4) Using well known transformations from probability theory for conditional probabilities.
(5) Since $x^{k+1} = \bar{x}^{k+1}$ and $x^k = \bar{x}^k$ already entails $u^k = \bar{u}^k$. Furthermore,x^{k+1} is independent of y^k.
(6) By the definition of τ_X in Section 4.1.2.

- Transition from $s^k \in S \setminus \{s_G, s_H\}$ to $s^{k+1} \in S \setminus \{s_H\}$:
The transition between states $s^{k+1} \neq s_H$ from $s^k \neq s_H$ happens if and only if $x^{k+1} \notin y^{k+1}$ and $x^k \notin y^k$. Hence if $\bar{s}^{k+1} = (\bar{q}^{k+1}, \bar{x}^{k+1})$ and $\bar{s}^k = (\bar{q}^k, \bar{x}^k)$, then for any k and $\bar{u}^k \in U$

$$
\begin{aligned}
\tau_S^k(\bar{s}^{k+1} \,|\, \bar{s}^k, \bar{u}^k) &= P(s^{k+1} = \bar{s}^{k+1} \,|\, s^k = \bar{s}^k, u^k = \bar{u}^k) \\
&\overset{(1)}{=} P(x^{k+1} \notin y^{k+1}, q^{k+1} = \bar{q}^{k+1}, x^{k+1} = \bar{x}^{k+1} \,|\, x^k \notin y^k, q^k = \bar{q}^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(2)}{=} P(x^{k+1} \notin y^{k+1}, q^{k+1} = \bar{q}^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, q^k = \bar{q}^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\times P(x^{k+1} = \bar{x}^{k+1} \,|\, x^k \notin y^k, q^k = \bar{q}^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(3)}{=} P(x^{k+1} \notin y^{k+1} \,|\, x^k \notin y^k, q^{k+1} = \bar{q}^{k+1}, x^{k+1} = \bar{x}^{k+1}, q^k = \bar{q}^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\times P(q^{k+1} = \bar{q}^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, q^k = \bar{q}^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\times P(x^{k+1} = \bar{x}^{k+1} \,|\, x^k \notin y^k, q^k = \bar{q}^k, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(4)}{=} P(x^{k+1} \notin y^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k) \\
&\times P(q^{k+1} = \bar{q}^{k+1} \,|\, x^{k+1} = \bar{x}^{k+1}, q^k = \bar{q}^k) \cdot P(x^{k+1} = \bar{x}^{k+1} \,|\, x^k = \bar{x}^k, u^k = \bar{u}^k) \\
&\overset{(5)}{=} \Big(1 - P(x^{k+1} \in y^{k+1} \,|\, x^k \notin y^k, x^{k+1} = \bar{x}^{k+1}, x^k = \bar{x}^k)\Big)
\end{aligned}
$$

$$
\begin{aligned}
&\times \tau_Q(\bar{q}^{k+1} \,|\, \bar{q}^k, \bar{x}^{k+1}) \cdot \tau_X(\bar{x}^{k+1} \,|\, \bar{x}^k, \bar{u}^k) \\
&\overset{(6)}{=} (1 - p_H^k(\bar{x}^{k+1}, \bar{x}^k)) \cdot \tau_Q(\bar{q}^{k+1} \,|\, \bar{q}^k, \bar{x}^{k+1}) \cdot \tau_X(\bar{x}^{k+1} \,|\, \bar{x}^k, \bar{u}^k), \qquad (43)
\end{aligned}
$$

where we used the following steps:

(1) Due to the definition of $\bar{s}^{k+1}$ and $\bar{s}^k$.

(2) Using well known transformations from probability theory for conditional probabilities.

(3) Similarly to Step (2).

(4) Since $x^{k+1} = \bar{x}^{k+1}$ and $x^k = \bar{x}^k$ already entails $u^k = \bar{u}^k$, and x^{k+1} and y^{k+1} are independent of q^k and q^{k+1}. Furthermore, q^{k+1} is also independent of x^k, y^k and u^k, and x^{k+1} is independent of y^k and q^k.

(5) Due to well known properties of probabilities. Furthermore, by the definition of τ_Q and τ_X in Section 4.1.4 and 4.1.2, respectively.

(6) By the definition of p_H^k in Section 4.1.5.

Combining Equations (41), (42) and Equation (43), we finally get

$$
\tau_S^k(\bar{s}^{k+1} \,|\, \bar{s}^k, \bar{u}^k) = \begin{cases}
1 & \text{if } \bar{s}^{k+1} = \bar{s}^k \in \{s_G, s_H\}, \\
\sum\limits_{\bar{x}^{k+1} \in X} p_H^k(\bar{x}^{k+1}, \bar{x}^k) & \\
\quad \times \tau_X(\bar{x}^{k+1} \,|\, \bar{x}^k, \bar{u}^k) & \text{if } \bar{s}^{k+1} = s_H \\
& \wedge\ \bar{s}^k = (\bar{q}^k, \bar{x}^k) \notin \{s_G, s_H\}, \\
\big(1 - p_H^k(\bar{x}^{k+1}, \bar{x}^k)\big) & \\
\quad \times \tau_Q(\bar{q}^{k+1} \,|\, \bar{q}^k, \bar{x}^{k+1}) & \\
\quad \times \tau_X(\bar{x}^{k+1} \,|\, \bar{x}^k, \bar{u}^k) & \text{if } \bar{s}^{k+1} = (\bar{q}^{k+1}, \bar{x}^{k+1}) \neq s_H \\
& \wedge\ \bar{s}^k = (\bar{q}^k, \bar{x}^k) \notin \{s_G, s_H\}, \\
0 & \text{otherwise.}
\end{cases}
$$

8.3 Proof of Equation (13)

This section aims to prove how the dynamic programming algorithm (see Equation (12)) solves the single-agent safe planning problem (see Equation 11), i.e., Equation (13). Let us reformulate Equation (10) the following way

$$
\begin{aligned}
f_r(\mu, T_r) = P(s^N = s_G \,|\, \mu) &\overset{(1)}{=} \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot P(s^N = \bar{s}^N \,|\, \mu) \\
&\overset{(2)}{=} \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \sum_{\bar{s}^{N-1} \in S} P(s^N = \bar{s}^N, s^{N-1} = \bar{s}^{N-1} \,|\, \mu) \\
&\overset{(3)}{=} \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \sum_{\bar{s}^{N-1} \in S} P(s^N = \bar{s}^N \,|\, s^{N-1} = \bar{s}^{N-1} \,|\, \mu) \cdot P(s^{N-1} = \bar{s}^{N-1} \,|\, \mu)
\end{aligned}
$$

$$
\begin{aligned}
&\overset{(4)}{=} \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \sum_{\bar{s}^{N-1} \in S} \tau_S^{N-1}(\bar{s}^N \,|\, \bar{s}^{N-1}, \mu^{N-1}(\bar{s}^{N-1})) \cdot P(s^{N-1} = \bar{s}^{N-1} \,|\, \mu) \\
&\overset{(5)}{=} \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \sum_{\bar{s}^{N-1} \in S} \tau_S^{N-1}(\bar{s}^N \,|\, \bar{s}^{N-1}, \mu^{N-1}(\bar{s}^{N-1})) \\
&\cdots \sum_{\bar{s}^0 \in S} \tau_S^0(\bar{s}^1, |\, \bar{s}^0, \mu^0(\bar{s}^0)) \cdot P(s^0 = \bar{s}^0), \qquad (44)
\end{aligned}
$$

where we made the following steps:

(1) Due to the definition of the indicator function in Section 2.
(2) Using marginalization of discrete random variables.
(3) Using the definition of conditional probabilities.
(4) By the definition of τ_S^k in Section 4.1.5.
(5) Continue iterating Steps (2)-(4) and finally adding that the initial state $s_r^0 = (\emptyset, x_r^0)$ is known.

Now, by combining equations (44) and (11) with (12), we can write

$$
\begin{aligned}
f_r(T_r) &= f_r(\mu_r, T_r) = \max_{\mu} f_r(\mu, T_r) \\
&= \max_{\mu} \Big\{ \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \sum_{\bar{s}^{N-1} \in S} \tau_S^{N-1}(\bar{s}^N \,|\, \bar{s}^{N-1}, \mu^{N-1}(\bar{s}^{N-1})) \\
&\cdots \sum_{\bar{s}^0 \in S} \tau_S^0(\bar{s}^1, |\, \bar{s}^0, \mu^0(\bar{s}^0)) \cdot \mathbb{1}_{s_r^0}(\bar{s}^0) \Big\} \\
&= \max_{\mu} \Big\{ \sum_{\bar{s}^0 \in S} \cdots \sum_{\bar{s}^{N-1} \in S} \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \tau_S^{N-1}(\bar{s}^N \,|\, \bar{s}^{N-1}, \mu^{N-1}(\bar{s}^{N-1})) \\
&\cdots \tau_S^0(\bar{s}^1, |\, \bar{s}^0, \mu^0(\bar{s}^0)) \cdot \mathbb{1}_{s_r^0}(\bar{s}^0) \Big\} \\
&= \sum_{\bar{s}^0 \in S} \max_{\mu^0} \Big\{ \cdots \max_{\mu^{N-2}} \Big\{ \sum_{\bar{s}^{N-1} \in S} \max_{\mu^{N-1}} \Big\{ \sum_{\bar{s}^N \in S} \mathbb{1}_{s_G}(\bar{s}^N) \cdot \tau_S^{N-1}(\bar{s}^N \,|\, \bar{s}^{N-1}, \mu^{N-1}(\bar{s}^{N-1})) \Big\} \\
&\times \tau_S^{N-2}(\bar{s}^{N-1} \,|\, \bar{s}^{N-2}, \mu^{N-2}(\bar{s}^{N-2})) \Big\} \cdots \tau_S^0(\bar{s}^1, |\, \bar{s}^0, \mu^0(\bar{s}^0)) \Big\} \cdot \mathbb{1}_{s_r^0}(\bar{s}^0) \\
&= \sum_{\bar{s}^0 \in S} \max_{\mu^0} \Big\{ \cdots \max_{\mu^{N-2}} \Big\{ \sum_{\bar{s}^{N-1} \in S} \max_{\mu^{N-1}} \Big\{ \sum_{\bar{s}^N \in S} V^N(\bar{s}^N) \cdot \tau_S^{N-1}(\bar{s}^N \,|\, \bar{s}^{N-1}, \mu^{N-1}(\bar{s}^{N-1})) \Big\} \\
&\cdot \tau_S^{N-2}(\bar{s}^{N-1} \,|\, \bar{s}^{N-2}, \mu^{N-2}(\bar{s}^{N-2})) \Big\} \cdots \tau_S^0(\bar{s}^1, |\, \bar{s}^0, \mu^0(\bar{s}^0)) \Big\} \cdot \mathbb{1}_{s_r^0}(\bar{s}^0) \\
&= \sum_{\bar{s}^0 \in S} \max_{\mu^0} \Big\{ \cdots \max_{\mu^{N-2}} \Big\{ \sum_{\bar{s}^{N-1} \in S} V^{N-1}(\bar{s}^{N-1}) \cdot \tau_S^{N-2}(\bar{s}^{N-1} \,|\, \bar{s}^{N-2}, \mu^{N-2}(\bar{s}^{N-2})) \Big\} \\
&\cdots \tau_S^0(\bar{s}^1, |\, \bar{s}^0, \mu^0(\bar{s}^0)) \Big\} \cdot \mathbb{1}_{s_r^0}(\bar{s}^0) \\
&= \cdots = \sum_{\bar{s}^0 \in S} V^0(\bar{s}^0) \cdot \mathbb{1}_{s_r^0}(\bar{s}^0) = V^0(s_r^0), \qquad (45)
\end{aligned}
$$

which coincides with Equation (13).

8.4 Full-fleet safe planning problem

Based on the formulation used in Section 4.1 we define the full-fleet safe planning problem for multiple agents. Let us consider the target list $T \in X$ and the set of robots R, where $|R| = n > 1$. At each step k, instead of considering the position of a single-robot $x^k \in X$, we use the position of the fleet as a tuple $x_M^k = (x_1^k, \dots, x_n^k)$, where x_r^k is the position of agent r at step k. We can write $x_M^k \in X_M = X^n$. We assume that multiple agents can occupy the same location at the same time. We modify the rest of the notations introduced in Section 4.1 to be consistent with the definition of X_M.

Let $U_M = U^n$ be the set of possible inputs for the fleet. The input applied at step k can be described by $u_M^k = (u_1^k, \dots, u_n^k)$, where u_r^k is the input of agent r. Furthermore, $U_M(x_M) = \{u_M \in U_M \,|\, x_r + d_{u_r} \in X, \ \forall r \in R\} \subseteq U_M$ is the set of available inputs for the fleet. The transition kernel $\tau_{X,M} : X_M \times X_M \times U_M \to [0, 1]$ defines the fleet dynamics, where

$$\tau_{X,M}(x_M^{k+1} \,|\, x_M^k, u_M^k) = \prod_{r \in R} \tau_X(x_r^{k+1} \,|\, x_r^k, u_r^k),$$

consistently with Section 4.1.2. The hazard dynamics τ_Y is defined the same way as in Section 4.1.3.

The definition of the target execution state at step k denoted by $q^k \in Q = 2^T$ coincides with that of Section 4.1.4. The transition kernel, however, can now be described by $\tau_{Q,M} : Q \times Q \times X_M \to [0, 1]$, where

$$T_{Q,M}(q^{k+1} \,|\, q^k, x_M^{k+1}) = \begin{cases} 1 & \text{if } \ q^{k+1} = q^k, x_r^{k+1} \notin T, \ \ \forall r \in R \\ & \vee \ \ q^{k+1} = q^k \cup \{x_r^{k+1} \,|\, x_r^{k+1} \in T\}, \\ 0 & \text{otherwise.} \end{cases}$$

If none of the agents step on a target location $x_r^{k+1} \notin T$ for all $r \in R$, then $q^{k+1} = q^k$. If any of the robots visit a target location, it is added to the target execution state, hence $q^{k+1} = q^k \cup \{x_r^{k+1} \,|\, x_r^{k+1} \in T\}$.

We can now define the combined state space S_M based on Section 4.1.5. First, we define the contamination state $s_{H,M}$, which indicates an unsuccessful mission. The fleet transmits into state $s_{H,M}$ at step k if at least one of the robots get contaminated, hence there exist $r \in R$ such that $x_r^k \in y^k$. Second, we can reduce the state space of S_M by removing the set of impossible states $\{(q, x_M) \,|\, \exists r \in R \text{ st. } x_r \in T \wedge x_r \notin q\}$, where not all visited target positions were added to the target execution state. Now we can define the combined state space

$$S_M = \{s_{H,M}\} \cup (Q \times X_M) \setminus \{(q, x_M) \,|\, \exists r \in R \text{ st. } x_r \in T \wedge x_r \notin q\}. \tag{46}$$

Furthermore, we can define the goal position $x_G \in X$, the goal position of the fleet $x_{G,M} = (x_G, \dots, x_G)$ and the goal state $s_{G.M} = (x_{G,M}, T)$ of the fleet. Reaching $s_{G,M}$ indicates a successful mission, where every robot has reached the goal position and all targets have been

visited by the fleet. The initial robot positions $x_M^0 = (x_1^0, \ldots, x_n^0)$ and the initial state $s_M^0 = (\emptyset, x_M^0)$ is assumed to be known.

We now define the contamination risk for the fleet $p_{H,M}^k : X_M \times X_M \to [0,1]$ based on Appendix 8.1. The value of $p_{H,M}^k(\bar{x}_M^{k+1}, \bar{x}_M^k)$ is defined as the probability of at least one robot getting contaminated at step $k+1$ (there exist $r \in R$ such that $\bar{x}_r^{k+1} \in y^{k+1}$) given that no robots are contaminated at step k ($\bar{x}_r^k \notin y^k$ for all $r \in R$) and the transition between $x_M^k = \bar{x}_M^k = (\bar{x}_1^k, \ldots, \bar{x}_n^k)$ and $x_M^{k+1} = \bar{x}_M^{k+1} = (\bar{x}_1^{k+1}, \ldots, \bar{x}_n^{k+1})$, i.e.,

$$p_{H,M}^k(\bar{x}_M^{k+1}, \bar{x}_M^k) = P(y^{k+1} \in \bar{Y}^{k+1}(\bar{x}_M^{k+1}) \,|\, y^k \in \bar{Y}^k(\bar{x}_M^k), x_M^{k+1} = \bar{x}_M^{k+1}, x_M^k = \bar{x}_M^k),$$

where we define the following sets

$$\bar{Y}^k(\bar{x}_M^k) = \{\bar{y}^k \in Y \,|\, \bar{x}_r^k \notin \bar{y}^k, \ \forall r \in R\},$$
$$\bar{Y}^{k+1}(\bar{x}_M^{k+1}) = \{\bar{y}^{k+1} \in Y \,|\, \exists r \in R \text{ st. } \bar{x}_r^{k+1} \in \bar{y}^{k+1}\}.$$

Similarly to p_H^k, calculating the values of $p_{H,M}^k$ is computationally intractable, hence the usage of the Monte-Carlo sampling based approximation is advised (see [13, Algorithm 1]).

Similarly to Section 4.1.6, we propose the controller synthesis and provide the solution via dynamic programming. Given the initial state s_M^0 and the time horizon $N \in \mathbb{N}_{>0}$, for a generic control policy $\pi = \{\mu_M^0, \ldots, \mu_M^{N-1}\}$ (where $\mu_M^k : S_M \to U_M$ is the control law for the fleet at step k) and target list T, the probability of success is defined by the following equation (consistently with Equation (10))

$$F_M(\pi, T) = P(s_M^N = s_{G,M} \,|\, \pi).$$

The aim of the safe planning problem is to find the optimal control policy $\pi_M(T)$ by maximizing the probability of success, hence (consistently with Equation (11))

$$\pi_M(T) = \arg\max_{\pi} F_M(\pi, T).$$

We can now formulate the dynamic programming algorithm for solving the problem (consistently with Equation (12)): For $k = N$, let us define $V_M^N(s_M^N) = \mathbb{1}_{s_{G,M}}(s_M^N)$, while for $1 \le k \le N$,

$$V_M^{k-1}(s_M^{k-1}) = \max_{u_M \in U_M(x_M)} \left\{ \sum_{s_M^k \in S_M} \tau_{S,M}^{k-1}(s_M^k \,|\, s_M^{k-1}, u_M) \cdot V_M^k(s_M^k) \right\}.$$

The control law $\mu_M^k(s_M^k)$ can be obtained as the optimal $u_M \in U_M(x_M)$ at step k. Furthermore similarly to Appendix 8.3 it holds, that

$$F_M(T) = F_M(\pi_M, T) = \max_{\pi} F_M(\pi, T) = V_M^0(s_M^0).$$

8.5 Mild conditions for multiplicative group success to be a lower bound on the probability of group success

We introduce the following proposition.

Proposition 1 *If knowing that a robot $r \in R$ succeeds does not decrease the probability for other robots $R' \subset R \setminus \{r\}$ succeeding, then the multiplicative group success $F(T)$ (see Equation (14)) is a lower bond on the probability of group success $F_M(T)$ (see Appendix 8.4).*

Proof (Proposition 1) *First, we define $c_r^{\pi_r}$ to be a discrete random variable. Let $c_r^{\pi_r} = True$ indicate that robot $r \in R$ ended its mission successfully under control policy π_r and task allocation $T_r \subset T$. Hence, we can write $P(c_r^{\pi_r} = True) = P(s^N = s_G|\pi) = f_r(T_r)$ consistently with Equation (10). Furthermore let $P(c_r^{\pi_r} = False) = 1 - P(c_r^{\pi_r} = True)$. Now, we can express proposition 1 as follows*

$$P\left(\{c_{r'}^{\pi_{r'}} = True\}_{r' \in R'} \,|\, c_r^{\pi_r} = True\right) \geq P\left(\{c_{r'}^{\pi_{r'}} = True\}_{r' \in R'}\right),$$

for all $r \in R$ and $R' \subset R \setminus \{r\}$. This implies

$$P\left(\{c_r^{\pi_r} = True\}_{r \in R}\right) \geq \prod_{r \in R} P\left(c_r^{\pi_r} = True\right) = \prod_{r \in R} f_r(T_R) = F(T). \tag{47}$$

Furthermore, according to Appendix 8.4, we have $F_M(T) = \max_\pi F_M(\pi, T)$ and consequently

$$F_M(T) \geq P\left(\{c_r^{\pi_r} = True\}_{r \in R}\right) \geq \prod_{r \in R} P\left(c_r^{\pi_r} = True\right), \tag{48}$$

for any set of policies $\{\pi_r\}_{r \in R}$. Finally, by combining Equation (47) and (48) we can write

$$F_M(T) \geq P\left(\{c_r^{\pi_r} = True\}_{r \in R}\right) \geq \prod_{r \in R} P\left(c_r^{\pi_r} = True\right) \geq F(T),$$

which concludes the proof. ■

8.6 Performance guarantee for the forward greedy algorithm

In this section we provide a proof for the performance guarantee introduced in Theorem 1.

Lemma 1 *For any $M \in \mathcal{I}$, $|M| = K$, the elements of $M = \{m^1, \dots, m^K\}$ can be ordered, so that*

$$\rho(m^k|A^{k-1}) \geq \rho^k = \rho(a^k|A^{k-1}),$$

where the sets $A^k = \{a^1, \dots, a^k\}$ for all k and $A_{fg} = A^{|K|}$ are obtained using Algorithm 1. Furthermore, if $m^k \in A_{fg}$, then $m^k = a^k$ holds.

Proof (Proof of Lemma 1) *First, we need to prove by induction, that $A_m^k = \{a^1, \dots, a^{k-1}, m^k, m^{k+1}, \dots, m^K\} \in \mathcal{I}$ and consequently $A_m^k \supset \{a^1, \dots, a^{k-1}, m^k\} \in \mathcal{I}$ holds for all k.*

This statement holds for $k = K$. Since $(P, \mathcal{I})$ is a matroid and $A_{fg} = \{a^1, \dots, a^K\} \in \mathcal{I}$, therefore $A^{K-1} = \{a^1, \dots, a^{K-1}\} \in \mathcal{I}$. Note that $M = \{m^1, \dots, m^K\} \in \mathcal{I}$ and because $|M| > |A^{K-1}|$, due to matroid properties, there exists $m^K \in M \setminus A^{K-1}$ for which $A^{K-1} \cup \{m^K\} = \{a^1, \dots, a^{K-1}, m^K\} = A_m^K \in \mathcal{I}$.

Assuming the statement holds for $k+1$, we know, that $A_m^{k+1} = \{a^1, \dots, a^t, m^{k+1}, m^{k+2}, \dots, m^K\} \in \mathcal{I}$. Consequently $\{a^1, \dots, a^{k-1}, m^{k+1}, m^{k+2}, \dots, m^K\} \in \mathcal{I}$. Now, due to matroid properties, since $M \in \mathcal{I}$ and $|M| > |A_m^{k+1}|$, there exist $m^k \in M \setminus A_m^{k+1}$ for which $A_m^{k+1} \cup \{m^t\} = \{a^1, \dots, a^{k-1}, m^k, m^{k+1}, \dots, m^K\} = A_m^k \in \mathcal{I}$. It is now easy to see, that $A_m^k \supset \{a^1, \dots, a^{k-1}, m^k\} \in \mathcal{I}$ also holds.

If $m^k \in A_{fg}$, then at step k of the induction proof above, $m^k = a^k$ can be chosen. Since for all $k' > k$ it holds that $m^k = a^k \in A_m^{k'+1}$ and $m^{k'} \in M \setminus A_m^{k'+1}$, therefore $m^{k'} \neq m^k$, it cannot get chosen at an earlier step. Consequently, if $m^k \in A_{fg}$, then $m^k = a^k$ can be written.

Note that if $\{a^1, \dots, a^{k-1}, m^k\} = A^{k-1} \cup \{m^k\} \in \mathcal{I}$, then the following holds for all k

$$\rho(m^k|A^{k-1}) \geq \min_{a: A^{k-1} \cup \{a\} \in \mathcal{I}} \rho(a|A^{k-1}) = \rho(a^k|A^{k-1}),$$

which concludes the proof. ■

Lemma 2 *For all $A \subseteq P$, and for any $a \notin A$, we have that*

$$\rho(a|A) \geq (1 - \alpha) \cdot \rho(a|\emptyset).$$

Proof (Proof of Lemma 2) *Trivial, directly comes from the definition of curvature, see Definition 6.* ■

Lemma 3 *For all $A \subseteq P$, and for any $a \notin A$, we have that*

$$\rho(a|\emptyset) \geq \gamma \cdot \rho(a|A).$$

Proof (Proof of Lemma 3) *Trivial, directly comes from the definition of submodularity ratio, see Definition 4.* ■

Proof (Proof of Theorem 1) *Let $A_* = \{a_*^1, \ldots, a_*^K\}$, where the elements a_*^k are ordered according to lemma 1. Let $A_*^k = \{a_*^1, \ldots, a_*^k\}$ for $k = 1, \ldots, K$, and $A_*^0 = \emptyset$. Using this definition and Lemma 2, we obtain*

$$F_{fg}(A_*) - F_{fg}(\emptyset) = \sum_{k=1}^{K} \rho(a_*^k|A_*^{k-1}) \geq (1-\alpha) \cdot \sum_{k=1}^{K} \rho(a_*^k|\emptyset). \tag{49}$$

Let us further define the following sets:

$$\begin{aligned} K^{int.} &= \left\{k \in \{1, \ldots, K\} \,|\, a^k \in A_{fg} \cap A_*\right\}, \\ K^{fg} &= \left\{k \in \{1, \ldots, K\} \,|\, a^k \in A_{fg} \setminus A_*\right\}, \\ K^{*} &= \left\{k \in \{1, \ldots, K\} \,|\, a_*^k \in A_* \setminus A_{fg}\right\}. \end{aligned}$$

It is easy to see, that $K^* = K^{fg}$, which follows from Lemma 1. Now, using Lemma 1, we obtain the following for the greedy solution

$$\begin{aligned} F_{fg}(A_{fg}) - F_{fg}(\emptyset) = \sum_{k=1}^{K} \rho^k &= \sum_{k \in K^{int.}} \rho(a^k|A^{k-1}) + \sum_{k \in K^{fg}} \rho(a^k|A^{k-1}) \\ &\leq \sum_{k \in K^{int.}} \rho(a^k|A^{k-1}) + \sum_{k \in K^{*}} \rho(a_*^k|A^{k-1}). \end{aligned}$$

By noticing, that for any $a_*^k \notin A_{fg}$ we have $a_*^k \notin A^{k-1}$, furthermore $a^k \notin A^{k-1}$ holds for any $k \in K^{int.}$, and by invoking Lemma 3, it holds, that

$$\begin{aligned} F_{fg}(A_{fg}) - F_{fg}(\emptyset) &\leq \sum_{k \in K^{int.}} \rho(a^k|A^{k-1}) + \sum_{k \in K^{*}} \rho(a_*^k|A^{k-1}) \\ &\leq \sum_{k \in K^{int.}} \rho(a^k|A^{k-1}) + \frac{1}{\gamma} \cdot \sum_{k \in K^{*}} \rho(a_*^k|\emptyset) \\ &\leq \frac{1}{\gamma} \cdot \sum_{k \in K^{int.}} \rho(a^k|\emptyset) + \frac{1}{\gamma} \cdot \sum_{k \in K^{*}} \rho(a_*^k|\emptyset). \end{aligned}$$

Now, by Lemma 1, if $a^k \in A_*$, then $a^k = a_*^k$, therefore

$$F_{fg}(A_{fg}) - F_{fg}(\emptyset) \leq \frac{1}{\gamma} \cdot \sum_{k \in A^{int.}} \rho(a^k|\emptyset) + \frac{1}{\gamma} \cdot \sum_{k \in K^{*}} \rho(a_*^k|\emptyset)$$

$$= \frac{1}{\gamma} \cdot \sum_{k \in K^{int.}} \rho(a_*^k|\emptyset) + \frac{1}{\gamma} \cdot \sum_{k \in K^*} \rho(a_*^k|\emptyset)$$
$$= \frac{1}{\gamma} \cdot \sum_{k=1}^{K} \rho(a_*^k|\emptyset). \tag{50}$$

Finally, combining Equations (49) *and* (50) *completes the proof*

$$\frac{F_{fg}(A_{fg}) - F_{fg}(\emptyset)}{F_{fg}(A_*) - F_{fg}(\emptyset)} \leq \frac{\frac{1}{\gamma} \cdot \sum_{k=1}^{K} \rho(a_*^k|\emptyset)}{(1-\alpha) \cdot \sum_{k=1}^{K} \rho(a_*^k|\emptyset)} = \frac{1}{\gamma \cdot (1-\alpha)}.$$

■

8.7 Performance guarantee for the reverse greedy algorithm

In this section we provide a proof for the performance guarantee introduced in Theorem 2.

Lemma 4 *For any $M \in \bar{\mathcal{I}}$, $|M| = \bar{K}$, the elements of $M = \{m^1, \ldots, m^K\}$ can be ordered, so that*

$$\rho(m^k|\bar{A}^{k-1}) \leq \bar{\rho}^k = \rho(\bar{a}^k|\bar{A}^{k-1}),$$

where the sets $\bar{A}^k = \{\bar{a}^1, \ldots, \bar{a}^k\}$ for all k and $\bar{A}_{rg} = \bar{A}^{|\bar{K}|}$ are obtained using Algorithm 3. Furthermore, if $m^k \in \bar{A}_{rg}$, then $m^k = \bar{a}^k$ holds.

Proof (Proof of Lemma 4) *First, we need to prove by induction, that $\bar{A}_m^k$ $= \{\bar{a}^1, \ldots, \bar{a}^{k-1}, m^k, m^{k+1}, \ldots, m^{\bar{K}}\} \in \bar{\mathcal{I}}$ and consequently $\bar{A}_m^k \supset \{\bar{a}^1, \ldots, \bar{a}^{k-1}, m^k\} \in \bar{\mathcal{I}}$ holds for all k.*

This statement holds for $k = \bar{K}$. Since $(P, \bar{\mathcal{I}})$ is a matroid and $\bar{A}_{rg} = \{\bar{a}^1, \ldots, \bar{a}^{\bar{K}}\} \in \bar{\mathcal{I}}$, therefore $\bar{A}^{K-1} = \{\bar{a}^1, \ldots, \bar{a}^{\bar{K}-1}\} \in \bar{\mathcal{I}}$. Note that $M = \{m^1, \ldots, m^{\bar{K}}\} \in \bar{\mathcal{I}}$ and because $|M| > |\bar{A}^{\bar{K}-1}|$, due to matroid properties, there exists $m^{\bar{K}} \in M \setminus \bar{A}^{\bar{K}-1}$ for which $\bar{A}^{\bar{K}-1} \cup \{m^{\bar{K}}\} = \{\bar{a}^1, \ldots, \bar{a}^{\bar{K}-1}, m^{\bar{K}}\} = \bar{A}_m^{\bar{K}} \in \bar{\mathcal{I}}$.

Assuming the statement holds for $k+1$, we know, that $\bar{A}_m^{k+1} = \{\bar{a}^1, \ldots, \bar{a}^k, m^{k+1}, m^{k+2}, \ldots, m^{\bar{K}}\} \in \bar{\mathcal{I}}$. Consequently $\{\bar{a}^1, \ldots, \bar{a}^{k-1}, m^{k+1}, m^{k+2}, \ldots, m^{\bar{K}}\} \in \bar{\mathcal{I}}$. Now, due to matroid properties, since $M \in \bar{\mathcal{I}}$ and $|M| > |\bar{A}_m^{k+1}|$, there exist $m^k \in M \setminus \bar{A}_m^{k+1}$ for which $\bar{A}_m^{k+1} \cup \{m^k\} =$ $\{\bar{a}^1, \ldots, \bar{a}^{k-1}, m^k, m^{k+1}, \ldots, m^{\bar{K}}\} = \bar{A}_m^k \in \bar{\mathcal{I}}$. It is now easy to see, that $\bar{A}_m^k \supset \{\bar{a}^1, \ldots, \bar{a}^{k-1}, m^k\}$ $\in \bar{\mathcal{I}}$ also holds.

If $m^k \in \bar{A}_{rg}$, then at step k of the induction proof above, $m^k = \bar{a}^k$ can be chosen. Since for all $k' > k$ it holds that $m^k = \bar{a}^k \in \bar{A}_m^{k'+1}$ and $m^{k'} \in M \setminus \bar{A}_m^{k'+1}$, therefore $m^{k'} \neq m^k$, it cannot get chosen at an earlier step. Consequently, if $m^k \in \bar{A}_{rg}$, then $m^k = \bar{a}^k$ can be written.

Note that if $\{\bar{a}^1, \ldots, \bar{a}^{k-1}, m^k\} = \bar{A}^{k-1} \cup \{m^k\} \in \bar{\mathcal{I}}$, then the following holds for all k

$$\rho(m^k|\bar{A}^{k-1}) \leq \max_{\bar{a}: \bar{A}^{k-1} \cup \{\bar{a}\} \in \bar{\mathcal{I}}} \rho(\bar{a}|\bar{A}^{k-1}) = \rho(\bar{a}^k|\bar{A}^{k-1}),$$

which concludes the proof. ■

Proof (Proof of Theorem 2) *In this section we extend the existing results shown in [?, Theorem 2.9] by considering the submodularity ratio of function F_{rg} as well. By using the definition of the curvature (see Definition 6), we obtain the following*

$$\begin{aligned} F_{rg}(\bar{A}_* \cup \bar{A}_{rg}) - F_{rg}(\bar{A}_*) &= \sum_{k=1}^{\bar{K}} \rho(\bar{a}^k | \bar{A}_* \cup \bar{A}^{k-1}) \\ &\geq (1-\bar{\alpha}) \cdot \sum_{k=1}^{\bar{K}} \rho(\bar{a}^k | \bar{A}^{k-1}) \\ &= (1-\bar{\alpha}) \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)). \end{aligned} \tag{51}$$

Furthermore, by invoking the definition of the submodularity ratio (see Definition 4) and Lemma 4, we can write

$$\begin{aligned} F_{rg}(\bar{A}_* \cup \bar{A}_{rg}) - F_{rg}(\bar{A}_{rg}) &= \sum_{k=1}^{\bar{K}} \rho(\bar{a}_*^k | \bar{A}_*^k \cup \bar{A}_{rg}) \\ &\leq \frac{1}{\bar{\gamma}} \cdot \sum_{k=1}^{\bar{K}} \rho(\bar{a}_*^k | \bar{A}^{k-1}) \\ &\leq \frac{1}{\bar{\gamma}} \cdot \sum_{k=1}^{\bar{K}} \rho(\bar{a}^k | \bar{A}^{k-1}) \\ &= \frac{1}{\bar{\gamma}} \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)). \end{aligned} \tag{52}$$

Let us now combine Equations (51) *and* (52) *to conclude the proof*

$$\begin{aligned} F_{rg}(\bar{A}_*) + (1-\bar{\alpha}) \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)) &\leq F_{rg}(\bar{A}_*) + F_{rg}(\bar{A}_* \cup \bar{A}_{rg}) - F_{rg}(\bar{A}^*) \\ &= F_{rg}(\bar{A}_* \cup \bar{A}_{rg}) \\ &\leq F_{rg}(\bar{A}_{rg}) + \frac{1}{\bar{\gamma}} \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)). \end{aligned}$$

Now, by substracting $F_{rg}(\emptyset)$ from both sides and rearranging the equation, we obtain

$$\begin{aligned} F_{rg}(\bar{A}_*) - F_{rg}(\emptyset) + (1-\bar{\alpha}) \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)) &\leq F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset) + \frac{1}{\bar{\gamma}} \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)) \\ F_{rg}(\bar{A}_*) - F_{rg}(\emptyset) &\leq \left(1 + \frac{1}{\bar{\gamma}} - (1-\bar{\alpha})\right) \cdot (F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)) \\ \frac{1}{1 + \frac{1}{\bar{\gamma}} + (\bar{\alpha} - 1)} &\leq \frac{F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)}{F_{rg}(\bar{A}_*) - F_{rg}(\emptyset)} \\ \frac{\bar{\gamma}}{1 + \bar{\gamma} \cdot \bar{\alpha}} &\leq \frac{F_{rg}(\bar{A}_{rg}) - F_{rg}(\emptyset)}{F_{rg}(\bar{A}_*) - F_{rg}(\emptyset)}, \end{aligned}$$

which concludes the proof. ■

8.8 Example hazard model τ_Y

In the following, we propose the hazard dynamics τ_Y also used in Section 6 based on [9, Section IV./C.]. Let $y^k \in Y$ be the hazard state and $x \in X \setminus y^k$ be an uncontaminated cell at time step k. We also introduce the scalar parameter $\theta \in [0,1]$ as the *spread speed* parameter which controls the speed of the evolving hazard. Now, let us say, that x can be ignited by any of its contaminated neighbours $\bar{x}_N \in N(x) \cap y^k$ with probability θ and diagonal neighbours $\bar{x}_D \in D(x) \cap y^k$ with probability $\theta \,/\, \sqrt{2}$, to take the distance into account. We also say, that $n_N(x, y^k) = |N(x) \cap y^k|$ and $n_D(x, y^k) = |D(x) \cap y^k|$ denote the numbers of contaminated and diagonally contaminated neighbours of x. Now, we can define the following function $p_{nc} : X \times Y \to [0,1]$ as the probability of uncontaminated position x remaining non-contaminated given hazard state y^k at step k

$$p_{nc}(x \,|\, y^k) = (1-\theta)^{n_N(x,y^k)} \cdot \left(1 - \frac{\theta}{\sqrt{2}}\right)^{n_D(x,y^k)}.$$

Note that the smaller the value of θ, the less likely position x becomes contaminated, hence the slower the hazard spreads. Once the hazard reaches a cell, it remains contaminated throughout the whole process. In order to model this behaviour, we define $p_c : X \times Y \to [0,1]$ representing the probability of position x getting contaminated given hazard state y^k at step k the following way

$$p_c(x \,|\, y^k) = \begin{cases} 1 - p_{nc}(x \,|\, y^k) & \text{if } x \notin y^k, \\ 1 & \text{if } x \in y^k. \end{cases}$$

Finally, we can define transition kernel $\tau_Y : Y \times Y \to [0,1]$ as

$$\tau_Y\left(y^{k+1} \,|\, y^k\right) = \prod_{x \in y^{k+1}} p_c(x \,|\, y^k) \cdot \prod_{x \in (X \setminus y^{k+1})} 1 - p_c(x \,|\, y^k). \tag{53}$$

Function τ_Y defined this way is a valid transition kernel for any k if the following conditions hold.

Condition 8.1 *For all $y^{k+1}, y^k \in Y$ it holds that $\tau_Y\left(y^{k+1} \,|\, y^k\right) \geq 0$.*

Condition 8.2 *For all $y^k \in Y$ it holds that $\sum_{y \in Y} \tau_Y\left(y \,|\, y^k\right) = 1$.*

We can verify that these conditions hold for τ_Y defined by Equation (53) in the following.

Proof (Condition 8.1) *For all $x \in X$ and $y^k \in Y$ we can write the following. Since $\theta \in [0,1]$ and both $n_N(x, y^k) \geq 0$ and $n_D(x, y^k) \geq 0$, it also holds that $p_{nc}(x|y^k) \in [0,1]$. This implies that $p_c(x|y^k) \in [0,1]$ and finally $\tau_Y(y^{k+1} \,|\, y^k) \in [0,1]$. Hence, $\tau_Y(y^{k+1} \,|\, y^k) \geq 0$ for any pair $y^{k+1}, y^k \in Y$.* ■

Proof (Condition 8.2) *Let $\{x_1, \ldots, x_{|X|}\}$ be an ordering of the elements in set X. Furthermore, we define the discrete random variable $x_i^c \in \{True, False\}$. The random variable takes*

the value $x_i^c = True$ if x_i is contaminated ($x_i \in y$), and $x_i^c = False$, if it is not contaminated ($x_i \notin y$), given hazard state $y \in Y$. We can rewrite τ_Y the following way

$$\tau_Y\left(y \,|\, y^k\right) = \tau_Y\left(x_1^c, \ldots, x_{|X|}^c \,|\, y^k\right) = \prod_{i=1}^{|X|} \mathbb{1}_{True}(x_i^c) \cdot p_c(x_i|y^k) \cdot \mathbb{1}_{False}(x_i^c) \cdot (1 - p_c(x_i|y^k)), \quad (54)$$

where we also used Equation (53) *and the definition of the indicator function (see Section 2). Based on the above, we can write*

$$\begin{aligned}
\sum_{y \in Y} \tau_Y\left(y \,|\, y^k\right) &\overset{(1)}{=} \sum_{x_1^c \in \{True, False\}} \cdots \sum_{x_{|X|}^c \in \{True, False\}} \tau_Y\left(x_1^c, \ldots, x_{|X|}^c \,|\, y^k\right) \\
&\overset{(2)}{=} \sum_{x_1^c \in \{True, False\}} \cdots \sum_{x_{|X|}^c \in \{True, False\}} \cdot \prod_{i=1}^{|X|} \mathbb{1}_{True}(x_i^c) \cdot p_c(x_i|y^k) \cdot \mathbb{1}_{False}(x_i^c) \cdot (1 - p_c(x_i|y^k)) \\
&\overset{(3)}{=} \sum_{x_1^c \in \{True, False\}} \cdots \sum_{x_{|X|-1}^c \in \{True, False\}} \cdot \prod_{i=1}^{|X|-1} \mathbb{1}_{True}(x_i^c) \cdot p_c(x_i|y^k) \cdot \mathbb{1}_{False}(x_i^c) \cdot (1 - p_c(x_i|y^k)) \\
&\times \left(\sum_{x_{|X|}^c \in \{True, False\}} \mathbb{1}_{True}\left(x_{|X|}^c\right) \cdot p_c\left(x_{|X|}|y^k\right) \cdot \mathbb{1}_{False}\left(x_{|X|}^c\right) \cdot \left(1 - p_c\left(x_{|X|}|y^k\right)\right) \right) \\
&\overset{(4)}{=} \sum_{x_1^c \in \{True, False\}} \cdots \sum_{x_{|X|-1}^c \in \{True, False\}} \cdot \prod_{i=1}^{|X|-1} \mathbb{1}_{True}(x_i^c) \cdot p_c(x_i|y^k) \cdot \mathbb{1}_{False}(x_i^c) \cdot (1 - p_c(x_i|y^k)) \\
&\times \left(p_c\left(x_{|X|} \,|\, y^k\right) + \left(1 - p_c\left(x_{|X|} \,|\, y^k\right)\right)\right) \\
&\overset{(5)}{=} \sum_{x_1^c \in \{True, False\}} \cdots \sum_{x_{|X|-1}^c \in \{True, False\}} \cdot \prod_{i=1}^{|X|-1} \mathbb{1}_{True}(x_i^c) \cdot p_c(x_i|y^k) \cdot \mathbb{1}_{False}(x_i^c) \cdot (1 - p_c(x_i|y^k)) \\
&\overset{(6)}{=} \cdots = \sum_{x_1^c \in \{True, False\}} \mathbb{1}_{True}(x_1^c) \cdot p_c(x_1|y^k) \cdot \mathbb{1}_{False}(x_1^c) \cdot (1 - p_c(x_1|y^k)) \\
&\overset{(7)}{=} p_c(x_1|y^k) + (1 - p_c(x_1|y^k)) = 1, \qquad (55)
\end{aligned}$$

by using the steps below:

(1) *By Equation* (54).
(2) *By Equation* (54).
(3) *By rearranging the sum.*
(4) *By evaluating the last term.*
(5) *By evaluating the last term.*
(6) *Continue iterating Steps* (3)-(5).
(7) *By evaluating the last term.*

This concludes the proof. ■