# A Foraging Strategy with Risk Response for Individual Robots in Adversarial Environments

KAI DI and YIFENG ZHOU, Southeast University
FUHAN YAN, Chongqing University of Posts and Telecommunications
JIUCHUAN JIANG, Nanjing University of Finance and Economics
SHAOFU YANG and YICHUAN JIANG, Southeast University

As an essential problem in robotics, foraging means that robots collect objects from a given environment and return them to a specified location. On many occasions, robots are required to perform foraging tasks in adversarial environments, such as battlefield rescue, where potential adversaries may damage robots with a certain probability. The longer an individual robot moves through adversarial environments, the higher the probability of being damaged by adversaries. The robot system can gain utility only when the robot brings carried objects back to a predetermined home station. Such a risk of being damaged makes returning home at different locations potentially relevant to the expected utility produced by the robot. Thus, the individual robot faces a dilemma when it responds to the potential risks in adversarial environments: whether to return the carried resources home or continue foraging tasks. In this article, two fundamental environment settings are discussed, homogeneous cases and heterogeneous cases. The former is analyzed as having both the optimal substructure property and the non-aftereffect property. Then, we present a dynamic programming (DP) algorithm that can find an optimal solution with polynomial time complexity. For the latter, it is proven that finding an optimal solution is $\mathcal{NP}$-hard. We then propose a heuristic algorithm: A division hierarchical path planning (DHPP) algorithm that is based on the idea of dividing the foraging routes generated initially into a certain number of subroutes to dilute risks. Finally, these algorithms are extensively evaluated in simulations, concluding that in adversarial environments, they can significantly improve the productivity of an individual robot before it is damaged.

CCS Concepts: • **Computing methodologies → Robotic planning;**

Additional Key Words and Phrases: Mobile robot foraging, path planning, robotics in adversarial environments, integer programming, dynamic programming, heuristic

**ACM Reference format:**
Kai Di, Yifeng Zhou, Fuhan Yan, Jiuchuan Jiang, Shaofu Yang, and Yichuan Jiang. 2022. A Foraging Strategy with Risk Response for Individual Robots in Adversarial Environments. *ACM Trans. Intell. Syst. Technol.* 13, 5, Article 83 (June 2022), 29 pages.
https://doi.org/10.1145/3514499

This research is supported by the National Natural Science Foundation of China (62076060, 61932007, 71971109, 61703097), and the Natural Science Foundation of Jiangsu Province of China (BK20201394).
Authors' addresses: K. Di, Y. Zhou, S. Yang, and Y. Jiang (corresponding author), Southeast University, Nanjing 211189, China; emails: {dikai, yfzhou, sfyang, yjiang}@seu.edu.cn; F. Yan, Chongqing University of Posts and Telecommunications, Chongqing 400065, China; email: yanfh@cqupt.edu.cn; J. Jiang (corresponding author), Nanjing University of Finance and Economics, Nanjing 210023, China; email: jcjiang@nufe.edu.cn.

2157-6904/2022/06-ART83 $15.00
https://doi.org/10.1145/3514499

## 1 INTRODUCTION

Robot foraging is an essential task in many real-world robotics applications, e.g., pickup and delivery of objects, rescuing of victims, and gathering of information, where one or more robots will collect the randomly distributed objects and transport them back to a specially designated location, i.e., the home station.

Most existing studies of robot foraging assume that nothing in environments can hinder robots [45, 64]. In some real-world applications, however, robots are required to fetch objects from adversarial environments where potential adversaries exist, which may damage or destroy robots independently [1, 2]. A motivating example is in adversarial information gathering, an **unmanned aerial vehicle** (**UAV**) collects information about enemy forces on battlefields and then returns the information to a home station with the goal of collecting as much information as possible. Due to the existence of patrol agents on battlefields, the UAV faces a certain probability of being discovered by enemies' patrol agents [48]. The more information points visited on battlefields, the higher the probability of such losses. The system will access the information collected only when the UAV returns home. Thus, when the UAV has collected certain information, it is necessary for the UAV to decide whether to continue visiting information points or bring the collected information back to the home station and yield the associated utility in a timely manner.

This research investigates the problem of robot foraging in adversarial environments that are potentially harmful to a robot with an objective to obtain as much utility as possible by fetching and sending back objects from such an environment before being destroyed. Existing foraging strategies that address safe settings are not applicable to foraging tasks in adversarial environments. In safe settings, a robot does not need to return home during foraging to yield timely utility; returning wastes the robot's energy and prolongs the time it takes to collect all objects from a safe environment. However, different from safe settings, the behavior that the robot returns home at the right location in adversarial environments is beneficial to this system and can create a timely utility for the system. When a robot has collected certain objects, it faces a question: whether to return the collected resources home or continue foraging. Therefore, it is important to design suitable algorithms with risk response for such a case.

The challenges that arise when designing foraging strategies for a robot in adversarial environments are twofold: (1) The risk factor adds new constraints to original problems, including how many times the robot returns to the storage point during foraging and where the robot returns, complicating the structure of the solution space; and (2) The order in which foraging points are visited by the robot must be designed, which affects the probability of the robot traveling the assigned foraging route successfully, increasing the dimension of the solution space.

We address these challenges under two fundamental environment settings: homogeneous cases and heterogeneous cases. For the former, all regions in an environment are associated with the same level of risks, and the objects available at each foraging location have the same utility. We find that this homogeneous setting has both the optimal substructure property and the non-aftereffect property. Thus, we propose a **dynamic programming** (**DP**)-based algorithm that can find an optimal solution in polynomial time. For the latter, we prove that finding an optimal solution is $\mathcal{NP}$-hard and thus propose a **Division Hierarchical Path Planning** (**DHPP**) algorithm for solving the issue heuristically in polynomial time. First, this algorithm generates a single route that includes all foraging locations. Second, this algorithm calculates the associated utility gained by following this route heuristically before a hierarchical structure is constructed via iteratively dividing the current route into subroutes. Finally, this algorithm will traverse the entire hierarchical structure, and the route corresponding to the hierarchy with the highest utility will be selected as the final foraging route. These algorithms are evaluated in simulations and compared

to other existing benchmark algorithms for performance in robot foraging [18, 75]. The results show that the proposed algorithms outperform the benchmark methods on scalability and solution quality.

We highlight the main contributions as follows:

(1) We provide a new formulation for the adversarial robot foraging problem where the robot performing foraging tasks may be stopped by adversaries in environments. The formulation extends the basic form of the previous robot foraging problem.
(2) We systematically analyze two common situations in real-life scenarios: homogeneous cases and heterogeneous cases. For the former, we propose a DP algorithm that can return an optimal solution in polynomial time. For the latter, we prove that finding an optimal solution is $\mathcal{NP}$-hard and propose a heuristic algorithm called the DHPP algorithm, the idea of which is to divide the route into a certain number of subroutes to increase the total expected utility.
(3) We evaluate the proposed algorithms in simulations and compare their performance against existing benchmark algorithms for the robot foraging problem in this article. Through both rigid theoretical analyzes and extensive simulations, we demonstrate that the proposed algorithms outperform the benchmark algorithms for the robot foraging problem under adversarial settings. There is a notable phenomenon behind the experimental results, where there does not appear to be a statistical difference in the expected utilities between the optimal results calculated by the optimal solver CPLEX and our proposed DHPP algorithm. In addition, the runtime performance of DHPP has been improved significantly by comparing with CPLEX.

The remainder of this article is organized as follows. Section 2 presents the related work. Section 3 defines the adversarial robot foraging problem. Section 4 presents the proposed algorithms and analyzes them theoretically. Section 5 presents experimental results evaluating these algorithms. Section 6 concludes with a summary and brief overview of future work.

## 2 RELATED WORK

In this section, we introduce the related background and previous studies of the foraging problem in the context of its development. The study of the robot foraging problem originates from the analysis of the foraging behavior of animals in biological systems, and the foraging mechanisms of animals are revealed by modeling and analyzing the observed foraging behavior of animals. Inspired by the foraging mechanisms from biological systems, roboticists have created robot foraging systems that can address complex foraging tasks. As robot foraging systems continue to develop, robots can perform dangerous tasks in place of humans. The risk factors in environments pose new challenges to the traditional robot foraging problem.

### 2.1 Foraging Behavior in Biological Systems: From Behaviors to Mechanisms

Foraging behavior is one of the ubiquitous biological behaviors performed by animals in nature, making it a natural place to look for inspiration [9]. The studies of foraging behavior in biological systems start from behaviors to mechanisms. Through modeling and analyzing observed animal foraging behavior, biological studies have revealed animal foraging mechanisms in natural environments [27].

Driven by the individual or systemic foraging motivations, animals move from their nest to gather food from environments and return it to their nest [49]. Many species have evolved over a long period of time to display behavior that is highly suitable for addressing foraging tasks [50].

In recent years, we have seen an increasing interest in identifying mechanisms from observed foraging behavior in biological systems.

Gautrais et al. [29, 53] focused on identifying the distributed decision-making mechanisms from observed foraging behavior of fish schools. Schools of fish are at risk of being captured by a predator during foraging. Even if they were all aware of the presence and location of the predator, they could not broadcast the decisions they make to all other members in the school, nor do they have a central leader to tell them how to avoid it [53]. Through the distributed decision-making mechanisms embedded in the fish schools, they can operate as a whole to avoid predators during foraging [53].

Deneubourg et al. [20] focused on identifying behavioral mechanisms from the observed foraging behavior of ants. Even though food may be far away, ants can gather food over a large area and then return it to their nest. They can establish paths from the nest to the food, even if none of these ants know where they are or where the food is located [27]. In this foraging process, there is no central leader with all the information making decisions for each agent [9, 20]. After modeling and analyzing the foraging behavior of ants, researchers discovered a leaderless collaboration mechanism [67].

### 2.2 Robot Foraging Problem: From Mechanisms to Behaviors

The foraging mechanisms identified in biological systems are useful starting points for roboticists to construct robot foraging systems, as the restrictions on the individual physical capability are similar [5]. In such a robot foraging system, the robot should develop strategic behaviors relying on information sensing from the given environment, as many biological systems do. Roboticists can leverage the mechanisms learned from biological systems to build intelligent robot foraging systems.

Hunt et al. [37, 60] discovered that bees can transfer sensed information to other members in the bee colony during foraging in unknown environments. Inspired by the distributed communication mechanisms learned from this phenomenon, Alers et al. [4] designed a decentralized robot foraging system with distributed communication rules to sense an unknown environment and manage foraging tasks.

Bonabeau et al. [13, 35] found that termites build new nests based on the distance between food sources in the environment as they gather food, with the goal of reducing moving distance during food transport. Inspired by the termite foraging mechanisms learned from the biological systems, Lu et al. [49] proposed a home location selection algorithm and redesigned the foraging behavior of robots based on the locations of new home stations.

The idealization of robot foraging systems has several real-world applications, including mine-clearing, waste clean-up, and victim rescue [8, 32, 58]. In these applications, foraging robots may be single robots operating individually or multiple robots operating collectively [70]. The single robot foraging problem has many similarities to the **traveling salesman problem** (**TSP**) problem [44, 51, 61], which is $\mathcal{NP}$-complete even for simple graphs, such as grid graphs. More specifically, both problems require the agent to start from a point and then return to that point after visiting certain points. The single robot foraging problem also differs from the TSP problem and its variants in a number of ways. In this article, we also focus on how to design foraging strategies for an individual robot and need to determine whether to return the collected objects home or continue foraging. More than one cyclic path will be generated and not all vertices will be visited. Conversely, the TSP problem and its variants need to generate "one" cyclic path through "all" vertices. These essential differences result in previous approaches to TSP and its variants not applying to the proposed single robot foraging problem.

### 2.3 Risk Factors in Environment: New Challenges for the Robot Foraging Problem

Many of these related studies of the robot foraging problem focus on robot models or resource models, and do not reference the existence of risks in environments [34, 46, 76]. The presence of robots in environments containing risks is becoming more prevalent due to their ability to perform missions accurately, efficiently, and with little risk to humans [1, 63]. In this article, we consider the adversarial robot foraging tasks in adversarial environments, including the capacity-insensitive foraging tasks and the capacity-sensitive ones. Both types of foraging tasks are common in real-world applications, and our study contributes to both of them.

For the capacity-insensitive foraging tasks, the most common example is in the information gathering scenario [74, 75], where the robot has enough storage capacity to store all the information gathered from the environment. In this case, we focus on the challenges that the risk factor brings to such problems and seek to find a foraging route for the robot that maximizes the expected utility. For the capacity-sensitive foraging tasks, the most familiar example is in the robot rescue scene [10, 58], where the robot can only carry a fixed number of victims. In this case, a small capacity may result in a decrease in the number of victims the robot can return, and a large capacity will lead to high manufacturing cost of robots. Then, "how to determine the capacity of the robot for the adversarial foraging events? This becomes a very important issue. Our study provides answers to this important issue.

Articles in the robotic literature that consider the presence of risk factors include [14, 22, 47, 72, 73], and they are primarily concerned with presenting algorithms and methods for risk avoidance. These studies focus on the path planning problem for an individual robot to bypass and avoid risks. Considering the adversarial robot coverage problem as an example, an individual robot needs to cover a given closed area where risks exist. The robot may be damaged while performing coverage tasks [74]. The robot coverage problem is similar to the robot foraging problem, as both require a robot or group of robots to visit points in a given terrain. However, foraging tasks seek to maximize the total utility of the objects returning to the home station (requiring the formation of cyclic paths), while coverage tasks typically seek to maximize the number of locations visited (without the formation of cyclic paths). Other optimization problems related to the proposed adversarial foraging problem include the Canadian Traveler Problem [51, 52], where the goal is to find the expected shortest path between two points in a partially observable graph, some of whose edges may be impassable. Conversely, the graph is fully observable in this problem, and the agent must visit each node in the graph.

Another closely related work was performed by Agmon et al. [74, 75], which introduces risk factors to traditional robot problems. These studies focus on the challenges that arise when traditional robot problems must be solved in the presence of an intention to damage robots. While that study is similar to this study regarding the existence of risks in adversarial environments, the task is inherently different. In [74, 75], the robot is required to travel throughout the environment; in this study, the robot must find the most efficient way to fetch objects from a given adversarial environment without being harmed. This study thus enriches this series of studies by introducing adversarial environments into the robot foraging problem. In addition, our follow-up work to this article studies the collaborative foraging behavior of multi-robots in hazardous environments [23]. The two have different emphases, with this article focusing on robot foraging strategies and the follow-up work [23] focusing more on collaboration between robots.

### 2.4 Comparisons to the Traditional Path Planning Problems

Adversarial robot foraging problem can be regarded as a variant of traditional path planning problem, which has the similarity in finding a set of paths that maximize the collected utility [54, 68].

We now compare our problem with two typical path planning problems (i.e., the orienteering problem and the **vehicle routing problem** (**VRP**)).

*2.4.1 The Orienteering Problem.* A closely-related area of research is represented by the orienteering problem [40, 68], which is a family of problems being focused on determining a path that visits some vertices and maximizes the sum of the collected utilities. In the orienteering problem, the utility is accumulated at a location if the location is visited by the robot [15, 68]. We perform comparative analyzes on the settings about the utility on the orienteering problems, as shown in Table 1.

key difference in this work

Simply visiting this location does not make the robot system gain in the adversarial robot foraging problem. The system gains the corresponding utility only when the object at the location is returned home. If the collected objects are not brought back immediately, the survivability of the robot decreases as the foraging behavior continues, and the expected utility gained by bringing these objects home also decreases.

Through the previous analysis, we can know that the behavior that the robot returns home at the right location in adversarial environments is beneficial to this system and can create a timely utility for the system. When a robot has collected certain objects, it faces a question: whether to return the collected resources home or continue foraging. Therefore, the algorithms designed for the orienteering problem do not apply to the adversarial foraging problem.

*2.4.2 The Vehicle Routing Problem.* A second related branch of previous work is the VRP [54, 57], which seeks to find a set of paths that maximize the quality of service subject to budget or time constraints. In the VRP, the utility is accumulated at a location if the associated demand is satisfied by vehicles without violating any problem-specific constraint. We perform comparative analyzes on the settings about the utility on the VRPs, as shown in Table 2.

The adversarial foraging problem differs from the VRP and its variants in problem settings. In the VRP and its variants, the utility gained by visiting a location is dependent and not related to the travel cost [54, 57]. However, in the adversarial foraging problem, the utility gained by bringing the collected objects home is independent and related to the travel cost (or known as the survival probability). If the collected objects are not brought back immediately, the survivability of the robot decreases as the foraging behavior continues, and the expected utility gained by bringing these objects home also decreases. Thus, the robot faces a dilemma when it carries certain objects: whether to return the carried resources home or continue foraging tasks.

Overall, both the different forms of utility and the risk factors in environment make the algorithms designed for the traditional path planning problem inapplicable to the adversarial foraging problem. To further analyze the adversarial foraging problem, the mathematical formalization of this problem is required.

## 3 PROBLEM FORMULATION AND ANALYSES

### 3.1 Preliminaries

Here we define the *adversarial robot foraging* problem by the following: There is a robot that needs to fetch objects from a target environment $G$, and when foraging resources, the robot may be destroyed by risks in the given environment.

**Environment:** In considering a model for this adversarial environment, we characterize it by a fully connected undirected graph $G = \langle L, E \rangle$. Let $L = \{l_0\} \cup L_t$ be the set of locations in the adversarial environment, where $l_0$ and $L_t = \{l_1, \ldots, l_{|L_t|}\}$ are the home station and the set of foraging locations, respectively [64]. Let $g_i$ be the utility of objects available at location $l_i \in L_t$, and the robot can gain utility $g_i$ by transporting the objects located at $l_i$ to home station $l_0$ [45]. If

Table 1. Comparisons between the Orienteering Problems and the Adversarial Robot Foraging Problem

| The Orienteering Problems | Problem Descriptions | Settings of the Utility |
|---|---|---|
| Classic Orienteering Problem (OP) [68] | The OP can be seen as a combination between two classical combinatorial problems, the Knapsack Problem and TSP. The objective is to maximize the total utility collected from visited (selected) nodes within the limited time budget. | The utility is accumulated at a location if the location is visited by the robot within the limited time budget. |
| The OP with Time Windows (OPTW) [43] | The OPTW considers the time window constraints that arise in the context when the service at a particular node has to start within a predefined time window. | The utility is accumulated at a location if the location is visited by the robot within the predefined time window. |
| Capacitated Orienteering Problem (COP) [6] | The COP is a variant of the OP where each node is associated with a demand and a utility. The main objective is to determine a path for each available vehicle in order to maximize the total utility, without violating the capacity of the robot. | The utility is accumulated at a location if its demend does not violate the capacity of the robot. |
| Stochastic Orienteering Problem (SOP) [15] | The SOP is another variant of the OP where the travel costs between nodes are difficult or even impossible to predict in a deterministic way. | The utility is accumulated at a location if the location is visited by the robot. |
| Generalized Orienteering Problem (GOP) [30] | The Generalized OP (GOP) is a generalized version of the OP in which each node is assigned a set of utilities with respect to a set of attributes. | The multi-attributed utility is accumulated at a location if the location is visited by the robot. |
| Team Surviving Orienteers Problem (TSO) [40] | The TSO aims at maximizing the expected number of nodes collectively visited, subject to the predefined constraints on the probabilities. | The utility is accumulated at a location if the constraint on the survival probability is not violated. |

the robot is stopped before depositing its carried objects at home station $l_0$, it cannot obtain their associated utility. The utility of $g_i$ corresponds to different practical meanings in different real-world scenarios. For example, it may correspond to the associated utility of information available at $l_i$ in information collection applications [17, 56] or the number of victims located at $l_i$ in the rescuing scenarios [8]. Let $E \subseteq L \times L$ be the set of edges between locations.

Table 2. Comparisons between the VRPs and the Adversarial Robot Foraging Problem

| The Vehicle Routing Problems | Problem Descriptions | Settings of the Utility |
|---|---|---|
| Classic Vehicle Routing Problem (VRP) [33] | The goal of the VRP is to optimize the routing design (distribution process from depots to customers) in such a way that customers' demand of goods is satisfied without violating any problem-specific constraint (e.g., route maximum distance or time-related restrictions). | The utility is accumulated at a location if the associated customer is served without violating any predefined constraints. |
| The VRP with Time Windows (VRPTW) [16] | The VRPTW considers the time window constraints that arise in the context when the service at a particular node has to start within a predefined time window. | The utility is accumulated at a location if the location is visited by the robot within the predefined time window. |
| Open Vehicle Routing Problem (OVRP) [69] | The OVRP is a variant of the VRP where the planned routes can end on several points distinct to the depot location. | The utility is accumulated at a location if the associated demand is satisfied by the vehicle. |
| Pickup-and-Delivery Vehicle Routing Problem (PDVRP) [3] | The PDVRP is a variant of the VRP where each customer is associated by two quantities, representing one demand to be delivered at the customer and another demand to be collected by vehicles. | The utility is accumulated at a location if the associated demand (i.e., being delivered or being collected) is satisfied by vehicles. |
| Stochastic Vehicle Routing Problem (SVRP) [31] | The SVRP is a variant of the VRP, where a random behaviour is considered. This is typically the presence of a customer, its demand, its service time or the travel time between customers. | The utility is accumulated at a location if the associated customer is served by the vehicle. |
| Chance-Constrained Vehicle Routing Problem (CCVRP) [25] | The CCVRP is a version of the SVRP with the stochastic program being modeled as a chance constraint program, where a limit is imposed on the travel costs. | The utility is accumulated at a location if it is visited by the vehicle within the predefined travel costs. |

**Risk Factor**: Each edge $e_{ij} \in E$ is associated with a probability $p_{ij}$, which measures the likelihood that a robot traverses from location $l_i$ to $l_j$ successfully. The value of $p_{ij}$ is derived from the risks that may exist in the edge $e_{ij}$ that can destroy the robot. There are two kinds of edges in adversarial environments (i.e., *passable* edges and *impassable* edges), which also correspond to two kinds of survival probabilities. The risk factors associated with these two edges are shown below.

— *Passable* edges: We define the *passable* edges are those with $0 < p_{ij} \leq 1$. For any *passable* edges jointed by $l_i$, $l_j$, and $l_k$, we have $p_{ik} \geq p_{ij} \cdot p_{jk}$ [18]. In the subsequent text, we will refer to this property as the triangle inequality property. Due to the presence of this property, when we determine a trajectory for the foraging robot, going directly to the target point is optimal, and passing through other points (including the visited points) may make the survival probability lower;
— *Impassable* edges: We define the *impassable* edges are those with $p_{ij} = 0$. A robot can move freely along the *passable* edges mentioned above, if the robot cannot always move arbitrarily between two locations in practice, i.e., there are *impassable* edges between two locations, we can set the associated survival probabilities as 0 to describe this case.

**Known Information:** The map of risks and the utility of objects available at each foraging location is determined in advance (e.g., determined through satellite imagery) [41, 66]. With regard to the known risk information, both known cases and unknown ones are common in reality [26, 59, 75]. We can use UAVs to obtain accurate information before foraging [41, 66].

**Objective:** The objective is to maximize the total expected utility gained before the robot is destroyed. To provide a formal formulation of the objective function, we denote the foraging route by the sequence $R = \{l_0, \ldots, l_i, e_{ij}, l_j, \ldots, l_0\}$. The motion from home station $l_0$ to the foraging location and return is considered to be one iteration for the robot. Note that there may be more than one iteration in the final foraging route $R$. Based on the number of iterations within it, route $R$ can be divided into a sequence of subroutes, i.e., $R = \{r_1, r_2, \ldots, r_m\}_{1 \leq m \leq |L_t|}$. Let $m$ denote the number of subroutes in the foraging route, the value of which cannot exceed $|L_t|$ (i.e., the number of foraging locations) and must not be less than 1. We say that a robot survives a subroute if it manages to complete it unharmed. The survivability measure is defined as follows:

*Definition 1.* **Survivability:** the probability that the robot can survive the $k$th subroute $r_k \in R$ is calculated as

$$\phi_k = \prod_{n=1}^{k} \prod_{e_{ij} \in r_n} p_{ij}. \tag{1}$$

Then, the expected utility gained by following the sequence $R = \{r_1, r_2, \ldots, r_m\}_{1 \leq m \leq |L_t|}$ can be presented as

$$E(R) = \sum_{k=1}^{m} \sum_{l_i \in r_k} g_i \phi_k, \tag{2}$$

where $m$ denotes the number of subroutes in foraging route $R$.

Based on the introduced preliminaries, we formulate the definition of the **Foraging Route with the Maximum Expected Utility problem (*FRMEU*)** as follows:

*Definition 2.* Given an adversarial environment represented by a fully connected undirected graph $G$ that contains risks and valuable objects, the FRMEU can be defined as finding a foraging route of the graph with maximum total expected utility.

### 3.2 Mathematical Model

To provide a mathematical formulation of the problem, we first introduce the following notations:

— For each subroute $r_k \in R$ and each edge $e_{ij} \in E$, binary variable $x_{ij}^k = 1$ if the edge $e_{ij}$ appears in subroute $r_k \in R$ and 0 otherwise (note that when $x_{00}^k = 1$, subroute $r_k$ is empty).
— For each subroute $r_k \in R$ and each location $l_i \in L_t$, binary variable $y_i^k = 1$ indicates the location $l_i$ is visited in the subroute $r_k \in R$ and 0 otherwise.

Then, the foraging route $R$ can be presented as a stacked vector containing all the optimization variables defined above,

$$R = [\underbrace{y_1^1, \ldots, y_{|L_t|}^1, x_{00}^1, \ldots, x_{ij}^1, \ldots, x_{|L||L|}^1}_{\text{variables for subroute } r_1}, \ldots, \underbrace{y_1^k, \ldots, y_{|L_t|}^k, x_{00}^k, \ldots, x_{ij}^k, \ldots, x_{|L||L|}^k}_{\text{variables for subroute } r_k}, \ldots, \underbrace{y_1^{|L_t|}, \ldots, y_{|L_t|}^{|L_t|}, x_{00}^{|L_t|}, \ldots, x_{ij}^{|L_t|}, \ldots, x_{|L||L|}^{|L_t|}}_{\text{variables for subroute } r_{|L_t|}}]^T. \tag{3}$$

The *FRMEU* can be then formulated as follows:

$$\max \quad \sum_{k=1}^{|L_t|} \sum_{l_i \in L_t} g_i y_i^k \phi_k, \tag{FRMEU}$$

$$\text{s.t.} \quad \psi_{ij}^k = (1 - p_{ij}) x_{ij}^k, \quad \forall e_{ij} \in E, 1 \le k \le |L_t|, \tag{4}$$

$$\phi_k = \prod_{n=1}^{k} \prod_{e_{ij} \in E} 1 - \psi_{ij}^n, \quad 1 \le k \le |L_t|, \tag{5}$$

is k optimized too?
the number of subrouts

$$\sum_{k=1}^{|L_t|} y_i^k \le 1, \quad \forall l_i \in L_t, \tag{6}$$

$$\sum_{l_j \in L_t} x_{ij}^k = y_i^k, \quad \forall l_i \in L_t, 1 \le k \le |L_t|, \tag{7}$$

$$\sum_{l_i \in L} x_{iv}^k = \sum_{l_j \in L} x_{vj}^k, \quad \forall l_v \in L, 1 \le k \le |L_t|, \tag{8}$$

$$\sum_{l_i \in L} x_{0i}^k = 1, \quad 1 \le k \le |L_t|, \tag{9}$$

$$\sum_{l_i \in L} x_{i0}^k = 1, \quad 1 \le k \le |L_t|, \tag{10}$$

$$x_{ij}^k \in \{0, 1\}, \quad \forall e_{ij} \in E, 1 \le k \le |L_t|, \tag{11}$$

$$y_i^k \in \{0, 1\}, \quad \forall l_i \in L_t, 1 \le k \le |L_t|. \tag{12}$$

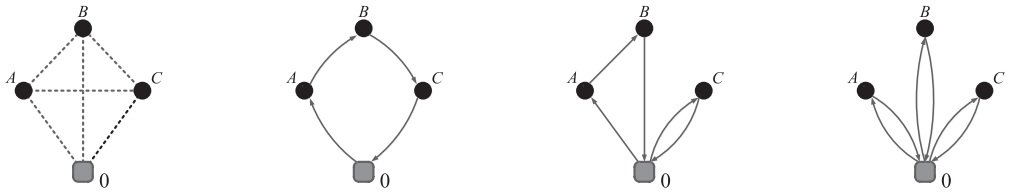
Constraints (4) and (5) represent the accumulated probability that the robot survives the subroute $r_k$. Constraint (6) states that each location should be visited at most once, because revisits will not generate additional utility, and there is no need to go through other locations to access the locations not visited. Constraints (7)–(10) are the flow constraints that describe the subroute $r_k \in R$.

### 3.3 Complexity Analysis

In this subsection, the complexity of the *FRMEU* problem proposed above is discussed. For this problem, finding a foraging route with maximum total expected utility is $\mathcal{NP}$-hard.

Theorem 1. *The FRMEU problem is $\mathcal{NP}$-hard.*

Proof. We first show that the *FRMEU* problem belongs to $\mathcal{NP}$. Given an instance of the problem, we can construct a foraging route and then verify its expected utility in polynomial time.




(a) initial environment (b) $R_1$ (c) $R_2$ (d) $R_3$

Fig. 1. A simple case used to illustrate the inefficiency of previous path planning algorithms. The probability successfully traversing per edge is 0.9 (i.e., $p_{ij} = 0.9, \forall e_{ij} \in E$). The utility for each task is $g_i = 1$.

To prove that *FRMEU* is $\mathcal{NP}$-hard, we will use a reduction from the **Hamiltonian Cycle** (**HC**) problem, which has been proven $\mathcal{NP}$-complete [39]. Let $G^h = \langle L^h, E^h \rangle$ be an arbitrary Graph, and we will show how to answer the question of whether $G^h$ has an HC using an *FREMU* problem with only $O(n^2)$ extra steps.

We form the complete graph $G' = \langle L^h, E' \rangle$, where $E' = \{(l_i, l_j) : l_i, l_j \in L^h, l_i \neq l_j\}$. Without loss of generality, the first element $l_0 \in L^h$ is defined as the home station. Let $g_i = 1$ for all $l_i \in L^h \backslash \{l_0\}$. We define the survival probability function $p_{ij}$ by

$$p_{ij} = \begin{cases} 0, & \text{if } e_{ij} \notin E^h \\ 0.5, & \text{if } e_{ij} \in E^{h0} \\ 1, & \text{if } e_{ij} \in E^h \backslash E^{h0}, \end{cases} \tag{13}$$

where $E^{h0}$ is the set of edges in $E^h$ jointed by home station $l_0$. This construction can be done in polynomial time. (Note that the *passable* edges are those with $0 < p_{ij} \leq 1$ and the triangle inequality property of their survival probabilities is also satisfied in this constructed instance. In other words, we have $p_{ik} \geq p_{ij} \cdot p_{jk}$ for any *passable* edges in $G'$. Therefore, this construction describes a valid *FRMEU* problem.)

We now show that graph $G^h$ has an HC (noted as **Conclusion A**), if and only if the constructed *FRMEU* problem has a foraging route of expected utility at least $\frac{|L^h|-1}{4}$ in $G'$ (noted as **Conclusion B**).

**The "if" direction (B → A):** Assume that the constructed *FRMEU* has a foraging route $h'$ of expected utility at least $\frac{|L^h|-1}{4}$ in $G'$. Since the survival probabilities of the edges in $E'$ are 0, 0.5, and 1, the expected utility of $h'$ is exactly $\frac{|L^h|-1}{4}$, and $h'$ contains only edges in $E^h$. If $h'$ contains more than one cyclic subroute, then there will be subroutes of survival probability at most 1/16, making the expected utility lower than $\frac{|L^h|-1}{4}$. Therefore, $h'$ is a cyclic route that contains only edges in $E^h$. We conclude that $h'$ is an HC in graph $G^h$.

**The "only if" direction (A → B):** Assume that graph $G^h$ has an HC $h$. Since it is a cycle, it returns back to the home station $l_0$. Each edge in $h$ belongs to $E^h$ and thus has accumulated survival probability $0.5 \times 0.5 \times 1 = 1/4$ in $G'$. Thus, $h$ is a foraging route of expected utility $\frac{|L^h|-1}{4}$ in $G'$. □

### 3.4 Inefficiency of Previous Algorithms

There are many classic algorithms for the traditional path planning problems (e.g., the orienteering problem [40, 68], the vehicle routing problem [54, 57]). Then, we use a simple case presented in Figure 1 to analyze the inefficiency of these previous path planning algorithms under our problem settings.

The objective of previous path planning algorithms is to find a route that visits some vertices and maximizes the sum of collected utilities, which is similar to the adversarial foraging problem studied in this article. Without taking into account the risk factors in adversarial environments,

all foraging locations will be included within one single foraging route, and a foraging route with the minimal risk level will be chosen by the previous path planning algorithms. In this simple case presented in Figure 1(a), all edges are associated with the same survival probability, i.e., $p_{ij} = 0.9, \forall e_{ij} \in E$. According to the idea of previous path planning algorithms, the execution of these algorithms yields foraging route $R_1 = \{l_0, e_{0A}, l_A, e_{AB}, l_B, e_{BC}, l_C, e_{C0}, l_0\}$ as shown in the Figure 1(b) with expected utility $E(R_1) = \mathbf{1.968}$. Different from the results computed by the traditional path planning algorithms mentioned above, generating more subroutes may improve the expected utility. As shown in Figure 1(c), foraging route $R_2 = \{\{l_0, e_{0A}, l_A, e_{AB}, l_B, e_{B0}, l_0\}, \{l_0, e_{0C}, l_C, e_{C0}, l_0\}\}$ yields expected utility $E(R_2) = 2 \times 0.9^3 + 1 \times 0.9^5 = \mathbf{2.049} > E(R_1)$. It can be seen that the traditional path planning algorithms cannot achieve good results in this case.

After the above analysis, we can also learn that it is possible to enhance the expected utility by splitting one single route into multiple subroutes. Another new question arises: Is the policy of visiting only one foraging location per subroute an optimal policy? As we can learn from the simple case presented in Figure 1(d), the total expected utility gained by executing one task within one single subroute is $E(R_3) = 1 \times 0.9^2 + 1 \times 0.9^4 + 1 \times 0.9^6 = \mathbf{1.998}$, whereas foraging route $R_2$ yields expected utility $E(R_2) = \mathbf{2.049} > E(R_3)$. It can be seen that the policy of visiting only one foraging location per subroute is not the optimal policy, and further research is needed to find out which is the optimal solution.

Through the above analysis of the simple case, we find some drawbacks that lead to the inefficiency of traditional path planning algorithms in this problem, listed as follows:

- If the robot serves all locations in one single route, the survival probability associated with this strategy can definitely be the highest. As shown in Figure 1(b), the survival probability of following $R_1$ is $0.9^4$, which is larger than the survival probabilities following all other routes. However, in the adversarial foraging problem, the utility gained by bringing the collected objects home is related to the survival probability. When the robot follows $R_1$ presented in Figure 1(b), the utility gained by bringing the objects in the location $l_A$ home is $1 \times 0.9^4$. By comparison, we can learn that it is lower than all other feasible routes. It can be learned from this example that returning the carried resources home at the right location is beneficial to the robot system;
- If the robot executes only one task in each subroute, the survival probability of this strategy is the lowest. As shown in Figure 1(d), the survival probability of following $R_3$ is $0.9^6$, which is the lowest among all other feasible routes. The lowest survival probability makes it difficult for the robot to complete the planned path, resulting in a suboptimal expected utility. It can be learned from this example that choosing the right location to return the carried resources home is an issue worth discussing.

Therefore, the robot faces a dilemma when it carries certain objects: whether to return the carried resources home or continue foraging tasks. We address these challenges in two fundamental environment settings of the adversarial robot foraging problem in this article: homogeneous cases and heterogeneous cases.

## 4 THE FORAGING STRATEGY

In this section, we systematically discuss the *FRMEU* under two fundamental environment settings: homogeneous cases and heterogeneous ones.

### 4.1 An Optimal Solution for Homogeneous Cases

In this section, we study how the robot performs adversarial foraging tasks in adversarial environments, where both the resources utilities and the risk levels are assumed homogeneous. A

motivating example is in adversarial information-gathering applications, where a UAV is required to collect information about opponents on the battlefields; the risks in this adversarial environment exist in the form of enemy agent patrolling [28, 48]. Many studies in multi-agent adversarial patrolling domains generally assume that patrolling points often have the same utilities, and conclude that it is optimal for patrolling agents to visit these patrolling points with the same probabilities [2, 65]. The rationalities of the hypothesis about the homogeneous environments in this motivating example are listed as follows:

— *Hypothesis on the homogeneous resource utilities:* Patrolling points are usually of equal importance in robot adversarial patrolling scenarios [48, 65]. If a UAV is required to collect information in this adversarial environment, the utilities gained by sending any collected information back are the same. Therefore, the resource utilities in the environment are assumed homogeneous;
— *Hypothesis on the homogeneous risk levels:* Based on the above hypothesis on homogeneous resource utilities, many studies have pointed out that it is optimal for patrolling agents to visit these patrol points of equal utility with the same frequency according to Nash equilibrium [2, 28]. The same frequency leads to the same probability of the robot being detected on each edge. Therefore, the risk levels in this adversarial environment are also assumed homogeneous.

From this, it is shown that in many studies, the resource utilities are usually assumed homogeneous. Based on the hypothesis on homogeneous resource utilities, Agmon et al. [2, 28] concluded that homogeneous risk levels result in an optimal adversarial strategy. Without loss of generality, the entire environment is also assumed homogeneous in this section, where all regions are associated with the same risk level (i.e., the survival probability $p_{ij}$ is equal for each edge $e_{ij} \in E$), and the objects available at each foraging location have the same utility (i.e., the utility $g_i$ is also equal for each $l_i \in L_t$).

Depending on the homogeneous features presented in the environment, in this subsection, we discuss how to design foraging strategies for the robot under homogeneous cases. By taking advantage of the properties of the homogeneous environments, we can simplify the proposed original problem as: given $|L_t|$ foraging locations, design a foraging strategy containing $k$ subroutes with the objective of maximizing the resulting expected utility, i.e., $\max_{1\le k\le |L_t|} f(|L_t|, k)$. The specific form of $f(|L_t|, k)$ is

$$f(|L_t|,k) = \begin{cases} |L_t| p^{|L_t|+1}, & \text{if } k = 1 \\ f(|L_t|-1, k-1) + p^{2|L_t|}, & \text{if } |L_t| = k > 1 \\ \max\limits_{1\le n_1\le |L_t|-k+1} n_1 p^{n_1+1} + f(|L_t|-n_1, k-1) p^{n_1+1}, & o.w. \end{cases} \tag{14}$$

where $n_i \in \mathbb{N}^+$ denotes the number of locations visited in the $i$th subroute, the route $R$ gained by solving $f(|L_t|, k)$ can then be presented as $\{n_1, n_2, \ldots, n_k\}$. Each $n_i \in R$ is optimized to achieve the optimization goal. All possible conditions that occur in Equation (14) are analyzed as follows:

(1) When all foraging locations are visited within only one subroute, i.e., $k = 1$, we can obtain the expected utility from executing this path as $f(|L_t|, 1) = |L_t| p^{|L_t|+1}$;
(2) The second possible scenario is that $|L_t| = k$, which indicates that the robot visits only one foraging location and returns in each subroute. In this case, by following this subroute, the robot can gain the expected utility $f(|L_t|, k) = f(|L_t|-1, k-1) + p^{2|L_t|}$;
(3) After analyzing these two special cases, we analyze the more general case where $|L_t| \neq k$. For this case, the final expected utility can be made up of two parts, the expected utility obtained by visiting $n_1 \in \mathbb{N}^+$ foraging locations in the first subroute and the expected utility obtained by visiting the remaining $|L_t| - n_1$ foraging locations, i.e., $f(|L_t| - n_1, k-1) p^{n_1+1}$.

Next, we prove that the optimal goal can be achieved by using the DP algorithm. We begin by analyzing the completeness of the recursive equation $f(|L_t|, k)$, i.e., the foraging route obtained by the DP algorithm must visit all foraging locations.

Lemma 1. *The optimal foraging route must include $|L_t|$ foraging locations.*

Proof. We can use reduction ad absurdum to prove Lemma 1. If the optimal foraging route $R^* = \{n_1^*, n_2^*, \ldots, n_k^*\}$ does not include all foraging locations, i.e., $\sum_{i=1}^{k} n_i^* + \Delta = |L_t|$, $\Delta > 0$, we can construct a new solution $\{n_1^*, \ldots, n_k^*, \Delta\}$ that results in a positive increase on utility of $\Delta \cdot p^{|L_t|+k+1}$. Therefore, the optimal foraging route must include $|L_t|$ foraging locations. □

We next analyze the optimal substructure of the *FRMEU* problem under homogeneous cases. A problem is said to exhibit the optimal substructure property if an optimal solution can be constructed from optimal solutions of its subproblems [21, 38]. This property is used to determine the usefulness of DP algorithms for a problem.

Lemma 2. *The FRMEU problem under homogeneous cases has the optimal substructure property.*

Proof. This problem exhibits the optimal substructure property. In other words, if $\{n_1, n_2, \ldots, n_k\}$ is the optimal solution to the problem $f(|L_t|, k)$, then $\{n_2, \ldots, n_k\}$ is the optimal solution to the subproblem $f(|L_t| - n_1, k-1)$. We can use reductio ad absurdum to prove Lemma 2.

Assume that $\{n_2, \ldots, n_k\}$ is not the optimal solution to the subproblem $f(|L_t| - n_1, k-1)$, then there exists another optimal solution $\{n_2', \ldots, n_k'\}$. From the recursive relationship in Equation (14), we can learn that $f(|L_t|, k) = n_1 p^{n_1+1} + f(|L_t| - n_1, k-1)p^{n_1+1}$ for $k > 1$, where both $n_1$ and $p^{n_1+1}$ have constant value. One can conclude that $\{n_1, n_2', \ldots, n_k'\}$ is the optimal solution to the problem $f(|L_t|, k)$, under the assumption that $\{n_2', \ldots, n_k'\}$ is the optimal solution to this subproblem $f(|L_t| - n_1, k-1)$. This conclusion contradicts the fact that $\{n_1, n_2, \ldots, n_k\}$ is the optimal solution to the problem $f(|L_t|, k)$. In other words, if $\{n_1, n_2, \ldots, n_k\}$ is the optimal solution to the problem $f(|L_t|, k)$, then $\{n_2, \ldots, n_k\}$ is the optimal solution to the problem $f(|L_t| - n_1, k-1)$. Therefore, this problem exhibits the optimal substructure property. □

The non-aftereffect property means that once the state of a certain stage is determined, it is not affected by the state of future decision-making; that is, the process after a state will not affect the previous state but is only related to the current state [11, 12]. We next show that this problem exhibits non-aftereffect.

Lemma 3. *The FRMEU problem under homogeneous cases has the non-aftereffect property.*

Proof. This problem exhibits the non-aftereffect property, if the calculation process of $f(|L_t|, k)$ is independent once the values of $f(|L_t| - i, k-1)$ for all $i \leq |L_t| - k - 1$ are determined. When we need to calculate $f(|L_t|, k)$, we only need to know the values of $f(|L_t| - i, k-1)$ for all $i \leq |L_t| - k - 1$, since $f(|L_t|, k) = \max_{i \leq |L_t|-k-1}\{ip^{i+1} + f(|L_t| - i, k-1)p^{i+1}\}$. Meanwhile, how to calculate the value of $f(|L_t| - i, k-1)$ will not make any difference to the remainder of this problem. In other words, given the state of a stage (i.e., $f(|L_t|, k)$), the development of the process beyond that stage is unaffected by the states of the preceding stages (i.e., $f(|L_t| - i, k-1)$). Therefore, this problem has the non-aftereffect property. □

Through the above analysis, we can finally conclude that the DP algorithm yields an optimal solution to the *FRMEU* problem under homogeneous cases.

Theorem 2. *The DP algorithm can return an optimal solution to the FRMEU problem under homogeneous cases.*

Proof. Because the problem has both the optimal substructure property (Lemma 2) and the non-aftereffect property (Lemma 3), the DP algorithm can return an optimal solution to this problem [19]. □

We can use the state transition equation proposed in Equation (14) to design the DP algorithm. The details of the proposed DP algorithm are shown in Algorithm 1. The mathematical symbols that appear in the pseudo code have been defined above, we do not repeat them here.

**ALGORITHM 1:** *Dynamic Programming Algorithm*

**Input**: Environment topology $G =< L, E >$
**Output**: The optimal expected utility matrix $f(|L_t|, |L_t|)$

```
utility ← 0;
f ← zeros([|L_t|, |L_t|]);        // Initialize the optimal expected utility matrix f(|L_t|, |L_t|)
for j = 1 → |L_t| do
    for i = j → |L_t| do
        if j = 1 then
            f[i][j] ← i p^{i+1};
        else if i = j > 1 then
            f[i][j] ← f[i − 1][j − 1] + p^{2i};
        else
            max ← 0;
            for k = 1 → i − j + 1 do
                utility ← k p^{k+1} + f[i − k][j − 1] p^{k+1};
                if utility ≥ max then
                    max ← utility;
            f[i][j] ← max;
return f(|L_t|, |L_t|);
```

Next, we make some improvements to this DP algorithm to reduce the time complexity of this algorithm. Based on the optimal substructure property of the problem analyzed in Lemma 2 and the state transition equation in Equation (14), we find that the size of the operation can be reduced by constraining the value of $n_1$. In response to this phenomenon, we then present a theorem to bound each element in the final solution, with the goal of reducing the computing scale of the DP algorithm.

Theorem 3. *Let* $R = \{n_1^*, n_2^*, n_3^*, \ldots, n_k^*\}$ *be the optimal solution to problem* $\max_{1 \le k \le |L_t|} f(|L_t|, k)$, *the value of* $n_1^*$ *will not be greater than the value of* $\hat{n_1}$ *returned by solving problem* $f(|L_t|, 2)$.

Proof. We can use reduction ad absurdum to prove Theorem 3. If $n_1^* > \hat{n_1}$ in the optimal route $R^*$, we can construct an instance $R' = \{\hat{n_1}, n_1^* + n_2^* - \hat{n_1}, n_3^*, \ldots, n_k^*\}$, and the other elements $\{n_3^*, \ldots, n_k^*\}$ are the same as the associated element in $R^*$. The expected utility gained by following $R'$ is $E(R') = \hat{n_1} p^{\hat{n_1}+1} + (n_1^* + n_2^* - \hat{n_1}) p^{n_1^*+n_2^*+2} + p^{n_2^*+2} \sum_{i=3}^{k} n_i^* p^{i+\sum_{j=1}^{i} n_j^*}$. Meanwhile, the expected utility gained by following $R$ is $E(R) = n_1^* p^{n_1^*+1} + n_2^* p^{n_1^*+n_2^*+2} + p^{n_2^*+2} \sum_{i=3}^{k} n_i^* p^{i+\sum_{j=1}^{i} n_j^*}$.

We can obtain the difference between the expected utilities gained by following $R$ and $R'$, i.e., $\Delta = E(R') - E(R) = \hat{n_1} p^{\hat{n_1}+1} - n_1^* p^{n_1^*+1} + (n_1^* - \hat{n_1}) p^{n_1^*+n_2^*+2}$. Since $\hat{n_1}$ is returned by solving problem $f(|L_t|, 2)$, we can obtain $\hat{n_1} p^{\hat{n_1}+1} + (|L_t| - \hat{n_1}) p^{|L_t|+2} \ge n_1^* p^{n_1^*+1} + (|L_t| - n_1^*) p^{|L_t|+2}$, i.e., $\hat{n_1} p^{\hat{n_1}+1} - n_1^* p^{n_1^*+1} \ge -(n_1^* - \hat{n_1}) p^{|L_t|+2}$. Then, we can obtain $\Delta \ge (n_1^* - \hat{n_1}) p^{n_1^*+n_2^*+2} - (n_1^* - \hat{n_1}) p^{|L_t|+2}$. Because

$n_1^* + n_2^* \leq |L_t|$ and $0 < p \leq 1$, we can obtain the conclusion $\Delta \geq 0$, i.e., $E(R') \geq E(R)$, which contradicts the assumption $n_1^* > \hat{n_1}$ in $R$. □

Concurrently, the bounds on the number of foraging locations in subroutes obtained analytically in Theorem 3 are also used in the proposed **improved dynamic programming** (**IDP**) algorithm to reduce the running time. The details of the proposed IDP algorithm are shown in Algorithm 2. The IDP algorithm under homogeneous cases has $O(n^3)$ time complexity.

Observation 1. *For the IDP algorithm proposed in Algorithm 2, the use of the bound of each first element derived from Theorem 3 can reduce the running time.*

Two reasons for this observed issue have been summarized as follows:

(1) First, the primary body of the proposed DP algorithm is to iteratively find the first element $n_1$ in the optimal foraging route, as shown in Line 10–15 of Algorithm 1. The above characterization is obtained from the description of the optimal substructure property in Lemma 2. Therefore, tightening the bounds of the first element can effectively reduce the execution time;
(2) Second, the value of $n_1$ returned by solving problem $f(|L_t|, 2)$, i.e., the maximum utility of $n_1^*$ has been calculated in the previous steps and does not need to be repeated. The proposed DP algorithm adopts a structure with memoization.[1] From the above analysis, we know that the solution of any subproblem depends only on the solution of the "smaller" subproblem. We thus rank the subproblems in order of size and solve them in order of smallest to largest. When solving certain subproblems, the upper bound of the first element it depends on has been solved, and the result is saved. When solving the subproblem $\max_{1 \leq k \leq |L_t|} f(|L_t|, k)$, the value of $n_1$ returned by solving problem $f(|L_t|, 2)$, i.e., the maximum value of $n_1^*$ has been calculated in previous steps.

### 4.2 A Division Hierarchical Path Planning Method for Heterogeneous Cases

In real-life scenarios, both the heterogeneous cases discussed in this subsection and the homogeneous cases mentioned above exist simultaneously [16, 55, 64]. In heterogeneous cases, attributes such as risk levels and resource utilities in adversarial environments show heterogeneity. More specifically, all regions in the adversarial environment are associated with a different risk level (i.e., the survival probability $p_{ij}$ is different for each edge $e_{ij} \in E$), and the objects available at each foraging location have a different utility (i.e., the utility $g_i$ is different for each $l_i \in L_t$).

For heterogeneous cases, based on the computational complexity of *FRMEU* analyzed in Theorem 1, it is $\mathcal{NP}$-hard to find an optimal solution of this problem within a polynomial time. To obtain a suitable result in a reasonable time, we propose a heuristic algorithm called the DHPP Algorithm that exploits the idea of the **Hierarchical Clustering** (**HC**) Algorithm [62, 71] in this subsection. Recently, the idea of the HC algorithm has been innovatively used to solve the single robot path planning problem [7, 42]. The underlying purpose of the HC algorithm is to group locations that can gain higher utility into a single cluster. Then, a path planning algorithm will be executed to form a cyclic route that includes all locations within a cluster. The reason why the HC algorithm is appealing for the robot path planning problem is that the HC algorithm is parameter-free, fully automatic and, in particular, requires no assumption with regard to the number of clusters (or the number of subroutes) [71]. In robot path planning problems such as robot foraging problems, it can be difficult to tune the number of cyclic subroutes and even more challenging if a change in the number drastically changes the final results.

[1] *Memoization* is derived from *memo* [19].

**ALGORITHM 2:** *Improved Dynamic Programming Algorithm*

**Input**: Environment topology $G =< L, E >$
**Output**: The optimal expected utility matrix $f(|L_t|, |L_t|)$

$index \leftarrow |L_t|$;
$utility \leftarrow 0$;
$f \leftarrow zeros([|L_t|, |L_t|])$; // Initialize the optimal expected utility matrix $f(|L_t|, |L_t|)$
**for** $j = 1 \rightarrow |L_t|$ **do**
  **for** $i = 1 \rightarrow |L_t|$ **do**
    **if** $j = 1$ **then**
      $f[i][j] \leftarrow ip^{i+1}$;
      $index \leftarrow i$;
    **else if** $i = j > 1$ **then**
      $f[i][j] \leftarrow f[i-1][j-1] + p^{2i}$;
      **if** $j = 2$ **then**
        $index \leftarrow 1$;
    **else**
      $max \leftarrow 0$;
      **if** $j \leq 2$ **then**
        $index \leftarrow i - j + 1$;
      **else**
        **if** $index \geq i - j + 1$ **then**
          $index \leftarrow i - j + 1$;
      **for** $k = 1 \rightarrow index$ **do**
        $utility \leftarrow kp^{k+1} + f[i-k][j-1]p^{k+1}$;
        **if** $utility \geq max$ **then**
          $max \leftarrow utility$;
          $temp \leftarrow k$;
      $f[i][j] \leftarrow max$;
      **if** $j = 2$ **then**
        $index \leftarrow temp$;
**return** $f(|L_t|, |L_t|)$;

In traditional HC algorithms, two typical frameworks are widely used, either by repeatedly merging two smaller clusters into a larger one or by splitting a larger cluster into smaller ones, i.e., the bottom-up method and the top-down method, respectively [24]. The bottom-up method termed the aggregation HC algorithm is similar to the enumeration method under the described problem settings. When following the aggregation HC algorithm, each location will be initially generated as a single cluster and then merged together as a larger cluster following certain rules. However, combining these clusters into new ones following certain rules leads to high time complexity. The running time is primarily spent constructing cyclic paths including all foraging locations in each cluster. The computational complexity of the calculation process is equivalent to calculating the TSP problem. Due to the lack of high time complexity from the bottom-up hierarchical method, we prefer the division HC algorithm (a top-down hierarchical method) for this problem. This algorithm combines all foraging locations into a single route initially. The next step is to split this initial route until each foraging location is identified in a single subroute. The entire running

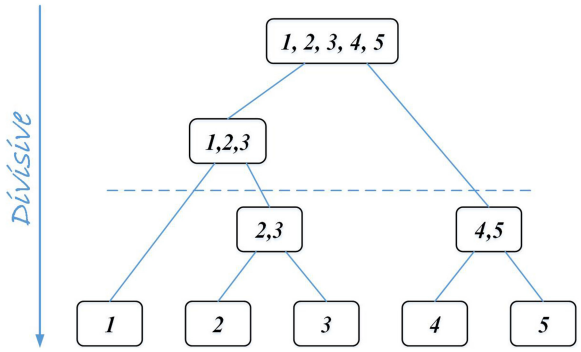



Fig. 2. An example dendrogram for five instances, where horizontal lines indicate a division of one route. The dotted line is an example of using a distance cutoff utility to generate a set of subroutes. In this case, the resulting clusters are {1}, {2, 3}, and {4, 5}.

process of the algorithm can be presented by a dendrogram, as shown in Figure 2. By moving from top to down, a series of division operations are performed. Based on the utility designed before, we can obtain a set of subroutes. The details are shown in Algorithm 3.

In this *DHPP* algorithm, there is no need to determine the number of subroutes in advance. At the beginning of this algorithm, we need to construct a feasible route initially, and this constructed route can then be divided into a certain number of subroutes. The greedy algorithm for solving the traveling salesman problem is used in the proposed algorithm to construct an initial route in a reasonable time, which visits all locations within a cyclic route with suitable risks. After the construction of the initial route, a certain number of pairs of home stations $l_0$ can be added into the initial route greedily to generate subroutes with the aim at improving the total expected utility. The details for the DHPP algorithm are shown as follows:

- **Step 1.** [Lines 1–2] Construct the initial route and initialize the associated parameter. Function Calculate($R$) is used to return the expected utility gained by executing route $R$. Note that $R$ is a list of locations in environment $G$, and there is more than one pair of home station $l_0$ in route $R$ as the algorithm is executed. Thus, we find the sequence of subroutes within route $R$ and then calculate the expected utility gained by executing route $R$ according to Equation (2).
- **Step 2.** [Lines 3–15] Perform the division operation iteratively. The number of subroutes in the foraging route $R$ cannot exceed $|L_t|$ (i.e., the number of foraging locations), and thus the maximum number of iterations is $|L_t|$ times. If dividing route $R$ into subroutes can bring greater expected utility, we will perform this division operation. Otherwise, the loop is terminated and the resulting route $R$ is output. Function Divisive($R, i$) is used to return resulting route $R$ after the division operation, where foraging locations in the generated subroute are ordered by the Greedy-TSP algorithm before being added into the foraging route $R$.

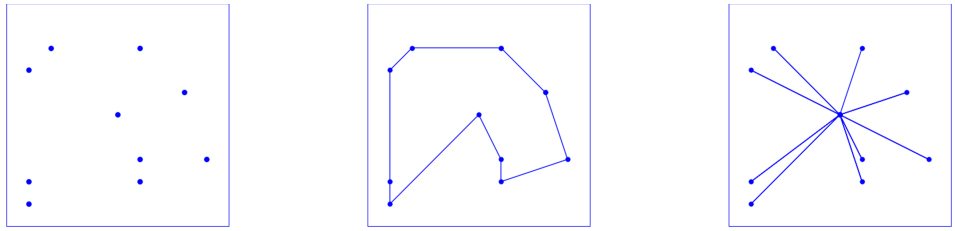
We now present the analysis of the DHPP algorithm and first prove that DHPP is complete, i.e., it generates a path that visits every reachable foraging location in the graph. This means that a robot following this path is guaranteed to visit the entire graph with a certain non-zero probability. We start with the following observation, which proves the completeness of DHPP.

Observation 2 (Completeness). *The procedure DHPP creates a path that visits all the locations in its input. In Line 12 of the procedure, this algorithm splits the current routes into subroutes. Each of these subroutes can generate a cyclic path containing all foraging locations by the greedy TSP method,*

**ALGORITHM 3:** Division Hierarchical Path Planning Algorithm

**Input**: Environment topology $G = \langle L, E \rangle$
**Output**: The foraging route $R$

$R \leftarrow$ Greedy-TSP($\langle L, E \rangle$);
$max \leftarrow$ Calculate($R$);
**for** $num = 1 \rightarrow |L_t|$ **do**
  $index \leftarrow 0$;
  **for** $i = 1 \rightarrow |R| - 1$ **do**
    $R^{temp} \leftarrow$ Divisive($R, i$);
    $utility \leftarrow$ Calculate($R^{temp}$);
    **if** $utility \geq max$ **then**
      $max \leftarrow utility$;
      $index \leftarrow i$
  **if** $index \neq 0$ **then**
    $R \leftarrow$ Divisive($R, index$);
  **else**
    break;
**return** $R$;
**Function** Calculate($R$):
  $utility \leftarrow 0$;
  $p \leftarrow 1$;
  $u \leftarrow 0$;
  **for** $i = 1 \rightarrow |R| - 1$ **do**
    $p \leftarrow p \times p_{R[i-1],R[i]}$; // Calculate the survivability according to Equation (1)
    $u \leftarrow u + g_{R[i]}$;
    **if** $R[i] = l_0$ **then**
      $utility \leftarrow utility + p \times u$; // Calculate the expected utility according to Equation (2)
      $u \leftarrow 0$;
  **return** $utility$;
**Function** Divisive($R, i$):
  $\hat{R} \leftarrow R$;
  $\hat{L} \leftarrow \emptyset$;
  **if** $\hat{R}[i-1] \neq l_0$ *and* $\hat{R}[i] \neq l_0$ **then**
    $\hat{R}.insert(i, l_0)$; // Insert home station $l_0$ into position $i$ of route set $\hat{R}$
    **for** $j = i + 1 \rightarrow |\hat{R}| - 1$ **do**
      $\hat{L}.add(\hat{R}[j])$;
      **if** $\hat{R}[j] = l_0$ **then**
        $r \leftarrow$ Greedy-TSP($\langle \hat{L}, E \rangle$);
        $\hat{R}.replace(i, j, r)$; // Replace the element from $\hat{R}[i]$ to $\hat{R}[j]$ in route set $\hat{R}$ with $r$
        break;
    **return** $\hat{R}$;



(a) initial environment (b) $R_1$ (c) $R_2$

Fig. 3. A simple case used to illustrate the survival probability of the foraging route calculated by DHPP. The probability successfully traversing per edge is 0.9 (i.e., $p_{ij} = 0.9, \forall e_{ij} \in E$). $R_1$ is the foraging route with the maximum survival probability $P = 0.9^{10}$, and $R_2$ is the foraging route with the minimum survival probability $P = 0.9^{18}$.

*which is known to be complete. Because all locations in L are reachable from the starting location $l_0$ (this is a precondition on the input), there must be a connecting edge between all locations in L. Thus, the DHPP algorithm creates a path that visits all the foraging locations in its input L.*

We now discuss the run-time complexity of DHPP.

Theorem 4. *Let n be the number of foraging locations in the group $G = \langle L, E \rangle$ given as input to DHPP. DHPP can find a foraging route of G in $O(n^5)$ time.*

Proof. First, DHPP runs the Greedy-TSP Algorithm on the graph induced from the given adversarial environment with computation complexity $O(n^2)$. Second, DHPP runs the function Divisive$(R, i)$ on the connected area, which divides a cyclic route including all foraging locations into subroutes in $O(n^3)$ time. Third, DHPP iterates the division process $O(n^2)$ times until the expected utility will no longer increase, yielding a total of $O(n^5)$. Thus, the entire computation complexity of DHPP is $O(n^5)$. □

We now establish bounds on the completion probability of the foraging route generated by DHPP. The following theorem provides more specific bounds on the survival probability of the foraging route returned by the DHPP for various types of environments.

Theorem 5. *Let $P_i$ denote the set that includes the survival probabilities of successfully traversing from location $l_i$ to any other locations in location set L. Denote the maximum survival probability in $P_i$ be $P_{i1} = \max(P_i)$ and the second-largest survival probability in $P_i$ be $P_{i2} = \max(P_i \setminus \max(P_i))$.The survival probability of traversing the path $\{l_0, e_{0i}, l_i, e_{i0}, l_0\}$ can be calculated as $p_{0i}^2$ for all $l_i \in L_t$. Let P be the survival probability of completing the final foraging route. Then, DHPP can calculate a foraging route associated with the survival probability $\prod_{i=1}^{|L_t|} p_{0i}^2 \leq P \leq \prod_{i=0}^{|L_t|} \sqrt{P_{i1}P_{i2}}$.*

Proof. For the upper bound of the survival probability of the final foraging route, the cyclic route with no subroutes can yield the largest survival probability, as shown in Figure 3(b). Due to the constraint that the survival probability of each edge in the environment satisfies the triangle inequality, i.e., $p_{ij} \geq p_{ik} \cdot p_{kj}$, the additional construction for subroutes will reduce the final survival probability. Therefore, traveling through all foraging locations via only one cyclic route yields the maximum survival probability. According to the characteristics of the cyclic path, each foraging location has two neighboring edges, one in and one out. Assuming that the survival probabilities of going to each foraging location and leaving from it are the maximum as well as the second-largest values, we can then obtain the upper bound of the survival probability as $\prod_{i=0}^{|L_t|} \sqrt{P_{i1}P_{i2}}$.

For the lower bound of the survival probability of the final foraging route, the foraging route that contains $|L_t|$ subroutes can yield the lowest survival probability, as shown in Figure 3(c).

Due to the constraint that the survival probability of each edge in the environment satisfies the triangle inequality, i.e., $p_{ij} \geq p_{ik} \cdot p_{kj}$, the combination of subroutes will improve the final survival probability. Therefore, traveling through $|L_t|$ cyclic paths formed by one single foraging location $l_i \in L_t$ and the home location $l_0$ yields the lowest survival probability. We can obtain the lower bound of the survival probability as $\prod_{i=1}^{|L_t|} p_{i0}^2$. □

## 5 EXPERIMENTS

We now conduct simulation experiments for the proposed approaches in various settings by comparing them with the previous benchmark approaches. We verify the scalability and solution quality of the proposed algorithm on these simulations. All computations are performed on a 64-bit PC with a dual-core 3.60 GHz CPU and 16 GB memory. Real-time experiments are conducted here, and the robot will adjust its strategy in time according to different environmental instances. All results are averaged over 5,000 instances. Each record is statistically significant at 95% confidence level, and the error bars in the figure represent the 95% confidence interval.

### 5.1 Benchmark Approaches for Comparison

In the experiments, we compare the proposed adversarial robot foraging algorithms against the three benchmark approaches: the **Greedy Adversarial Coverage** (**GAC**) Algorithm, the **Risk-Aware Graph Search** (**RAGS**) Algorithm, and the optimal result *Cpl* calculated by solver CPLEX (version 12.6). We outline these three benchmark algorithms as follows:

- The *GAC* algorithm is a conventional approach that has been used in previous risk-aware path planning problems [75]. In this approach, the entire environment $G$ is first split into a certain number of parts based on the survival probability of each edge $e_{ij} \in E$ when following this algorithm. The robot will then perform foraging tasks along the cyclic path formed by the edges with the highest survival probability. After successfully traversing these edges and taking the obtained objects home, the robot will go on performing foraging tasks along the cyclic path formed by the edges with the second highest survival probability. The robot will then repeat the process mentioned above until the end;
- The *RAGS* algorithm is an efficient method to solve the risk-aware robotic path planning problem [18], the idea of which is to use a total ordering of vertices based on the calculated cost-to-arrive to find a single optimal path between defined start and goal vertices. In this series of experiments, both the start and goal vertices are set as home station $l_0$, because the robot needs to start from the home station $l_0$ and return the foraged objects home under these problem settings;
- The *Cpl* is denoted as the result returned by solving the *FRMEU* with the optimal solver CPLEX (version 12.6).

### 5.2 Performance Indices for Comparison

Based on the optimization objective of this article, we define below four indices to evaluate the performances of the proposed approaches, in comparison with previous benchmark approaches:

- *Expected Utility*: We define this index to evaluate the proposed optimization objective of maximizing the expected utility gained by following the given foraging route. This index is used to measure the solution quality of the foraging strategies.
- *Task Completion Proportion*: Based on Equation (1), we assume that the probability of successfully fetching the objects located at $l_i$ back is $p_i$ and that there are a total of $m$ foraging locations. The completion proportion of all foraging tasks is defined as $\sum_{i=1}^{m} p_i/m$.

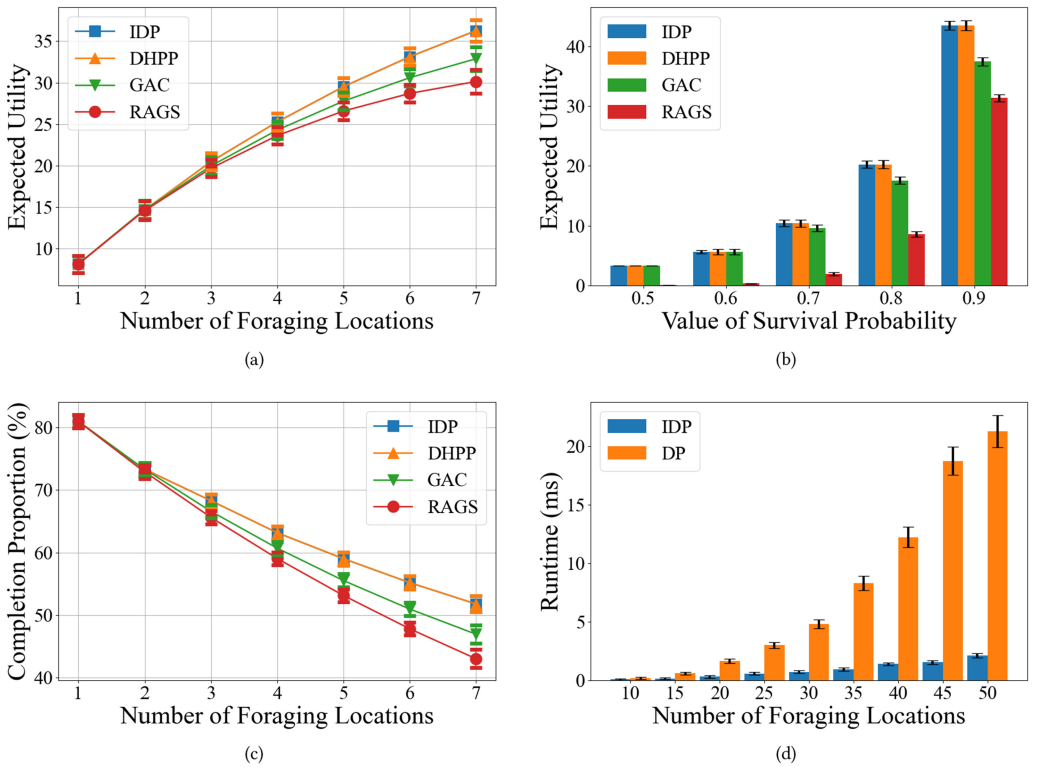



Fig. 4. The performance of our approach and baseline algorithms. Test on the Influence of Number of Foraging Locations: (a); Test on the Influence of Survival Probability: (b); Test on the Task Completion Proportion: (c); Test on the Running Time: (d).

— *Robustness*: Robustness means that the performance of an algorithm should not be significantly affected by changes in the data [36]. This index is introduced here to measure the robustness of the foraging strategies under different environmental settings, including different numbers of foraging locations, different values of survival probability, and different values of utility.
— *Running Time of Algorithm*: This index is used to measure the scalability and computational efficiency of the foraging strategies.

## 5.3 Test on the Homogeneous Cases

*5.3.1 Test on the Influence of the Number of Foraging Locations.* In this subsection, we vary the problem scale to test the effectiveness of the IDP algorithm presented in Algorithm 2 and the other three benchmark algorithms, where the IDP algorithm can return an optimal solution as analyzed in Theorem 2, and the DHPP algorithm is the heuristic algorithm presented in Algorithm 3 for the heterogeneous cases. The utility available at each foraging location $l_i \in L_t$ is set as 10, and the survivability probability $p_{ij}$ corresponding with the likelihood that a robot traverses between location $l_i$ and $l_j$ successfully is set as 0.9.

Figure 4(a) shows the result on the expected utility gained by adapting the IDP algorithm and the other three benchmark algorithms with the number of foraging locations $|L_t|$ varying from 1 to 7. Figure 4(a) shows that both the proposed IDP algorithm and DHPP algorithm outperform the other

two benchmark algorithms. Also, the expected utility obtained by executing the DHPP algorithm is near the optimal result calculated by IDP. The IDP has been proven to return an optimal solution to robots under homogeneous cases and therefore achieves better performance than the tested methods. The DHPP outperforms the other two benchmarks because it takes returning home at the right location into consideration. The order in which the foraging points are visited also does not need to be considered under homogeneous cases and will not influence the survivability of the robot. This characteristic under homogeneous cases makes the DHPP algorithm achieve similar performance to the IDP algorithm.

*5.3.2 Test on the Influence of Survival Probability.* In this subsection, we test the effectiveness of the IDP algorithm presented in Algorithm 2, the DHPP algorithm presented in Algorithm 3 and the other two benchmark algorithms with the survival probability $p_{ij}$ of each edge $e_{ij} \in E$ varying from 0.5 to 0.9. The number of foraging locations $|L_t|$ is set as 10.

Figure 4(b) shows the changes in the expected utility gained by adapting the IDP algorithm, the DHPP algorithm, and the other two benchmark algorithms with the survival probability $p_{ij}$ of each edge $e_{ij} \in E$ varying from 0.5 to 0.9. We can also learn that both the proposed IDP algorithm and the DHPP algorithm outperform the other two benchmark algorithms, and the expected utility obtained by executing the DHPP algorithm is near the optimal result calculated by IDP. The conclusions are the same as those analyzed in the previous subsection. Whether we vary the number of foraging locations $|L_t|$ or the value of survival probability $p_{ij}$, these series of experiments yield the same conclusions.

*5.3.3 Test on the Task Completion Proportion.* In this subsection, we vary the problem scale to test the task completion proportion of the IDP algorithm presented in Algorithm 2 and the other three benchmark algorithms, where the IDP algorithm can return an optimal solution as analyzed in Theorem 2, and the DHPP algorithm is the heuristic algorithm presented in Algorithm 3 for the heterogeneous cases. The survivability probability $p_{ij}$ corresponding with the likelihood that a robot traverses between location $l_i$ and $l_j$ successfully is set as 0.9.

Figure 4(c) shows the result on the task completion proportion gained by adapting the IDP algorithm and the other three benchmark algorithms with the number of foraging locations $|L_t|$ varying from 1 to 7. Figure 4(c) shows that both the proposed IDP algorithm and DHPP algorithm outperform the other two benchmark algorithms. Also, the expected utility obtained by executing the DHPP algorithm is near the optimal result calculated by IDP. The IDP has been proven to return an optimal solution to robots under homogeneous cases and therefore yields better performance than the tested methods. The reason why the DHPP outperforms the other three benchmarks is similar to the reason analyzed in the above subsection, which will not be repeated in this study.

*5.3.4 Test on the Running Time.* In this subsection, we vary the problem scale to test the running time of the DP algorithm presented in Algorithm 1 and the IDP algorithm presented in Algorithm 2. The survivability probability $p_{ij}$ associated with the likelihood that a robot traverses between location $l_i$ and $l_j$ successfully is set as 0.9.

Figure 4(d) demonstrates the change in running time of DP and IDP with the number of foraging locations varying from 10 to 50. The error bars in the figure represent the 95% confidence interval. Both the DP algorithm and the IDP algorithm have $O(n^3)$ time complexity, all of which can be used in large-scale cases. However, Figure 4(d) shows that the IDP algorithm has a lower running time than the DP algorithm under the same problem scale. As mentioned in Section 4.1, the use of the bound of each first element in the IDP algorithm can effectively reduce the running time. This conclusion is mentioned in Observation 1, and is also evident in the experimental results.

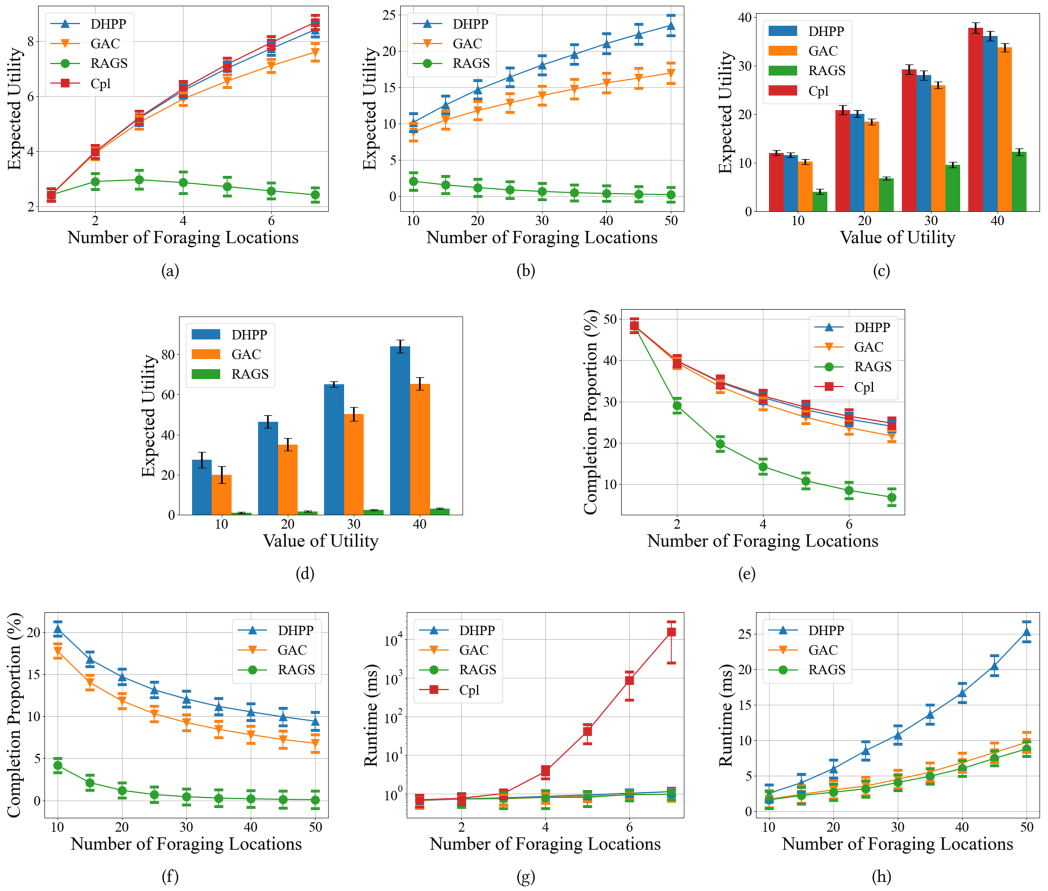



Fig. 5. The performance of our approach and baseline algorithms. Test on the Influence of Problem Scale: (a) and (b); Test on the Influence of utility Range: (c) and (d); Test on the Task Completion Proportion: (e) and (f); and Test on the Running Time: (g) and (h).

## 5.4 Test on the Heterogeneous Cases

*5.4.1 Test on the Influence of Problem Scale.* In this subsection, we vary the problem scale to evaluate the effectiveness of the DHPP algorithm and the other three benchmark algorithms. The CPLEX solver requires much more time than the others; therefore, we divide this series of experiments into two cases, i.e., small-scale cases, and large-scale cases. The utility available at each foraging location $l_i \in L_t$ is randomly assigned within the interval $g_i \in [5, 25]$. The survivability probability $p_{ij}$ associated with the likelihood that a robot traverses between location $l_i$ and $l_j$ successfully is a random value between the interval $(0, 1]$.

For the small-scale cases, we vary the number of foraging locations $|L_t|$ from 1 to 7 and present the test results in Figure 5(a). First, we find that the proposed DHPP algorithm outperforms both the GAC algorithm and the RAGS algorithm. The expected utility obtained by executing the DHPP algorithm is near the optimal result *Cpl* calculated by solver CPLEX. For the large-scale cases, we alter the number of foraging locations $|L_t|$ from 10 to 50, and the associated test results are shown in Figure 5(b). Similar to the above results, the DHPP algorithm outperforms the other benchmark algorithms. Let us present a simple case to explain why the final utility obtained by following

the RAGS algorithm goes down as the scale of the problem increases. Assuming that the survival probability of each edge is set as a constant variable $p$, then the utility calculated by the RAGS algorithm equals $|L_t|p^{|L_t|+1}$. The value of final utility begins to decrease when the number of foraging locations $|L_t|$ grows greater than $\lceil \frac{p}{1-p} \rceil$.

*5.4.2 Test on the Influence of Utility Range.* Figure 5(c) and (d) shows the performance of the proposed approach and benchmark algorithms with the upper bound of $g_i$ varying from 10 to 40 under settings where the value of $|L_t|$ equals 5 and 30, respectively. The survivability probability $p_{ij}$ associated with the likelihood that a robot traverses between locations $l_i$ and $l_j$ successfully is a random value between the interval $(0, 1]$. The error bars in the figures represent the 95% confidence interval.

We can learn from Figure 5(c) and (d) that the proposed DHPP algorithm outperforms the other algorithms except *Cpl*. The reason why the RAGS algorithm does not work well under this problem setting is that it can only find a single route with the lowest risk without considering generating a certain number of cyclic routes to obtain timely utility. The GAC algorithm considers returning home to avoid risk, but it does not take the associated utility into account when generating subroutes. As mentioned above, the GAC algorithm generates subroutes based on the survival probability of the edge.

*5.4.3 Test on the Task Completion Proportion.* In this subsection, we vary the problem scale to test the task completion proportion of the DHPP presented in Algorithm 3 for the heterogeneous cases and the other three benchmark algorithms, where the *Cpl* is an optimal solution calculated by the solver CPLEX. The survivability probability $p_{ij}$ associated with the likelihood that a robot traverses between locations $l_i$ and $l_j$ successfully is a random value between the interval $(0, 1]$.

Figure 5(e) and (f) shows the results on the task completion proportion gained by adapting the DHPP algorithm and the other three benchmark algorithms with different numbers of foraging locations $|L_t|$. Figure 5(e) and (f) shows that the proposed DHPP algorithm outperforms the other three benchmark algorithms in terms of task execution completion proportion. The task completion proportion obtained by executing the DHPP algorithm is near the optimal result calculated by *Cpl*. The reason why the DHPP outperforms the other three benchmarks is similar to the reason analyzed in the above subsection, which will not be repeated in this study.

*5.4.4 Test on the Running Time.* Figure 5(g) and (h) demonstrates the change in running time of the four algorithms mentioned above under the small-scale cases and the large-scale cases, respectively. The time complexities of the DHPP, GAC, and RAGS algorithms are all polynomial, and all can be used for the large-scale case. Also, the proposed DHPP algorithm has $O(n^5)$ time complexity, as mentioned in Section 4.2. The GAC algorithm first divides the entire environment into several parts within $O(n^2 \log n)$ time complexity and then generates routes within $O(n^2)$ time complexity. Therefore, the GAC algorithm has time complexity $O(n^2 \log n)$. The RAGS algorithm is an improved method based on the naive $A^*$ algorithm, both of which have similar time complexity.

## 6 CONCLUSION

This paper aims at addressing two challenges that arise when designing foraging strategies for the robot performing foraging tasks in adversarial environments: (1) The risk factor adds new constraints to traditional problems, complicating the structure of the solution space; and (2) the order in which the robot visits foraging points affects the probability of its survival, making the dimension of the solution space increase. We address these challenges in two fundamental environment settings of the adversarial robot foraging problem: homogeneous cases and heterogeneous cases.

For the former, we present a DP algorithm that can find an optimal solution with polynomial time complexity. For the latter, we propose a heuristic algorithm: a DHPP.

This article performs theoretical analyzes and extensive experiments to prove the effectiveness and efficiency of the proposed approaches. First, the optimization objective of the proposed approaches of increasing the resulting expected utility is validated by comparing with the benchmark risk-aware path planning algorithms, concluding that, in adversarial environments, they can significantly increase the utility of objects returned by a robot before it is stopped. In particular, there does not appear to be a statistical difference in the expected utilities between the optimal results calculated by the optimal solver CPLEX and our proposed DHPP algorithm. Then, it is shown that the proposed approaches can consume less running time than the optimal result calculated by the CPLEX solver (version 12.6).

In the future, there are several areas that we plan to pursue. First, we are interested in handling the adversarial robot foraging problem in unknown environments, where the foraging must be completed without the use of a map of the area. Second, we would like to consider non-stationary environments, where the associated survival probability of each foraging location can vary over time. Third, we will extend our conclusions on how to choose a reasonable capacity for capacity-sensitive foraging tasks to the case of fixed capacity. With the addition of tight constraints on the capacity, the robot needs to consider not only its survival probability but also the relationship between its capabilities and the task demands when performing foraging tasks. In addition, we would like to extend the algorithms for multi-robot foraging systems. The use of multiple robots for foraging has the potential to improve foraging efficiency and greater robustness; even if one robot is totally damaged, others may take over its foraging subtask.

## REFERENCES

[1] Noa Agmon. 2017. Robotic strategic behavior in adversarial environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 5106–5110.

[2] Noa Agmon, Gal A. Kaminka, and Sarit Kraus. 2011. Multi–robot adversarial patrolling: Facing a full-knowledge opponent. *Journal of Artificial Intelligence Research* 42, 1 (2011), 887–916.

[3] The Jin Ai and Voratas Kachitvichyanukul. 2009. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers and Operations Research* 36, 5 (2009), 1693–1702.

[4] Sjriek Alers, Daan Bloembergen, Daniel Hennes, Steven De Jong, Michael Kaisers, Nyree Lemmens, Karl Tuyls, and Gerhard Weiss. 2011. Bee-inspired foraging in an embodied swarm. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 1311–1312.

[5] Sjriek Alers, Daniel Claes, Karl Tuyls, and Gerhard Weiss. 2014. Biologically inspired multi-robot foraging. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*. 1683–1684.

[6] Claudia Archetti, Nicola Bianchessi, and M. Grazia Speranza. 2013. The capacitated team orienteering problem with incomplete service. *Optimization Letters* 7, 7 (2013), 1405–1417.

[7] Omur Arslan, Dan P. Guralnik, and Daniel E. Koditschek. 2016. Coordinated robot navigation via hierarchical clustering. *IEEE Transactions on Robotics* 32, 2 (2016), 352–371.

[8] Chris A. B. Baker, Sarvapali Ramchurn, W. T. Luke Teacy, and Nicholas R. Jennings. 2016. Planning search and rescue missions for UAV teams. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 1777–1782.

[9] Tucker Balch and Maria Hybinette. 2000. Social potentials for scalable multi–robot formations. In *Proceedings of the International Conference on Robotics and Automation*. 73–80.

[10] Zoltán Beck, Luke Teacy, Alex Rogers, and Nicholas R. Jennings. 2016. Online planning for collaborative search and rescue by heterogeneous robot teams. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 1024–1033.

[11] Richard Bellman. 1966. Dynamic programming. *Science* 153, 3731 (1966), 34–37.

[12] Richard E. Bellman and Stuart E. Dreyfus. 2015. *Applied Dynamic Programming*. Princeton University Press.

[13] Eric Bonabeau, Guy Theraulaz, J.-L. Deneubourg, Nigel R. Franks, Oliver Rafelsberger, J.-L. Joly, and Stephane Blanco. 1998. A model for the emergence of pillars, walls and royal chambers in termite nests. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 353, 1375 (1998), 1561–1576.

[14] Scott A. Bortoff. 2000. Path planning for UAVs. In *Proceedings of the American Control Conference*. IEEE, 364–368.
[15] Ann M. Campbell, Michel Gendreau, and Barrett W. Thomas. 2011. The orienteering problem with stochastic travel and service times. *Annals of Operations Research* 186, 1 (2011), 61–81.
[16] Zhiguang Cao, Hongliang Guo, and Jie Zhang. 2017. A multiagent-based approach for vehicle routing by considering both arriving on time and total travel time. *ACM Transactions on Intelligent Systems and Technology* 9, 3 (2017), 1–21.
[17] Shih-Fen Cheng, Cen Chen, Thivya Kandappu, Hoong Chuin Lau, Archan Misra, Nikita Jaiman, Randy Tandriansyah, and Desmond Koh. 2017. Scalable urban mobile crowdsourcing: Handling uncertainty in worker movement. *ACM Transactions on Intelligent Systems and Technology* 9, 3 (2017), 1–24.
[18] Jen Jen Chung, Andrew J. Smith, Ryan Skeele, and Geoffrey A. Hollinger. 2019. Risk-aware graph search with dynamic edge cost discovery. *International Journal of Robotics Research* 38, 2–3 (2019), 182–195.
[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts.
[20] J. L. Deneubourg, Serge Aron, Simon Goss, and Jacques M. Pasteels. 1990. The self-organising exploratory pattern of the argentine ant. *Journal of Insect Behavior* 3, 2 (1990), 159–168.
[21] Kevin W. DeRonne and George Karypis. 2013. Pareto optimal pairwise sequence alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 10, 2 (2013), 481–493.
[22] Kai Di, Shaofu Yang, Wanyuan Wang, Fuhan Yan, Haokun Xing, Jiuchuan Jiang, and Yichuan Jiang. 2019. Optimizing evasive strategies for an evader with imperfect vision capacity. *Journal of Intelligent & Robotic Systems* 96, 3–4 (2019), 419–437.
[23] Kai Di, Yifeng Zhou, Jiuchuan Jiang, Fuhan Yan, Shaofu Yang, and Yichuan Jiang. 2022. Risk-aware collection strategies for multirobot foraging in hazardous environments. *ACM Transactions on Autonomous and Adaptive Systems* 1, 1 (2022), 1–39.
[24] Chris Ding and Xiaofeng He. 2002. Cluster merging and splitting in hierarchical clustering algorithms. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE, 139–146.
[25] Thai Dinh, Ricardo Fukasawa, and James Luedtke. 2018. Exact algorithms for the chance-constrained vehicle routing problem. *Mathematical Programming* 172, 1 (2018), 105–138.
[26] John A. Doucette, Graham Pinhey, and Robin Cohen. 2016. Multiagent resource allocation for dynamic task arrivals with preemption. *ACM Transactions on Intelligent Systems and Technology* 8, 1 (2016), 1–27.
[27] Ofer Feinerman and James F. A. Traniello. 2016. Social complexity, diet, and brain evolution: Modeling the effects of colony size, worker size, brain size, and foraging behavior on colony fitness in ants. *Behavioral Ecology and Sociobiology* 70, 7 (2016), 1063–1074.
[28] Natalie Fridman, Doron Amir, Yinon Douchan, and Noa Agmon. 2019. Satellite detection of moving vessels in marine environments. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 9452–9459.
[29] Jacques Gautrais, Christian Jost, and Guy Theraulaz. 2008. Key behavioural factors in a self-organised fish school model. In *Proceedings of the Annales Zoologici Fennici*. 415–428.
[30] Zong Woo Geem, Chung-Li Tseng, and Yongjin Park. 2005. Harmony search for generalized orienteering problem: Best touring in China. In *Proceedings of the International Conference on Natural Computation*. Springer, 741–750.
[31] Michel Gendreau, Gilbert Laporte, and René Séguin. 1996. Stochastic vehicle routing. *European Journal of Operational Research* 88, 1 (1996), 3–12.
[32] Brian P. Gerkey and Maja J. Matarić. 2004. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research* 23, 9 (2004), 939–954.
[33] Bruce L. Golden, Subramanian Raghavan, and Edward A. Wasil. 2008. *The Vehicle Routing Problem: Latest Advances and New Challenges*, Vol. 43. Springer Science & Business Media.
[34] John Harwell. 2019. A unified mathematical approach for foraging and construction systems in a 1,000,000 robot swarm. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 6438–6439.
[35] Bert Hölldobler and Edward O. Wilson. 1990. *The Ants*. Harvard University Press.
[36] Peter J. Huber. 2004. *Robust Statistics*, Vol. 523. John Wiley & Sons.
[37] Greg J. Hunt, R. E. Page, M. Kim Fondrk, and Charles J. Dullum. 1995. Major quantitative trait loci affecting honey bee foraging behavior. *Genetics* 141, 4 (1995), 1537–1545.
[38] Ilias Iliadis. 2000. Optimal PNNI complex node representations for restrictive costs and minimal path computation time. *IEEE/ACM Transactions on Networking* 8, 4 (2000), 493–506.
[39] David S. Johnson. 1985. The NP-completeness column: An ongoing guide. *Journal of Algorithms* 6, 3 (1985), 434–451.
[40] Stefan Jorgensen, Robert H. Chen, Mark B. Milam, and Marco Pavone. 2018. The team surviving orienteers problem: Routing teams of robots in uncertain environments with survival constraints. *Autonomous Robots* 42, 4 (2018), 927–952.
[41] Asif Khan, Evsen Yanmaz, and Bernhard Rinner. 2014. Information merging in multi-UAV coperative search. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 3122–3129.

[42] Dana Kulić, Wataru Takano, and Yoshihiko Nakamura. 2008. Incremental learning, clustering and hierarchy formation of whole body motion patterns using adaptive hidden markov chains. *The International Journal of Robotics Research* 27, 7 (2008), 761–784.
[43] Nacima Labadie, Jan Melechovskỳ, and Roberto Wolfler Calvo. 2011. Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics* 17, 6 (2011), 729–753.
[44] Gilbert Laporte, Ardavan Asef-Vaziri, and Chelliah Sriskandarajah. 1996. Some applications of the generalized traveling salesman problem. *Journal of the Operational Research Society* 47, 12 (1996), 1461–1467.
[45] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J. Matarić. 2006. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research* 25, 3 (2006), 225–241.
[46] Somchaya Liemhetcharat, Rui Yan, Rui Yan, and Keng Peng Tee. 2015. Continuous foraging and information gathering in a multi-agent team. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 1325–1333.
[47] Maxim Likhachev and Anthony Stentz. 2007. Goal directed navigation with uncertainty in adversary locations. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*. IEEE, 4127–4134.
[48] Efrat Sless Lin, Noa Agmon, and Sarit Kraus. 2019. Multi-robot adversarial patrolling: Handling sequential attacks. *Artificial Intelligence* 274, 1 (2019), 1–25.
[49] Qi Lu, Joshua P. Hecker, and Melanie E. Moses. 2016. The MPFA: A multiple-place foraging algorithm for biologically-inspired robot swarms. In *Proceedings of the International Conference on Intelligent Robots and Systems*. IEEE, 3815–3821.
[50] Manfred Milinski and Rolf Heller. 1978. Influence of a predator on the optimal foraging behaviour of sticklebacks (gasterosteus aculeatus L.). *Nature* 275, 5681 (1978), 642–644.
[51] Evdokia Nikolova and David R. Karger. 2008. Route planning under uncertainty: The canadian traveler problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 969–974.
[52] Christos H. Papadimitriou and Mihalis Yannakakis. 1991. Shortest paths without a map. *Theoretical Computer Science* 84, 1 (1991), 127–150.
[53] Julia K. Parrish, Steven V. Viscido, and Daniel Grunbaum. 2002. Self-organized fish schools: An examination of emergent properties. *The Biological Bulletin* 202, 3 (2002), 296–305.
[54] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L. Medaglia. 2013. A review of dynamic vehicle routing problems. *European Journal of Operational Research* 225, 1 (2013), 1–11.
[55] Lenka Pitonakova, Richard Crowder, and Seth Bullock. 2016. Information flow principles for plasticity in foraging robot swarms. *Swarm Intelligence* 10, 1 (2016), 33–63.
[56] Ryan R. Pitre, X. Rong Li, and R. Delbalzo. 2012. UAV route planning for joint search and track missions–an information-value approach. *IEEE Trans. Aerospace Electron. Systems* 48, 3 (2012), 2551–2565.
[57] Harilaos N. Psaraftis, Min Wen, and Christos A. Kontovas. 2016. Dynamic vehicle routing problems: Three decades and counting. *Networks* 67, 1 (2016), 3–31.
[58] Sarvapali D. Ramchurn, Alessandro Farinelli, Kathryn S. Macarthur, and Nicholas R. Jennings. 2010. Decentralized coordination in robocup rescue. *The Computer Journal* 53, 9 (2010), 1447–1461.
[59] N. S. V. Rao, Neal Stoltzfus, and S. S. Iyengar. 1991. A 'retraction' method for learned navigation in unknown terrains for a circular robot. *IEEE Transactions on Robotics and Automation* 7, 5 (1991), 699–707.
[60] Leslie A. Real. 1981. Uncertainty and pollinator-plant interactions: The foraging behavior of bees and wasps on artificial flowers. *Ecology* 62, 1 (1981), 20–26.
[61] Gerhard Reinelt. 1991. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing* 3, 4 (1991), 376–384.
[62] Costel Sarbu, Katharina Zehl, and Jürgen W. Einax. 2007. Fuzzy divisive hierarchical clustering of soil data using gustafson–kessel algorithm. *Chemometrics and Intelligent Laboratory Systems* 86, 1 (2007), 121–129.
[63] Yaniv Shapira and Noa Agmon. 2015. Path planning for optimizing survivability of multi-robot formation in adversarial environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4544–4549.
[64] Dylan A. Shell and Maja J. Matarić. 2006. On foraging strategies for large-scale multi-robot systems. In *Proceedings of the International Conference on Intelligent Robots and Systems*. 2717–2723.
[65] Efrat Sless, Noa Agmon, and Sarit Kraus. 2014. Multi-robot adversarial patrolling: Facing coordinated attacks. In *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems*. 1093–1100.
[66] Mason Thammawichai, Sujit P. Baliyarasimhuni, Eric C. Kerrigan, and João B. Sousa. 2017. Optimizing communication and computation for multi-UAV information gathering applications. *IEEE Transactions on Aerospace and Electronic Systems* 54, 2 (2017), 601–615.
[67] Benjamin J. Toscano, Natasha J. Gownaris, Sarah M. Heerhartz, and Cristián J. Monaco. 2016. Personality, foraging behavior and specialization: Integrating behavioral and food web ecology at the individual level. *Oecologia* 182, 1 (2016), 55–69.

[68] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. 2011. The orienteering problem: A survey. *European Journal of Operational Research* 209, 1 (2011), 1–10.
[69] F. Yu Vincent, Parida Jewpanya, and A. A. N. Perwira Redi. 2016. Open vehicle routing problem with cross-docking. *Computers and Industrial Engineering* 94, 1 (2016), 6–17.
[70] Alan F. T. Winfield. 2009. Foraging robots. *Encyclopedia of Complexity and Systems Science* 15, 3 (2009), 3682–3700.
[71] Tengke Xiong, Shengrui Wang, André Mayers, and Ernest Monga. 2012. DHCC: Divisive hierarchical clustering of categorical data. *Data Mining and Knowledge Discovery* 24, 1 (2012), 103–135.
[72] Fuhan Yan, Kai Di, Jiuchuan Jiang, Yichuan Jiang, and Hui Fan. 2019. Efficient decision-making for multiagent target searching and occupancy in an unknown environment. *Robotics and Autonomous Systems* 114, 1 (2019), 41–56.
[73] Fuhan Yan, Jiuchuan Jiang, Kai Di, Yichuan Jiang, and Zhifeng Hao. 2019. Multiagent pursuit-evasion problem with the pursuers moving at uncertain speeds. *Journal of Intelligent & Robotic Systems* 95, 1 (2019), 119–135.
[74] Roi Yehoshua and Noa Agmon. 2015. Adversarial modeling in the robotic coverage problem. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. 1–7.
[75] Roi Yehoshua, Noa Agmon, and Gal A. Kaminka. 2016. Robotic adversarial coverage of known environments. *The International Journal of Robotics Research* 35, 12 (2016), 1419–1444.
[76] Ruohan Zhang and Zhao Song. 2016. Maximum sustainable yield problem for robot foraging and construction system. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 2725–2731.

Received November 2020; revised November 2021; accepted January 2022