Data-Driven Monitoring with Mobile Sensors and Charging Stations using Multi-Arm Bandits and Coordinated Motion Planners

Nayak, Siddharth; Greiff, Marcus; Raghunathan, Arvind; Di Cairano, Stefano; Vinod, Abraham P.

TR2024-078 June 26, 2024

Abstract

We study the problem of data-driven monitoring using an autonomous search team. We consider a typical monitoring task of classifying a search environment into in- teresting and uninteresting regions as quickly as possible using a search team comprised of mobile sensors (e.g., drones) and mobile charging stations (e.g., ground vehicles). For widespread deployment in the physical world, the search team must also accommodate noisy data collected by the mobile sensors, overcome energy constraints on the mobile sensors which limit their range, and ensure collision avoidance for the charging stations. We address these challenges using a novel, bi-level approach for the monitoring task, where a high-level planner uses past data to determine the potential regions of interest for the drones to visit, and a low-level path planner coordinates the paths for the entire search team to visit these regions subject to the posed constraints. We design the high-level planner using a multi-armed bandit framework. For the low-level planner, we propose two approaches: an optimal integer program-based motion planner and a real-time implementable graph-based heuristic planner. We characterize several theoretical properties of the proposed approaches, including anytime guarantees, upper bounds on computing time, and task completion time. We show the efficacy of our approach in simulations, including one in Gazebo where we identify harvest-ready trees using an autonomous heterogeneous search team.

American Control Conference (ACC) 2024

Data-Driven Monitoring with Mobile Sensors and Charging Stations using Multi-Arm Bandits and Coordinated Motion Planners

Siddharth Nayak, Marcus Greiff, Arvind Raghunathan, Stefano Di Cairano, Abraham P. Vinod*

***Abstract*— We study the problem of data-driven monitoring using an autonomous search team. We consider a typical monitoring task of classifying a search environment into interesting and uninteresting regions as quickly as possible using a search team comprised of mobile sensors (e.g., drones) and mobile charging stations (e.g., ground vehicles). For widespread deployment in the physical world, the search team must also accommodate noisy data collected by the mobile sensors, overcome energy constraints on the mobile sensors which limit their range, and ensure collision avoidance for the charging stations. We address these challenges using a novel, bi-level approach for the monitoring task, where a high-level planner uses past data to determine the potential regions of interest for the drones to visit, and a low-level path planner coordinates the paths for the entire search team to visit these regions subject to the posed constraints. We design the high-level planner using a multi-armed bandit framework. For the low-level planner, we propose two approaches: an optimal integer program-based motion planner and a real-time implementable graph-based heuristic planner. We characterize several theoretical properties of the proposed approaches, including anytime guarantees, upper bounds on computing time, and task completion time. We show the efficacy of our approach in simulations, including one in Gazebo where we identify harvest-ready trees using an autonomous heterogeneous search team.**

I. Introduction

Monitoring large areas using autonomous search teams has several critical applications, including environmental monitoring, search and rescue operations, and wildlife tracking [1]–[4]. In this paper, we propose algorithms for autonomous search teams to identify regions containing interesting objects or phenomena. Our focus is on heterogeneous search teams comprising mobile sensors, such as range-limited drones with noisy sensors, and ground robots. The ground robots collaborate with drones to address constraints from the physical environment, such as the energy limitations of drones. Our approach is motivated by the limitations of drone platforms available in the market today, which are often constrained in range due to battery and communication limitations and are often only equipped with low-cost, lightweight sensors that provide noisy measurements.

Several strategies have been proposed for multi-agent search [2]–[4]. Popular approaches include algorithms based on submodular maximization [5], algorithms combining Voronoi-based search [6] with function approximation [7], [8], active sensing/perception algorithms [9], graph-based search algorithms [10], [11], and algorithms based on statistical learning [12], [13]. However, these works may require perfect sensing, may not have finite-time guarantees on the search performance, and/or may generate trajectories for the mobile sensors that are suboptimal in terms of energy usage.

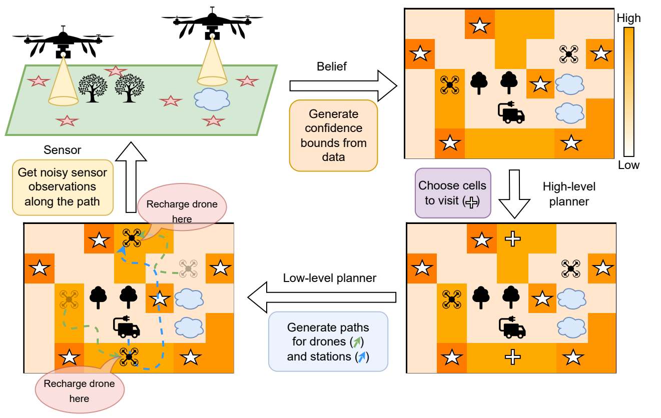

Belief
High
Low
Generate confidence bounds from data
Sensor
Get noisy sensor observations along the path
Recharge drone here
Choose cells to visit
High-level planner
Low-level planner
Generate paths for drones and stations
Recharge drone here

Fig. 1: Data-driven multi-agent search under noisy observations. Measurements of cells determine confidences that inform the high-level planner on which cells to visit. Based on this, the low-level planner computes trajectories for the agents subject to various constraints, ensuring safe sensor deployment when collecting additional data.

Recently, bandit-based algorithms have been proposed for data-driven monitoring [1], [14], [15]. While [14], [15] propose algorithms to identify the top-k interesting regions, their approaches require prior knowledge of the number of interesting regions in the search environment and may not extend to large search teams or mobile sensors with energy limitations. Recently, a bandit-based approach [1] was proposed to identify *all* interesting regions, but [1] treats the motion and energy constraints on the search team as soft constraints that may be violated. In this work, we build upon the approach in [1] to propose a bandit-based data-driven monitoring strategy, and use coordinated control of the heterogeneous search team to guarantee satisfaction of motion and energy constraints.

The problem of coordinated control of heterogeneous teams of aerial and ground vehicles have gained attention to tackle the energy limitations of drones [2]–[4]. Examples of recent works include approaches based on satisfiability modulo theories [16], approaches that rely on assigned roles for the search team members via partitioning [17], and approaches that solve vehicle routing problems [18]. However,

*Corresponding author. Email: abraham.p.vinod@ieee.org

S. Nayak is with Massachussetts Institute of Technology, Cambridge, MA 02139, USA. This work was done during Nayak's internship at MERL.

M. Greiff, A. Raghunathan, S. Di Cairano, and A. Vinod are with Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA.

these works may be limited to static goals due to the need for offline planning, may not scale to large search teams, and may not adapt to dynamic assignments of sub-teams formed by a ground vehicle and its associated aerial vehicles. In this paper, we propose an optimal, integer program-based motion planner and a real-time implementable, graph-based heuristic planner to overcome these limitations.

The main contribution of this work is *a bi-level approach for the monitoring task that uses online data for agent deployment and uses optimization or graph-based planning to coordinate the path planning for the search teams*. We provide theoretical guarantees for the proposed approach, including anytime guarantees (the algorithm provides a useful result, even when terminated prematurely), and finite upper bounds on compute time and monitoring completion. We also demonstrate our approach in several simulations, including a Gazebo simulation based on precision agriculture.

Notation: For a set $\mathcal{S}$, $|\mathcal{S}|$ denotes its cardinality, and $\mathcal{S}^{\mathsf{C}}$ denotes its complement with respect to some set $\mathcal{G} \supset \mathcal{G}$.

II. Problem statement

Search environment: We model the search environment as a set $\mathcal{G}$ of grid cells. We collect all cells that are occupied by obstacles or are no-fly zones in a known set $\mathcal{O} \subset \mathcal{G}$. We will use $\mathcal{I} \subset \mathcal{G} \setminus \mathcal{O}$ to denote the set of *interesting cells*, which is *a priori* unknown. Specifically, $\mathcal{I}$ is the set of cells that are occupied by objects/phenomena of interest. We wish to characterize $\mathcal{I}$ using data collected by a search team.

Search team: We use a search team with $N_d \in \mathbb{N}$ mobile sensors (e.g., drones) from now on simply *sensors*, and $N_c \in \mathbb{N}$ charging stations (e.g., autonomous trucks) from now on simply *stations*, where $N_c \leq N_d$. While both the sensors and stations can move in the search environment, we constrain their motion to enforce energy limitations on the sensors and slower relative speed of the stations with respect to the sensors. We assume that the sensors can move to one of their neighboring cells in any direction at each time step (like a king piece in chess), excluding the cells in $\mathcal{O}$. However, the sensors run out of battery after traversing T_d cells and need to be recharged to continue. Assuming that the sensors and stations have a speed of v_d and v_c respectively with $v_d > v_c$, $v_d/v_c \in \mathbb{N}$, $T_c \triangleq T_d \frac{v_c}{v_d} \in \mathbb{N}$, the stations can move at most $T_c < T_d$ cells in the same duration as that of the mobile sensors covers T_d cells. Similarly to the sensors, the stations can move to any of their neighbouring cells, but can move only at every $\frac{T_d}{T_c}$ time steps. For any cell $l \in \mathcal{G}$, we denote the set of neighboring cells in all directions by $\mathcal{N}(l) \subseteq \mathcal{G} \setminus \mathcal{O}$. We can also accommodate other motion models for the sensors and the stations, e.g., axis-aligned movements.

Sensing: When a sensor visits a grid cell, it receives a noisy, binary measurement (data) of whether that cell is interesting or not. Formally, every grid cell $i \in \mathcal{G}$ has a corresponding Bernoulli random variable ν_i with *a priori* unknown mean μ_i. The mean μ_i may be influenced by the underlying spatial distribution of the interesting cells as well as the imperfections of the noisy sensors and the perception algorithms used by the search team. We assume that the Bernoulli random variables for any two cells in $\mathcal{G}$ are mutually independent.

Labeling error criterion: For a known threshold $\theta \in (0, 1)$,

$$\mathcal{S}_\theta \triangleq \{i \in \mathcal{G} : \mu_i \geq \theta\} \tag{1}$$

the set $\mathcal{S}_\theta$, is the set of grid cells that are sensed as interesting with a probability of at least θ and is an approximation of $\mathcal{I}$ given the noisy data collected by the search team. Motivated by [1], we tolerate a labeling error for any grid cells $i \in \mathcal{G}$ close to the threshold θ. Formally, for a (small) labeling error tolerance $\epsilon > 0$, a (keep) set $\mathcal{K} \subseteq \mathcal{G}$ such that $\mathcal{S}_{\theta+\epsilon} \subseteq \mathcal{K} \subseteq \mathcal{S}_{\theta-\epsilon}$, is an acceptable approximation of $\mathcal{S}_\theta$ (thereby of $\mathcal{I}$). We encode this as a probabilistic low *labeling error* criterion,

$$\mathbb{P}\left[(\mathcal{S}_{\theta+\epsilon} \setminus \mathcal{K}(p_{\text{term}})) \cup (\mathcal{K}(p_{\text{term}}) \setminus \mathcal{S}_{\theta-\epsilon}) = \emptyset\right] \geq 1 - \delta, \tag{2}$$

at some termination time $p_{\text{term}} \in \mathbb{N}$ for the search algorithm, and for a given *labeling error probability* $\delta \in (0, 1)$.

We now state the two problems tackled by this paper:

Problem 1. *Design a data-driven algorithm for the search team that terminates in finite time, meets the labeling error criterion* (2) *upon termination, and ensures that motion and energy constraints on the search team are satisfied.*

Problem 2. *Determine upper bounds on the time to terminate the search for the algorithm solving Problem 1.*

III. Proposed approach

We propose a bi-level iterative approach to solve Problems 1 and 2, as illustrated in Figure 1. Our approach consists of a high-level and a low-level planner.

The high-level planner identifies a set of *epoch goals* based on the past data (Section III-A). Epoch goals are cells that the search team must visit in the current epoch to collect new measurements. We use *epoch* $p \in \mathbb{N}$ to denote time in a slower time scale used by the high-level planner, as compared to *time steps* $t \in \mathbb{N}$ which denotes time in a faster time scale used by the low-level planner.

Given the current set of epoch goals, the low-level planners generate motion plans for the search team to visit every epoch goal and take measurements while satisfying motion and energy constraints on the team. We propose two low-level planners — an optimal, integer program-based motion planner (Section III-B) and a real-time implementable, graph-based heuristic planner (Section III-C). Finally, the high-level planner uses the noisy data collected by the search team to generate a new set of epoch goals, and we repeat these steps until a termination criterion.

A. High-level planner: Data-driven monitoring via bandits

The proposed high-level planner is inspired by the bandit-based monitoring algorithm proposed in [1]. Specifically, we cast the search problem as a $|\mathcal{G}|$-armed bandit problem, where the bandit arms are grid cells. The high-level planner is a sequential decision maker that processes the collected information to decide on which grid cells must be visited at each epoch We use *upper confidence bounds*, typical

Algorithm 1 Bandit-based monitoring under motion and energy constraints (High-level planner)

```
Input: Set of grid cells $\mathcal{G}$, threshold $\theta \in (0,1)$, labeling error tolerance $\epsilon > 0$, labelling error probability $\delta \in (0,1)$, number of epoch goals $D \in \mathbb{N}$, obstacle set $\mathcal{O}$
Output: $\{\mathcal{K}(p)\}_{p\in\mathbb{N}}$, sequence of (keep) sets
1: Initialize $p \leftarrow 0$, $\mathcal{K}(p) \leftarrow \emptyset$, and $\mathcal{R}(p) \leftarrow \emptyset$
2: while $\mathcal{R}(p) \cup \mathcal{K}(p) \neq \mathcal{G}$ do
3:     Define $\mathcal{E}_p$ as the top $D$ elements in the list of unlabelled cells $i \in \mathcal{G} \setminus (\mathcal{K}(p) \cup \mathcal{R}(p))$, sorted in descending order based on $J(i,p,\delta)$ (3)
4:     Use a low-level planner Algorithm 2 or Algorithm 3 to deploy the search team to visit cells in $\mathcal{E}_p$ while avoiding $\mathcal{O}$, take measurements along the way, and update history $\mathcal{H}(p+1)$
5:     Update sets of labelled cells $\mathcal{K}(p+1)$ and $\mathcal{R}(p+1)$ based on (4) using $\mathcal{H}(p+1)$, $\mathcal{G}$, $\theta$, $\delta$, and $\epsilon$
6:     Increment epoch $p \leftarrow p+1$
7: end while
```

of bandit-based algorithms [1], [19]–[21], to sample the unlabeled cells that are "most likely" to be interesting.

Let $\mathcal{H}_i(p)$ be the history of measurements taken at cell $i \in \mathcal{G}$ collected by the search time until epoch p, and define $\mathcal{H}(p) = \{\mathcal{H}_i(p)\}_{i\in\mathcal{G}}$. Then, at epoch p, we choose D (typically, $D \geq N_d$) distinct cells that achieve the top-D values of a function $J : \mathcal{G} \times \mathbb{N} \times (0,1) \to \mathbb{R} \cup \{\infty\}$,

$$J(i,p,\delta) = \hat{\mu}_i(p) + U_i(p,\delta), \tag{3a}$$

$$\hat{\mu}_i(p) = \frac{\sum_{h\in\mathcal{H}_i(p)} h}{|\mathcal{H}_i(p)|}, \tag{3b}$$

$$U_i(p,\delta) = 2\sqrt{\frac{2\log(\log_2(2|\mathcal{H}_i(p)|)) + \log(12|\mathcal{G}|/\delta)}{2|\mathcal{H}_i(p)|}}, \tag{3c}$$

with $\hat{\mu}_i(p) = U_i(p,\delta) = \infty$, whenever $\mathcal{H}_i(p) = \emptyset$. Here, $\delta \in (0,1)$ is a given (small) labelling error probability. After getting the data for the current epoch, we update the sets $\mathcal{K}(p+1)$ and $\mathcal{R}(p+1)$ as follows,

$$\mathcal{K}(p+1) = \{i \in \mathcal{G} : \hat{\mu}_i(p+1) - U_i(p+1,\delta) \geq \theta - \epsilon\}, \tag{4a}$$

$$\mathcal{R}(p+1) = \{i \in \mathcal{G} : \hat{\mu}_i(p+1) + U_i(p+1,\delta) \leq \theta + \epsilon\}. \tag{4b}$$

Since $U_i(p,\delta)$ is a non-increasing function of $|\mathcal{H}_i(p)|$, the sets $\mathcal{K}(p)$ and $\mathcal{R}(p)$ are monotonic in p, and their cardinality are non-decreasing in p. Equations (3) and (4) are motivated by the desire to obtain *anytime guarantees* as well as ensuring a low labelling time (see Section IV).

We summarize the high-level planner in Algorithm 1, which also summarizes the proposed approach illustrated in Figure 1. In Step 4, Algorithm 1 assumes access to a low-level planner that can deploy the search team to visit all epoch goals, identified in Step 3, in finite time. We discuss two such low-level planners in Sections III-B and III-C.

B. Low-level planner: Optimal method via integer program

We now describe an optimization-based planner that provides an optimal deployment plan for the search team. In large search environments, we can not expect the mobile sensors to cover all cells in $\mathcal{E}_p$ using a single charge of their batteries. In the proposed optimization-based low-level planner, we deploy the search team such that the mobile sensors always start and end their flights on a charging station within T_d moves, while minimizing the number of such *flight cycles*. The use of flight cycles ensures that the mobile sensors never run out of battery, and the search team constantly switches between having all mobile sensors being charged on charging stations and having all mobile sensors moving around in the search environment taking measurements. Informally, the optimization-based low-level planner solves the following problem at each epoch p,

$$\begin{aligned} \text{min.}\quad & \text{Number of flight cycles,} && \text{(5a)}\\ \text{s. t.}\quad & \text{Search team respects motion constraints,} && \text{(5b)}\\ & \text{Search team visits all cells in } \mathcal{E}_p \text{ but none in } \mathcal{O}, && \text{(5c)}\\ & \text{Mobile sensors never run out of battery.} && \text{(5d)} \end{aligned}$$

We cast the optimization problem (5) as an integer program (IP), and solve it using off-the-shelf solvers [22].

We use the following set of binary decision variables in the IP formulation: $x_{ijkl}, y_{abkl}, \lambda_k \in \{0,1\}$ for every

$$\begin{aligned} &\text{(Sensor/station index):} && i \in \mathcal{D} \triangleq \mathbb{N}_{[1,N_d]}, \quad a \in \mathcal{C} \triangleq \mathbb{N}_{[1,N_c]},\\ &\text{(Time in a flight cycle):} && j \in \mathcal{T}_d \triangleq \mathbb{N}_{[1,T_d]}, \quad b \in \mathcal{T}_c \triangleq \mathbb{N}_{[1,T_c]},\\ &\text{(Flight cycle count):} && k \in \mathcal{K} \triangleq \mathbb{N}_{[1,K]},\\ &\text{(Grid cell location):} && l \in \mathcal{G} \triangleq \mathbb{N}_{[1,|\mathcal{G}|]}, \end{aligned} \tag{6}$$

where $x_{ijkl} = 1$ if mobile sensor i is at grid cell l at time step j in the flight cycle k, $x_{ijkl} = 0$ otherwise; $y_{abkl} = 1$ if station a is at grid cell l at time step b in the flight cycle k, $y_{abkl} = 0$ otherwise; and, $\lambda_k = 1$ if flight cycle k is necessary for the search team to solve (5), $\lambda_k = 0$ otherwise. Here, K is a user-specified upper bound on the number of flight cycles. The number of variables in the IP for each epoch p is $K(1 + |\mathcal{G}|(N_d T_d + N_c T_c))$. In what follows, the constraints are enforced for all indices i, j, k, l, a, and b as described in (6), unless stated otherwise.

Motion constraints:

$$x_{i00l} = 1 \text{ and } y_{a00d} = 1, \qquad \forall l \in X_0, d \in Y_0, \tag{7a}$$

$$\sum\nolimits_{m\in\mathcal{N}(l)} x_{i(j-1)km} \geq x_{ijkl}, \tag{7b}$$

$$\sum\nolimits_{m\in\mathcal{N}(l)} y_{a(b-1)km} \geq y_{abkl}, \tag{7c}$$

$$\sum\nolimits_{a\in\mathcal{C}} y_{abkl} \leq 1, \tag{7d}$$

$$\sum\nolimits_{l\in\mathcal{G}} x_{ijkl} = \sum\nolimits_{d\in\mathcal{G}} y_{abkl} = \lambda_k, \tag{7e}$$

$$\lambda_k \geq \lambda_{k+1}, \quad \forall k \in \mathbb{N}_{[1,K-1]}. \tag{7f}$$

Constraint (7a) requires the mobile sensors and the charging stations start at the initial locations with $\mathcal{X}_0, \mathcal{Y}_0 \subset \mathcal{G}$ as the set of grid cells initially occupied by the search team (sensors and stations, respectively). Constraints (7b) and (7c) require that the sensors and the stations on grid cell $l \in \mathcal{G}$ can move only to one of their neighboring cells $\mathcal{N}(l)$. Constraint (7d) requires no two stations to occupy the same grid cell at any time to avoid collisions. Collision avoidance among mobile

```
Algorithm 2 Low-level planner: Integer program
Input: Epoch goals from high-level planner $\mathcal{E}_p$, initial
    search team configuration $(\mathcal{X}_0, \mathcal{Y}_0)$, obstacle set $\mathcal{O}$,
    search team parameters $(T_d, T_c, N_d, N_c, K, \mathcal{G})$
Output: Paths for the search team
 1: Compute $x^*_{ijkl}, y^*_{abkl}, \lambda^*_k$ by solving an integer program,
        $\begin{array}{ll} \text{minimize} & \sum_{k\in\mathbb{N}_{[1,K]}} \lambda_k, \\ \text{subject to} & (7), (8), \text{and } (9). \end{array}$    (10)
 2: Compute paths for the search team by extracting all
    indices $(i,j,k,l)$ and $(a,b,c,d)$ such that $x^*_{ijkl} = 1$
    and $y^*_{abcd} = 1$, then creating a sequence of grid cells
    to traverse through these indices $i, j, k, l, a, b, c$, and $d$
```

sensors may be enforced by a constraint similar to (7d), but may also be easily enforced by requiring the sensors to fly at different altitudes. Constraint (7e) links the binary variables corresponding to flight cycles λ_k to the search team paths. Specifically, $\lambda_k = 0$ requires the search team to visit no cells in $\mathcal{G}$ during the flight cycle k, i.e., the path of the search team terminates prior to flight cycle k. On the other hand, when $\lambda_k = 1$, each sensor and station visit exactly one cell in $\mathcal{G}$ at each time step in the flight cycle k. Constraint (7f) encodes the temporal constraint among flight cycles, i.e., $\lambda_{m+1} = 1$ for any $m \leq K-1$ implies that $\lambda_k = 1$ for every $1 \leq k \leq m$.

Visit constraints: The followig constraints enforce (5c) by requiring that every cell $m \in \mathcal{E}_p$ is visited by some sensors at some time step in some flight cycle, and the search team (both sensors and stations) never visits any grid cell in $\mathcal{O}$.

$$\sum\nolimits_{(i,j,k)\in\mathcal{D}\times\mathcal{T}_d\times\mathcal{K}} x_{ijkm} \geq 1, \qquad \forall m \in \mathcal{E}_p, \tag{8a}$$

$$x_{ijkm} = y_{abkm} = 0. \qquad \forall m \in \mathcal{O}, \tag{8b}$$

Battery constraints:

$$\sum\nolimits_{(j,l)\in\mathcal{T}_d\times\mathcal{G}} x_{ijkl} \leq T_d, \quad \sum\nolimits_{(b,l)\in\mathcal{T}_c\times\mathcal{G}} y_{abkl} \leq T_c, \tag{9a}$$

$$|x_{i0(k+1)l} - x_{iT_dkl}| \leq 2(1-\lambda_{k+1}), \quad \forall k \in \mathbb{N}_{[1,K-1]}, \tag{9b}$$

$$|y_{a0(k+1)l} - y_{aT_ckl}| \leq 2(1-\lambda_{k+1}), \quad \forall k \in \mathbb{N}_{[1,K-1]}, \tag{9c}$$

$$x_{i0kl} \leq \sum\nolimits_{a\in\mathcal{C}} y_{a0kl}, \quad x_{iT_dkl} \leq \sum\nolimits_{a\in\mathcal{C}} y_{aT_ckl} \tag{9d}$$

Constraints (9a) require the path lengths of sensors and stations to respect the energy and velocity constraints. Constraints (9b) and (9c) require the positions of sensors and stations to remain unchanged between flight cycles, and are trivially satisfied when $\lambda_{k+1} = 0$. Constraints (9d) require every sensor to start and end its path within a flight cycle on a grid cell occupied by one of the stations.

We summarize the IP-based low-level planner in Algorithm 2. However, IPs are NP-hard problems and it may become challenging to solve (10) for large $\mathcal{G}$. This motivates the heuristic-based low-level planner discussed next.

C. Low-level planner: Heuristic via graph-based planning

We will refer to the space-time coordinates at which a sensor runs out of energy as *restpoints*. In Algorithm 2, constraints (9d) require the stations to reach restpoints before the corresponding sensor reaches them. To propose a heuristic, low-level planner, we relax (9d), and allow a sensor to reach a restpoint before any station has arrived. If a sensor reaches the restpoint early, we assume that the sensor waits at the restpoint for the station to rendezvous, and then continues on with the sensor's assigned path upon begin recharged. Additionally, we relax the collision avoidance constraint (7d), since we can generate collision-free paths for the stations using a protocol-based conflict resolution algorithm as introduced in [23].

We summarize the heuristic, low-level planner in Algorithm 3. Algorithm 3 uses the relaxation on the charging constraint to further divide the low-level planning problem into two stages — the first stage that designs paths for the sensors to visit all epoch goals in $\mathcal{E}_p$, and the second stage that designs paths for the stations to visit all the restpoints necessitated by the sensor paths designed in the first stage. Motivated by the limited energy on the mobile sensors, we use Dijkstra's algorithm, a graph-based planning algorithm, to design obstacle-free shortest paths for the search team.

First, we call `assignSensors`, to compute the shortest (obstacle-free) paths from the sensors' initial locations $\mathcal{X}_0$ to the epoch goals. We also distribute these goals among the mobile sensors based on their shortest path distance with random breaking of ties to ensure that no sensor has more than $M_d \in \mathbb{N}$ epoch goals, where $M_d \geq |\mathcal{E}_p|/N_d$. `assignSensors` only assigns epoch goals to the sensors, and does not prescribe the sequence of visits.

Next, we call `getSensorPaths`, which assigns the sequence of visiting epoch goals for each sensor and generates feasible paths. We enumerate all possible visiting orders (at most $M_d!$) for each sensor, and choose the sequence and path that has the smallest path length. Alternatively, one could use traveling salesman problem solvers [24].

To follow the sensor paths obtained from `getSensorPaths`, we must interject these paths with restpoints that occurs after every T_d moves due to energy limitations of the sensors. We compute a list of restpoints by calling `getRestPoints`. Each sensor may have multiple restpoints, which we collect in a set $\mathcal{R} = \{\mathcal{R}_i\}_{i\in\mathcal{D}}$ with $\mathcal{R}_i \triangleq \{r_{i1}, r_{i2}, \ldots, \ldots r_{i|\mathcal{R}_i|}\}$. The stations must visit all restpoints in $\mathcal{R}$, while satisfying a visitation temporal constraint — r_{ij} must be visited before $r_{i(j+1)}$ for any $j \in \mathbb{N}_{[1,|\mathcal{R}_i|]}$ and $i \in \mathcal{D}$. The sets $\mathcal{R}_i$ are guaranteed to be finite, since the longest sensor path that may be computed by `getSensorPaths` is finite.

Next, we call `assignStations` to assign both the restpoints to stations as well as to satisfy the visitation temporal constraint. First, `assignStations` segregates the restpoints temporally by defining $\mathcal{R}^{(\tau)} = \{r_{i\tau} : i \in \mathcal{D}\}$ for $\tau \in \mathbb{N}_{[1,L_R]}$ with $L_R \triangleq \max_{i\in\mathcal{D}} |\mathcal{R}_i|$, and then distributes $\mathcal{R}^{(\tau)}$ among the stations based on the shortest path distance between $\mathcal{R}^{(\tau)}$ and the previous locations of the stations. Similarly to `assignSensors`, we cap the number of restpoints assigned to each station at each τ by $M_c \in \mathbb{N}$ with $M_c \geq L_R/N_c$.

Algorithm 3 Low-level planner: Graph-based heuristic

```
Input: Epoch goals from high-level planner $\mathcal{E}_p$, initial
    search team configuration $(\mathcal{X}_0, \mathcal{Y}_0)$, obstacle set $\mathcal{O}$,
    search team parameters $(T_d, N_d, N_c, M_d, M_c, \mathcal{G})$
Output: Sensor and station paths (SePaths,StPaths)
1: SeAlloc $\leftarrow$ assignSensors($\mathcal{E}_p, M_d, N_d, \mathcal{X}_0, \mathcal{G}, \mathcal{O}$)
2: SePaths$\leftarrow$getSensorPaths(SeAlloc,$\mathcal{X}_0, \mathcal{O}$)
3: $\mathcal{R}$ $\leftarrow$getRestPoints(SePaths,$T_d$)
4: StAlloc$\leftarrow$assignStations($\mathcal{R}, M_c, N_c, \mathcal{Y}_0, \mathcal{G}, \mathcal{O}$)
5: StPaths$\leftarrow$getStationPaths(StAlloc,$\mathcal{Y}_0, \mathcal{O}$)
```

Finally, we call `getStationPaths` to generate a shortest path for each station that connects the stations' initial location $\mathcal{Y}_0$ to cover all waypoints from `assignStations`.

Algorithm 3 uses graph-based planning to ensure real-time implementability (see Section IV). However, due to the relaxation of (9d) as well as the use of greedy assignment of epoch goals and restpoints, the paths generated by Algorithm 3 may be significantly suboptimal compared to Algorithm 2 in terms of the time taken to cover $\mathcal{E}_p$.

IV. Theoretical guarantees

We now state various theoretical guarantees provided by the proposed approach and address Problem 2.

Proposition 1 (Anytime algorithm)**.** *Let $\delta \in (0,1)$ be the labeling error probability. At any epoch $p \in \mathbb{N}$, the sets $\mathcal{K}(p)$ and $\mathcal{R}(p)$ maintained by Algorithm 1 satisfy $\mathcal{K}(p) \subseteq \mathcal{S}_{\theta-\epsilon}$ and $\mathcal{R}(p) \subseteq \mathcal{S}_{\theta+\epsilon}^{\complement}$, with probability of at least $1-\delta$.*

The key takeaway from Proposition 1 is that the keep set $\mathcal{K}(p)$ computed by Algorithm 1 always inner-approximates $\mathcal{S}_{\theta-\epsilon}$. Consequently, Algorithm 1 may be prematurely terminated if required, and the intermediate solution $\mathcal{K}(p)$ and $\mathcal{R}(p)$ will contain only interesting and uninteresting cells respectively (upto a tolerance ϵ), with high probability.

Proposition 2 (Finite time guarantees)**.** *Algorithm 1 terminates within $P^{\max} \in \mathbb{N}$ epochs, and satisfies the labeling error criterion (2) with*

$$P^{\max} \triangleq \max_{i\in\mathcal{D}_\triangle} O(\phi_i) + \frac{1}{N_d}\sum_{i\in\mathcal{D}_\triangle^{\complement}} O(\phi_i), \tag{11}$$

$$\phi_i = \frac{1}{\gamma_i^2}\log\left(\frac{|\mathcal{G}|}{\delta}\log\left(\frac{|\mathcal{G}|}{\gamma_i^4\delta}\right)\right), \tag{12}$$

$$\gamma_i = |\mu_i - \theta| + \epsilon, \tag{13}$$

with ϕ_i, γ_i defined for every $i \in \mathcal{G}$, and $\mathcal{D}_\triangle$ is the union of a grid cell with the smallest γ_i with a set of $d-1$ grid cells with the largest γ_i among all cells $i \in \mathcal{G}$.

By Proposition 2, Algorithm 1 terminates at some $p_{\text{term}} \leq P^{\max}$ epochs, and returns $\mathcal{K}(p_{\text{term}})$ that satisfies the labeling error criterion (2). Specifically, $\mathcal{K}(p_{\text{term}})$ outer-approximates $\mathcal{S}_{\theta+\epsilon}$ and inner-approximates $\mathcal{S}_{\theta+\epsilon}$ with high probability (2).

Corollary 1. *Given a search environment and a search team, let $K^{\max}$ be the maximum number of flight cycles needed by Algorithm 2 to cover any collection of epoch goals starting from any configuration of the search team. Then, Algorithm 1 with Algorithm 2 as the low-level planner terminates while satisfying (2) in not more than $P^{\max}K^{\max}$ time steps.*

Corollary 1 provides an upper bound on the time steps required by a search team that are deployed using Algorithms 1 and 2 to complete the search, and satisfy (2). Note that the computation of $K^{\max}$ involves a min-max computation, and while its solution always exists and is finite, it may be hard to compute explicitly. However, one can obtain reasonable estimates of $K^{\max}$ via Monte-Carlo simulations, if desired.

Proposition 3. (Computation time complexity) *The compute time for each call of the heuristic, low-level planner Algorithm 3 is $T^{\max} = O\left(N_d \cdot |\mathcal{G}|^2\left[|\mathcal{E}_p| + M_d! + N_c \cdot \frac{|\mathcal{G}|}{T_d} + M_c! \cdot \frac{N_c}{N_d}\right]\right)$.*

Corollary 2. *Given a search environment and a search team, the overall compute time for Algorithm 1 with Algorithm 3 as the low-level planner to terminate while satisfying (2) is of the order $P^{\max}T^{\max}$.*

Since protocol-based conflict resolution is also meant for real-time implementation [23], Algorithm 3 can be modified to generate collision-free paths using similar ideas without losing the presented real-time implementation guarantees.

V. Numerical simulations

We now validate the proposed algorithms via simulations. We investigate the time-optimality and the scalability of the proposed algorithms in varying sizes of search environments. We also demonstrate the ability of the proposed approach to quickly identify interesting cells, owing to the use of the bandit-based framework, and study the effect of varying team sizes on the proposed approach. We used a standard computer with Intel Core i7-7700K CPU (4.20GHz, 8 cores) and 62.8 GB RAM, running Python 3.8 for the computations.

We consider a grid $\mathcal{G}$ with some pre-defined known obstacles cells $\mathcal{O} \subseteq \mathcal{G}$ and pre-defined unknown interesting cells $\mathcal{I} \subseteq \mathcal{G}$. We consider three different configurations:

1) *Small:* Grid size $|\mathcal{G}| = 25$, $N_d = 4$ sensors, $N_c = 2$ stations, and $|\mathcal{O}| = |\mathcal{I}| = 4$ (16% of $\mathcal{G}$),

2) *Medium:* $|\mathcal{G}| = 100$, $N_d = 10$, $N_c = 5$, $|\mathcal{O}| = 16$ (16% of $\mathcal{G}$), and $|\mathcal{I}| = 10$ (10% of $\mathcal{G}$), and

3) *Large:* $|\mathcal{G}| = 400$, $N_d = 20$, $N_c = 10$, $|\mathcal{O}| = 41$ (10.25% of $\mathcal{G}$), and $|\mathcal{I}| = 20$ (5% of $\mathcal{G}$).

Observe that the search environment is sparsely populated by interesting cells with $|\mathcal{I}| \leq 0.16|\mathcal{G}|$, as typical in environmental monitoring problems.

Figure 2 shows a snapshot of the search environments. The drones are represented with green crosses (×), charging stations with blue circles (●), interesting cells with red stars (⋆), obstacles with black squares (■).

We perform a Monte-Carlo simulation where the values of μ_i for $i \in \mathcal{I}$ is chosen randomly between $[0.8, 1]$ and the values of μ_u for $u \in (\mathcal{G}\setminus\mathcal{O})\setminus\mathcal{I}$ are chosen randomly between $[0, 0.2]$ at the start of each simulation. Additionally, during the trial, the measurements h_l obtained upon a mobile sensor visiting cell $l \in (\mathcal{G}\setminus\mathcal{O})$ is drawn randomly from a Bernoulli

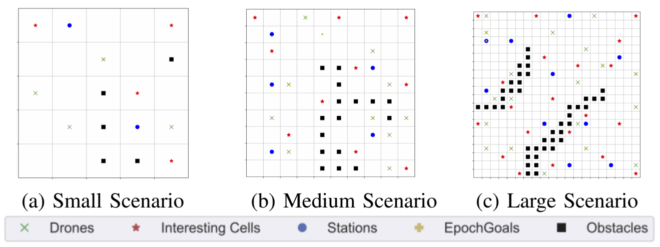


Fig. 2: Various scenarios considered in this paper.

distribution with mean μ_l. A trial is complete when all the cells are classified as either interesting or uninteresting. For $D = N_d$, $\theta = 0.5$, $\epsilon = 0.01$, and $\delta = 10^{-5}$, we empirically observed that the proposed approach classified the search environment correctly in all of the simulations.

Table I reports the median, the 10-th, and the 90-th percentiles over 100 seeds of the following metrics:

1) Compute time (s): The compute time required to execute the monitoring of the environment till completion,

2) Number of low-level planner steps (Total): The total number of time steps taken to complete the monitoring,

3) Number of low-level planner steps (Interesting): The number of time steps required to identify the interesting cells,

4) Number of high-level steps (total): The number of epochs required to complete the monitoring.

All the metrics mentioned above operate based on the principle of "lower is better". While Algorithm 2 (IP-based low-level planner) was able to solve the small and the medium scenarios, it timed-out in the large scenario. On the other hand, Algorithm 3 (heuristic low-level planner) was able solve all three scenarios, and generate motion plans faster than the IP-based solver. As expected, the number of epochs and the time steps taken by the heuristic low-level planner to complete the monitoring is much higher as compared to the IP-based low-level planner.

Figure 3 shows the progress of the fraction of cells classified with the number of epochs. First, observe that the proposed completes identifies all the interesting cells before identifying all the uninteresting cells. Additionally, the gap between the epoch at which Algorithm 1 identifies all the interesting cells and the epoch at which it terminates increases with increasing $\mathcal{G}$. Both of these observations may be attributed to the bandit framework used in Algorithm 1 that prioritizes the pursuit of the most promising cells. See [1] for more details.

Finally, we validate our approach in a high-fidelity simulation environment motivated by a precision agriculture application. Here, the orchard consists of fruit-bearing trees (interesting cells, red), trees without fruit (green) and impassable terrain (blue), see Figure 4. We implemented the search environment using `Gazebo` in `ROS` with the firefly drone models in the `rotors_simulation` package [25] and the waffle models in the `turtlebot3` package [26]. We used existing, standard controllers to achieve set-point stabilization, track paths generated by the low-level planner, land drones land next to the charging stations when charging. The setup corresponds to a Medium scenario (see Fig. 3), and illustrates how the proposed bi-level planner may be deployed in practice.

VI. Conclusion

We introduced a bi-level approach for the multi-agent environment monitoring problem. The high-level bandit-based planner is data-driven and does not rely on pre-determined search trajectories. The low-level planner creates paths for the drones and the charging stations which are practically implementable and satisfy the physical constraints in the environment. We introduce two different methods for the low-level planner — an optimal IP-based planner, and a heuristic graph-based planner. We also present several theoretical properties of the proposed approach and demonstrate its efficacy in simulations. Albeit the IP formulation guarantees optimal paths, they are computationally expensive for real-time implementation. The heuristic planner is sub-optimal compared to the IP-based planner, but it offers quick solutions that allow for real-time implementation.

References

[1] P. Thaker, S. Di Cairano, and A. P. Vinod, "Bandit-based multi-agent search under noisy observations," in *IFAC World Congress*, 2023.

[2] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Rep.*, vol. 2, pp. 189–200, 2021.

[3] J. Queralta, J. Taipalmaa, B. Pullinen, V. Sarker, T. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," *IEEE Access*, vol. 8, pp. 191 617–191 643, 2020.

[4] M. Dunbabin and L. Marques, "Robots for environmental monitoring: Significant advancements and applications," *IEEE Rob. & Autom. Mag.*, vol. 19, no. 1, pp. 24–39, 2012.

[5] A. Krause and C. Guestrin, "Near-optimal observation selection using submodular functions," in *AAAI*, vol. 7, 2007, pp. 1650–1654.

[6] F. Bullo, J. Cortés, and S. Martinez, *Distributed control of robotic networks*. Princeton Univ. Press, 2009.

[7] M. Schwager, M. Vitus, S. Powers, D. Rus, and C. J. Tomlin, "Robust adaptive coverage control for robotic sensor networks," *IEEE Tran. Ctrl. Netw. Syst.*, vol. 4, no. 3, pp. 462–476, 2015.

[8] W. Luo, C. Nam, G. Kantor, and K. Sycara, "Distributed environmental modeling and adaptive sampling for multi-robot sensor coverage," in *Proc. Intl. Conf. Auto. Agents Multi-Agent Syst.*, 2019, pp. 1488–1496.

[9] R. Bajcsy, Y. Aloimonos, and J. K. Tsotsos, "Revisiting active perception," *A. Robots*, vol. 42, no. 2, pp. 177–196, 2018.

[10] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos, "DARP: Divide areas algorithm for optimal multi-robot coverage path planning," *J. Intelli. Robotic Syst.*, vol. 86, no. 3, pp. 663–680, 2017.

[11] G. Best, J. Faigl, and R. Fitch, "Online planning for multi-robot active perception with self-organising maps," *A. Robots*, vol. 42, no. 4, pp. 715–738, 2018.

[12] R. Marchant and F. Ramos, "Bayesian optimisation for intelligent environmental monitoring," in *IEEE Int'l Conf. Intelli. Robots Syst.*, 2012, pp. 2242–2249.

[13] R. Ghods, A. Banerjee, and J. Schneider, "Decentralized multi-agent active search for sparse signals," in *Proc. Conf. Uncertainty Artif. Intelli.*, vol. 161, 2021, pp. 696–706.

[14] E. Rolf, D. Fridovich-Keil, M. Simchowitz, B. Recht, and C. Tomlin, "A successive-elimination approach to adaptive robotic source seeking," *IEEE Tran. Robotics*, vol. 37, no. 1, pp. 34–47, 2021.

[15] B. Du, K. Qian, H. Iqbal, C. Claudel, and D. Sun, "Multi-robot dynamical source seeking in unknown environments," in *Intl Conf. Robotics and Autom.*, 2021, pp. 9036–9042.

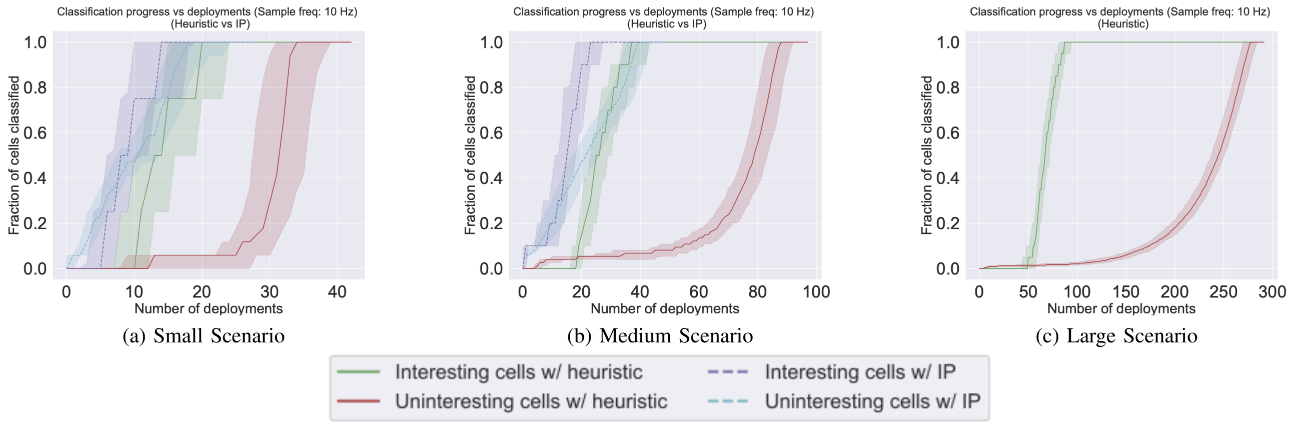
[16] T. Kundu and I. Saha, "Mobile recharger path planning and recharge scheduling in a multi-robot environment," in *IEEE Intn'l Conf. Intell. Rob. Syst.*, 2021, pp. 3635–3642.

[17] X. Lin, Y. Yazıcıoğlu, and D. Aksaray, "Robust planning for persistent surveillance with energy-constrained uavs and mobile charging stations," *IEEE Rob. Autom. Lett.*, vol. 7, no. 2, pp. 4157–4164, 2022.

[18] W. Jing, D. Deng, Y. Wu, and K. Shimada, "Multi-uav coverage path planning for the inspection of large and complex structures," in *IEEE Int'l Conf. Intelli. Robots Syst.*, 2020, pp. 1480–1486.

TABLE I: Comparing IP and Heuristic (HR) in different scenarios. Experiments done with 10 different seeds. IP was not able to give a solution before time-out of 2 hours for the large scenario.

Problem size	Compute time (s)		Number of low-level steps (Total)		Number of low-level steps (Interesting)		Number of high-level steps (total)	
	IP-based planner	Heuristic planner	IP-based planner	Heuristic planner	IP-based planner	Heuristic planner	IP-based planner	Heuristic planner
Small	10.306 $(8.828, 12.500)$	0.131 $(0.122, 0.159)$	160 $(136, 200)$	252 $(229, 282)$	112 $(79, 142)$	156 $(125, 194)$	19 $(16, 24)$	34 $(31, 39)$
Medium	380.274 $(346.984, 430.158)$	2.436 $(2.287, 2.665)$	328 $(296, 360)$	865 $(811, 925)$	184 $(144, 216)$	298 $(244, 365)$	40 $(36, 44)$	88 $(84, 92)$
Large	>7200	35.924 $(34.875, 37.357)$	N/A	5043 $(4849, 5203)$	N/A	653 $(559, 795)$	N/A	278 $(271, 284)$



(a) Small Scenario (b) Medium Scenario (c) Large Scenario

Fig. 3: Number of epochs (median, 10-th and 90-th percentile) required to complete classification for different scenarios.

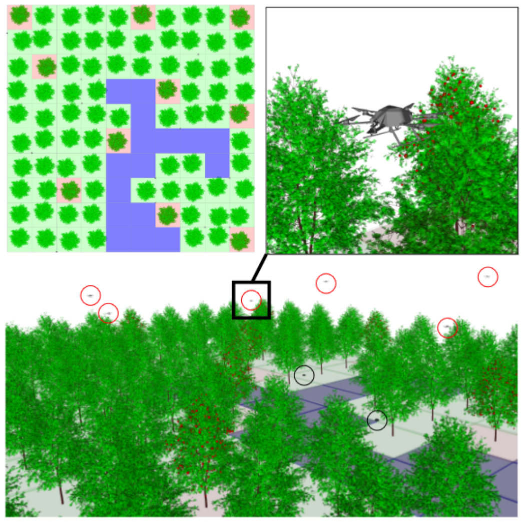


Fig. 4: View of the medium scenario, with uninteresting cells (green), interesting cells (red), and impassable terrain (blue). 2D view (left), zoom on a UAV (right) and 3D view (bottom).

[19] A. Locatelli, M. Gutzeit, and A. Carpentier, "An optimal algorithm for the thresholding bandit problem," in *Intl. Conf. Machine Learning*, 2016, pp. 1690–1698.
[20] B. Mason, L. Jain, A. Tripathy, and R. Nowak, "Finding all ϵ-good arms in stochastic bandits," *Adv. Neural Info. Process. Syst.*, 2020.
[21] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge Univ. Press, 2020.
[22] Gurobi Opt., LLC, "Gurobi Optimizer Reference Manual," https://www.gurobi.com (Last accessed: 2023).
[23] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 3–19.
[24] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Concorde tsp solver," 2006, https://www.math.uwaterloo.ca/tsp/index.html.
[25] Furrer, et. al., *RotorS—A Modular Gazebo MAV Simulator Framework*. Cham: Springer International Publishing, 2016, pp. 595–625.
[26] Robotis, last accessed: 07-13-2023. [Online]. Available: github.com/ROBOTIS-GIT/turtlebot3
[27] K.-S. Jun, K. Jamieson, R. Nowak, and X. Zhu, "Top arm identification in multi-armed bandits with batch arm pulls," in *Proc. Intl Conf. Artifi. Intelli. Stats.*, vol. 51, 2016, pp. 139–148.
[28] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, pp. 263–, 1998.

APPENDIX

A. Proof sketches for Section IV

Proof of Proposition 1): The proof follows from similar arguments used in [1, Prop. 1] and Lemma 1 in [27]. □

Proof of Proposition 2): The proof follows from similar arguments used in [1, Thm. 1]. The key insight used is that the sufficient number of samples required for successful classification, with high confidence, of a grid cell $i \in \mathcal{G}$ can be tightly upper-bounded by $\mathcal{O}(\phi_i)$ (see (6) in [27] with $\omega = \sqrt{\delta/(2|\mathcal{G}|)}$). □

Proof of Proposition 3): Recall that the shortest paths are evaluated using Dijkstra's algorithm which has a complexity of $O(|G|^2)$ [28]. `assignSensors` assigns each sensor to each epoch goal in $\mathcal{E}_p$ to one of N_d sensors after evaluating the shortest path $\left(O\left(|\mathcal{E}_p| \cdot N_d \cdot |\mathcal{G}|^2\right)\right)$. `getSensorPaths` computes at most $M_d!$ paths for each one of the N_d to arrive at the shortest path corresponding to the optimal sequence of epoch goal visits $O\left(M_d! \cdot N_d \cdot |\mathcal{G}|^2\right)$, ignoring the complexity of `getRestPoints` $(O(N_d|G|))$ The maximum number of restpoints for N_d sensors is $O(N_d|\mathcal{G}|/T_d)$. `assignStations` uses Dijkstra's algorithm to identify the assignment to N_c stations $\left(O\left(N_d \cdot N_c \cdot |\mathcal{G}|^3/T_d\right)\right)$. The complexity for `getStationPaths` follows similarly to `getSensorPaths` with M_d replaced with M_c $\left(O\left(M_c! \cdot N_c \cdot |\mathcal{G}|^2\right)\right)$. □