2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

Reduced State Value Iteration for Multi-Drone Persistent Surveillance with Charging Constraints

Patrick H. Washington and Mac Schwager

***Abstract*— This paper presents Reduced State Value Iteration (RSVI), an algorithm to compute policies for Markov Decision Processes (MDPs) that have natural checkpoints, allowing for a solution based on a reduced state space. The algorithm is applied to find policies for multiple drones to persistently surveil an environment subject to charging constraints. RSVI leverages the structure of the true MDP to build an MDP with a smaller state-action space. Monte Carlo simulations are used to estimate transitions between the states in the reduced MDP, which are used in value iteration to compute a policy for the reduced MDP. States in the true MDP are mapped to reduced states. Actions in the reduced space from the policy are then mapped to actions in the full space for execution on the true MDP. Performance of the RSVI policy improves as the state discretization becomes finer, but with increasing computational requirements, thus giving a natural trade-off between computational resources and policy suboptimality. Results of simulated persistent surveillance experiments show that our RSVI policy outperforms a baseline heuristic.**

I. Introduction

We consider a problem in which multiple drones with finite battery life must collaborate to perpetually monitor an environment, while each drone periodically recharges its batteries. As some drones recharge, others perform surveillance, so that the surveillance task is performed indefinitely. The difficulty is in scheduling the charging and surveillance activities of all the drones so that no drone runs out of battery life, while all the required surveillance paths are continually flown by some drone. We call this problem persistent surveillance, and we propose a method to compute a state feedback policy to control the collaborative decision making of the drones. This problem has applications in monitoring for traffic, crime, or other activities, monitoring disaster sites for survivors, and monitoring ecological phenomena such as ocean processes [1], temperature distributions [2], wildfire threat [3], and movements and populations of wildlife [4].

While we specifically consider multiple drones in this work, one can use the same methods for persistent surveillance with multiple cars, boats, satellites, or heterogeneous groups of surveillance vehicles. Similarly the monitoring paths may be tours over a city (or other area), through the streets of a city, or on the surface of a body of water. In all cases, the persistent surveillance problem is challenging because the agents have limited energy or fuel available, requiring them to periodically recharge or refuel. Any time an agent spends recharging is time that it is not monitoring. However, the cost associated with an agent running out of energy and crashing or getting stranded can be very large. So, planning the agents' paths and scheduling their recharging periods is critical to persistent surveillance.

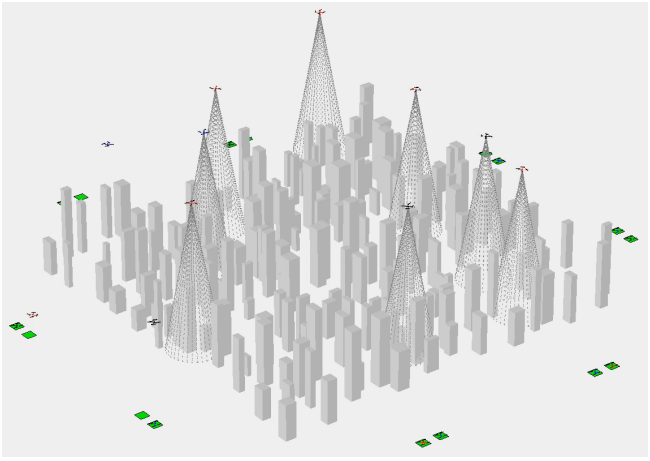

Fig. 1. Scene from a simulation in the included video (can also be found at `https://youtu.be/B4LGU-DZULM`) with several instances of the problem running in parallel. The agents fly surveillance paths over a city and periodically return to stations to charge their batteries, indicated by the green squares around the perimeter. We plan a policy by which the drones coordinate their times on the chargers vs in the air, such that each surveillance path is always occupied by some drone.

There are two main aspects of the problem. The first is concerned with finding paths that maximize information gathering. The second aspect supposes the surveillance paths have been planned, and is concerned with scheduling when each agent should follow a surveillance path, and when it should recharge while another agent follows the surveillance path. This paper focuses on solving the second part of the problem.

Our main contribution is to introduce Reduced State Value Iteration (RSVI), an algorithm that leverages the structure of certain Markov Decision Processes (MDPs) to solve for optimal policies, and apply it to the persistent surveillance problem. Specifically, we discretize the continuous surveillance paths, charging locations, and battery charge levels for the drones, and formulate the presistant surveillance problem as a discrete state MDP. We then solve for an optimal feedback policy for this reduced state representation with RSVI. Finally, we map back to the original continuous problem, using the policy on the reduced state to drive the continuous control actions. We find that the original continuous MDP

The authors are with the Department of Aeronautics & Astronautics, Stanford University. {phw, schwager} @ stanford.edu

This research was supported in part by DARPA YFA grant D18AP00064 and ONR grant N00014-18-1-2830. The first author was supported on a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

is intractable, but with an appropriate discretization, we can compute an optimal policy efficiently. This policy leads to adequate performance on the original system. With finer and finer discretization, the performance reaches an asymptote, which appears to be the optimal performance for the continous system. We achieve near-optimal performance for 3 drones using an RSVI policy computed in 7 min 39 sec on a standard laptop machine. This policy strongly outperforms a baseline heuristic, and requires significantly less computation time that existing optimal MDP methods reported in the literature [5] and similar computation time to other near-optimal methods [6].

The rest of the paper is organized as follows. Section II discusses related work. Section III reviews the basic value iteration algorithm. Section IV describes the persistent surveillance MDP and how to generate the reduced MDP. Section V describes the RSVI algorithm. Section VI contains simulated experiments that run RSVI on the persistent surveillance problem with various model resolutions.

II. Related Work

There is overlap between the path planning and scheduling aspects of the persistent surveillance problem, but the two are often approached in different ways.

A. Path Planning

Some work has focused on finding surveillance paths to follow. There can be temporal constraints regarding how often to visit regions of the space [7], [8] or objectives that seek to maximize the number of observed events and minimize time between consecutive event observations at the same location [9]. Others assign importance levels to regions of the space and have the agents decide when and where to move based on the importance levels [10], [11].

In the case of an evolving environment, one might seek to plan a trajectory that minimizes the error in environment knowledge. Sampling-based approaches have been applied to this version of the persistent surveillance problem [12], [1]. Optimal control techniques have also been applied to solve for trajectories in 1D and 2D mission spaces [13], [14]. A similar problem is planning how to follow a path in such a changing environment, where an agent uses a speed controller to decide how to monitor [15]. In some applications, communication is limited. There, agents must coordinate monitoring the environment and communicating with each other [3], [6].

The orienteering problem deals with planning routes that visit many nodes for agents that have limited range. This has been applied to persistent surveillance in [2].

Some work attempts to improve surveillance performance by coordinating teams consisting of both ground agents and flying agents, where the former can act as charging stations for the latter [16], [17], [18], [19]. The path planning problem then includes how to route the mobile charging stations.

B. Scheduling

Once the desired surveillance locations are known, the problem becomes a scheduling problem dealing with when agents should perform tasks and when they should charge. Some work has used dynamic programming or approximate dynamic programming to solve for policies that anticipate the possibility of agent failures and plan accordingly [5], [6]. The work in [5] presents an optimal dynamic programming solution for a scheduling problem similar to the one in this paper but its optimality comes at the cost of long computation times. The near-optimal solution in [6] requires similar computation time to RSVI when finding a policy for another related problem.

Other approaches seek to improve performance of the surveillance task by modifying the recharging process in addition to applying a scheduling algorithm. One approach is to swap batteries instead of recharging them to reduce downtime [20].

Some approaches try to decompose the problem into subproblems. [21] solves for an optimal partitioning for agents to divide and monitor the region. [22] is a more general hierarchical decomposition approach for multiagent MDPs and is shown to apply to persistent surveillance.

C. Model Reduction

There have been other techniques, similar to the one presented in this paper, that focus on taking large or continuous systems and reducing the problem to a more manageable form. This work mostly deals with partially observable Markov Decision Processes (POMDPs), whereas the algorithm presented in this paper assumes full state knowledge. Systems that can be described with POMDPs need to deal with large or continuous state and belief spaces. Monte Carlo Value Iteration [23] samples the space to avoid having to compute a policy over a large discretized space. Macro-MCVI [24] takes this further to improve performance for long planning horizons using macro-actions. Another similar approach from a different point of view is restricted value iteration [25], which works with subsets of the belief space to solve for policies for POMDPs.

III. Preliminaries

Here we review standard Value Iteration (VI) for solving for optimal infinite horizon discounted policies for discrete Markov Decision Processes (MDPs). In this work, we reduce a continuous-state multi-drone persistent surveillance problem to a discrete MDP approximation, solve a policy with VI, and apply the policy to control the original continuous system. Let $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ define an MDP, where $\mathcal{S}$ is a discrete set of states s, $\mathcal{A}$ a discrete set of actions a, $T(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ returns the probability of arriving at state s' when taking action a from state s, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ a reward function, and γ a discounting factor. The goal is to solve for the value function, $V(s_0) = \max_{\pi(s)} \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1})$, which returns the infinite-horizon value of being at state s_0 and acting optimally from that time forward, as well as the associated

optimal feedback policy $\pi(s) : \mathcal{S} \mapsto \mathcal{A}$. Similarly, the Q-function, $Q(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, returns the value of taking action a at state s, then acting optimally according to $\pi(s)$ afterward. Value iteration requires knowledge of the distribution $T(s,a,s')$, which is the probability of arriving at state s' when taking action a at state s, and the reward function $R(s,a,s')$, which is the reward received when taking action a at state s and arriving at state s'. The parameter $\gamma \in [0,1)$ is chosen based on how important immediate rewards are compared to future rewards.

Computing the optimal value involves iterating between the two update rules

$$Q(s,a) = \sum_{s' \in \mathcal{S}} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right], \quad (1)$$

and

$$V(s) = \begin{cases} 0 & \text{if terminal state} \\ \max_{a \in \mathcal{A}} Q(s,a) & \text{otherwise,} \end{cases} \quad (2)$$

until $V(s)$ converges. This is equivalent to computing Bellman's iteration repeatedly until convergence. After running value iteration until convergence, $Q(s,a)$ determines the optimal policy $\pi(s) : \mathcal{S} \to \mathcal{A}$ with (3).

$$\pi(s) = \arg \max_{a \in \mathcal{A}(s)} Q(s,a) \quad (3)$$

The policy acts as a lookup table where each state is mapped to the optimal action to take from that state.

IV. Model

A. Stations

There are several stations in the system that the agents fly between. There are charging stations, which are always at fixed points in space, and a surveillance station, which follows a known, cyclic path. For a system with N agents, then there are $N-1$ charging stations and 1 surveillance station. Define $c_j \in \mathbb{R}^3$ as the position of the jth charging station. Define $s(t) : \mathbb{Z} \to \mathbb{R}^3$ as the mapping from time t to the position of the surveillance station. Note that this s is different from the state s used in Section III. From this point forward, $s(t)$ refers to the surveillance path.

B. Action Representation

There are two types of actions, full and reduced. Full actions consist of a vector in $a \in \mathbb{Z}^N$ that tell each agent which station it should go to. Reduced actions consist of a single integer $\tilde{a} \in \mathbb{Z}$ that indicates whether or not a replacement agent should be sent to the surveillance station and if so, which agent should go. A reduced action $\tilde{a} = i < N$ means that the agent at charging station i should be sent as a replacement. A reduced action $\tilde{a} = N$ means that the agent at the surveillance station should stay there.

C. State Representation

There are two types of states, full and reduced.

1) Full State: The full state consists of positions and battery levels for each agent, in addition to the current time.

Let $x_i^t \in \mathbb{R}^3$ denote the position of agent i at time t. Let v denote the maximum movement speed and $p_m \in (0,1]$ denote the probability of moving. This probability accounts for potential gusts of wind or other environmental effects that could slow down the agent.

The action a^t determines the goal position for each agent at time t. If an agent is assigned to the station that it is currently at, then the agent stays at that station. For the charging station, that means staying still. For the surveillance station, that means staying locked to the surveillance path. Assume that the surveillance path is chosen such that the agents are able to follow it perfectly. If agent i is assigned to move to a charging station, its goal $g_i^t \in \mathbb{R}^3$ is simply the position of that station, $c_{a_i^t}$. If agent i is assigned to move to the surveillance station, its goal is determined by (4), which computes the first point along the surveillance path that the agent can expect to reach before the surveillance station reaches that point.

$$g_i^t = s\left(t + \min_{\Delta t \in \mathbb{Z}_+} \left\{ \Delta t \mid \|s(t+\Delta t) - x_i^t\| \le p_m v \Delta t \right\} \right) \quad (4)$$

Once each agent has a goal position, the agents move according to (5).

$$x_i^{t+1} = \begin{cases} g_i^t & \text{at station} \\ x_i^t + \frac{g_i^t - x_i^t}{\|g_i^t - x_i^t\|} \min(\|g_i^t - x_i^t\|, v) & \text{moving}, P = p_m \\ x_i^t & \text{moving}, P = 1 - p_m \end{cases} \quad (5)$$

Let $b_i^t \in [0, b_{\max}]$ denote the battery level of agent i at time t. During each time step that an agent is charging, its battery level increases by an amount $r_c \in \mathbb{R}_+$ with probability $p_c \in (0,1]$ up to a maximum battery level of $b_{\max}$. During each time step that an agent is flying, its battery level decreases by an amount $r_d \in \mathbb{R}_+$ with probability $p_d \in (0,1]$ down to a minimum battery level of 0, which corresponds to the agent dying.

$$b_i^{t+1} = \begin{cases} 0 & b_i^t = 0, \text{ with } P = 1 \\ \min(b_i^t + r_c, b_{\max}) & \text{charging, with } P = p_c \\ b_i^t & \text{charging, with } P = 1 - p_c \\ \max(b_i^t - r_d, 0) & \text{flying, with } P = p_d \\ b_i^t & \text{flying, with } P = 1 - p_d \end{cases} \quad (6)$$

2) Reduced State: The reduced state representation consists of an integer battery level for each agent and the time. No explicit position is required because the reduced state is only valid when every agent is at its own station and the battery levels can be ordered by station index (i.e., b_2^t is the battery of the agent at station 2 at time t).

The dynamics of the reduced state are very similar to those of the full state. If action $\tilde{a} = N$, then no replacement is sent and the system only steps forward by one time step and no agents switch stations. If $\tilde{a} = i < N$, then the agent at charging station i is sent as a replacement. The replacement agent will travel to the surveillance station. When that agent arrives, the agent that was at the surveillance station goes

to the charging station that the replacement came from. In this case, the position of each agent is set according to the station it is at and then the position propagates according to (4) and (5). Once the replacement is complete, the agents are ordered again according to their new stations. Some number of time steps Δt passes, where $\Delta t = 1$ if no replacement was sent and Δt is the time steps for the replacement if there was one.

At every time step during the state transition, the battery evolves almost identically to the full system. The structure is the same but the rates, probabilities, and maximum level change. The maximum battery level $\tilde{b}_{\text{max}} \leq b_{\text{max}}$ is an integer parameter to be chosen. Battery levels change in increments of 1. The charge and discharge probabilities are set to make the expected charge and discharge times the same as the corresponding full system.

$$\tilde{p}_c = r_c p_c \frac{\tilde{b}_{\text{max}}}{b_{\text{max}}}, \qquad \tilde{p}_d = r_d p_d \frac{\tilde{b}_{\text{max}}}{b_{\text{max}}}$$

The battery levels then evolve according to (7)

$$\tilde{b}_i^{t+1} = \begin{cases} \min(\tilde{b}_i^t + 1, \tilde{b}_{\text{max}}) & \text{charging, with } P = \tilde{p}_c \\ \tilde{b}_i^t & \text{charging, with } P = 1 - \tilde{p}_c \\ \max(\tilde{b}_i^t - 1, 0) & \text{flying, with } P = \tilde{p}_d \\ \tilde{b}_i^t & \text{flying, with } P = 1 - \tilde{p}_d \end{cases} \tag{7}$$

for the Δt time steps that the state transition takes. If any battery level hits 0, transition to a special dead state, $\tilde{\sigma}_{\text{dead}}$.

The full process for computing a successor state in the reduced state space is detailed in Algorithm 1.

3) State Scaling: For a system with N agents, the state representation using the full position model is

$$\sigma = (b_1^t, ..., b_N^t, x_1^t, ..., x_N^t, t)$$

Suppose there are n_b possible battery levels, n_x possible positions, and n_t points on the surveillance path. The number of unique states scales with $O(n_b^N n_x^N n_t)$. For real systems with continuous battery levels and positions, n_b and n_x can be infinite. Discretization is possible, particularly for the battery level. However, discretizing $\mathbb{R}^3$ for a finite n_x can still give a very large number of unique states.

For the same system, the state representation using the reduced position model is

$$\tilde{\sigma} = (\tilde{b}_1^t, ..., \tilde{b}_N^t, t)$$

The number of unique states scales with $O(\tilde{b}_{\text{max}}^N n_t)$.

D. Conversion

The policy is computed on the reduced-state system but execution happens on the full-state system. If all agents are at unique stations, then the full state can be converted to a reduced state. Algorithm 2 details the procedure. If any battery levels in the full state are 0, then the reduced state is the special dead state, $\tilde{\sigma}_{\text{dead}}$. Otherwise, the battery level in the reduced state is set such that the ratio of the battery level and the maximum capacity are approximately equal for the full state and the reduced state. The reduced state level is always rounded down to an integer because it means that decisions are made with conservative estimates of the battery level. Acting on an overestimate could easily lead to an agent's battery dying. Note that the minimum reduced battery level for agents that are not dead is 1. If a reduced battery level would otherwise be 0 but the full battery level is nonzero, then it is raised to 1 because it should act as if it has a very low battery level but is not dead yet. It would be possible to have a reduced level of 0 that is not the dead state but that does not meaningfully change anything.

The policy maps reduced states to reduced actions. The full-state system requires full actions to operate. Algorithm 3 details the procedure to compute the full action given a reduced action and a full state. If the reduced action is to not send a replacement, then the corresponding full action is a vector $a \in \mathbb{Z}^N$ where a_i is the index of the station that agent i is at in the full state. If the reduced action calls for a replacement, then there are two corresponding full actions.

Algorithm 1 GetReducedSuccessor($\tilde{\sigma}, \tilde{a}$)

```
Compute a successor state given a reduced state and action
(b̃_1^t, ..., b̃_N^t, t) ← σ̃
if ã = N then                                   ▷ No replacement
    compute each b̃_i^{t+1} with (7)
    if any b̃_i^{t+1} = 0 then
        return σ̃_dead
    end if
    σ̃' ← (b̃_1^{t+1}, ..., b̃_N^{t+1}, t + 1)
else                                            ▷ Sending replacement
    for i = 1, ..., N − 1 do
        x_i^t ← c_i
    end for
    x_N^t ← s(t)
    for i = 1, ..., N do
        a_i ← i
    end for
    a_ã ← N                                     ▷ Assign replacement
    while true do
        compute each x_i^{t+1} according to (4) and (5)
        compute each b̃_i^{t+1} according to (7)
        if any b̃_i^{t+1} = 0 then
            return σ̃_dead
        end if
        if x_ã^{t+1} ≠ s(t + 1) then            ▷ If replacement arrived
            a_N ← ã                             ▷ Send to vacated charger
        end if
        if x_N^{t+1} = c_ã then
            break                               ▷ Replacement is complete
        end if
        t ← t + 1
    end while
    swap b̃_ã^t and b̃_N^t
    σ̃' ← (b̃_1^t, ..., b̃_N^t, t)
end if
return σ̃'
```

Algorithm 2 ConvertToReducedState(σ)

Convert a full state to a reduced state
Assume all agents are at unique stations
$(b_1^t, ..., b_N^t, x_1^t, ..., x_N^t, t) \leftarrow \sigma$
for $i = 1, ..., N$ **do**
  if $b_i^t = 0$ **then**
    return $\tilde{\sigma}_{\text{dead}}$
  end if
  if $x_i^t = s(t)$ **then**
    $\tilde{b}_N \leftarrow \max\left(\text{floor}(b_i^t \cdot \tilde{b}_{\max}/b_{\max}), 1\right)$
  else
    for $j = 1, ..., N-1$ **do**
      if $x_i^t = c_j$ **then**
        $\tilde{b}_j \leftarrow \max\left(\text{floor}(b_i^t \cdot \tilde{b}_{\max}/b_{\max}), 1\right)$
      end if
    end for
  end if
end for
$\tilde{\sigma} \leftarrow (\tilde{b}_1, ..., \tilde{b}_N, t)$
return $\tilde{\sigma}$

Algorithm 3 ConvertToFullAction($\tilde{a}, \sigma$)

Convert a reduced action to a single full action or a pair of full actions given the full state
Assume all agents are at unique stations
$(b_1^t, ..., b_N^t, x_1^t, ..., x_N^t, t) \leftarrow \sigma$
for $i = 1, ..., N$ **do**
  $a_i \leftarrow i$
end for
if $\tilde{a} = N$ **then**
  return a ▷ single action to do nothing
end if
$a^1 \leftarrow a$ ▷ first action to send replacement
$a^2 \leftarrow a$ ▷ second action to return to charger
for $i = 1, ..., N$ **do**
  if $x_i^t = c_j$ **then**
    $a_{\tilde{a}}^1 \leftarrow N$
    $a_{\tilde{a}}^2 \leftarrow N$
  end if
  if $x_i^t = s(t)$ **then**
    $a_i^2 \leftarrow \tilde{a}$
  end if
end for
return (a^1, a^2)

The first, a^1 sends the replacement from a charging station to the surveillance station with $a_{\tilde{a}}^1 = N$ where $\tilde{a}$ is the reduced action, which is the index of the charging station where the replacement agent starts. Once the replacement reaches the surveillance station, the second action keeps the replacement at the surveillance station with $a_{\tilde{a}}^2 = N$ and sends the agent that was already at the surveillance station to the now-vacated charging station with $a_N^2 = \tilde{a}$.

V. Reduced State Value Iteration

Our main contribution is to demonstrate that a policy for the full state MDP can be computed using the reduced state MDP. The former is able to express the complete state of the system but scales too poorly to be of any practical use in many problems. The latter is not able to fully describe the state of the system at any arbitrary point in time but can describe the state at times relevant to making decisions and has much better scaling.

A. Computing the Policy

The proposed method leverages the structure of the surveillance problem. First, agents are almost always at stations since replacements are infrequent. This is what allows the reduced state to be sufficient at most time steps. Second, at most one agent is ever flying between stations at a given time and no other decisions are needed during that time. This means that the policy only needs to be defined at states that can be represented by the reduced state model since it is not making decisions when the full state model is required.

When using the full state representation, determining the possible successor states is a matter of applying (5) and (6) for the position and battery transitions. However, determining the possible successor states is nontrivial when using the reduced state representation. When no agents are commanded to switch stations, the transition only uses (7). However, when an agent is told to switch stations, there are potentially infinitely many combinations of travel times, battery levels, and intercept positions from the stochastic position and battery evolutions, meaning that determining the exact state transition distributions is impractical.

Monte Carlo simulations are used instead. With a finite number of simulations, some possible successors are missed, but the nontrivial probabilities can be estimated. Apply Algorithm 4 to generate transition distribution estimates. The idea is that when an action that involves station switching is commanded, each agent is assigned a position based on its current station. Then, many simulations are run. Each simulation runs until the station switching is completed, at which point the agents are at unique stations and a reduced state can describe the system again. The resulting reduced states provide an estimate for the transition distribution.

After estimating the transition distributions for each state-action pair that involves station switching, value iteration is performed. Initialize all state values to zero then iterate over all state-action pairs to update Q_{opt} and V_{opt} with (1) and (2) until convergence. The full procedure to compute a policy is detailed in Algorithm 5.

B. Executing the Policy

Once the policy is computed, the full-state system must execute it. At each relevant time step, this involves converting

Algorithm 4 EstimateReducedTransitions($\tilde{\sigma}, \tilde{a}$)

Initialize $\hat{T}(\tilde{\sigma}, \tilde{a}, \tilde{\sigma}')$ to 0 for each possible successor $\tilde{\sigma}'$
for $k = 1, ..., n_{\text{sim}}$ **do**
  $\tilde{\sigma}' \leftarrow$ GetReducedSuccessor($\tilde{\sigma}, \tilde{a}$)
  $\hat{T}(\tilde{\sigma}, \tilde{a}, \tilde{\sigma}') \leftarrow \hat{T}(\tilde{\sigma}, \tilde{a}, \tilde{\sigma}') + 1/n_{\text{sim}}$
end for
return $\hat{T}$

Algorithm 5 ReducedStateValueIteration

Enumerate all reduced states $\tilde{\sigma}$
Enumerate all reduced actions $\tilde{a}$
for each $(\tilde{\sigma}, \tilde{a})$ pair **do**
  $\hat{T} \leftarrow$ EstimateReducedTransitions($\tilde{\sigma}, \tilde{a}$)
end for
$V(\tilde{\sigma}) \leftarrow 0$ for each $\tilde{\sigma}$
$Q(\tilde{\sigma}, \tilde{a}) \leftarrow 0$ for each $(\tilde{\sigma}, \tilde{a})$ pair
while V not converged **do**
  Update Q with (1), using $\hat{T}$ in place of T
  Update V with (2)
end while
Compute policy $\pi(\tilde{\sigma})$ with (3)
return π

Algorithm 6 ExecutePolicy(σ, π)

while $\sigma \neq \sigma_{\text{dead}}$ **do** ▷ Can use any stopping metric
  $\tilde{\sigma} \leftarrow$ ConvertToReducedState(σ)
  $\tilde{a} \leftarrow \pi(\tilde{\sigma})$
  if $\tilde{a} = N$ **then**
    $a \leftarrow$ ConvertToFullAction($\tilde{a}, \sigma$)
    Step forward using (5), (6) with action a
  else
    $(a^1, a^2) \leftarrow$ ConvertToFullAction($\tilde{a}, \sigma$)
    while agent going to surveillance **do**
      Propagate σ using (4),(5),(6) with action a^1
    end while
    while agent returning to charge **do**
      Propagate σ using (4),(5),(6) with action a^2
    end while
  end if
end while

the full state to a reduced state, looking up the optimal reduced action, converting it to the corresponding full action(s), and executing the action(s). This procedure is detailed in Algorithm 6.

VI. Numerical Experiments

These tests involve three agents maintaining coverage at a surveillance station that follows a circular path. The parameters are $b_{\max} = 50$, $r_c = 1$, $p_c = 1$, $r_d = 1$, $p_d = 1$, $p_m = 0.9$, and $v = 1$. Agents that start at charging stations start with full battery. The agent that starts at the surveillance station starts with 50% battery. The charging positions and surveillance path are defined by

$$c_1 = \begin{bmatrix} -0.25 & 0 & 0 \end{bmatrix}^T,$$

$$c_2 = \begin{bmatrix} 0.25 & 0 & 0 \end{bmatrix}^T,$$

$$s(t) = \begin{bmatrix} 2\cos\left(\frac{2\pi t}{25}\right) & 2\sin\left(\frac{2\pi t}{25}\right) + 3 & 4 \end{bmatrix}^T,$$

where $2\pi/25$ was chosen to keep the required speed at approximately 0.5. A circular path is relatively simple but it is a realistic path and serves an important purpose. The surveillance station moves toward and away from the charging stations, forcing the policy to consider both battery levels and changing travel times for replacements. In this arrangement, the difference between the closest point and furthest point in the expected time to complete a replacement corresponds to depleting an extra 10% of the total battery capacity. Repeatedly calling for longer replacements wastes a large amount of battery, increasing the chance of an agent dying.

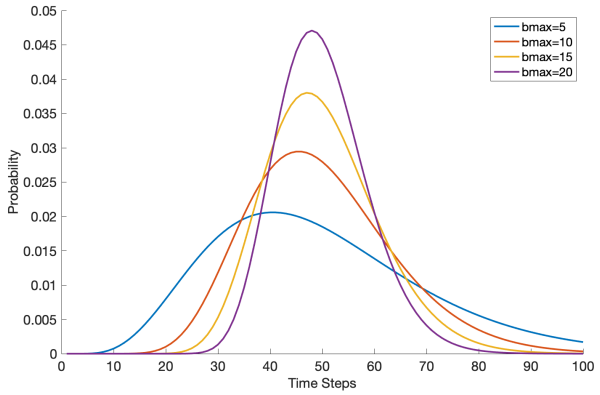

Fig. 2. Probability distribution for the charge or discharge time of an agent's battery for the four values of $\tilde{b}_{\max}$ used. The expected time is 50 for all levels but smaller $\tilde{b}_{\max}$ leads to more uncertainty in how long the battery will take to charge or discharge because of the smaller maximum level and corresponding charge and discharge probabilities. The distribution is 0 for time steps less than $\tilde{b}_{\max}$ and only time steps up to double the expected time are shown.

The trials use four different maximum battery levels for the reduced state. The values tested are $\tilde{b}_{\max} = 5, 10, 15$, and 20. Figure 2 demonstrates how $\tilde{b}_{\max}$ affects the uncertainty in the time to fully charge or discharge the battery. There are 100 Monte Carlo simulations run for every state-action pair.

Each evaluated policy has 1000 trials, each running for up to 100000 time steps, which is 1000 times the expected time for a full charge-discharge cycle of a battery.

1) State: The full state of the system is

$$\sigma = (b_1^t, b_2^t, b_3^t, x_1^t, x_2^t, x_3^t, t),$$

where b_i^t is the battery level of agent i and x_i^t is the position of agent i. The reduced state of the system is

$$\tilde{\sigma} = (\tilde{b}_1^t, \tilde{b}_2^t, \tilde{b}_3^t, t),$$

where $\tilde{b}_1^t$ is the reduced state battery level of the agent at the first charging station, $\tilde{b}_2^t$ is the reduced state battery level of the agent at the second charging station, and $\tilde{b}_3^t$ is the reduced state battery level of the agent at the surveillance station.

2) Action Space: Only the reduced state is used to determine the action at any given time. Since one agent is at the surveillance station and two agents are charging and ready to send as replacements, the reduced action space is $\tilde{\mathcal{A}} = \{1, 2, 3\}$, which correspond to sending the agent at the first charging station as a replacement, sending the agent at the second charging station as a replacement, and sending no replacement, respectively.

3) Reward: The reward signal for this system gives a small reward for each time step that all of the agents have positive battery levels and a large penalty for any agent dying. This is described in (8).

$$R(\sigma, a, \sigma') = \begin{cases} 1 & \text{if } \sigma' \neq \sigma_{\text{dead}} \\ -1000 & \text{otherwise} \end{cases} \tag{8}$$

The discount factor used in value iteration is $\gamma = 0.99$. Value iteration is run until no value changes by more than 0.001 in an iteration.

4) Baseline Policy: A baseline policy is used as a point of comparison. It estimates the remaining battery life of the agent at the surveillance station and the expected time for a replacement if the charging agent with the highest battery level is sent, then sends that when it expects the current surveillance agent to be able to get to a charging station with some threshold battery level remaining. The estimated replacement time uses (4) to determine the travel distance. A replacement is sent as soon as the inequality in (9) is satisfied.

$$\left(\frac{b_3^t}{r_d p_d} - \frac{2\|g_{\text{max}}^t - c_{\text{max}}\|}{v p_m} \right) r_d p_d \leq b_{\text{thresh}}, \tag{9}$$

where c_{max} is the position of the charging station where the charging agent with the highest battery level is at time t and g_{max}^t is the goal position of that agent if it is assigned as a replacement to go to the surveillance station. The threshold battery level, b_{thresh}, for these tests is 5, which is 10% of the maximum battery level.

5) Results: Table I shows the results of the 1000 trials for each policy. The policy computed with $\tilde{b}_{\text{max}} = 5$ performs poorly. This is to be expected because the corresponding reduced-state model has very low resolution. It places a nonzero probability on an agent going from a full battery to a dead battery in only 5 time steps and must plan accordingly. Increasing $\tilde{b}_{\text{max}}$ leads to much better performance, though there are diminishing returns. The policy using $\tilde{b}_{\text{max}} = 10$ is significantly better than the policy using $\tilde{b}_{\text{max}} = 5$ and the policy using $\tilde{b}_{\text{max}} = 15$ is even better. However, the policy computed with $\tilde{b}_{\text{max}} = 20$ shows little improvement over $\tilde{b}_{\text{max}} = 15$ despite a sizeable increase in computation time.

Policy	States	Compute	Mean	Median	Finished
Baseline	-	-	1118	757.5	0
5	3126	15s	1287	198	0
10	25001	2m 21s	89781	100000	82.4%
15	84376	7m 39s	95238	100000	93.8%
20	200001	19m 57s	96939	100000	95.2%

TABLE I

This shows the computation time, mean end time, median end time, and the percentage of simulations that reached 100000 time steps. The mean and median for the policies generated with $\tilde{b}_{\text{max}} = 10, 15$, and 20 are kept artificially low by the 100000 time step limit.

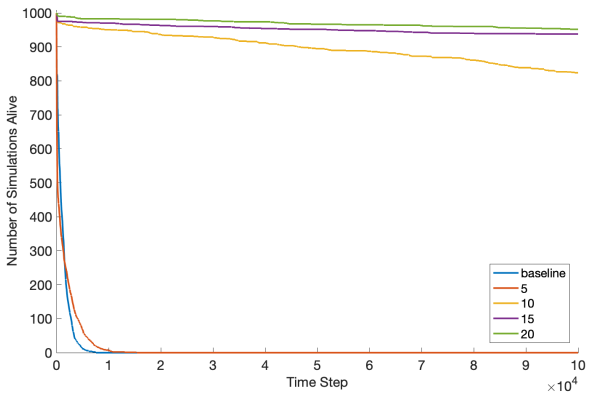

Fig. 3. Number of simulations still alive at each time step for each $\tilde{b}_{\text{max}}$ value. The number increases with $\tilde{b}_{\text{max}}$, which means that a higher $\tilde{b}_{\text{max}}$ kept more of the 1000 simulations alive for longer. There are diminishing returns, with $\tilde{b}_{\text{max}} = 15$ and 20 having very similar results.

These times compare favorably to the required 36 hours to find the optimal solution for a simpler problem in [5]. The times are similar to the 5 minutes required for a near-optimal solution in [6], which computes a policy for a system that has far fewer possible agent positions but adds inter-agent communication requirements.

Fig. 3 shows how many simulations still had living agents at each time step. The plot demonstrates why the $\tilde{b}_{\text{max}} = 5$ policy has a higher mean than the baseline but a lower median. Many of its simulations fail very early but then some stay alive for long enough to drag the mean up. The plot shows that the policy computed with $\tilde{b}_{\text{max}} = 5$ and the baseline are similar, the policy computed with $\tilde{b}_{\text{max}} = 10$ is significantly better than those, and the policies computed with $\tilde{b}_{\text{max}} = 15$ and 20 are clearly better than the others, though not very different from each other.

Fig. 4 shows the average battery level of the agents in trials that have not died yet. This demonstrates how well the policies perform up until an agent dies. The baseline policy has the lowest average value, followed by the computed policies in order of $\tilde{b}_{\text{max}}$. This indicates that the better policies are operating further from death than the worse policies, which makes sense given how many simulations reached the

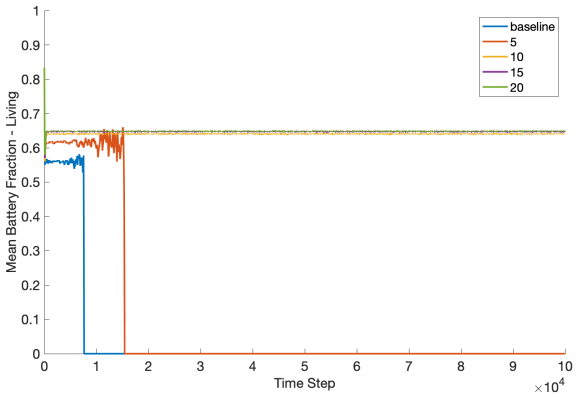

Fig. 4. Mean battery level of agents at each time step for each $\tilde{b}_{\max}$ value, discarding dead states. The data is smoothed. As the last few die in the baseline and $\tilde{b}_{\max} = 5$ policy, their mean values oscillate rapidly because there are so few left alive that losing each one can have a large effect. The performance difference is not as large, though the mean battery level is still ordered by $\tilde{b}_{\max}$.

time step limit for each one.

VII. Conclusion

We have presented Reduced State Value Iteration and demonstrated its ability to compute policies for the persistent surveillance problem. The performance levels and diminishing improvements for increasing $\tilde{b}_{\max}$ indicate that the better policies are near-optimal, though we do not have a formal guarantee of the distance to optimal. Some failures are to be expected due to the stochastic nature of the problem so there is limited room for improvement, if any. Removing the restriction that replacement agents may not be sent unless all agents are at unique stations could help but that would remove the ability to only compute policies that act on reduced states. Similarly, removing the requirement that an agent must always be at the surveillance station could further reduce the failure rate, though that relaxation may compromise performance of the surveillance task.

References

[1] X. Lan and M. Schwager, "Rapidly exploring random cycles: Persistent estimation of spatiotemporal fields with multiple sensing robots," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1230–1244, 2016.

[2] J. Yu, M. Schwager, and D. Rus, "Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 342–349.

[3] R. N. Haksar, S. Trimpe, and M. Schwager, "Spatial Scheduling of Informative Meetings for Multi-Agent Persistent Coverage," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3027–3034, Apr. 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9001230/

[4] K. Shah, G. Ballard, A. Schmidt, and M. Schwager, "Multidrone aerial surveys of penguin colonies in Antarctica," *Science Robotics*, vol. 5, no. 47, p. eabc3000, Oct. 2020. [Online]. Available: https://robotics.sciencemag.org/lookup/doi/10.1126/scirobotics.abc3000

[5] B. Bethke, J. P. How, and J. Vian, "Group health management of uav teams with applications to persistent surveillance," in *2008 American Control Conference*, 2008, pp. 3145–3150.

[6] B. Bethke, J. How, and J. Vian, "Multi-uav persistent surveillance with communication constraints and health management," 08 2009.

[7] K. Leahy, D. Zhou, C.-I. Vasile, K. Oikonomopoulos, M. Schwager, and C. Belta, "Provably Correct Persistent Surveillance for Unmanned Aerial Vehicles Subject to Charging Constraints," in *Experimental Robotics*, M. A. Hsieh, O. Khatib, and V. Kumar, Eds., vol. 109. Cham: Springer International Publishing, 2016, pp. 605–619. [Online]. Available: http://link.springer.com/10.1007/978-3-319-23778-7_40

[8] ——, "Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints," *Autonomous Robots*, vol. 40, no. 8, pp. 1363–1378, Dec. 2016. [Online]. Available: http://link.springer.com/10.1007/s10514-015-9519-z

[9] J. Yu, S. Karaman, and D. Rus, "Persistent monitoring of events with stochastic arrivals at multiple stations," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 521–535, 2015.

[10] N. Michael, E. Stump, and K. Mohta, "Persistent surveillance with a team of mavs," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2708–2714.

[11] E. Stump and N. Michael, "Multi-robot persistent surveillance planning as a vehicle routing problem," in *2011 IEEE International Conference on Automation Science and Engineering*, 2011, pp. 569–575.

[12] X. Lan and M. Schwager, "Planning periodic persistent monitoring trajectories for sensing robots in gaussian random fields," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2415–2420.

[13] C. G. Cassandras, X. Lin, and X. Ding, "An optimal control approach to the multi-agent persistent monitoring problem," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 947–961, 2013.

[14] X. Lin and C. G. Cassandras, "An optimal control approach to the multi-agent persistent monitoring problem in two-dimensional spaces," *IEEE Transactions on Automatic Control*, vol. 60, no. 6, pp. 1659–1664, 2015.

[15] S. L. Smith, M. Schwager, and D. Rus, "Persistent monitoring of changing environments using a robot with limited range sensing," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5448–5455.

[16] S. Seyedi, Y. Yazicioğlu, and D. Aksaray, "Persistent surveillance with energy-constrained uavs and mobile charging stations," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 193–198, 2019, 8th IFAC Workshop on Distributed Estimation and Control in Networked Systems NECSYS 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319320087

[17] N. Mathew, S. L. Smith, and S. L. Waslander, "A graph-based approach to multi-robot rendezvous for recharging in persistent tasks," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3497–3502.

[18] K. Yu, A. K. Budhiraja, and P. Tokekar, "Algorithms for routing of unmanned aerial vehicles with mobile recharging stations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 5720–5725.

[19] P. Maini, K. Yu, P. B. Sujit, and P. Tokekar, "Persistent monitoring with refueling on a terrain using a team of aerial and ground robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 8493–8498.

[20] N. K. Ure, G. Chowdhary, T. Toksoz, J. P. How, M. A. Vavrina, and J. Vian, "An automated battery management system to enable persistent missions with multiple aerial vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 1, pp. 275–286, 2015.

[21] N. Nigam and I. Kroo, "Persistent surveillance using multiple unmanned air vehicles," in *2008 IEEE Aerospace Conference*, 2008, pp. 1–14.

[22] Y. F. Chen, N. K. Ure, G. Chowdhary, J. P. How, and J. Vian, "Planning for large-scale multiagent problems via hierarchical decomposition with applications to uav health management," in *2014 American Control Conference*, 2014, pp. 1279–1285.

[23] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, *Monte Carlo Value Iteration for Continuous-State POMDPs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 175–191. [Online]. Available: https://doi.org/10.1007/978-3-642-17452-0_11

[24] Z. Lim, L. Sun, and D. Hsu, "Monte carlo value iteration with macro-actions," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/eefc9e10ebdc4a2333b42\\b2dbb8f27b6-Paper.pdf

[25] W. Zhang and N. Zhang, "Restricted value iteration: Theory and algorithms," *J. Artif. Intell. Res.*, vol. 23, pp. 123–165, 2005.