Reinforcement Learning Based Persistent Surveillance on Graphs Under Energy Constraints

Nishchal Hoysal G[1] Aravind S[1] Lalit Jayanti[2] and Pavankumar Tallapragada[1]

Abstract— **In this work, we consider the problem of optimally routing a single agent on a graph for persistent surveillance and reporting under energy constraints. We aim to minimize a combination of weighted graph idleness, based on node priorities, and the time elapsed since the agent's last report. This requires repeatedly solving 'similar' computationally challenging optimization problems. To this end, we adopt a rolling horizon approach and use a deep Q-network (D3QN) based algorithm to learn policies that, given the graph state and the agent's location, decide the agent's next node to visit, while guaranteeing the satisfaction of energy constraints. Through exhaustive simulations and comparisons against an integer programming (IP) solver, we demonstrate near-optimality of our approach. Our method also provides several orders of magnitude reduction in computation time over the IP solver. We also compare our approach with five other frameworks, namely, Q learning based Black Box Learning Agent (BBLA), Entropy Maximized Patrolling (EMP), Travelling salesman problem (TSP) heuristic, Concurrent Bayesian Learning Strategy (CBLS), a graph attention (GAT) based learning method.**

I. INTRODUCTION

Surveillance by mobile agents is critical in areas of public safety, border security, search and rescue operations etc. We model such a problem as one of optimal routing on a graph under energy constraints. Typically, such problems are computationally challenging and not scalable. In this paper, we propose a reinforcement learning (RL) based scalable and near-optimal approach to solve this problem.

Related work: Persistent surveillance/patrolling problem has been studied for several decades and a recent survey [1] describes the regular and adversarial surveillance/patrolling problems in the literature. Among the several approaches to the surveillance problem in the literature, a popular one is that of optimization-based routing on graphs. Some prominent examples of this approach for single or multi-robot surveillance include [2]–[6]. Much of this literature is in the context of centralized computation though. A recent work [7] describes a distributed algorithm for communication constrained and coordinated multi-robot path planning. For persistent surveillance, a common approach is to use a rolling horizon planning or model predictive control as in [8]. Works like [9] take travelling salesman problem (TSP) based approach to persistent surveillance.

A major drawback of optimization based routing for surveillance is that it often tends to be a combinatorial problem with very poor computational scalability. Hence, in recent years, several research groups are exploring the use of learning-based approaches to surveillance problems. Work [10] proposes an RL-based patrolling problem with the aim of minimizing average inter-visit times to different nodes. Works like [11], [12] explore a Bayesian Learning approach whereas [13] proposes an LSTM-based multi-agent patrolling algorithm. Works like [14] and [15] use graph attention and CommNet. While most of these works focus on just persistenBBLAt 'surveillance', there is a recent body of work which also introduces the notion of reporting the collected information. Works like [16]–[20] have discussed the notion of base nodes/stations where an agent has to regularly visit/connect to upload the collected information. These works provide sub-optimal heuristic solutions since naive optimization methods scale badly in terms of computation.

Literature gap and Contributions: Existing literature in optimal graph surveillance either propose (i) optimization-based methods like [8], which are computationally inefficient and not scalable, (ii) heuristic solutions [9], [16], [17] are specific to a certain set of scenarios and it is not directly clear how to adapt the methods for other scenarios, (iii) learning-based methods which are computationally efficient but present comparisons against some existing heuristics [10], [12], [14] or do not work with general graphs with zero surveillance priority nodes [21] and without any provision for reporting. Further, the possibility of dynamic nature of some problem parameters, for eg. time varying survey node priorities, are not considered.

The following are the contributions of this work.

(i) We propose an RL aided rolling horizon framework, with attention based novel Q-network architecture, for single-agent persistent surveillance and reporting on a graph with dynamic node surveillance priorities, with guaranteed energy constraint satisfaction.

(ii) The framework allows for time varying (possibly stochastic) node priorities, indicating their relative importance in the overall surveillance and reporting task. Due to good generalizability and low inference times of RL based policies, it is possible for our approach to adapt quickly and re-plan near-optimal solutions when the surveillance parameters change.

Through extensive simulations, we demonstrate that the proposed framework provides near-optimal solutions with great computational efficiency. We train and test on a set of randomly generated graphs with various number of nodes and topologies. We also demonstrate that our proposed method performs significantly better compared five other algorithms, namely, Q learning based Black Box Learning Agent (BBLA) [10], TSP based heuristic [9], Entropy Maximized Patrolling (EMP) [21], CBLS [12] and GAT based learning [14]. This strongly indicates that the proposed method is generalizable to graphs of various sizes and topologies.

Notation: We denote the sets of all real numbers, non-negative real numbers, integers and non-negative integers by $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{Z}$ and $\mathbb{N}_0$, respectively. For $a, b \in \mathbb{R}$, we let

[1]Nishchal Hoysal G, Aravind S and Pavankumar Tallapragada are associated with Indian Institute of Science `nishchalg, aravinds1@iisc.ac.in, pavant@iisc.ac.in`

[2]Lalit Jayanti is associated with the Department of Mechanical Engineering, Indian Institute of Technology Madras `jayanti.lalit@gmail.com`

$[a,b]_{\mathbb{Z}} := [a,b] \cap \mathbb{Z}$ and $[a,b)_{\mathbb{Z}} := [a,b) \cap \mathbb{Z}$. A weighted directed graph is an ordered tuple (V,E,W), where V is the set of nodes, $E \subseteq \{(i,j) \mid i,j \in V\}$ is the set of directed edges between the nodes and $W = \{w(i,j) \in \mathbb{N}_0 | (i,j) \in E\}$ is a set of weights associated with the edges. We denote the set of out-neighbors of node i along with node i as $\bar{\mathcal{N}}(i) := \{j \in V | (i,j) \in E\} \cup \{i\}$. By $\Omega(i,\bar{V})$, we denote the cost along the least cost path among all paths going from node $i \in V$ to some node $j \in \bar{V} \subset V$, where the edge costs are the weights W. We denote a function $f : \mathbb{R}^n \to \mathbb{R}^m$ parametrized by $\theta \in \mathbb{R}^l$, evaluated at $b \in \mathbb{R}^n$ as $f(b;\theta)$. We represent the uniform distribution on a set S as $\mathbb{U}(S)$.

II. PROBLEM SET-UP

We consider the problem of persistent surveillance and reporting on a strongly connected directed graph $G = (V,E,W)$ by a single agent, in a discrete-time framework. The node set V modelling the set of places to be visited, consists of two kinds of nodes, namely, survey nodes and base nodes. The set of survey nodes, denoted by V_s, models the set of places to be surveyed and the set of base nodes, denoted by V_b, models the places where the agent has to occasionally report the survey data. Edge $(i,j) \in E$ models a direct route from node $i \in V$ to node $j \in V$ and the weight $w(i,j) \in W$ models the travel time along the associated edge (i,j). Since we consider a discrete-time model with $t \in \mathbb{N}_0$ as the discrete time variable, we also let the travel times $w(i,j) \in \mathbb{N}_0$ for each $(i,j) \in E$. Unless mentioned otherwise, we assume the graph has a self-loop at each node $i \in V$, i.e., $(i,i) \in E$, and we let $w(i,i) = 1$ for all $i \in V$.

We model the surveillance and reporting problem as one of routing an agent on the graph. The sequence of actions of the agent is the sequence of nodes it visits. In other words, the agent takes an action only when it is at a node and not when it is on an edge between two nodes. Hence, we keep track of $(t_m)_{m \in \mathbb{N}_0}$, the sequence of time steps when the agent is at a node. In particular, t_m is the m^{th} time step at which the agent is at a node in V. In the sequel, for brevity, we refer to the overall surveillance and reporting task (problem) as the surveillance task (problem), unless mentioned otherwise.

We associate a *priority* $p_i(t) \in \mathbb{R}_{\geq 0}$ to each node $i \in V$ at each time-step $t \in \mathbb{N}$. Priority models the nodes' importance at time t in the overall surveillance task. In the current work, the priority of a base node is *zero* for all time, i.e., $p_j(t) = 0,\ \forall j \in V_b,\ \forall t \in \mathbb{N}_0$. Priorities are user-defined, possibly random quantities and are useful for incorporating exogenous information into the surveillance problem or to make the routing pattern of the agent time-varying and difficult to predict by an adversary. For example, the priorities may be determined based on historical data on events of interest or topography of the regions that are abstracted as the nodes in the graph. In this work, we do not assume any particular model or pattern in how the priorities change, except that they change somewhat slowly and that the priorities $p_i(t)$ of all nodes are known only at and after time step t.

For this reason, we pose the persistent surveillance problem as a rolling horizon optimal control problem in order to adapt and re-plan for possible changes in priorities as well as for computational tractability. In particular, we solve a fixed horizon optimal control problem with planning horizon length $T > 0$ at each time step in the sequence $(t_m)_{m \in \mathbb{N}_0}$ and at t_m, the agent takes the first action in the plan computed at t_m and discards the rest. Thus, to keep the exposition simple, we mainly describe a single instance of the finite horizon problem, wherein the priorities are assumed to be constant for the considered horizon.

We denote the *location* of the agent at a time step $t \in [0,T]_{\mathbb{Z}}$ in the planning horizon by $l(t)$. Thus,

$$l(t) = i, \text{ if agent is at node } i \in V \text{ at time } t \tag{1a}$$
$$l(t) \notin V, \text{ if the agent is on an edge at time } t. \tag{1b}$$

In the plan computed at t_m, the agent takes the action $a(m) \in \bar{\mathcal{N}}(l(t_m))$, which is the set of neighbors of the node $l(t_m)$ (including $l(t_m)$ itself) and the agent would reach the node $a(m)$ at time $t_m + w(l(t_m), a(m))$. That is,

$$a(m) \in \bar{\mathcal{N}}(l(t_m)) \tag{2a}$$
$$t_{m+1} = t_m + w(l(t_m), a(m)) \tag{2b}$$
$$l(t_{m+1}) = a(m). \tag{2c}$$

We assume that the agent has a finite energy capacity $\bar{e} \in \mathbb{N}_0$. The agent fully recharges its energy whenever it visits a base node. Whenever the agent is not in a base node, it consumes one unit of energy per time step. Thus, the energy of the agent evolves as:

$$e(t) = \begin{cases} \bar{e}, & \text{if } l(t) \in V_b \\ e(t-1) - 1, & \text{otherwise.} \end{cases} \tag{3}$$

For persistent surveillance, the agent must maintain its energy $e(t)$ above a threshold to perform continuous surveillance without interruption. Thus, we impose the constraint $e(t) \geq 0$, for all $t \in \mathbb{N}_0$.

To each node $i \in V$, we associate a state variable called *demand*, $d_i(t) \in \mathbb{R}$, which is the time since the agent's last visit to node i. Thus the demand, at each node $i \in V$ and for all $t \in \mathbb{N}_0$, evolves as

$$d_i(t) = \begin{cases} 0, & \text{if } l(t) = i \\ d_i(t-1) + 1, & \text{otherwise.} \end{cases} \tag{4}$$

Similarly, the time since last base node visit for the agent, denoted by $g(t)$, evolves as

$$g(t) = \begin{cases} 0, & \text{if } l(t) \in V_b \\ g(t-1) + 1, & \text{otherwise.} \end{cases} \tag{5}$$

We now formally define the fixed horizon optimization problem for surveillance in (6). Without loss of generality, we assume that the initial time for the optimization problem is $t = 0$ and that the agent starts at a node, i.e., $l(0) = b$ for some $b \in V$. $\hat{d}_i \in \mathbb{N}_0$ denotes the demand of node $i \in V$ at $t = 0$ with $\hat{d}_b = 0$, $\hat{g} \in \mathbb{N}_0$ denotes the time since last visit to a base node by the agent at $t = 0$ with $\hat{g} = 0$ if $b \in V_b$, and $\hat{e} \in \mathbb{N}_0$ denotes the initial energy of the agent with $\hat{e} = \bar{e}$ if $b \in V_b$. Since we assume that the node priorities $p_i(t),\ \forall i \in V$ and $\forall t \in \mathbb{N}_0$, change very slowly compared to the planning time horizon T, in Problem (6), we let $p_i = p_i(0)$ for all $i \in V$ to be constants. Notice that Problem (6) is combinatorial in nature.

$$\min_{\{a(m) \in V\}_{m \in \mathbb{N}_0}} \sum_{t=1}^{t=T} \left(\sum_{i \in V} p_i d_i(t) + g(t) \right) \tag{6a}$$
$$\text{s.t. } (1), (2), (3), (4), (5), \forall i \in V, e(t) \geq 0\ \forall t \in [1,T]_{\mathbb{Z}} \tag{6b}$$
$$l(0) = b,\ g(0) = \hat{g},\ e(0) = \hat{e},\ d_i(0) = \hat{d}_i\ \forall i \in V. \tag{6c}$$

Problem statement: Given a graph $G = (V, E, W)$, we aim to develop an optimal persistent surveillance and reporting strategy for G using a rolling horizon framework. This requires repeatedly solving "structurally" similar but varying instances of Problem (6), with different parameters,

$$\mathcal{D} := \{\{p_i, \hat{d}_i\}_{i\in V}, b, \hat{g}, \hat{e}\}.$$

This poses serious computational challenges. Thus, we develop an RL based method to obtain policies that provide near-optimal solutions to instances of Problem (6), given the parameters $\mathcal{D}$ as inputs, for a broad set of possible parameters. Since we want to solve such problem instances repeatedly, we also want the method to be computationally efficient for online plan generation.

III. RL FRAMEWORK

In this section, we propose an RL based framework for learning policies that give solutions to arbitrary instances of Problem (6), for a variety of parameters $\mathcal{D}$. We first introduce the state space ($\mathcal{S}$), action space ($\mathcal{A}$), state transition dynamics, and the reward function for the underlying Markov decision process (MDP). Recall that, it is sufficient to define the state, actions, and rewards only at time steps $(t_m)_{\{m\in\mathbb{N}_0 | t_m \leq T\}}$ when the agent visits a node and chooses the action of which node to visit next.

Consider an arbitrary instance of Problem 6. At a time-step t_m in the planning horizon, we define the state as

$$s(m) := \Big(\Big(p_i, d_i(t_m), x_i(t_m), \big(y_i(j)\big)_{j\in V_b}, W_i\Big)_{i\in V}, \\ g(t_m), e(t_m), T - t_m\Big),$$

where $y_i(j)$ is the travel-time along an arbitrarily predetermined path with the least number of hops from the node $i \in V \setminus V_b$ to the base node $j \in V_b$, $y_j(j) = 0 \ \forall j \in V_b$ and W_i is the i^{th} row of the weighted adjacency matrix of the graph G with the diagonal elements replaced by -1. $x_i(t_m)$ is the travel-time along an arbitrarily pre-determined path with the least number of hops from the node $l(t_m)$ to node $i \in V$. We call the set of all such feasible states $s(m)$, as the state space $\mathcal{S}$.

Given a state $s(m)$, we define the action, $a(m)$, as the agent's next node to visit. Note that due to graph connectivity constraints, $a(m) \in \bar{\mathcal{N}}(l(t_m))$. Also, given the energy constraints, the next node the agent visits should be chosen such that the agent can still visit a base node for a recharge before it depletes all its energy. Hence, the admissible set of actions at state $s(m)$ is the following.

$$\mathcal{A}_{s(m)} := \left\{v \in \bar{\mathcal{N}}\left(l(t_m)\right) | \Omega(v, V_b) \leq e(t_m) - w\left(l(t_m), v\right)\right\}$$

We let the reward associated with the state-action pair $(s(m), a(m))$ at time step t_m in the planning horizon be

$$r(m) := -\sum_{\tau=t_m+1}^{\min\{t_{m+1},T\}} \left(\sum_{i\in V} p_i d_i(\tau) + g(\tau)\right).$$

Notice that the reward $r(m)$ is the negative of the sum of priority-weighted demands at all nodes and the time since last visit to a base node by the agent over all time instants from $t_m + 1$ to $\min\{t_{m+1}, T\}$. Also, notice that the rewards over an episode sum up to negative of the very cost function that we want to minimize in (6a).

A. Duelling double deep Q-network

To learn a policy that gives the agent's actions, we intend to learn a function $Q(s, a)$, which approximates the state-action value function of the state s and action a. For this, we use Double Duelling Deep Q-Learning (D3QN) algorithm, similar to the one used in [22]. In this regard, we use a neural network, denoted henceforth as the Q-Network, for function approximation. We use the standard experience-replay based DQN-like algorithm which maintains policy and target Q-networks for Q-value estimation. Due to space constraints, we do not present the complete pseudo-code of this standard algorithm. Specifically, the D3QN algorithm is a combination of Double DQN [23] and Duelling DQN [24], and it is designed to avoid inaccurate Q-value estimation.

Q-Network architecture and Q-value estimate computation for D3QN: We denote our Q-Network as function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Note that the dimension of the state $s(m)$ grows with the number of nodes in V and it may be fairly large even for not so big graphs. This would require a large neural network with many parameters if $Q(.,.)$ is represented by a fully connected network. To avoid this, we construct our Q-network out of the following four building blocks, a schematic of which is presented in Figure 1.

(i) A bidirectional gated recurrent unit (BGRU), $\mathcal{G}(.;\theta_{\mathcal{G}})$
(ii) A Bahdanau Attention block [25], $\mathcal{B}(.;\theta_{\mathcal{B}})$
(iii) A fully connected state-value block, $\mathcal{X}(.;\theta_{\mathcal{X}})$
(iv) A custom advantage-value kernel, $\mathcal{P}(.,.;\theta_{\mathcal{P}})$.

The BGRU block $\mathcal{G}(.;\theta_{\mathcal{G}})$ has a hidden state dimension of 128. The state-value block $\mathcal{X}(.;\theta_{\mathcal{X}})$ consists of *four* fully connected hidden layers with $256, 512, 256$ and 64 units respectively. The first and third hidden layers have rectified linear (ReLu) activation and the remaining layers have linear activation. The advantage kernel $\mathcal{P}$ is modelled by a neural network described by the equation $\mathcal{P}(x;\theta_{\mathcal{P}}) = Y_3((\mathbf{1} - Y_1 x)\tanh(Y_2 x))$, where, x is the concatenated input and $\theta_{\mathcal{P}} = (Y_i)_{i=1}^{i=3}$ are learnable parameters, $\mathbf{1}$ is the vector of ones with appropriate dimensions.

At $t = t_m$, the input to the BGRU is constructed from $s(m)$, as the sequence of features $\hat{\mathbf{s}}\big(s(m)\big)$ of the nodes.

$$\hat{\mathbf{s}}\big(s(m)\big) = (p_i, d_i(t_m), x_i(t_m), g(t_m), \\ \big(y_i(j)\big)_{j\in V_b}, W_i, T - t_m)_{i\in V}$$

The difference between $s(m)$ and $\hat{\mathbf{s}}\big(s(m)\big)$ is that in the latter we do not include $e(t_m)$ and we include $g(t_m)$ and $T - t_m$ as many times as the number of nodes in V. The BGRU block outputs a sequence $(h_i(s(m)))_{i\in V}$ and the last entry in this sequence, $h_{|V|}(s(m))$. That is, $\mathcal{G}\big(\hat{\mathbf{s}}\big(s(m)\big)\big) = \big((h_i(s(m)))_{i\in V}, h_{|V|}(s(m))\big)$.

$h_{|V|}(s(m))$ is fed as the query and sequence $(h_i(s(m)))_{i\in V}$ is fed as the keys to the attention block $\mathcal{B}(.)$ which outputs a context vector $c(s(m))$. That is, $c(s(m)) = \mathcal{B}\big((h_i(s(m)))_{i\in V}, h_{|V|}(s(m))\big)$. The context $c(s(m))$ is fed to the state-value block, $\mathcal{X}(.;\theta_{\mathcal{X}})$, and its output is treated as the value estimate for state $s(m)$.

Suppose that we are evaluating the value of the action $j \in \mathcal{A}_{s(m)}$, then we pass the context vector $c(s(m))$ and $h_j(s(m))$ (the j^{th} entry in the output sequence of $\mathcal{G}(.)$) through the advantage kernel $\mathcal{P}(.,.;\theta_{\mathcal{P}})$. The output of the

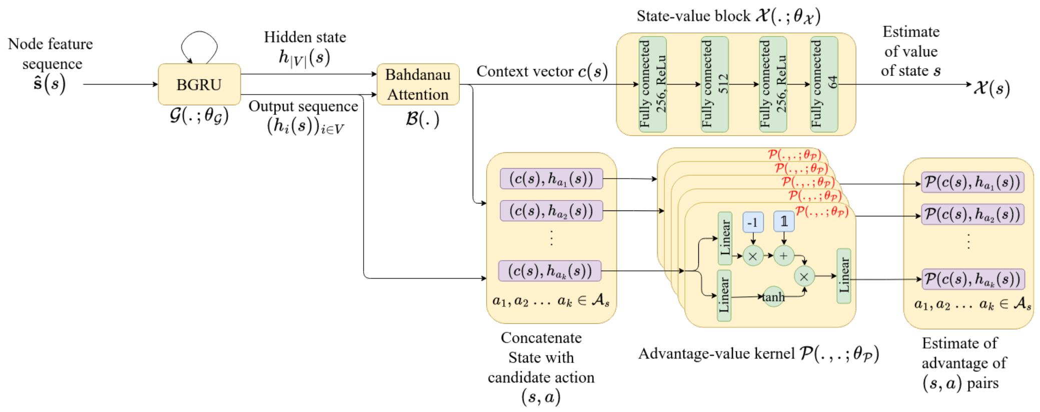

Node feature sequence $\hat{\mathbf{s}}(s)$
BGRU
$\mathcal{G}(.;\theta_{\mathcal{G}})$
Hidden state $h_{|V|}(s)$
Output sequence $(h_i(s))_{i\in V}$
Bahdanau Attention
$\mathcal{B}(.)$
Context vector $c(s)$
State-value block $\mathcal{X}(.;\theta_{\mathcal{X}})$
Fully connected 256, ReLu
Fully connected 512
Fully connected 256, ReLu
Fully connected 64
Estimate of value of state s
$\mathcal{X}(s)$
$(c(s), h_{a_1}(s))$
$(c(s), h_{a_2}(s))$
$(c(s), h_{a_k}(s))$
$a_1, a_2 \dots a_k \in \mathcal{A}_s$
Concatenate State with candidate action (s,a)
$\mathcal{P}(.,.;\theta_{\mathcal{P}})$
Linear
-1
1
tanh
Advantage-value kernel $\mathcal{P}(.,.;\theta_{\mathcal{P}})$
$\mathcal{P}(c(s), h_{a_1}(s))$
$\mathcal{P}(c(s), h_{a_2}(s))$
$\mathcal{P}(c(s), h_{a_k}(s))$
Estimate of advantage of (s,a) pairs

Fig. 1: Schematic of the neural network architecture used for Q value estimation. Notice that the advantage kernel $\mathcal{P}(.,.)$ uses the same parameters $\theta_{\mathcal{P}}$ for all state-action pairs.

kernel $\mathcal{P}(c(s(m)), h_j(s(m)))$ is treated as the advantage value for action j when the environment is in state $s(m)$.

Finally, the state-action value, $Q(s,a)$, for a state s and candidate action $a \in \mathcal{A}_s$ pair is estimated as $Q(s,a) = \mathcal{X}(c(s)) + \mathcal{P}(c(s), h_a(s)) - \frac{1}{|\mathcal{A}_s|}\sum_{a' \in \mathcal{A}_s} \mathcal{P}(c(s), h_{a'}(s))$.

Once trained, the policy that the agent uses is to sample an action from $\mathbb{U}(\operatorname{argmax}_a Q(s,a))$ when the state is s.

IV. Simulations

In this section, we present some simulations, comparisons and their results demonstrating the near-optimality, and computation time benefits of our approach. We also compare our approach with a greedy heuristic, Q learning based Black Box Learning Agent (BBLA [10]), travelling salesman problem based approach (TSP [9]), Entropy Maximized Patrolling (EMP [21]), the CBLS algorithm [12] and a graph attention based learning algorithm (GAT [14]). We first describe the various graphs we use, then describe the training, testing and comparison simulations. Finally, we discuss the simulation results.

Most of the simulations we present were carried out on a set of 15 strongly connected random directed graphs, $\mathbb{G}_1$, with various number of nodes and topologies. These are generated by removing edges randomly from random Watts-Stogartz graphs. $\mathbb{G}_1$ includes 3 graphs each with 10, 15, 20, 25 and 100 nodes. Base nodes for each graph are selected randomly (see Table I) and the travel time for each edge is sampled uniformly from $[1,3]_{\mathbb{Z}}$. For convenience, say $\mathcal{G}_n := \{G : G \in \mathbb{G}_1$ and G has n nodes$\}$.

In the RHP framework, we also compare the D3QN policies against the CBLS algorithm [12] and a graph attention based learning algorithm (GAT) [14] on a grid graph (25 nodes) and an irregular graph (34 nodes) shown in Figures 3(b) (grid) and 3(c) (irregular) in [14]. Let $\mathbb{G}_2$ be the set of these grid and irregular graphs.

A. Training, testing and comparisons

1) D3QN Training: We separately train 5 D3QN policies for each graph. Training process involves running a PyTorch implementation of D3QN algorithm. At the start of training a policy for a given graph, we sample the parameters $\mathcal{D}$ of Problem (6) (more details in specific simulations given below). At each time step, given the state, we use an $\epsilon-$greedy exploration policy to get the action. The action is taken and the next state and reward are observed. The (state, action, reward, next state) tuple is stored in the replay buffer, replacing the oldest tuple if the replay memory is full. At each learning iteration, we sample 16 tuples from the replay buffer and update the weights of the policy and target Q-networks according to the standard D3QN algorithm. At the end of the horizon of an instance, the parameters of Problem (6) are sampled again. The exploration probability, ϵ, decays exponentially from 0.5 to 10^{-4} along learning iterations. The learning rate starts at a value of 0.0005 and is scheduled to decrease by a factor of 0.75 after every 2×10^4 learning iterations. We set the replay buffer size to 5×10^4. We run one learning iteration per time step.

2) Sim-1 – D3QN evaluation on arbitrary instances of Problem (6): Here, we use the graphs in $\mathbb{G}_1$. For graphs in $\mathbb{G}_1 \setminus \mathcal{G}_{100}$ and $\mathcal{G}_{100}$, we set the horizon length to be $T = 15$ and $T = 30$, respectively. We empirically compare the computation times and relative percentage difference of cost between D3QN and SCIP. We also compare the performance of D3QN against the greedy heuristic, wherein the agent always moves to an admissible neighbor node that would have the highest product of priority and demand at the next time step if the agent were to take no action.

An instance is generated by randomly selecting *high priority nodes* (HPNs), *low priority nodes* (LPNs), their priorities, initial demands (see Table I), and the initial agent location. Priorities of HPNs and LPNs are respectively sampled from $\mathbb{U}([5,7]_{\mathbb{Z}})$ and $\mathbb{U}([1,2]_{\mathbb{Z}})$. Initial demands of HPNs and LPNs are respectively sampled from $\mathbb{U}([10,20)_{\mathbb{Z}})$ and $\mathbb{U}([1,4]_{\mathbb{Z}})$. The initial agent position is sampled uniformly from the set of nodes V. The initial energy and time since last base node visit are sampled from $\mathbb{U}([e_m, \bar{e}]_{\mathbb{Z}})$ and $\mathbb{U}([e_m, 3T]_{\mathbb{Z}})$ respectively, where, e_m is the least energy required to travel

No. of nodes	No. of HPNs	No. of LPNs	Average No. of neighbours	No. of base nodes
10	3	7	5	1
15	3	12	7	2
20	3	17	5	3
25	5	20	7	3
100	25	75	4	10

TABLE I: Number of high priority nodes (HPNs), low priority nodes (LPNs), base nodes and average number of neighbors in random graphs with varied number of nodes.

from the initial position to a base node and $\bar{e}$ is the maximum energy capacity of the agent. We train the 5 D3QN polices for each graph as mentioned in Section IV-A.1. We run 2.25×10^5 learning iterations (equivalently 1.5×10^4 instances) for graphs in $\mathbb{G}_1 \setminus \mathcal{G}_{100}$ and 5.5×10^5 learning iterations (equivalently 3×10^4 instances) for graphs in $\mathcal{G}_{100}$.

For comparisons, we generate and use the same 50 test instances of each graph in $\mathbb{G}_1$. We run greedy heuristic and corresponding trained RL policies on the same test instances. We also solve an integer program (IP) corresponding to each of these test instances using the SCIP solver in Python supported by PySCIPOpt. Due to computational limitations, we limit the solution time of the solver to 10800s and we use the best solution obtained by then. Let $J_{IP}(i)$, $J_{RL}(i)$, $J_{GP}(i)$, $J_{BBLA}(i)$ and $J_{TSP}(i)$ represent the cost obtained by solving the test instance i using SCIP, D3QN, greedy heuristic, BBLA and TSP-heuristic respectively. We compute $\beta_H(i) := \frac{J_H - J_{IP}}{J_{IP}}\%$ for policy H (eg. RL, IP, TSP etc.), and use these values to judge the performance of a policy. We record 150 values each for SCIP, BBLA, TSP and greedy heuristic (3 graphs with 50 values each) and 750 values for D3QN (3 graphs with 50 values each on 5 trained policies), over all graphs with same number of nodes.

3) Sim-2 – Rolling horizon comparisons (D3QN vs SCIP): Here, we only use the graphs in $\mathcal{G}_{25}$. We generate 5 RHP test instances for each graph, as explained in Sim-1, but with demands of all nodes set to 0. For each RHP test instance, the priorities change randomly. The sequence of time steps on which the priorities change is given by $(z_i)_{i\in\mathbb{N}_0}$ where, $z_{i+1} = z_i + \kappa_i$, $z_0 = 0$ and $\kappa_i \sim \mathbb{U}([1, 10]_{\mathbb{Z}})$. At these time steps, only the HPNs and LPNs are sampled and the priorities are assigned randomly like a new instance in Sim-1.

Here, we re-plan for the next planning horizon after implementing the first decision of the previous horizon. We set planning time horizon, $T = 15$ and the simulation time to 150 time steps. We run the 5 trained D3QN polices (trained on corresponding graphs from Sim-1) and SCIP on each of these RHP test instances. For each instance i, we compute the cost (6a) with D3QN ($\hat{J}_{RL}(i, \tau)$) and with SCIP ($\hat{J}_{IP}(i, \tau)$) with changing priorities over various values of RHP horizon $\tau \leq 150$. Say, $G_{RHP}(i, \tau) = \frac{\hat{J}_{RL}(i,tau) - \hat{J}_{IP}(i,tau)}{\hat{J}_{IP}(i,tau)}\%$ for an RHP test instance i for some RHP horizon length τ. We use $G_{RHP}(i, \tau)$ values to compare the performance of a policy in the RHP framework. Here for each value of τ, we will have 25 values of $G_{RHP}(., \tau)$.

4) Sim-3 – Comparison with EMP [21], CBLS [12] and GAT [14]: Here, we only use the graphs in $\mathbb{G}_2$. A test instance refers to a graph with all node demands initialized to 0, node priorities set to 1 and the agent initiated randomly. There are no base nodes, no energy constraints and the priorities do not change. Hence we generate 25 and 34 test instances for grid and irregular graphs, respectively (each instance with the agent in a distinct node). We use the RHP framework with re-planning after implementing the first decision with the planning horizon, $T = 15$ time steps and the simulation time as 150 time steps.

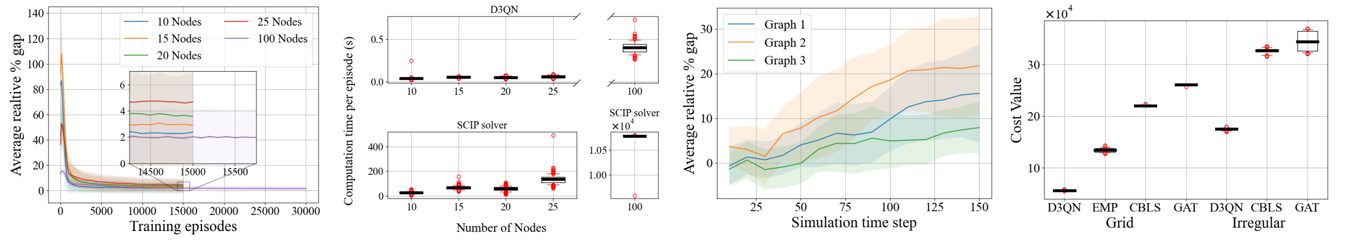
For D3QN, we freshly train 5 policies on the grid and irregular graphs separately, as given in Section IV-A.1. While training, in each instance, the demands are sampled randomly from $\mathbb{U}[1, 4]_{\mathbb{Z}}$ for all nodes with a horizon length of $T = 15$ time steps. We train each policy for 2.25×10^5 learning iterations (equivalent to 1.5×10^4 instances).

For CBLS and GAT we use the code available in the link https://github.com/glx15534565855/MARL_to_solve_patrol and for EMP, we use the code available at https://github.com/kevinkang1125/EM-Patroller. CBLS uses the policy in [12]. For GAT and EMP, we set the number of agents to 1 and use the policy trained with default settings in the code. We compare the cost values, as in Sim-2, with RHP over 150 time steps on 25 and 34 instances of each of the grid and irregular graphs respectively. We run 10 simulations for each instance with EMP's stochastic policy. Also, we run EMP simulations only on grid graph, since it needs the travel times between all neighbors to be equal. For each problem instance, we consider average objective value over 5 policies for D3QN and 10 simulations for EMP. For grid graph, this gives 25 values each for D3QN, EMP, CBLS and GAT. For irregular graph, we have 34 values each for D3QN, CBLS and GAT.

B. Results

Now, we present our simulation results.

1) Sim-1: In Figure 2a, we present the evolution of μ_β (average relative percentage gap between D3QN and SCIP) with training iterations for various graphs. We observe that D3QN's performance improves with the training iterations. Table II summarizes the average and standard deviation values at the end of training, showing the method's robustness to network initialization. The table also presents the same for BBLA and TSP-heuristic. As expected, BBLA and TSP perform poorly (comparable to or worse than greedy policy) due to missing information on the node priorities and time since last visit to base node. The proposed method also outperforms the greedy heuristic. Note the lower average relative percentage difference 100 node graphs compared to



(a) D3QN training curves (b) Computation time comparison (c) Variation in μ_{RHP} with horizon (d) Comparison with CBLS and GAT

Fig. 2: Figures 2a and 2b present results for Sim-1, Figure 2c presents results for Sim-2 and 2d presents results for Sim-3. For Figures 2a and 2c, the thick line indicates the average, while the shaded region denotes one standard deviation from the average. For Figures 2b and 2d, black line represents the average, box represents the range of values between first and third quartile, whiskers represent range of values between 90^{th} and 10^{th} percentile. Red circles are outliers.

Nodes	D3QN (%)		Greedy (%)		BBLA (%)		TSP (%)	
	μ_β	σ_β	μ_β	σ_β	μ_β	σ_β	μ_β	σ_β
10	2.4	4.8	26.8	20.8	29.9	22	58.2	4.3
15	2.9	3.7	30.7	18.2	37.5	26.9	37	8.2
20	3.6	4.8	18.6	12.6	14.7	14.3	55.4	5.2
25	4.7	4.8	19.3	10.8	23.2	16.8	58.9	5.5
100	1.7	2.1	6.8	3.4	6.1	2.6	89.3	2

TABLE II: Results from Sim-1. μ_β and σ_β are average and standard deviation in relative %-gap between fully trained policy with D3QN, greedy policy, BBLA and TSP vs. SCIP.

others is due to a combination of computation time limit on the SCIP solver and larger SCIP objective values for 100 node graphs. Figure 2b compares computation time per episode for SCIP solver and D3QN on a i7-8700 processor machine with 40GB of RAM. D3QN provides up to 4 orders of magnitude benefit in computation time over SCIP. The average and variance in computation times per episode for the SCIP solver increases with the number of nodes in the graph. Also, due to increase in the number of network parameters, we observe an increase in D3QN computation time for 100 node graphs. Note that the training time per policy is roughly 6.5 hours for 1.5×10^4 learning iterations on the same machine.

2) Sim-2: Figure 2c presents the effect of the fixed horizon planning on the long term cost. As the simulation time increases, the performance of the D3QN policies degrades, though the rate of degradation reduces with the simulation time. We observe that the maximum (over D3QN policies) optimality gaps for the three graphs (from Sim-1) are $4.5\%, 6.9\%$ and 4.6%. However, the maximum rate of increase of average gap in RHP framework are $0.29\%, 0.51\%$ and 0.32% in order. This suggests that the rate of accumulation of errors over the RHP simulation time is less compared to errors in individual planning horizons. The average relative percentage gap after 150 time steps for the 3 graphs are 15.57, 21.75 and 7.91 respectively.

3) Sim-3: Figure 2d compares the cost for RHP test instances of Sim-3 with D3QN, CBLS and GAT on the grid and irregular graphs. We observe that D3QN performs significantly better compared to CBLS and GAT.

V. Conclusions and Future Work

In this work, we proposed a rolling horizon based RL framework for single-agent persistent surveillance and reporting with energy constraints. We compared the same against SCIP solver and other state of the art surveillance algorithms called BBLA, TSP-heuristic, EMP, CBLS and GAT. We observed that the method performs comparably to the SCIP solver and provides several orders of magnitude reduction in the computation time. The proposed also provides a significant improvement over CBLS and GAT algorithms. Future directions include improving the long term performance in rolling horizon framework, and extension to multi-agent surveillance.

References

[1] L. Huang, M. Zhou, K. Hao, and E. Hou, "A survey of multi-robot regular and adversarial patrolling," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 894–903, 2019.

[2] P. H. Washington and M. Schwager, "Reduced state value iteration for multi-drone persistent surveillance with charging constraints," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6390–6397.

[3] S. Alamdari, E. Fata, and S. L. Smith, "Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 138–154, 2014.

[4] N. Rezazadeh and S. S. Kia, "A sub-modular receding horizon solution for mobile multi-agent persistent monitoring," *Automatica*, vol. 127, p. 109460, 2021.

[5] S. C. Pinto, S. B. Andersson, J. M. Hendrickx, and C. G. Cassandras, "Multiagent persistent monitoring of targets with uncertain states," *IEEE Transactions on Automatic Control*, vol. 67, no. 8, pp. 3997–4012, 2022.

[6] A. B. Asghar, S. L. Smith, and S. Sundaram, "Multi-robot routing for persistent monitoring with latency constraints," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 2620–2625.

[7] 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, 2020.

[8] M. Kido, K. Kobayashi, and Y. Yamashita, "MPC-based surveillance over graphs by multiple agents," *SICE Journal of Control, Measurement, and System Integration*, vol. 10, no. 3, pp. 253–258, 2017.

[9] C. D. Alvarenga, N. Basilico, and S. Carpin, "Combining coordination and independent coverage in multirobot graph patrolling," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 4413–4419.

[10] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch, "Multi-agent patrolling with reinforcement learning," in *Autonomous Agents and Multiagent Systems, International Joint Conference on*, vol. 4. IEEE Computer Society, 2004, pp. 1122–1129.

[11] D. Portugal, M. S. Couceiro, and R. P. Rocha, "Applying bayesian learning to multi-robot patrol," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2013, pp. 1–6.

[12] D. Portugal and R. P. Rocha, "Cooperative multi-robot patrol with bayesian learning," *Autonomous Robots*, vol. 40, no. 5, pp. 929–953, 2016.

[13] M. Othmani-Guibourg, A. El Fallah-Seghrouchni, and J.-L. Farges, "Lstm path-maker: a new lstm-based strategy for multiagent patrolling," in *Conférence Nationale en Intelligence Artificielle*, 2019.

[14] L. Guo, H. Pan, X. Duan, and J. He, "Balancing efficiency and unpredictability in multi-robot patrolling: A marl-based approach," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3504–3509.

[15] W. J. Yun, S. Park, J. Kim, M. Shin, S. Jung, D. A. Mohaisen, and J.-H. Kim, "Cooperative multiagent deep reinforcement learning for reliable surveillance via autonomous multi-UAV control," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 10, pp. 7086–7096, 2022.

[16] J. Banfi, N. Basilico, and F. Amigoni, "Minimizing communication latency in multirobot situation-aware patrolling," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 616–622.

[17] J. Scherer and B. Rinner, "Persistent multi-UAV surveillance with energy and communication constraints," in *2016 IEEE international conference on automation science and engineering (CASE)*. IEEE, 2016, pp. 1225–1230.

[18] B. Z. H. Rozaliya, I.-L. Wang, and A. Muklason, "Multi-UAV routing for maximum surveillance data collection with idleness and latency constraints," *Procedia Computer Science*, vol. 197, pp. 264–272, 2022.

[19] J. Scherer and B. Rinner, "Persistent multi-UAV surveillance with data latency constraints," *arXiv preprint arXiv:1907.01205*, 2019.

[20] K. Kobayashi, S. Ueno, and T. Higuchi, "Multi-robot patrol algorithm with distributed coordination and consciousness of the base station's situation awareness," *arXiv preprint arXiv:2307.08966*, 2023.

[21] H. Guo, Q. Kang, W.-Y. Yau, M. H. Ang, and D. Rus, "Empatroller: Entropy maximized multi-robot patrolling with steady state distribution approximation," *IEEE Robotics and Automation Letters*, 2023.

[22] Y. Huang, G. Wei, and Y. Wang, "VD D3QN: the variant of double deep Q-learning network with dueling architecture," in *2018 37th Chinese Control Conference (CCC)*. IEEE, 2018, pp. 9130–9135.

[23] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[24] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[25] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.