Reinforcement Learning based Persistent Surveillance by Robots

Aravind S
Advisor: Dr. Pavankumar Tallapragada

***Abstract*— In this project, we consider a rolling horizon optimal routing problem on a graph for persistent surveillance by energy constrained single and multiple agents, which need to occasionally visit base nodes to upload surveillance data. We aim to minimize the weighted (by node priorities) graph idleness and time between visits to the base node. This requires repeatedly solving similar combinatorial opitmization problems, which is computationally challenging. To address this important challenge, we propose a learning based solution. In particular, we propose a reinforcement learning (RL) based policy that given the priorities and the state of the graph decides the next node to visit for the agent(s). We use a sequential decision making approach to perform multi agent planning and guarantee satisfaction of energy constraints by projecting the actions to a safe set. We train the RL policy using deep Q-network (DQN). Through exhaustive simulations, we establish near-optimality of the learnt policies by comparing them against an integer programming (IP) solver. Compared to the IP solver, our method provides several orders of magnitude reduction in computation time.**

I. Introduction

Effective surveillance systems with optimal resource deployment are critical in the areas of public safety, border security, search and rescue operations, and environmental and wildlife monitoring in remote locations. In this project, we model the region to be surveyed as a connected graph. The nodes represent the places to be surveyed, and the edges represent the direct paths between these nodes. We pose the surveillance problem as a route optimization problem. Solving such routing problems with generic optimization solvers is often computationally very demanding. It is not scalable in terms of the size of the graph, the number of agents or repeated re-planning, which is required for persistent surveillance. In this project, we propose a reinforcement learning (RL) based approach to obtain policies that give near-optimal solutions with very little computation time for a variety of parameters in the surveillance optimization problem.

A. Related Work

One of the earliest frameworks for the surveillance problem was provided through the art gallery problem in [1], which was posed as a centralized static optimization problem. [2] gives an overview of designing fixed cyclic tours on which multiple robots traverse to carry out the patrolling task. [3], [4] provide different algorithms for designing cyclic tours with 'refresh time' and latency as the performance criteria. Another popular approach to the surveillance problem is that of an optimization-based path planning, where the region of interest is modeled as a graph, and the aim is to determine the optimal routes of one or more robots under some constraints. In [5], each node in the graph has an internal state that evolves with stochastic linear dynamics, The goal is to minimize the mean squared estimation error of these nodes, which is solved as a centralized problem. [6] assigns a concave and increasing reward function of time for each node that resets to zero after an agent's visit and proposes a centralized algorithm to get a sub-optimal solution. [7]–[9] also solve optimization problems in a centralized manner for single or multi-robot surveillance. [10] considers multi-robot path planning on a weighted graph, with constraints on the maximum time spent by robots between visits to the nodes, called the latency. The objective is to find the minimum number of robots that can satisfy the latency constraints. Another important area relevant to surveillance problems is that of vehicle routing problems (VRP) [11]. [12] considers the m-vehicle dynamic traveling salesman problem (TSP). A special class of VRP, particularly suited to surveillance, is Orienteering problems (OP) [13]. There is a body of work with a learning-based approach to various routing problems. [14] proposes a methodology for solving generic combinatorial problems using RL. [15] proposes an RL approach to solving a multi-agent TSP problem, where an RL policy is used to assign nodes to different agents, and then a single-agent TSP problem is solved for each agent. Works like [16], [17] propose RL frameworks to solve different variants of VRP. [18] proposes an RL-based patrolling problem with the aim of minimizing average inter-visit times to different nodes. [19], [20] adopt a Bayesian Learning approach for multi-robot patrolling with the aim of minimizing average graph idleness. [21] proposes an LSTM-based multi-agent patrolling algorithm. [22] presents an RL-based solution for a multi-agent patrolling problem using graph attention Networks to find a stochastic policy while minimizing the inter-visit times of the nodes by the agents. Another important need in surveillance is to report information collected from nodes to some base station node while also obeying battery constraints. [23] and [24] give an overview of this literature. [25] proposes an integer programming solution for a problem consisting of cooperating UAVs that collect and transfer sensor data in a store-and-forward manner while optimizing data transmission to the base station within latency and idleness constraints. [26] addresses limited battery and communication constraints, by proposing an offline path planning algorithm for multiple UAVs while ensuring connectivity to one or more base stations for battery replacement and data transfer. Two planning algorithms with varying horizons and cooperation are proposed and compared in [27], where short horizon uncooperative strategy excels with sufficient UAVs, while the full horizon strategy ensures coverage if specific conditions are met. [28] minimizes idleness and latency by employing minimum-latency paths

(MLP) to ensure data transfer to the base station within a pre-defined bound. [29] formulate a Mixed Integer Linear Program (MILP) to minimize the maximum weighted time between consecutive visits to nodes within a given cycle length. They also propose sub-optimal iterative schemes that enforce pre-specified node visit counts, significantly speeding up computation.

II. Preliminaries

We denote the set of non-negative real numbers and the set of non-negative integers by $\mathbb{R}_{\geq 0}$ and $\mathbb{N}_0$. A (directed) weighted 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 $W = \{w(i,j) \in \mathbb{N}_0 \mid \{i,j\} \in E\}$ is a set of weights associated with the edges. A directed edge (i,j) represents a path from node $i \in V$ to node $j \in V$. A path in the graph $G = (V, E, W)$ is an ordered set of nodes $(u_1, u_2, \ldots, u_{n+1})$, i.e. $u_i \in V$ for each $i \in \{1, 2, \ldots, n\}$, such that $(u_i, u_{i+1}) \in E$, for each $i \in \{1, 2, \ldots, n\}$. A directed graph $G = (V, E)$ is strongly connected if a path exists between every pair of nodes $i, j \in V$. In a graph $G = (V, E, W)$, we denote the set of neighbors of node i as $\mathcal{N}(i) := \{j \mid (i,j) \in E\} \cup \{i\}$. Note that we have included $\{i\}$ in the set of neighbors of i.

III. Problem Setup

We consider the problem of persistent surveillance and reporting on a strongly connected directed graph $G = (V, E, W)$ by a set of agents, denoted by A, 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 agents have to occasionally report the surveillance data. Edge $(i,j) \in E$ models a directed route between two nodes $i, j \in V$ 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 $i \in V$. An example of such a graph with different survey and base nodes is shown in Fig 1.

We model the surveillance problem as one of routing the set of agents A on the graph. At some time $t \in \mathbb{N}_0$, an agent $k \in A$ decides the next node to visit if and only if it is in a node at time t (and not on an edge between two nodes). This decision is termed as agent k's action at time t, denoted by $a_k(t)$. Hence, we keep track of $\{t_m\}_{m \in \mathbb{N}_0}$, the sequence of time steps when some agent is at a node. In particular, t_m is the m^{th} time step at which some agent is at a node in V.

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, we let the priority of a base node to be *zero* for all time, i.e., $p_j(t) = 0 \ \forall j \in V_b, \ \forall t \in \mathbb{N}_0$. Priorities

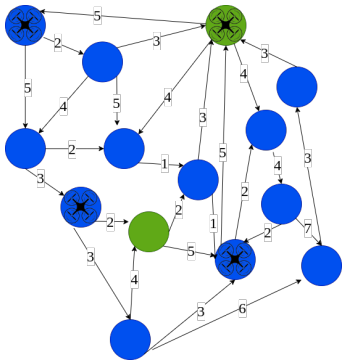


Fig. 1: An example of the type of graph considered in this work. Blue and green circles indicate survey and base nodes respectively. Lines and numbers between nodes are edges between corresponding nodes and the travel times along those edges, respectively.

are user-defined quantities and are useful for incorporating exogenous information into the surveillance problem. 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. The priorities $p_i(t)$'s could be periodic in time to model periodic patterns in the need for surveillance, or could change abruptly when new (external) information arises, in the middle of an ongoing surveillance task. In this work, we do not assume any particular model or pattern in how the priorities change, except that they change somewhat slowly and not every time-step. We also do not assume any foreknowledge about the priorities. We assume that at time t, only the priorities up to and including time t is known.

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. We solve a fixed horizon optimal control problem with planning horizon length $T > 0$ at each time step in the sequence $\{t_m\}_{m \in \mathbb{N}_0}$.

Notice that, due to the rolling horizon framework, there are two time scales involved. One is the actual time scale (where the node priorities can change) where the agent executes its actions traversing on the graph, and the other is the planning time scale (where the node priorities are assumed constant) in each instance of the rolling horizon problem. Note that not all planned actions are executed in the planning time scale. But, for all practical purposes, we are almost always concerned with the planning time scale. Hence, we allow for a slight abuse of notation. In all further discussions, unless mentioned otherwise, we use the notations t and t_m to denote time and m^{th} time step where an agent is in a node, respectively, in the planning time scale. Notation-wise, we do not make a distinction between planning and execution time scales. Note that, for each planning horizon problem, we assume that the priorities of all the nodes are constant.

We denote the *location* of agent $k \in A$ at a time step $t \in \{0, 1, \ldots, T\}$ in the planning horizon by $l_k(t)$. Thus,

$$l_k(t) = i, \ \text{if agent is at node } i \in V \text{ at time } t \tag{1a}$$

$$l_k(t) \notin V, \ \text{otherwise.} \tag{1b}$$

In the plan, at t_m, the agent $k \in A$ takes the action $a_k(t_m) \in \mathcal{N}(l_k(t_m))$, which is the set of neighbors of the node $l_k(t_m)$ and the agent will to reach the node $a(k)$ at time $t_m + w(l(t_m), a_k(t_m))$. That is,

$$a_k(t_m) \in \mathcal{N}(l_k(t_m)), \quad \forall m \in \{j \in \mathbb{N}_0 | t_j < T \wedge l_k(t_m) \in V\}\ \forall k \in A \tag{2a}$$

$$l_k(t_m + w(l(t_m))) = a_k(t_m), \quad \forall m \in \{j \in \mathbb{N}_0 | t_j < T \wedge l_k(t_m) \in V\}\ \forall k \in A \tag{2b}$$

$$t_{m+1} = \min\{t \in \mathbb{N}_0 | t > t_m \wedge \exists k : l_k(t) \in V\}, \quad \forall m \in \{j \in \mathbb{N}_0 | t_j < T\}. \tag{2c}$$

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

$$d_i(t+1) = \begin{cases} 0, & \text{if } \exists k : l_k(t+1) = i \\ d_i(t) + 1, & \text{otherwise.} \end{cases} \tag{3}$$

Notice that the demand at a node increments by 1 at each time step, except for time steps at which the agent visits the node, at which time the demand for that node resets to 0.

Similarly, the time since last base node visit for agent $k \in A$, denoted by $g_k(t)$, evolves as

$$g_k(t+1) = \begin{cases} 0, & \text{if } l_k(t+1) \in V_b \\ g_k(t) + 1, & \text{otherwise.} \end{cases} \tag{4}$$

Notice that, for agent $k \in A$, time since last base node visit increments by 1 at each time step, except if it visits some base node $j \in V_b$, where it is reset to 0.

Each agent, $k \in A$ is assumed to have a finite amount of energy to model the battery capacity of the robots. The battery can be recharged or exchanged by reaching one of the base nodes $V_b \subset V$. Upon reaching the base nodes, the battery of each agent is recharged to the maximum value, which could be different for each agent. The energy of each agent $k \in A$ evolves as:

$$e_k(t+1) = \begin{cases} \bar{e}_k, & \text{if } l_k(t+1) \in V_b \\ e_k(t) - 1, & \text{otherwise} \end{cases} \tag{5}$$

Notice that agent consumes 1 unit of energy per time step irrespective of whether it moves along an edge or chooses to remain at a node. $\bar{e}_k > 0$ is the maximum energy for agent k. In order to achieve persistence in surveillance, the energy of an agent, $e_k(t)$ must not drop to zero as that would indicate that the agent is lost and cannot continue the surveillance task. Thus an additional constraint $e_k(t) > 0$ is imposed.

Formally, without loss of generality, we now define a fixed horizon problem instance being solved at time step $t = 0$. Notice that at time $t = 0$, agents may be on a node or traveling on an edge towards a node as given in Problem 6, where $b_k \in V$ denotes the node occupied by the agent $k \in A$ at $t = 0$, $\hat{d}_i \in \mathbb{N}_0$ denotes the demand of node $i \in V$ at $t = 0$ with $\hat{d}_{b_k} = 0$, $\hat{g}_k \in \mathbb{N}_0$ denotes the time since last visit to base node of agent $k \in A$ at $t = 0$ with $\hat{g}_{b_k} = 0$ if $b_k \in V_b$, and $\hat{e_k} \in \mathbb{N}_0$ denotes the initial batteries of the agent $k \in A$ with $e_{b_k} = \bar{e}_k$ if $b_k \in V_b$.

$$\min_{a(k) \in V\ (k \in \mathbb{N}_0)} \sum_{\tau=1}^{\tau=T} \sum_{i \in V} p_i(t_n) d_i(\tau) + \sum_{\tau=1}^{\tau=T} \sum_{k \in A} g_k(\tau) \tag{6a}$$

$$\text{s.t. } (1), (3), (4), (5)\ \forall \tau \in \{1, \ldots, T-1\},\ \forall i \in V \tag{6b}$$

$$(2),\ \forall \tau_k \in \{q \in \mathbb{N}_0 \mid (0 \le q < T) \wedge (l(q) \in V)\} \tag{6c}$$

$$(5), e_k(t) > 0\ \forall k \in A\ \forall t \in \{1, 2, \ldots, T\} \tag{6d}$$

$$l(0) = b, d_i(0) = \hat{d}_i\ \forall i \in V \tag{6e}$$

$$g_k(0) = \hat{g}_k, e(0) = \hat{e_k}\ \forall k \in A \tag{6f}$$

Notice that Problem 6 is combinatorial in nature. For persistent surveillance, we need to solve "structurally" similar problems repeatedly, which poses computational challenges.

For ease of notation, we choose to omit the time argument in node priority $p_i(.)$ wherever the node priorities do not vary for the time horizon considered in the context.

A. Problem statement

In this work, we consider a graph $G = (V, E, W)$ with a set of heterogeneous agents A, and aim to develop an optimal persistent surveillance strategy for G using a rolling horizon framework. We assume slowly time-varying node priorities $p_i(t),\ i \in V,\ t \in \mathbb{N}_0$ in the surveillance task. This means solving "structurally" similar but varying instances of Problem (6) repeatedly, considering the initial positions of the agents, node priorities, initial node demands and initial battery of the agents as the parameters.

In this regard, we want to develop a reinforcement learning based solution method to obtain policies that provide a (near-) optimal solution to instances of Problem (6), 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 inference.

IV. RL Framework

In this section, we propose a reinforcement learning (RL) based solution to Problem (6). We first introduce the state space ($\mathcal{S}$), action space ($\mathcal{A}$), deterministic state transition dynamics, and the reward function for the underlying Markov Decision Process (MDP).

Note that we only solve arbitrary instances of Problem 6 using the RL framework. Hence, we restrict the discussion in this section to the planning horizon and planning time scale. Recall that in the planning horizon of length T, agents take action only on time steps $\{t_m\}_{\{m \in \{\mathbb{N}_0 | t_m < T\}}$ at which an agent plans to be in a node in V. So, it is sufficient to define the state, actions, and rewards at these time steps, for which an agent visits a node and chooses the action of which next node to visit.

At any given time step, each agent $k \in A$ can be in one of two states: active or inactive, represented by a binary variable $q_k(t)$. $q_k(t) = 1$ if agent k is active, else $q_k(t) = 0$. When an agent is active, it can make decisions and choose an action. Thus, $q_k(t) = 1$ is the agent k is on a node,

and $q_k(t) = 0$ if agent k is traveling on an edge or if it has already chosen an action. Additionally each agent $k \in A$ is associated with an *intent* node at each time-step, *intent*(k,t). The intent node is the node currently occupied by the agent or the node the agent is moving to when on an edge. Corresponding to the intent node, each agent is also associated with $t_{intent}(k,t)$ which is the time left to reach the intent node. If *intent*$(k,t) = l_k(t)$, then $t_{intent}(k,t) = 0$.

Consider an arbitrary instance of Problem 6. At a time-step t_m in the planning horizon, we define the features of a node, $i \in V$ as $f_{\text{node}}(i,m) := \big(p_i, d_i(t_m), \big(y_i(t_m,j)\big)_{j\in V_b}, W_i\big)$, where $y_i(t_m)$ is the travel-time along a randomly chosen path with the least number of hops from the node $i \in V$ to the base node $j \in V_b$ and W_i is the i^{th} row of the weighted adjacency matrix of the graph G with the diagonal elements being replaced with -1.

We define the features of an agent, $k \in A$ as $f_{\text{agent}}(k,m) := \big(q_k(t_m), g_k(t_m), f_{\text{node}}(\textit{intent}(k,t_m)), t_{\textit{intent}}(k,t), \big(x_i(t_m,z) + t_{\textit{intent}}(z,t)\big)_{z\in A}, e_k(t_m)\big)$, where $f_{\text{node}}(\textit{intent}(k,t))$ is the features of the intent node for agent k, $t_{\textit{intent}}(k,t)$ is the time left for the agent to reach the intent node, $x_i(t_m,z)$ is the travel-time along a randomly chosen path with the least number of hops from the node $i \in V$ to the intent node of agent z, *intent*(z,t) and $e_k(t_m)$ is the energy of the agent k at time t_m.

We define the state as: $s(m) := \big(\big(f_{\text{node}}(i,m)\big)_{i\in V}, \big(f_{\text{agent}}(k,m)\big)_{k\in A}\big)$. We call the set of all such feasible states $s(m)$, as the state space $\mathcal{S}$.

At a time-step t in the planning horizon, each agent can only move to one of its neighbors. Thus, given the state $s(m)$, We define the action space as the set $\mathcal{A}_{s(m)} := \bigtimes_{\{i|l_k(t)=i\ \forall k\in A\}} \mathcal{N}(i)$ which is the cartesian product of the set containing neighbors of all the agents that are at some node at time-step t_m in the planning horizon.

The reward associated with the state-action pair $(s(m), a(m))$ at time step t_m in the planning horizon, is defined as

$$r(m) = -\sum_{i\in V}\sum_{z=t_m+1}^{h(m)} p_i d_i(z) - \sum_{k\in A}\sum_{z=t_m+1}^{h(m)} g_k(z),$$

where $h(m)$ is the minimum of next time step t_m in the planning horizon at which an agent is in a node and the end of the time horizon T of the optimization problem (6).

Notice that the reward $r(m)$ is the negative of the sum of priority-weighted demands at all nodes and the sum of the time since last visit to a base node for all agents and over all time instants from $t_m + 1$ to $h(m)$.

We aim to learn a policy to solve the Problem (6) given the initial locations of the agents, the initial demands, the priorities, the initial values of the time since last visit to a base node and the initial battery values of the agents. We first developed an RL framework to solve a single agent version of the Problem (6) and then later extended the same to a multi-agent scenario.

We experimented with two reinforcement learning algorithms to learn a policy to solve problem (6). One of them was Double Dueling Deep Q-Learning (D3QN) similar to the one used in [30] and the other was Proximal Policy Optimization (PPO) [31]. The D3QN algorithm is a combination of Double DQN [32] and Dueling DQN [33], that avoids the inaccurate Q-value estimation problem of the Double DQN algorithm and the overestimation problem of the Dueling DQN algorithm. The PPO algorithm is a state of the art policy gradient algorithm that aims to optimize policies iteratively. It employs a clipped surrogate objective function to ensure stable policy updates and prevent large policy changes. The results regarding PPO have been omitted in this report due to space constraints.

A. Handling the energy constraints

The RL algorithms described above are unconstrained algorithms, that is, they cannot *guarantee* satisfaction of hard constraints. These algorithms can only provide a probabilistic guarantee by penalizing constraint violations in the reward function. The energy constraints for the agents are hard constraints and should never be violated. In order to guarantee satisfaction of these constraints, while using unconstrained RL algorithms like DQN or PPO, we leverage the structure of the action space in the RL framework. At time-step t_m in the planning horizon (where at least one agent is occupying a node), each agent can only move to a node that is a neighbor of its current location, that is $a_k(t) \in \mathcal{N}(l_k(t_m))$. In order to incorporate the energy constraints, we project the action space of each agent at that time step to a safe set. The safe set is the set of neighboring nodes that the agent can go to with its current energy while staying within range of a base node, from where the agent can charge its battery. Thus, the safe set, $\mathcal{N}_{\text{safe}}(l_k(t_m))$ is defined as the set of neighboring nodes, $j \in \mathcal{N}(l_k(t_m))$ such that:

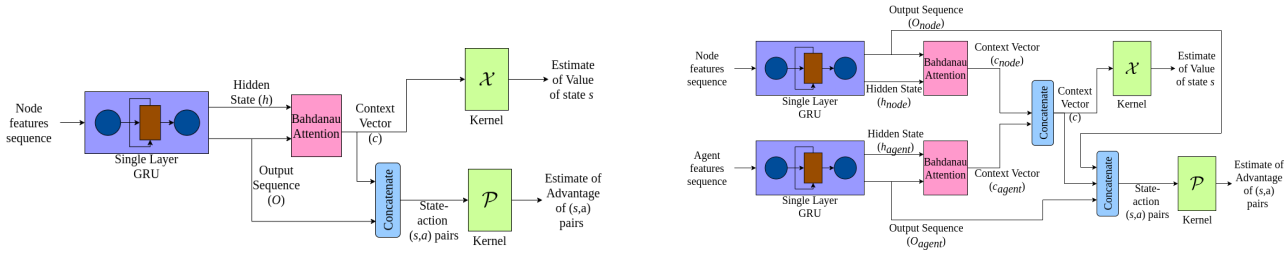
$$e_k(t_m) \geq w(l_k(t_m), j) + e_{\min}(j)$$

where $e_{\min}(j)$ is the minimum amount of energy needed to go from node j to a base node, $e_k(t_m)$ is the energy of agent $k \in A$ at time-step t_m, $l_k(t_m)$ is the location of agent $k \in A$ at the time step t_m in the planning horizon and $w(l_k(t_m), j)$ is the travel time (amount of energy) needed to go from node $l_k(t_m)$ to j. As long as each agent $k \in A$ chooses an action from $\mathcal{N}_{\text{safe}}(l_k(t_m))$, the energy constraints are guaranteed to be satisfied.

The following sections provide the details of the approaches used and the insights gained.

V. The Single Agent Problem

In the single agent problem, the set of agents, A is a singleton set. Hence, the action space for the RL framework described in section IV will reduce to being just the out-neighbors of the node occupied by the agent in the planning horizon, $\mathcal{N}(l(t_m))$. Since there is only one agent, the agent features defined in section IV can be absorbed into the node features. Thus, the state space, $\mathcal{S}$ for the single agent problem will be set of all feasible states $s(m) := \big(\big(p_i, d_i(t_m), x_i(t_m), \big(y_i(t_m,j)\big)_{j\in V_b}, W_i\big)_{i\in V}, g(t_m), T - t_m\big)$. Note that in this section the subscripts and arguments representing the



(a) Single Agent

(b) Multi Agent

Fig. 2: Figures 2a and 2b depict the neural network architectures used for the Single and Multi Agent problems respectively

agent have been dropped for ease of notation. The energy constraints are handled as described in IV-A.

A. Q-Network Architecture for the Single Agent Problem

Our Q-Network $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ consists of two real-valued non-linear kernels $\mathcal{X}(.;\theta_{\mathcal{X}})$ parameterized by $\theta_{\mathcal{X}}$ and $\mathcal{P}(.;\theta_{\mathcal{P}})$ parameterized by $\theta_{\mathcal{P}}$. Let us say, in the n^{th} instance of Problem 6 starting at t_n, the agent observes a state $s(m)$ at time t_m, for $t_n \leq t_m < t_n + T$. Say we are evaluating a candidate action $a \in \mathcal{N}_{\text{safe}}(l(t_m))$.

The input to kernel $\mathcal{X}$ is a modified vector $\hat{s}(m) = \big((p_i, d_i(t_m), x_i(t_m), g(t_m), \big(y_i(t_m, j)\big)_{j \in V_b}, W_i, T - t_m)_{i \in V}\big)$ and its output is interpreted as the estimate of the value function of the state $s(m)$. The input to kernel $\mathcal{P}$ is a state-action pair vector $\big(\hat{s}(m), p_a, d_a(t_m), x_a(t_m), g(t_m), \big(y_i(t_m, j)\big)_{j \in V_b}, W_a, T - t_m\big)$, and the output is interpreted as the advantage of the state-action pair.

The Q-value estimate of the given state-action pair (s, a) is computed as in (7), where $\mathcal{A}_s = \mathcal{N}_{\text{safe}}(l(t_m))$.

$$Q(s,a) = \mathcal{X}(s) + \mathcal{P}(s,a) - \frac{1}{|\mathcal{A}_s|} \sum_{a' \in \mathcal{A}_s} \mathcal{P}(s,a') \quad (7)$$

Initially, both the kernels $\mathcal{X}$ and $\mathcal{P}$ were modeled as fully connected neural networks with 4 layers having ReLU activation functions on alternate layers. But this architecture yielded poor results when the graphs contained nodes whose priorities were distributed unevenly and when the graph contained base nodes whose priorities are zero. One possible reason for this would be because the neural network consisting of only fully connected layers was unable to generate a good low dimensional representation of the state.

To fix this issue, we use Recurrent Neural Networks (RNNs) to first obtain a low dimensional representation of the state before feeding it as an input to the non-linear kernels. RNNs are good at generating a single vector representation of an input sequence. But in this problem, the state is represented by the nodes of a graph which do not have an inherent sequence. Hence, we use Bidirectional RNNs [34] (BRNNs) which process input sequences in both forward and backward directions simultaneously to obtain low dimensional representation of the state and features of each node.

The input to the BRNN is the modified state vector $\hat{s}(m)$ with the features of each node fed in a fixed sequence, $i = \{1, 2, \ldots, |V|\}$. The BRNN produces an output sequence O, containing a latent space representation of the features of each node and a hidden state vector h, containing a representation of the input sequence. The obtained hidden state vector and output sequence are fed into an attention mechanism with the hidden state vector as the query and the output sequence as the keys. The attention mechanism used is the Bahdanau Attention [35], which computes a context vector as:

$$c(m) = \text{softmax}\big(V_a^T \tanh(W_a h + U_a O)\big) K$$

where V_a, U_a and W_a are learnable matrices of appropriate dimensions. This context vector $c(m)$, is considered as the low dimensional representation of the state at time-step t_m. The vector $c(m)$ is fed as input to the kernel $\mathcal{X}$ to obtain the value of the state, $s(m)$. The kernel $\mathcal{P}$ is fed the state-action pair, $(s, a) = \big(c(m), O_a\big)$, where a is the action being evaluated and O_a is the vector corresponding to a in the output sequence, as the input to obtain the advantage of (s, a). The Q-value estimate is then computed as in (7). Fig 2a provides a graphical representation of the neural network architecture used.

The BRNN is a single layer bidirectional Gated Recurrent Unit (GRU) with a hidden vector size of 128. The kernel $\mathcal{X}$ is modeled by a neural network consisting of *four* consecutive fully connected hidden layers with $256, 512, 265$ and 64 units in each hidden layer respectively, with the first and third hidden layers having rectified linear (ReLu) activation, and the remaining layers having linear activation. The kernel $\mathcal{P}$ is modeled by a neural network described by the following equations:

$$\begin{aligned} r_t &= W_1 * x, & z_t &= W_2 * x \\ n_t &= \tanh(W_3 * x + r_t), & h_t &= (1 - z_t) * (n_t) \\ o_t &= W_4 * h_t \end{aligned}$$

where, W_1, W_2, W_3 and W_4 are learnable matrices of appropriate dimensions, x is the input to the neural network and o is the output.

VI. The Multi Agent Problem

The framework used in the single agent problem described in section V was then extended to the general multi-agent problem. The action space for the multi-agent problem is given in section IV. As the number of agents increases, the cardinality of the action space increases exponentially. Evaluating every possible action in a given state is computationally intractable. Hence, we adopt a *sequential optimization* approach to choose the actions for the agents. In this approach,

the agents choose their action, that is, the next node to go to, in a sequential fashion. For a given state, the sequence in which actions have to be taken could be different and hence has to be learnt.

A. Sequential Decision Making

At the m^{th} time-step where an agent is in a node, t_m in the planning horizon, only the active agents ($k \in A$ s.t $q_k(t_m) = 1$) can make decisions to choose which node to visit next. The action for each active agent is given by $a_k(t_m) \in \mathcal{N}(l_k(t_m))\ \forall k \in A$ s.t $q_k(t_m) = 1$. The Q-value estimate for each of the possible actions for each active agent is calculated. The active agent ($k_{\max}$) whose agent-action pair has the highest Q-value estimate among all the other agent-action pairs, chooses to take the corresponding action. In doing so, it becomes inactive ($q_{k_{\max}}(t_m) = 0$) and updates its intent node to $a_{k_{\max}}(t_m)$. This sets the intent node features of agent $k_{\max}$ to the features of the node $a_{k_{\max}}(t_m)$ and $t_{intent}(k_{\max}, t_m)$ to the travel time $w(l_{k_{\max}}(t_m), a_{k_{\max}}(t_m))$. By doing so the intent of this agent is shared with the other active agents who can then make decisions accordingly. This process of estimating Q-values for the actions of the active agents and letting the agent with highest Q-value estimate choose the corresponding action is repeated until all active agents have chosen an action. In this fashion, the sequence in which the agents have to take actions and the corresponding action for a given state is implicitly learnt.

The Duelling Double Deep Q-Learning (D3QN) algorithm is used to train the RL agent for the multi agent problem. This was chosen because of the success shown by the D3QN algorithm in the single agent problem. Satisfaction of energy constraints are guaranteed via projection to a safe set as described in section IV-A.

B. Q-Network Architecture for the Multi Agent Problem

The Q-Network architecture used for the Multi-Agent problem is similar to that of the single agent problem. The architecture consists of the same two non-linear kernels, $\mathcal{X}$ and $\mathcal{P}$ as defined in section V-A. To obtain a low dimensional representation of the state we use two BRNNs, one for the node features (B_{node}) whose input is the features of each node, $f_{\text{node}}(i, m)$ fed in a fixed sequence, $i = \{1, 2, \ldots, |V|\}$ and the other for the agent features B_{agent} whose input is the features of each agent, $f_{\text{agent}}(k, m)$ fed in a fixed sequence, $k = \{1, 2, \ldots, |A|\}$. B_{node} produces an output sequence O_{node} and a hidden state vector (h_{node}) and B_{agent} produces an output sequence O_{agent} and a hidden state vector (h_{agent}). These are then passed through a Bahdanau based attention mechanism as described in section V-A to obtain a node context vector $c_{\text{node}}(m)$ and an agent context vector $c_{\text{agent}}(m)$. The low dimensional representation of the state is taken as $\hat{s}(m) = (c_{\text{node}}(m), c_{\text{agent}}(m))$. This is fed into the kernel $\mathcal{X}$ to obtain the value of the state $s(m)$. The input to the kernel $\mathcal{P}$ is the state-action pair $(s, a) = \big(c_{\text{node}}(m), c_{\text{agent}}(m), O_{\text{agent}}(k), O_{\text{node}}(a)\big)$ where $a \in \mathcal{N}_{\text{safe}}(l_k(t_m))$ is the candidate action being evaluated for the active agent k, $O_{\text{agent}}(k)$ is the vector corresponding to agent k in O_{agent} and $O_{\text{node}}(a)$ is the vector corresponding to a in O_{node}. The output of kernel $\mathcal{P}$ is considered as the advantage of (s, a). The Q-value estimate is then computed as (7). Fig 2b provides a graphical representation of the neural network architecture used.

The BRNNs for the node and agent features are modeled using single layer GRUs with a hidden vector size of 128. The structure of the kernels $\mathcal{P}$ and $\mathcal{X}$ remain the same as in section V-A.

VII. Simulation

We test our approach empirically for both the single agent and multi agent problem, on a set of 12 directed graphs with various number of nodes and topology. These include graphs with 10, 15, 20, and 25 nodes (3 graphs each, respectively making a total of 12 random graphs). Additionally, the single agent problem is also tested on three large graphs containing 100 nodes each. These random directed graphs are generated by repeatedly removing a fixed number of random edges from an undirected graph, until the obtained directed graph is strongly connected. The undirected graphs used were random Watts-Strogatz graphs with the average number of neighbours as a parameter.

1) Training: We separately train *five* RL policies, each with different parameter initializations, for each of the graphs for the single and multi-agent problems. For the random graphs, in each episode we first randomly select *high-priority nodes* (HPNs), and *low-priority nodes* (LPNs) (check Table I for the number of HPNs, LPNs and number of agents for the multi-agent problem in various graphs). The priorities for HPNs and LPNs are then uniformly sampled from $\{5, 6, 7\}$ and $\{1, 2\}$, respectively, and the initial demands for HPNs and LPNs are uniformly sampled from $\{10, 11, ..., 19\}$ and $\{1, 2, 3, 4\}$ respectively. The initial position of the agent is sampled from a uniform distribution over the set of nodes V for all the graphs for the single agent problem. For the multi agent problem, the agents are allowed to start on an edge. Hence, their initial positions are sampled uniformly over the set of edges E for all the graphs.

No. of nodes	No. of HPNs	No. of LPNs	Average No. of neighbours	No. of agents for multi-agent problem
10	3	7	5	2
15	3	12	7	3
20	3	17	5	3
25	5	20	7	4
100	25	75	4	<NA>

TABLE I: Number of high priority nodes (HPNs), low priority nodes (LPNs), the average number of neighbors for each node and the number of agents in the multi-agent problem in graphs with varied number of total nodes in the 12 random graphs

DQN Training: The DQN based RL agent is trained by running a PyTorch implementation of the D3QN algorithm. The neural networks as described in sections V-A and VI-B are initialized randomly. An epsilon greedy policy is used for exploration with the value of ϵ decaying exponentially from 0.5 to 0.0001. The learning rate starts at a value of 0.5 and is scheduled to decrease by a factor 0.75 after every 20000

learning iterations. Training is done for 225000 learning iterations with a batch size of 16.

2) Testing on random instances: To empirically test the performance of the policies during or after the training process, we sample 50 instances of Problem 6 for each of the random graphs for both the single and multi agent problem. We formulate an integer program (IP) corresponding to each of these instances and solve them using the SCIP solver in Python supported by PySCIPOpt. Due to computational limitations, the solver is allowed to run for a maximum time period of 10800 seconds and the best solution obtained after that is used. For each of the graphs, for each of the 50 problem instances, we compute the percentage relative difference between objective function values due to the RL policy (J_{RL}) and the optimal solution (J_{IP}), i.e., for instance i, $g_i = \frac{J_{RL}-J_{IP}}{J_{IP}}\%$. We use the empirical average (μ_g) and standard deviation (σ_g) in these 50 values to compare any two RL policies.

3) Comparison of computation times: We compare the computation time per episode for the RL policies and the SCIP solver. The data is collected from the test instances on the random directed graphs discussed in the previous subsections. This comparison is done on a machine with an i7-8700 processor and 40GB of RAM.

4) Testing on long duration rolling horizon problems: We evaluate the performance of the trained policies in a rolling horizon framework for persistent surveillance on the three random graphs with 25 nodes for the single agent problems and 3 graphs with 15 nodes and 3 agents for the multi agent problem. We fix a planning horizon of 15 time steps for single agent and 10 time steps for multi agent, implement only the first decision, and re-plan again once an agent visits a node. For the three random graphs, we assume that the priorities change randomly. Say, the sequence of time steps that the priorities get changed is given by $\{z_i\}_{i\in\mathbb{N}_0}$. We determine the sequence by $z_{i+1} = z_i + \kappa_i$, $z_0 = 0$ and κ_i is sampled uniformly from $\{1, 2, 3, ..., 10\}$. At these time steps, HPNs and LPNs are sampled, and their priorities are assigned randomly, as discussed earlier. The initial node demands for all 3 graphs are set to zero. We run simulations for 150 time steps and compute the sum over the 150 time steps of the sum of the product of priorities and demands of all the nodes and the sum of time since last visit to a base node for all agents as $\hat{J}_{RL}$ (objective value with RL policy) and $\hat{J}_{SCIP}$ (objective value with rolling horizon optimal control problem solved using the SCIP solver) and compare the percentage relative difference $G_{RHP} = \frac{\hat{J}_{RL}-\hat{J}_{SCIP}}{\hat{J}_{SCIP}}\%$. For all rolling horizon problems, we use policies trained on the respective graphs for which they are being evaluated.

5) Studying the effect of battery life on the RL policy: The satisfaction of the energy constraints is guaranteed by projecting the action space of the agent to a safe set. Hence, we study the effect of varying the battery life on the optimality gap defined in VII-.2. We sample 50 instances each of problem (6) with a single agent, for each of the random graphs with 25 nodes and 15 nodes with 3 agents for the single and multi agent problems respectively, with different battery lives for the agent. The RL policy is trained with one battery life and evaluated on the remaining instances. The battery lives used for testing are $\{5, 10, 15, \ldots, 50\}$ and the RL policy is trained with a battery life of 10. The horizon length of optimization is set to 15 time steps.

A. Results

Now we present some results establishing the near-optimality and computation benefit we get from the proposed approach.

In figures 3a and 3b we present the μ_g (average relative percentage gap between RL and SCIP) at different training iterations for the single agent and multi agent problems respectively for graphs with various number of nodes. We observe that the average relative percentage gap against SCIP solver reduces with the training iterations.

Table II summarizes the results obtained by testing on problem instances as described in the previous sections. From this, we can see that the proposed RL-based solution outperforms the greedy heuristic. In the greedy heuristic, we use a strategy where, at each decision time step, each agent moves to a neighboring node with the highest product of priority and demand at the next time step if the agent were to take no action. Note that the reported values are with the data from 50 test instances each on 3 different graphs for the random graphs with $10, 15, 20$, and 25 nodes. Additionally, for the RL policies, the averages and standard deviations are over 5 different trained policies, showing the method's robustness to network initialization. Although the policies have converged (figure 3b), we observe that the reported gaps for the multi agent problem are not small. This could be because the features provided to the neural network are not expressive enough or are not giving full information of the current state. More investigation is needed to improve these results.

Graph	RL-Single Agent (%)		RL-Multi agent (%)		Greedy Single Agent (%)		Greedy Multi Agent (%)	
	μ_g	σ_g	μ_g	σ_g	μ_g	σ_g	μ_g	σ_g
10 nodes	2.38	4.8	28.11	14.79	26.84	20.81	53.83	33.15
15 nodes	2.91	3.74	43.32	14.03	30.71	18.22	67.11	25.6
20 nodes	3.56	4.81	41.3	15.38	18.62	12.65	61.83	24.63
25 nodes	4.67	4.83	46.32	15.61	19.32	10.84	56.49	19.62
100 nodes	1.7	2.11	<NA>	<NA>	6.81	3.44	<NA>	<NA>

TABLE II: Table with value of the average (μ_g) and standard deviation (σ_g) in the percentage relative gap between various policies (Single agent RL, Multi agent RL and greedy heuristic for single and multi agent) and SCIP solutions

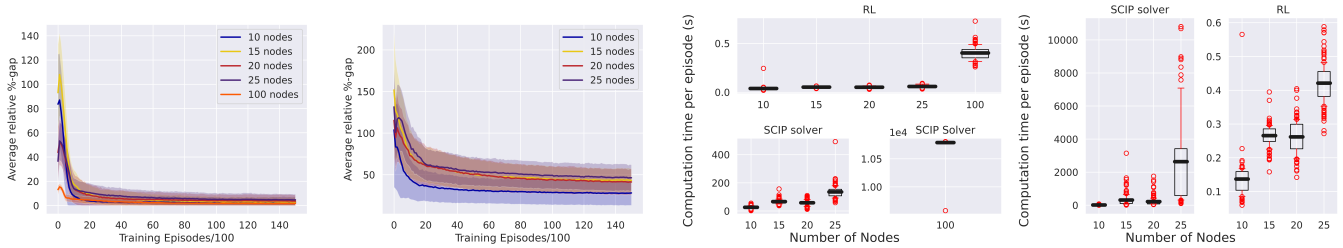
Graph	Duality Gap (%)	
	SA	MA
10 nodes	0.0	0.0
15 nodes	0.0	0.0
20 nodes	0.0	0.0
25 nodes	0.0	0.29
100 nodes	14.38	<NA>

TABLE III: Average percentage duality gap obtained from the SCIP solver with a maximum solving time of 10800s for single agent(SA) and multi agent (MA) problem

Graph	μ_{RHP} (%)	
	SA	MA
G1	15.57	150.58
G2	21.75	113.06
G3	7.91	149.78

TABLE IV: Average percentage relative gap between the SCIP solver and the RL policies for various graphs in the rolling horizon setting after 150 time steps for single agent (SA) and 100 time steps for multi agent (MA)

Table III gives the average percentage duality gap between



(a) Single agent Training curves (b) Multi agent Training curves (c) Computation time Comparison for Single Agent Problem (d) Computation time Comparison for Multi Agent Problem

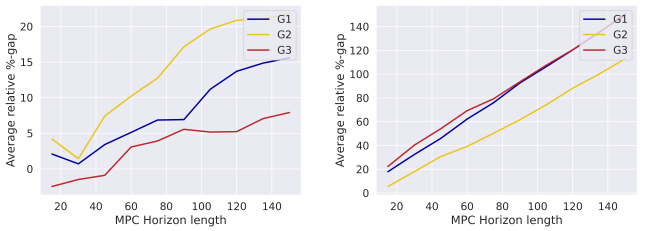
Fig. 3: Figures 3a and 3b present the evaluation of policies at various training stages for RL agent for sigle and multi agent problem respectively. Figures 3c and 3d present box plots of the computation time per episode for SCIP and RL policies for graphs with various number of nodes. For figures 3c and 3d, 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.

the primal and dual solutions obtained from the SCIP solver with a maximum solving time of 10800s. Duality Gap of 0 means that the optimal solution was obtained. We see that, for the 100 node graphs and some 25 node graphs for the multi agent problem were not solved to optimality in the given time period.

Figures 3c and 3d compare the computation time per episode for SCIP solver and RL policies. The data is collected from the test instances on the 12 Watts-Strogatz graphs discussed in the previous sections. We observe up to 4 orders of magnitude difference in the computation time between the SCIP solver and the RL approaches. We also note that the average and variance in computation times per episode for the SCIP solver increase with the number of nodes in graphs.

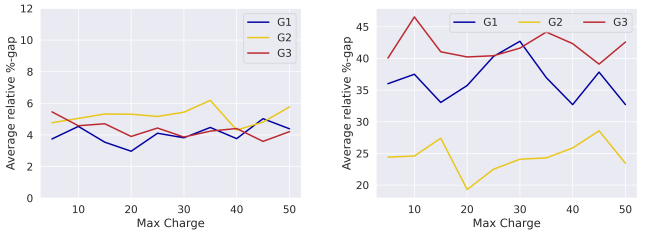
Table IV presents the average percentage relative gap between the SCIP solver and the RL policies for various graphs in the rolling horizon setting. Note that graphs G1, G2, and G3 are random Watts-Strogatz graphs as discussed in the previous sections. For each of these graphs, the average is over 5 trained policies (with random policy initialization) and 5 instances of rolling horizon problem with initial demands and time-varying priorities calculated, as discussed before. We observe that the reported gaps on a large simulation time of 150 for single agent and 100 for multi agent time steps are not small, although the policies for the single agent problem are near-optimal. This may be due to the accumulation of small errors over the simulation duration. Figure 4 presents a better picture of the effect of the fixed horizon on the long term objective function. We see that as the simulation time increases (relative to the planning horizon length), the performance of the RL policies degrade. However, it is interesting to see that, the performance degradation reduces with the simulation time for the single agent problem but not for the multi agent problem.

Figures 5a and 5b presents the effect of the maximum battery life on the average optimality gap of the RL policy for 3 of the random 25 node and 15 node graphs for the single and multi agent problems respectively. The values were obtained after averaging over 50 problem instances and over 5 trained policies on the respective graphs. We observe that, the maximum battery life does not affect the optimality gap by much, thus proving the robustness of the method to heterogenous agents having different battery lives.



(a) Single Agent (b) Multi Agent

Fig. 4: Variation of average relative percentage gap against SCIP with simulation time for various graphs in the rolling horizon setting. G1, G2 and G3 represent random Watts-Strogatz graphs



(a) Single Agent (b) Multi agent

Fig. 5: Variation of average relative percentage gap against SCIP with maximum battery life of agents. G1, G2 and G3 are 25 node and 15 node graphs for the single agent and multi agent problem respectively

VIII. Conclusions and Future Work

In this project, we proposed a rolling horizon based RL framework for persistent surveillance using energy constrained robots, where each robot has to not only perform surveillance but also report the collected data to base nodes. We first solved the single agent problem and then generalized the same for the multi agent case. We compared the RL policies against an SCIP solver. We observed that the method provides solutions that are almost comparable to those of the SCIP solver for the single agent problem but were highly sub-optimal for the multi agent problem. The method provides several orders of magnitude reduction in the computation time for both the single and multi agent problem.

Future directions include improving the performance in the multi agent case using more expressive neural networks and improving the long term performance with the rolling horizon framework.

REFERENCES

[1] J. Urrutia, "Art gallery and illumination problems," in *Handbook of computational geometry*, pp. 973–1027, Elsevier, 2000.

[2] 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.

[3] F. Pasqualetti, J. W. Durham, and F. Bullo, "Cooperative patrolling via weighted tours: Performance analysis and distributed algorithms," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1181–1188, 2012.

[4] F. Pasqualetti, A. Franchi, and F. Bullo, "On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms," *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 592–606, 2012.

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

[7] 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)*, pp. 6390–6397, IEEE, 2021.

[8] 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.

[9] E. Dasdemir, R. Batta, M. Köksalan, and D. T. Öztürk, "Uav routing for reconnaissance mission: A multi-objective orienteering problem with time-dependent prizes and multiple connections," *Computers & Operations Research*, vol. 145, p. 105882, 2022.

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

[11] P. Toth and D. Vigo, *The vehicle routing problem*. SIAM, 2002.

[12] M. Pavone, E. Frazzoli, and F. Bullo, "Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment," *IEEE Transactions on automatic control*, vol. 56, no. 6, pp. 1259–1274, 2010.

[13] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[14] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[15] Y. Hu, Y. Yao, and W. S. Lee, "A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs," *Knowledge-Based Systems*, vol. 204, p. 106244, 2020.

[16] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, "Reinforcement learning for solving the vehicle routing problem," *Advances in neural information processing systems*, vol. 31, 2018.

[17] G. Bono, J. S. Dibangoye, O. Simonin, L. Matignon, and F. Pereyron, "Solving multi-agent routing problems using deep attention mechanisms," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 12, pp. 7804–7813, 2020.

[18] 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, pp. 1122–1129, IEEE Computer Society, 2004.

[19] 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)*, pp. 1–6, IEEE, 2013.

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

[21] 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.

[22] 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)*, pp. 3504–3509, IEEE, 2023.

[23] F. Amigoni, J. Banfi, and N. Basilico, "Multirobot exploration of communication-restricted environments: A survey," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 48–57, 2017.

[24] J. Scherer and B. Rinner, "Multi-robot persistent surveillance with connectivity constraints," *IEEE Access*, vol. 8, pp. 15093–15109, 2020.

[25] 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.

[26] 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)*, pp. 1225–1230, IEEE, 2016.

[27] J. Scherer and B. Rinner, "Short and full horizon motion planning for persistent multi-uav surveillance with energy and communication constraints," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 230–235, IEEE, 2017.

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

[29] K. Kalyanam, S. Manyam, A. Von Moll, D. Casbeer, and M. Pachter, "Scalable and exact milp methods for uav persistent visitation problem," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 337–342, IEEE, 2018.

[30] 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)*, pp. 9130–9135, IEEE, 2018.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[32] 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, 2016.

[33] 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*, pp. 1995–2003, PMLR, 2016.

[34] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

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