Multi-Robot Routing for Persistent Monitoring with Latency Constraints

Ahmad Bilal Asghar   Stephen L. Smith   Shreyas Sundaram

Abstract— **In this paper we study a multi-robot path planning problem for persistent monitoring of an environment. We represent the areas to be monitored as the vertices of a weighted graph. For each vertex, there is a constraint on the maximum time spent by the robots between visits to that vertex, called the *latency*, and the objective is to find the minimum number of robots that can satisfy these latency constraints. The decision version of this problem is known to be PSPACE-complete. We present a $O(\log \rho)$ approximation algorithm for the problem where ρ is the ratio of the maximum and the minimum latency constraints. We also present an orienteering based heuristic to solve the problem and show through simulations that in most of the cases the heuristic algorithm gives better solutions than the approximation algorithm. We evaluate our algorithms on large problem instances in a patrolling scenario and in a persistent scene reconstruction application. We also compare the algorithms with an existing solver on benchmark instances.**

I. Introduction

With the rapid development in field robotics, teams of robots can now perform long term monitoring tasks. Examples of such tasks include infrastructure inspection [1] to detect presence of anomalies or failures; patrolling for surveillance [2], [3] to detect threats in the environment; 3D reconstruction of scenes [4], [5] in changing environments; and informative path planning [6]. In such persistent monitoring scenarios, locations in an environment need to be visited repeatedly by a team of robots. Since the duration of the events, or the rate of change of the properties to be monitored, can be different for different locations, each location will have a different latency constraint, which specifies the maximum time allowed between consecutive visits to that location. We study the problem of finding a set of paths that continually visit a set of locations while collectively satisfying the latency constraints on each location.

Related Work: Persistent monitoring problems have been extensively studied in the literature [7], [8]. In [9], persistent coverage using multiple robots in a continuous environment is considered. The problem of determining a visit sequence for a set of locations along with the time spent at each location to gather information is considered in [10], [11]. For the problem with latency constraints, the authors in [12] use incomplete greedy heuristics to find if a single robot can satisfy the constraints on all vertices of a graph. They show that if a solution exists, then a periodic solution also exists. In this paper, we consider the multi-robot problem and our objective is to minimize the number of robots that can satisfy the latency constraints on the given graph. This problem has been considered in [13], [14], where it is called *Cyclic Routing of Unmanned Aerial Vehicles*. The decision version of the problem for a single robot is shown to be PSPACE-complete in [14]. The authors also show that the length of even one period of a feasible walk can be exponential in the size of the problem instance. In [13], the authors propose a solver based on Satisfiability Modulo Theories (SMT). To apply an SMT solver, they impose an upper bound on the length of the period of the solution. Since an upper bound is not known *a priori*, the solver will not return the optimal solution if the true optimal period exceeds the bound. The authors generate a library of test instances, but since their algorithm scales exponentially with the problem size, they solve instances up to only 7 vertices. We compare our algorithms with their solver and show that our algorithms run over 500 times faster on average and return solutions with the same number of robots on 98% of the benchmark instances provided by [13].

A related single robot problem is studied in [15] where each vertex has a weight associated with it and the objective is to minimize the maximum weighted latency (time between consecutive visits) for an infinite walk. The authors provide an approximation algorithm for this problem. The authors in [2] study the single robot problem for a security application where the length of attack on each vertex of the graph is given. To intercept all possible attacks, they design an algorithm to repeatedly patrol all vertices with the maximum revisit time to each vertex less than its length of attack.

Several vehicle routing problems are closely related to persistent monitoring with latency constraints. In the *vehicle routing problem with time windows* [16], customers have to be served within their time windows by several vehicles with limited capacity. Since the problem does not require repeated visits, the length of the resulting tour is polynomially bounded, and thus the problem is in NP. In the *deadline-TSP* [17], the vertices have deadlines for first visits. The main difference between these problems and the cyclic routing problem with latency constraints is that the latency constraints need to be satisfied indefinitely which makes it harder than these problems.

Contributions: We present a $O(\log \rho)$ approximation algorithm for the problem where ρ is the ratio of the maximum and the minimum latency constraints (Section IV). We present a heuristic algorithm to solve the problem (Section V) and show through simulations that the heuristic algorithm gives better solutions than the approximation algorithm. We evaluate the performance of the algorithms on large problem instances and compare our algorithms against an existing

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Ahmad Bilal Asghar and Stephen L. Smith are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1 Canada. Shreyas Sundaram is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA.

solver on benchmark instances (Section VI).

II. Background and Notation

Given a directed graph $G = (V, E)$ with edge lengths $l(e)$ for each $e \in E$, a *walk* in graph G is defined as a sequence of vertices $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E$ for each $1 \leq i < k$. An *infinite walk* is a sequence of vertices $(v_1, v_2, \ldots)$ such that $(v_i, v_{i+1}) \in E$ for each $i \in \mathbb{N}$. Given walks W_1 and W_2, $[W_1, W_2]$ represents the concatenation of the walks. Given a finite walk W, an infinite periodic walk $\Delta(W)$ is constructed by concatenating infinite copies of W together. A *cycle* is a walk that starts and ends at the same vertex with no other vertex appearing more than once.

In general, a walk can stay for some time at a vertex before traversing the edge towards the next vertex. Therefore we define a *timed walk* W in graph G as a sequence $(o_1, o_2, \ldots, o_k)$, where $o_i = (v_i, t_i)$ is an ordered pair that represents the holding time t_i that the walk W spends at vertex v_i, such that $(v_i, v_{i+1}) \in E$ for each $1 \leq i < k$. The definitions of infinite walk and periodic walk can be extended to infinite timed walk and periodic timed walk. A walk with ordered pairs of the form $(v_i, 0)$ becomes a simple walk. The vertices traversed by walk W are given by $V(W)$ and the length of walk $W = ((v_1, t_1), (v_2, t_2), \ldots, (v_k, t_k))$ is given by $l(W) = \sum_{i=1}^{k-1} l(v_i, v_{i+1}) + \sum_{i=1}^{k} t_k$. Since we are considering a multi-robot problem, synchronization between the walks is important. Given a set of walks $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ on graph G, we assume that at time 0, each robot i is at the first vertex v_1^i of its walk W_i, and will spend the holding time t_1^i at that vertex before moving to v_2^i.

Given a graph G and length λ, the Minimum Cycle Cover Problem (MCCP) is to find minimum number of cycles that cover the whole graph such that the length of the longest cycle is at most λ. This problem is NP-hard and a 14/3-approximation algorithm for MCCP is given in [18].

Given a graph $G = (V, E)$ with vertex weights ψ_i for $i \in V$, vertices $s, t \in V$, and length λ, the Orienteering problem is to find a path from vertex s to t of length at most λ such that the sum of the weights on the vertices in the path is maximized. This problem is also NP-hard and a $(2 + \epsilon)$-approximation is given in [19].

III. Problem Statement

Let $G = (V, E)$ be a directed weighted graph with edge lengths $l(e)$ for each $e \in E$. The edge lengths are metric and represent the time taken by the robot to travel between the vertices. The latency constraint for each vertex v is denoted by $r(v)$ and represents the maximum time allowed between consecutive visits to v. The time taken by the robots to inspect a vertex can be added to the length of the incoming edges of that vertex to get an equivalent metric graph with zero inspection times and modified latency constraints [13]. Hence, we assume that the time required by the robots to inspect a vertex is zero. We formally define the problem statement after the following definition.

Definition III.1 (Latency)**.** Given a set of infinite walks $\mathcal{W} = \{W_1, W_2, \ldots, W_k\}$ on a graph G, let a_i^v represent the i^{th} arrival time for the walks to vertex v. Similarly, let d_i^v represent the i^{th} departure time from vertex v. Then the latency $L(\mathcal{W}, v)$ of vertex v on walks $\mathcal{W}$ is defined as the maximum time spent away from vertex v by the walks, i.e., $L(\mathcal{W}, v) = \sup_i (a_{i+1}^v - d_i^v)$.

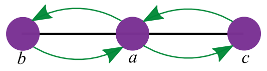


Fig. 1: A graph with three vertices and the walk (a, b, a, c). The length of shown edges is one. Equally placing two robots on this walk does not halve the latencies.

Problem III.2 (Minimizing Robots with Latency Constraints)**.** Given the latency constraints $r(v), \forall v \in V$, the optimization problem is to find a set of walks $\mathcal{W}$ with minimum cardinality such that $L(\mathcal{W}, v) \leq r(v), \forall v \in V$.

The decision version of the problem is to determine whether there exists a set of R walks $\mathcal{W} = \{W_1, W_2, \ldots, W_R\}$ such that $L(\mathcal{W}, v) \leq r(v)$ for all $v \in V$. Note that a general solution to Problem III.2 will be a set of timed walks with possibly non-zero holding times.

In this problem definition, the graph and its edge lengths capture the robot motion in the environment. This graph can be generated from a probabilistic roadmap, or any other environment decomposition method. The latency constraints provide the maximum allowable time between visits to a vertex. For example, in dynamic scene reconstruction, each vertex corresponds to a viewpoint [4]. The latency constraints may encode the maximum staleness of information that can be tolerated for the voxels captured at that viewpoint.

A. Multiple Robots on the Same Walk

In multi-robot problems that involve finding cycles in a graph, equally placing n robots on a cycle reduces the cost of that tour by a factor of n [20]. That is not true for Problem III.2: if a periodic walk W gives latency $L(W, v)$ on vertex v, equally spacing more robots on one period of that walk does not necessarily reduce the latency for that vertex. Figure 1 gives an example of such a walk. The latencies of vertices a, b and c on the walk (a, b, a, c) are 2, 4 and 4 respectively. The length of one period of the walk is 4. If we place another robot following the first robot with a lag of 2 units, the latency of vertex a remains the same. If we place the second robot at a lag of 1 unit, the latency will reduce to 1 for vertex a and 3 for vertices b and c. Hence we need more sophisticated algorithms than finding a walk for a single robot and adding more robots on that walk until the constraints are satisfied, unless that walk is a cycle.

IV. Approximation Algorithm

Since Problem III.2 is PSPACE-complete, we resort to finding approximate and heuristic solutions to the problem. An approximation algorithm is presented in this section.

A. $O(\log \rho)$ Approximation

We first mention a simple approach to the problem and then improve it incrementally to get the approximation

Algorithm 1: ApproximationAlgorithm

```
Input: Graph $G = (V, E)$, latency constraints $r(v), \forall v$
Output: A set of walks $\mathcal{W}$, such that $L(\mathcal{W}, v) \leq r(v)$
1: Let $r_{\texttt{max}} = \max_v r(v)$, $r_{\texttt{min}} = \min_v r(v)$, $\rho = \frac{r_{\texttt{max}}}{r_{\texttt{min}}}$
2: if $r_{\texttt{max}}/r_{\texttt{min}}$ is an exact power of 2 then $\rho = \frac{r_{\texttt{max}}}{r_{\texttt{min}}} + 1$
3: $\mathcal{W} = \{\}$
4: Let $V_i$ be the set of vertices $v$ such that $r_{\texttt{min}}2^{i-1} \leq r(v) < r_{\texttt{min}}2^i$ for $1 \leq i \leq \lceil \log_2 \rho \rceil$
5: for $i = 1$ to $\lceil \log_2 \rho \rceil$ do
6:     $\mathcal{C} = \text{MCCP}(V_i, r_{\texttt{min}}2^{i+1})$
7:     for $C$ in $\mathcal{C}$ do
8:         Equally place $\lceil l(C)/\min_{v \in V(C)} r(v) \rceil$ robots on cycle $C$ to get walks $\mathcal{W}'$
9:         $\mathcal{W} = \{\mathcal{W}, \mathcal{W}'\}$
```

algorithm. One naive solution is to find a TSP tour of the graph and equally place robots on that tour to satisfy all the latency constraints. However, a single vertex with a very small $r(v)$ can result in a solution with the number of robots proportional to $1/r(v)$. To solve this issue, we can partition the vertices of the graph such that the latencies in one partition are close to each other, and then place robots on the TSP tour of each partition. If more than one robot is required for a partition V', then another approach is to solve the MCCP for that partition. The benefit of using the MCCP is that if all the vertices in V' had the same latency requirement, then we have a guarantee on the number of cycles required for that partition. However, a general solution to the problem might not consist of simple cycles. Lemma IV.2 relates solutions consisting of cycles to general solutions and shows that a solution consisting of cycles will have latencies no more than twice that of any general solution with same number of robots. Therefore, if we solve the MCCP on a partition with its latency constraints multiplied by two, we have a guarantee on the number of cycles. We can then place multiple robots on each cycle to satisfy the latency constraints.

The approximation algorithm is given in Algorithm 1. The first four lines of the algorithm partition the vertices according to their latency constraints. For each of those partitions, the function MCCP(V, λ) called in line 6 uses an approximation algorithm for the minimum cycle cover problem from [18]. Then, the appropriate number of robots are placed on each cycle returned by the MCCP function to satisfy the latency constraints. We will need the following definition to establish the approximation ratio of Algorithm 1. A similar relaxation technique was used in [15].

Definition IV.1. Let $r_{\texttt{min}} = \min_v r(v)$. The latency constraints of the problem are said to be relaxed if for any vertex v, its latency constraint is updated from $r(v)$ to $\bar{r}(v) = r_{\texttt{min}}2^x$ such that x is the smallest integer for which $r(v) < r_{\texttt{min}}2^x$.

We will also need the following lemma that follows from Lemma 2 in [20] and we omit the proof for brevity.

Lemma IV.2. *For any set of walks $\mathcal{W}$ on an undirected metric graph with vertices V, there exists a set of walks $\mathcal{W}'$ on V with $|\mathcal{W}| = |\mathcal{W}'|$, such that each walk $W_i \in \mathcal{W}'$ is a cycle of vertices $V_j \subseteq V$, and the sets V_j partition V, and $\max_v L(\mathcal{W}', v) \leq 2 \max_v L(\mathcal{W}, v)$.*

Proposition IV.3. *Given an undirected metric graph $G = (V, E)$ with latency constraints $r(v)$ for $v \in V$, Algorithm 1 constructs R walks $\mathcal{W} = \{W_1, W_2, \ldots, W_R\}$ such that $L(\mathcal{W}, v) \leq r(v)$ for all $v \in V$ and $R \leq 4\alpha\lceil \log(\rho) \rceil R_{OPT}$, where R_{OPT} is the minimum number of robots required to satisfy the latency constraints and α is the approximation factor of MCCP.*

Proof. Given that $R_{\texttt{OPT}}$ robots will satisfy the latency constraints $r(v)$, they will also satisfy the relaxed constraints $\bar{r}(v)$ since $\bar{r}(v) > r(v)$. Therefore, there exists a set of at most $R_{\texttt{OPT}}$ walks $\mathcal{W}^*$ such that for $v \in V_i$, $L(\mathcal{W}^*, v) \leq r_{\texttt{min}}2^i$.

Using Lemma IV.2, given the set $\mathcal{W}^*$, $R_{\texttt{OPT}}$ cycles can be constructed in V_i such that the latency of each vertex in V_i is at most $r_{\texttt{min}}2^{i+1}$. Hence, running an α approximation algorithm for Minimum Cycle Cover Problem (MCCP) on the subgraph with vertices V_i and with maximum cycle length $r_{\texttt{min}}2^{i+1}$ will not return more than $\alpha R_{\texttt{OPT}}$ cycles.

Since MCCP returns cycles, equally placing k robots on each cycle will reduce the latency of each vertex on that cycle by a factor of k. As $r(v) \geq r_{\texttt{min}}2^{i+1}/4$ for each $v \in V_i$, we will need to place at most 4 robots on each cycle.

Finally, since there are at most $\lceil \log \rho \rceil$ partitions V_i, the algorithm will return $R \leq 4\alpha\lceil \log(\rho) \rceil R_{\texttt{OPT}}$ walks. □

Runtime: Since we run the approximation algorithm for MCCP on partitions of the graph, the runtime of Algorithm 1 is the same as that of the approximation algorithm of MCCP. That is because the runtime of MCCP is superlinear, so if $\sum |V_i| = |V|$, then $\sum |V_i|^p \leq |V|^p$ for $p \geq 1$.

V. Heuristic Algorithms

The approximation algorithm presented in Section IV is guaranteed to provide a solution within a fixed factor of the optimal solution. In this section, we propose a heuristic algorithm based on the orienteering problem, which in practice provides high-quality solutions.

A. Partitioned Solutions

In general, walks in a solution of the problem may share some vertices. However, sharing the vertices by multiple robots requires coordination and communication among the robots. Such strategies may also require the robots to hold at certain vertices for some time before traversing the next edge, in order to maintain synchronization. This is not possible for vehicles that must maintain forward motion, such as fixed-wing aircraft. The following example illustrates that if vertices are shared, lack of coordination or perturbation in edge weights can lead to large errors in latencies.

Example: Consider the problem instance shown in Figure 2. An optimal set of walks for this problem is given by $\{W_1, W_2, W_3\}$ where $W_1 = ((a, 1), (b, 1))$, $W_2 = ((b, 0), (c, 0))$ and $W_3 = ((c, 0), (d, 1), (c, 1))$. Note that

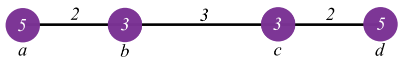


Fig. 2: A problem instance with an optimal set of walks that share vertices. The latency constraints for each vertex are written inside that vertex. The edge lengths are labeled with the edges. The optimal walks are $\{W_1, W_2, W_3\}$ where $W_1 = ((a,1),(b,1))$, $W_2 = ((b,0),(c,0))$ and $W_3 = ((c,0),(d,1),(c,1))$.

walk W_1 starts by staying on vertex a, while W_2 leaves vertex b and W_3 leaves vertex c. Also note that any partitioned solution will need 4 robots. Moreover, if the length of edge $\{b,c\}$ changes from 3 to $3-\epsilon$, (e.g., if the robot's speed increases slightly) the latencies of vertices b and c will keep changing with time and will go up to 5. Hence, a small deviation in robot speed can result in a large impact on the monitoring objective.

Since the above mentioned issues will not occur if the robots do not share the vertices of the graph, we focus on finding partitioned walks in this section. The general greedy approach used in this section is to find a single walk that satisfies latency constraints on a subset of vertices $V' \subseteq V$. Note that we do not know V' beforehand, but a feasible walk on a subset of vertices will determine V'. We then repeat this process of finding feasible walks on the remaining vertices of the graph until the whole graph is covered.

B. Greedy Algorithm

We now consider the problem of finding a single walk on the graph $G = (V, E)$ that satisfies the latency constraints on the vertices in $V' \subseteq V$. Given a robot walking on a graph, let $p(k)$ represent the vertex occupied by the robot after traversing k edges (after k steps) of the walk. Also, at step k, let the maximum time left until a vertex i has to be visited by the robot for its latency to be satisfied be represented by $s_i(k)$. If that vertex is not visited by the robot within that time, we say that the vertex expired. Hence, the vector $s(k) = [s_1(k), \dots, s_{|V'|}(k)]^T$ represents the time to expiry for each vertex. At the start of the walk, $s_i(0) = r(i)$, and $s_i(k)$ evolves according to the following equation:

$$s_i(k) = \begin{cases} r(i) & \text{if } p(k) = i \\ s_i(k-1) - l(p(k-1), p(k)) & \text{otherwise.} \end{cases} \tag{1}$$

We will use the notation s_i without the step k if it is clear that we are talking about the current time to expiry. An incomplete greedy heuristic for the decision version of the problem with $R = 1$ is presented in [12]. The heuristic is to pick the vertex with minimum value of $s_i(k)$ as the next vertex to be visited by the robot. This heuristic does not ensure that all the vertices on the walk will have their latency constraints satisfied since the distance to a vertex i to be visited might get larger than $s_i(k)$. To overcome this, we propose a modification to the heuristic to apply it to our problem. Given a walk W on graph G, the function PERIODICFEASIBILITY(W, G) determines whether the periodic walk $\Delta(W)$ is feasible on the vertices that are visited by W. This can be done simply in $O(|W|)$ by traversing the walk $[W, W]$ and checking if the time to expiry for any of the visited vertices becomes negative. Given this function, the simple greedy algorithm is to pick the vertex $i = \arg\min\{s_j\}$ subject to the constraint that PERIODICFEASIBILITY$([W, i], G)$ returns true, where W is the walk traversed so far. The algorithm terminates when all the vertices are either expired, or covered by the walk.

C. Orienteering Based Greedy Algorithm

Algorithm 2 also finds partitioned walks by finding a feasible walk on a subset of vertices and then considering the remaining subgraph. The idea is to visit more vertices on the way to the greedily picked vertex. From the current vertex x, the target vertex y is picked greedily as described in Section V-B. Then the time d is calculated in line 10 which is the maximum time to go from x to y for which the periodic walk remains feasible. In line 15, ORIENTEERING$(V - V_{\text{exp}}, x, y, d, \psi)$ finds a path in the vertices $V - V_{\text{exp}}$ from x to y of length at most d maximizing the sum of the weights ψ on the vertices of the path. The set V_{exp} represents the expired vertices whose latencies cannot be satisfied by the current walk, and they will be considered by the next robot. The vertices with less time to expiry are given more importance in the path by setting weight $\psi_i = 1/s_i$ for vertex i. The vertices that are already in the walk will remain feasible, and so their weight is discounted by a small number m to encourage the path to explore unvisited vertices.

Algorithm 2: ORIENTEERINGGREEDY

Input: Graph $G = (V, E)$, latency constraints $r(v), \forall v$
Output: A set of walks $\mathcal{W}$, such that $L(\mathcal{W}, v) \le r(v)$

```
1:  $j = 1$, $\mathcal{W} = \{\}$
2:  while $V$ is not empty do
3:      $V_{\text{exp}} = \{\}$
4:      $s_i = r(i)$ for all $i \in V$
5:      $W_j$ = pick vertex $a$ randomly from $V$
6:      while $V - V(W_j) - V_{\text{exp}}$ is not empty do
7:          $x$ = last vertex in $W_j$
8:          for $y \in V - V_{\text{exp}}$ in increasing order of $s$ do
9:              if PERIODICFEASIBILITY$([W_j, y], G)$ then
10:                 Use binary search between $l(x,y)$ and $s_y$
                    to get $d$ (time to go from $x$ to $y$) such that
                    $[W_j, y]$ remains feasible
11:                 for $z$ in $V - (V_{\text{exp}} \cup V(W_j))$ do
12:                     if $s_z < d + l(y,a)$ then $V_{\text{exp}} = V_{\text{exp}} \cup z$
13:                 $\psi_i = 1/s_i$ for all non expired vertices $i$
14:                 $\psi_i = m\psi_i$ for $i$ in $V(W_j)$
15:                 $W_j = [W_j$,ORIENTEERING$(V - V_{\text{exp}}, x, y, d, \psi)]$
16:                 Update $s$ using Equation (1)
17:             else
18:                 $V_{\text{exp}} = V_{\text{exp}} \cup y$
19:     $\mathcal{W} = \{\mathcal{W}, W_j\}$, $j = j + 1$
20:     $V = V - V(W_j)$
```

Lemma V.1. *Algorithm 2 returns a feasible solution, i.e., for the set of walks $\mathcal{W}$ returned by Algorithm 2, $L(\mathcal{W}, v) \le r(v)$, for all $v \in V$.*

The proof [21] is omitted due to space considerations. An approximation algorithm for ORIENTEERING can be used in

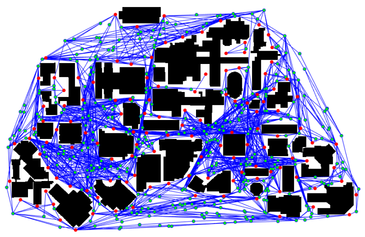


Fig. 3: The environment used for generation of random instances. The red dots represent the vertices to be monitored and the green dots represent the vertices in the PRM used to find shortest paths between red vertices.

line 15 of Algorithm 2. In our implementation, we used an ILP formulation to solve ORIENTEERING. To improve the runtime in practice, we pre-process the graph before calling the ORIENTEERING solver to consider only the vertices z such that $l(x,z)+l(z,y) \leq d$. We show in the next section that although the runtime of Algorithm 2 is more than that of Algorithm 1, it can still solve instances with up to 90 vertices in a reasonable time, and it finds better solutions.

VI. Simulation Results

We now present the performance of the algorithms presented in the paper. For Algorithm 1, we used the LKH implementation [22] to find the TSP of the graphs instead of the Christofides approximation algorithm [23]. This results in the loss of approximation guarantee but gives better results in practice. The orienteering paths in Algorithm 2 were found using the ILP formulation from [24] and the ILP's were solved using the Gurobi solver [25].

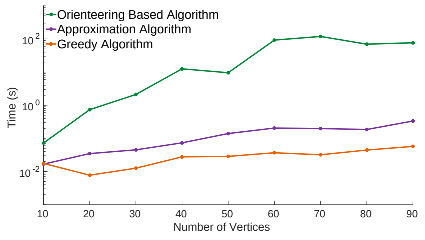
A. Patrolling an Environment

The graphs for the problem instances were generated randomly in a real world environment. The scenario represents a ground robot monitoring the University of Waterloo campus. Vertices around the campus buildings represent the locations to be monitored and a complete weighted graph was created by generating a probabilistic road-map to find paths between those vertices. Figure 3 shows the patrolling environment. To generate random problem instances of different sizes, n random vertices were chosen from the original graph. The latency constraints were generated uniformly randomly between TSP$/k$ and kTSP where k was chosen randomly between 4 and 8 for each instance. Here TSP represents the TSP length of the graph found using LKH.

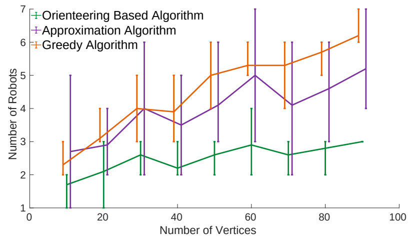
For each graph size, 10 random instances were created. The average run times of the algorithms are presented in Figure 4a. As expected, Algorithm 2 is considerably slower than the approximation and simple greedy algorithms due to multiple calls to the ILP solver. However, as shown in Figure 4b, Algorithm 2 also gives the minimum number of robots for most of these instances.

B. Persistent 3D Scene Reconstruction

Another application of our algorithms is in capturing images for 3D reconstruction of a scene. Since existing



(a)



(b)

Fig. 4: Average run times of the algorithms (a), and number of robots returned by each algorithm (b). The line plot shows the mean over 10 random instances for each graph size. The error bars in (b) show the minimum and maximum number of robots required for a graph size.

algorithms focus on computing robot paths to map a static scene [5], [4], our algorithms could be applied to persistently monitor and thus maintain an up-to-date reconstruction of a scene that changes over time. To demonstrate this, we create problem instances using a method similar to [4]. The viewpoints were generated on a grid around the building to be monitored. For each viewpoint, five camera angles were randomly generated, and best angle was selected for each viewpoint based on a view score that was calculated assuming a square footprint for the camera. For each camera angle, equally placed rays were projected onto the building within the footprint and a score was calculated based on the distance and incidence angle of the ray. This calculation is similar to that in [4], although they used a more detailed hemisphere coverage model.

After selecting the viewing angles, the final score of a camera pose was evaluated as in [4] by greedily picking the best viewpoint first and evaluating the marginal score of other viewpoints. The resulting graph had 109 vertices. The latencies were set such that the most informative viewpoint is visited every 8 minutes and on average each viewpoint is visited every 50 minutes. Algorithm 2 found a solution in 150 seconds using two walks, as shown in Figure 5. Note that Algorithm 1 returned a solution with 3 robots.

C. Comparison with Existing Algorithms in Literature

In [26], [13] the authors propose an SMT (Satisfiability Modulo Theory) based approach using Z3 solver [27] to solve the decision version of the problem. The idea is to fix an upper bound on the period of the solution and model the problem as a constraint program. The authors also provide benchmark instances for the decision version of the problem. We tested our algorithms on those benchmark instances and compare the results to the SMT based solver [13].

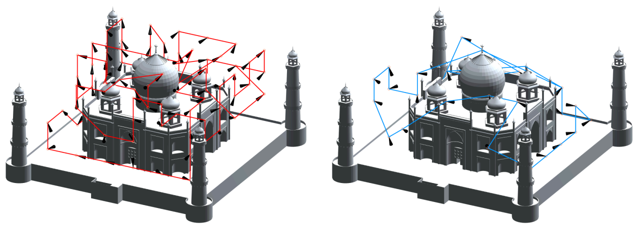


Fig. 5: The walks returned by Algorithm 2 to continually monitor the mausoleum of the Taj Mahal. The cones at each viewpoint show the camera angle. Note that the walk on the left is not a tour, as it visits the vertex with least latency twice within a period. The walk on the right is a tour and it visits the vertices that the first robot was unable to cover.

Out of 300 benchmark instances, given a time limit of 10 minutes, the Z3 solver returned 182 instances as satisfiable with the given number of robots. We ran our algorithms for each instance and checked if the number of robots returned are less than or equal to the number of robots in the instance. The approximation algorithm satisfied 170 instances whereas Algorithm 2 satisfied 178 instances. The four satisfiable instances that Algorithm 2 was unable to satisfy had optimal solutions where the walks share the vertices, and Algorithm 2 returned one more robot than the optimal in all those instances. The drawback of the constraint program is the scalability. It spent an average of 3.76 seconds on satisfiable instances whereas Algorithm 2 spent 3 ms on those instances on average. Moreover, on one such instance where Algorithm 2 returned one more robot than the Z3 solver, Z3 solver spent 194 seconds as compared to $\sim$ 5 ms for Algorithm 2. Note that these differences are for benchmark instances having up to 7 vertices. As shown above, Algorithm 2 takes $\sim$ 100 seconds for 90 vertex instances whereas we were unable to solve instances with even 15 vertices within an hour using the Z3 solver.

VII. Conclusion and Future Work

We presented and analyzed an approximation and a heuristic algorithm for the problem of finding the minimum number of robots that can satisfy the latency constraints for the vertices in a graph. We demonstrated the performance of the algorithms through simulations. Finding the relation between the partitioned optimal solution and a general optimal solution is an interesting direction for future work.

References

[1] G. Cabrita, P. Sousa, L. Marques, and A. De Almeida, "Infrastructure monitoring with multi-robot teams," in *Int. Conf. on Intelligent Robots and Systems*, 2010, pp. 18–22.

[2] N. Basilico, N. Gatti, and F. Amigoni, "Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder," *Artificial Intelligence*, vol. 184, pp. 78–123, 2012.

[3] A. B. Asghar and S. L. Smith, "Stochastic patrolling in adversarial settings," in *IEEE American Control Conf.*, 2016, pp. 6435–6440.

[4] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3D scanning," in *Int. Conf. on Computer Vision*, 2017, pp. 5334–5343.

[5] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics," in *IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 6423–6430.

[6] N. Cao, K. H. Low, and J. M. Dolan, "Multi-robot informative path planning for active sensing of environmental phenomena: A tale of two algorithms," in *Int. Conf. on Autonomous Agents and Multi-Agent Systems*, 2013, pp. 7–14.

[7] N. Nigam, S. Bieniawski, I. Kroo, and J. Vian, "Control of multiple uavs for persistent surveillance: algorithm and flight test results," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 5, pp. 1236–1251, 2012.

[8] Y. Elmaliach, N. Agmon, and G. A. Kaminka, "Multi-robot area patrol under frequency constraints," *Annals of Mathematics and Artificial Intelligence*, vol. 57, no. 3-4, pp. 293–320, 2009.

[9] J. M. Palacios-Gasós, E. Montijano, C. Sagues, and S. Llorente, "Multi-robot persistent coverage using branch and bound," in *IEEE American Control Conf.*, 2016, pp. 5697–5702.

[10] X. Yu, S. B. Andersson, N. Zhou, and C. G. Cassandras, "Optimal visiting schedule search for persistent monitoring of a finite set of targets," in *IEEE American Control Conf.*, 2018, pp. 4032–4037.

[11] ——, "Optimal dwell times for persistent monitoring of a finite set of targets," in *American Control Conf.*, 2017, pp. 5544–5549.

[12] J. Las Fargeas, B. Hyun, P. Kabamba, and A. Girard, "Persistent visitation under revisit constraints," in *IEEE Int. Conf. on Unmanned Aircraft Systems*, 2013, pp. 952–957.

[13] N. Drucker, M. Penn, and O. Strichman, "Cyclic routing of unmanned aerial vehicles," in *Int. Conf. on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer, 2016, pp. 125–141.

[14] H.-M. Ho and J. Ouaknine, "The cyclic-routing uav problem is pspace-complete," in *Int. Conf. on Foundations of Software Science and Computation Structures*. Springer, 2015, pp. 328–342.

[15] 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.

[16] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part I: Route construction and local search algorithms," *Transportation Science*, vol. 39, no. 1, pp. 104–118, 2005.

[17] J. N. Tsitsiklis, "Special cases of traveling salesman and repairman problems with time windows," *Networks*, vol. 22, no. 3, pp. 263–282, 1992.

[18] W. Yu and Z. Liu, "Improved approximation algorithms for some min-max and minimum cycle cover problems," *Theoretical Computer Science*, vol. 654, pp. 45–58, 2016.

[19] C. Chekuri, N. Korula, and M. Pál, "Improved algorithms for orienteering and related problems," *ACM Transactions on Algorithms*, vol. 8, no. 3, p. 23, 2012.

[20] Y. Chevaleyre, "Theoretical analysis of the multi-agent patrolling problem," in *IEEE Int. Conf. on Intelligent Agent Technology*, 2004, pp. 302–308.

[21] A. B. Asghar, S. L. Smith, and S. Sundaram, "Multi-robot routing for persistent monitoring with latency constraints," *arXiv preprint arXiv:1903.06105*, 2019.

[22] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[23] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.

[24] A. N. Letchford, S. D. Nasiri, and D. O. Theis, "Compact formulations of the steiner traveling salesman problem and related problems," *European Journal of Operational Research*, vol. 228, no. 1, pp. 83–92, 2013.

[25] Gurobi, "Gurobi optimizer," 2018. [Online]. Available: http://www.gurobi.com

[26] N. Drucker, "Cyclic routing of unmanned aerial vehicles," Master's thesis, Technion – Israel Institute of Technology, Israel, 2014.

[27] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.