Approximation Algorithms for Multi-Robot Patrol-Scheduling with Min-Max Latency

Peyman Afshani[1], Mark de Berg[2], Kevin Buchin[2], Jie Gao[3], Maarten Löffler[4], Amir Nayyeri[5], Benjamin Raichel[6], Rik Sarkar[7], Haotian Wang[8], and Hao-Tsung Yang[8]

[1] Department of Computer Science, Aarhus University, Denmark peyman@cs.au.dk
[2] Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands {M.T.d.Berg, k.a.buchin}@tue.nl
[3] Department of Computer Science; Rutgers University; New Brunswick, NJ 08901, USA jg1555@rutgers.edu
[4] Department of Information and Computing Sciences, Utrecht University, the Netherlands m.loffler@uu.nl
[5] School of Electrical Engineering and Computer Science, Oregon State University, OR 97330, USA nayyeria@eecs.oregonstate.edu
[6] Department of Computer Science; University of Texas at Dallas; Richardson, TX 75080, USA benjamin.raichel@utdallas.edu
[7] School of Informatics, University of Edinburgh, Edinburgh, U.K. rsarkar@inf.ed.ac.uk
[8] Department of Computer Science; Stony Brook University; Stony Brook, NY 11720, USA {haotwang, haotyang}@cs.stonybrook.edu

Abstract. We consider the problem of finding patrol schedules for k robots to visit a given set of n sites in a metric space. Each robot has the same maximum speed and the goal is to minimize the weighted maximum latency of any site, where the latency of a site is defined as the maximum time duration between consecutive visits of that site. The problem is NP-hard, as it has the traveling salesman problem as a special case (when $k = 1$ and all sites have the same weight). We present a polynomial-time algorithm with an approximation factor of $O(k^2 \log \frac{w_{\max}}{w_{\min}})$ to the optimal solution, where $w_{\max}$ and $w_{\min}$ are the maximum and minimum weight of the sites respectively. Further, we consider the special case where the sites are in 1D. When all sites have the same weight, we present a polynomial-time algorithm to solve the problem exactly. If the sites may have different weights, we present a 12-approximate solution, which runs in polynomial time when the number of robots, k, is a constant.

Keywords: Approximation, Motion Planning, Scheduling

1 Introduction

Monitoring a given set of locations over a long period of time has many applications, ranging from infrastructure inspection and data collection to surveillance for public or private safety. Technological advances have opened up the possibility to perform these tasks using autonomous robots. To deploy the robots in the most efficient manner is not easy, however, and gives rise to interesting algorithmic challenges. This is especially true when multiple robots work together in a team to perform the task.

We study the problem of finding a *patrol schedule* for a collection of k robots that together monitor a given set of n sites in a metric space, where k is a fixed parameter. Each robot has the same maximum speed—from now on assumed to be *unit speed*—and each site has a weight. The goal is to minimize the maximum weighted latency of any site. Here the *latency* of a site is defined as the maximum time duration between consecutive visits of that site (multiplied by its weight). A patrol schedule specifies for each robot its starting position and an infinitely long schedule describes how the robot moves over time from site to site.

Related Work. If $k = 1$ and all sites have the same weight, the problem reduces to the Traveling Salesman Problem (TSP) because then the optimal patrol schedule is to have the robot repeatedly traverse an optimal TSP tour. Since TSP is NP-hard even in Euclidean space [25], this means our problem is NP-hard for sites in Euclidean space as well. There are efficient approximation algorithms for TSP, namely, a (3/2)-approximation for metric TSP [9] and a polynomial-time approximation scheme (PTAS) for Euclidean TSP [5, 24], which carry over to the patrolling problem for the case where $k = 1$ and all sites are of the same weight.

Alamdari *et al.* [3] considered the problem with one robot (i.e., $k = 1$) and sites of possibly different weights. It can then be profitable to deviate from a TSP tour by visiting heavy-weight sites more often than low-weight sites. Alamdari *et al.* provided algorithms for general graphs with either $O(\log n)$ or $O(\log \varrho)$ approximation ratio, where n is the number of sites and ϱ is the ratio of the maximum and the minimum weight.

For $k > 1$ and even for sites of uniform weights, the problem is significantly harder than for a single robot, since it requires careful coordination of the schedules of the individual robots. The problem for $k > 1$ has been studied in the robotics literature under various names, including continuous sweep coverage, patrolling, persistent surveillance, and persistent monitoring [15, 18, 31, 23, 27, 28]. The dual problem has been studied by Asghar *et al.* [6] and Drucker *et al.* [12], where each site has a latency constraint and the objective is to minimize the number of robots to satisfy the constraint among all sites. They provide a $O(\log \rho)$-approximation algorithm where ρ is the ratio of the maximum and the minimum latency constraints. When the objective is to minimize the latency, despite all the works for practical settings, we are not aware of any papers that provide worst-case analysis. There are, however, several closely related problems that have been studied from a theoretical perspective.

The general family of *vehicle routing problems* (VRP) [11] asks for k tours, for a given k, that start from a given depot O such that all customers' requirements and operational constraints are satisfied and the global transportation cost is minimized. There are many different formulations of the problem, such as time window constraints in pickup and delivery, variation in travel time and vehicle load, or penalties for low quality services; see the monographs by Golden *et al.* [17] or Tóth and Vigo [29] for surveys.

In particular, the *k-path cover* problem aims to find a collection of k paths that cover the vertex set of the given graph such that the maximum length of the paths is minimized. It has a 4-approximation algorithm [4]. The *min-max tree cover* problem is to cover all the sites with k trees such that the maximum length of the trees is minimized. Arkin *et al.* [4] proposed a 4-approximation algorithm for this problem, which was improved to a 3-approximation by Kahni and Salavatipour [22] and to a (8/3)-approximation by Xu *et al.* [30]. The *k-cycle cover* problem asks for k cycles (instead of paths or trees) to cover all sites. For minimizing the maximum cycle length, there is an algorithm with an approximation factor of 16/3 [30]. For minimizing the sum of all cycle lengths, there is a 2-approximation for the metric setting and a PTAS in the Euclidean setting [20, 21]. Note that all problems above ask for tours visiting each site once (or at most once), while our patrolling problem asks for schedules where each site is visited infinitely often.

When the patrol tours are given (and the robots may have different speeds), the scheduling problem is termed the *Fence Patrolling Problem* introduced by Czyzowicz et al. [10]. Given a closed or open fence (a rectifiable Jordan curve) of length ℓ and k robots of maximum speed $v_1, v_2, \ldots, v_k > 0$ respectively, the goal is to find a patrolling schedule that minimizes the maximum latency L of any point on the fence. Notice that our problem focuses on a discrete set of n sites while the fence patrolling problem focuses on visiting all points on a continuous curve. For an open fence (a line segment), a simple partition strategy is proposed, in which each robot moves back and forth in a segment whose length is proportional to its speed. The best solution using this strategy gives the optimal latency if all robots have the same speed and a 2-approximation of the optimal latency when robots have different maximum speeds. Later, the approximation ratio was improved to $\frac{48}{25}$ by Dumitrescu et al. [13] allowing the robots to stop. Finally, this ratio is improved to $\frac{3}{2}$ by Kawamura and Soejima [19] and the speeds of robots are varied in the patrolling process.

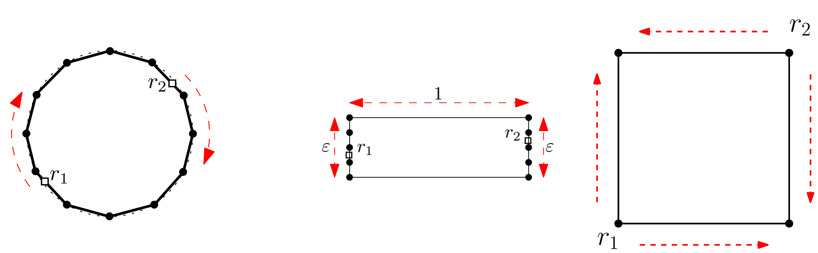

Fig. 1. Left: Two robots with n sites evenly placed on a unit circle. The optimal solution is to place two robots, maximum apart from each other, along the perimeter of a regular n-gon. Middle: Two robots with two clusters of vertices of distance 1 apart. The optimal solution is to have two robots each visiting a separate cluster. Right: A non-periodic optimal solution.

Challenges. For scheduling multiple robots, a number of new challenges arise. One is that already for $k = 2$ and all sites of weight 1 the optimal schedules may have very different structures. For example, if the sites form a regular n-gon for sufficiently large n, as in Figure 1 (left), an optimal solution would place the two robots at opposite points on the n-gon and let them traverse the n-gon at unit speed in the same direction. If there are two groups of sites that are far away from each other, as in Figure 1 (middle), it is better to assign each robot to a group and let it move along a TSP tour of that group. Figure 1 (middle) also shows that having more robots will not always result in a lower maximum latency. Indeed, adding a third robot in Figure 1 (middle) will not improve the result: during any unit time interval, one of the two groups is served by at most one robot, and then the maximum latency within that group equals the maximum latency that can already be achieved by two robots for the whole problem. The two strategies just mentioned—one cycle with all robots evenly placed on it, or a partitioning of the sites into k cycles, one cycle per robot exclusively—have been widely adopted in many practical settings [14, 26]. Chevaleyre [8] studied the performance of the two strategies but did not provide a bounded approximation ratio.

Note that the optimal solutions are not limited to the two strategies mentioned above. For example, for three robots it might be best to partition the sites into two groups and assign two robots to one group and one robot to the other group. There may even be completely unstructured solutions, that are not even periodic. See Figure 1 (right) for an example. There are four sites at the vertices of a square with two robots that initially stay on two opposite corners. r_1 will choose randomly between the horizontal or vertical direction. Correspondingly, robot r_2 always moves in the opposite direction of r_1. In this way, all sites have maximum latency 2 which is optimal. This solution is not described by cycles for the robots, and is not even periodic. Observe that for a single robot, slowing down or temporarily stopping never helps to reduce latency. But for multiple robots, it is not easy to argue that there is an optimal solution in which robots never slow down or stop.

When sites have different weights, intuitively the robots have to visit sites with high weights more frequently than others. Thus, coordination among multiple robots becomes even more complex.

Our results We present a number of exact and approximation algorithms which all run in polynomial time. In Section 3 we consider the weighted version in the general metric setting and presented an algorithm with approximation factor of $O(k^2 \log \frac{w_{\max}}{w_{\min}})$, where $w_{\max}$ and $w_{\min}$ are the maximum weight and minimum weight respectively. The main insight is to obtain a good assignment of the sites to the k robots. We first round up all the weights to powers of two, which only introduces a performance loss by a factor of two. The number of different weights is in the order of $O(\log \frac{w_{\max}}{w_{\min}})$. Given a target maximum weighted latency L, we obtain the t-min-max tree cover for each set of sites of the same weight w, for the smallest possible value $t \leq k$ such that the max tree weight in the tree cover is no greater than $O(L/w)$. Then we assign the sites to the k robots sequentially by decreasing weights. Each robot is assigned a depot tree with one of the vertices as the depot vertex. The subset of vertices of a new tree are allocated to existing depots/robots if they are sufficiently nearby; and if otherwise, allocated to a 'free' robot. We show that if we fail in any of the operations above (e.g., trees in a k-min-max tree cover are too large or we run out of free robots), L is too small. We double L and try again. We prove that the algorithm succeeds as soon as $L \geq L^*$, where L^* is the optimal weighted latency. At that point we can start to design the patrol schedules for the k robots, by using the algorithm in [3].

In Section 4 we consider the special case where all the sites are points in $\mathbb{R}^1$. When the sites have uniform weights, there is always an optimal solution consisting of k disjoint zigzag schedules (a zigzag schedule is a schedule where a robot travels back and forth along a single fixed interval in $\mathbb{R}^1$), one per robot. Such an optimal solution can be computed in polynomial time by dynamic programming.

When these sites are assigned different weights and the goal is to minimize the maximum weighted latency, we show that there may not be an optimal solution that consists of only disjoint zigzags. Cooperation between robots becomes important. In order to get an approximate solution, we run a series of relaxations to our problem and turn it into the Dyadic Time Window Problem (DTW) and Dyadic Time Window Tour Problem (DTT), the solution to which are constant approximations to our patrol problem. Again we round the weights to powers of two. Different from the patrol problem, in the time-window problems, we chop the time axis into time windows of length inversely proportional to the weight of a site – the higher the weight, the smaller its window size – and require each site to be visited within its respective time windows. Since the window sizes are powers of two, these are called dyadic windows. By the fact that the sites stay in 1D, we can represent the motion plan for each robot within a proper time window by four parameters: the starting position, the ending position, the leftmost position and the rightmost position. This is enough to conclude which site has been visited within the time window. The fact that the sub-schedules can be represented by a small number of parameters allows us to find a schedule for k robots with a 12-approximation solution of the min-max weighted latency in $\mathbb{R}^1$. The running time is $O((n/w_{\min})^{O(k)})$, where the maximum weight is 1 and the minimum weight is $w_{\min}$.

2 Problem Definition

As stated in the introduction, our goal is to design a schedule for a set of k robots visiting a set of n sites in such a way that the maximum weighted latency at any of the sites is minimized. It is most intuitive to consider the sites as points in Euclidean space, and the robots as points moving in that space. However, our solutions will actually work in a more general metric space, as defined next. Let (P, d) be a metric space on a set P of n sites, where the distance between two sites $s_i, s_j \in P$ is denoted by $d(s_i, s_j)$. Consider the undirected complete graph $G = (P, P \times P)$. We view each edge $(s_i, s_j) \in P \times P$ as an interval of length $d(s_i, s_j)$—so each edge becomes a continuous 1-dimensional space in which the robot can travel—and we define $C(P, d)$ as the continuous metric space obtained in this manner. From now on, and with a slight abuse of terminology, when we talk about the metric space (P, d) we refer to the continuous metric space $C(P, d)$.

Let $R := \{r_1, \ldots, r_k\}$ be a collection of robots moving in a continuous metric space $C(P, d)$. We assume without loss of generality that the maximum speed of the robots is 1. A *schedule* for a robot r_j is a continuous function $f_j : \mathbb{R}^{\geq 0} \to C(P, d)$, where $f_j(t)$ specifies the position of r_j at time t. A schedule must obey the speed constraint, that is, we require $d(f_j(t_1), f_j(t_2)) \leq |t_1 - t_2|$ for all t_1, t_2. A *schedule for the collection R of robots*, denoted $\sigma(R)$, is a collection of schedules f_j, one for each robot in $r_j \in R$. (We allow robots to be at the same location at the same time.) We call the schedule of a robot r_j *periodic* if there exists an offset $t_j^* \geq 0$ and period length $\tau_j > 0$ such that for any integer $i \geq 0$ and any $0 \leq t < \tau_j$ we have $f_j(t_j^* + i\tau_j + t) = f_j(t_j^* + (i+1)\tau_j + t)$. A schedule $\sigma(R)$ is periodic if there are $t_R^* \geq 0$ and $\tau_R > 0$ such that for any integer $i > 0$ and any $0 \leq t < \tau_R$ we have $f_j(t_R^* + i\tau_R + t) = f_j(t_R^* + (i+1)\tau_R + t)$ for all robots $r_j \in R$. It is not hard to see that in the case that all period lengths are rational, $\sigma(R)$ is periodic if and only if the schedules of all robots are periodic.

We say that a site $s_i \in P$ is *visited* at time t if $f_j(t) = s_i$ for some robot r_j. Given a schedule $\sigma(R)$, the *latency* L_i of a site s_i is the maximum time duration during which s_i is not visited by any robot. More formally,

$$L_i = \sup_{0 \leq t_1 < t_2} \{|t_2 - t_1| : s_i \text{ is not visited during the time interval } (t_1, t_2)\}$$

We only consider schedules where the latency of each site is finite. Clearly such schedules exists: if T_{opt} denotes the length of an optimal TSP tour for the given set of sites, then we can always get a schedule where $L_i = T_{\text{opt}}/k$ by letting the robots traverse the tour at unit speed at equal distance from each other. Given a metric space (P, d) and a collection R of k robots, the *(multi-robot) patrol-scheduling problem* is to find a schedule $\sigma(R)$ minimizing the *weighted latency* $L := \max_i w_i L_i$, where site i has weight w_i and maximum latency L_i.

Note that it never helps to move at less than the maximum speed between sites—a robot may just as well move at maximum speed and then wait for some time at the next site. Similarly, it does not help to have a robot start at time $t = 0$ "in the middle" of an edge. Hence, we assume without loss of generality that each robot starts at a site and that at any time each robot is either moving at maximum speed between two sites or it is waiting at a site.

3 Approximation Algorithms in a General Metric

For sites with weights in a general metric space (P, d), we design an algorithm with approximation factor $O(k^2 m)$ for minimizing the max weighted latency of all sites by using k robots of maximum speed of 1, where $m = \log \frac{w_{\max}}{w_{\min}}$. Without loss of generality, we assume that the maximum weight among sites is 1. We first round the weight of each site to the least dyadic value and solve the problem with dyadic weights. That is, if node i has weight w_i, we take $w'_i = \sup\{2^x | x \in \mathbb{Z} \text{ and } 2^x \geq w_i\}$. Clearly, $w_i \leq w'_i < 2w_i$. This will only introduce another factor of 2 in the approximation factor on the maximum weighted latency. In the following we just assume the weights are dyadic values. Suppose the smallest weight of all sites is $1/2^m$. Denote by W_j the collection of sites of weight $1/2^j$. W_j could be empty. Let $\mathcal{W}$ denote the collection of all non-empty sets W_j, $0 \leq j \leq m$. Note that $|\mathcal{W}| \leq m+1 = \log \frac{w_{\max}}{w_{\min}} + 1$. We assume we have a β-approximation algorithm $\mathcal{A}$ available for the min-max tree cover problem. The currently best-known approximation algorithm has $\beta = 8/3$ [30].

The intuition of our algorithm is as follows. We first guess an upper bound L on the optimal maximum weighted latency and run our algorithm with parameter L. If our algorithm successfully computes a schedule, its maximum weighted latency is no greater than $\beta k^2 m L$. If our algorithm fails, we double the value of L and run again. We prove that if our algorithm fails, the optimal maximum weighted latency must be at least L. Thus, when we successfully find a schedule, its maximum weighted latency is an $O(k^2 m)$ approximation to the optimal solution. The following two procedures together provide what is needed.

- Algorithm k-ROBOT ASSIGNMENT($\mathcal{W}, L$), returns FALSE when there does not exist a schedule with max weighted latency $\leq L$, or, returns k groups: $\mathcal{T}(r_1), \mathcal{T}(r_2), \cdots \mathcal{T}(r_k)$, where $\mathcal{T}(r_i)$ includes a set of trees that are assigned to robot r_i. Every site belongs to one of the trees and no site belongs to two trees in the union of the groups. For robot r_i, one of the trees in $\mathcal{T}(r_i)$ is called a depot tree $T_{\text{dep}}(r_i)$ and one vertex with the highest weight on the depot tree is a *depot* for r_i, denoted by $x_{\text{dep}}(r_i)$.
- With the trees $\mathcal{T}(r_i)$ assigned to one robot r_i, Algorithm SINGLE ROBOT SCHEDULE($\mathcal{T}(r_i)$) returns a single-robot schedule such that every site covered by $\mathcal{T}(r_i)$ has maximum weighted latency $O(k^2 m \cdot L)$.

Denote by $V(T)$ the set of vertices of a tree T and by $d(s_i, s_j)$ the distance between two sites s_i and s_j. See the pseudo code of the two algorithms.

The following observation is useful for our analysis later.

Lemma 1. *In k-ROBOT ASSIGNMENT($\mathcal{W}, L$), the depots s_i and s_j, with $w_i \geq w_j$, for different robots have distance more than kL/w_i.*

Proof. The depot vertices, in the order of their creation, have non-increasing weight. Thus, we could assume without loss of generality that s_j is the depot that is created later than s_i. s_j is more than kL/w_i away from the depot s_i. □

Lemma 2. *Let $s_0, \cdots, s_k$ be $k+1$ depot sites, ordered such that $w_0 \geq \cdots \geq w_k$, defined as in Algorithm k-ROBOT ASSIGNMENT($\mathcal{W}, L$). The optimal schedule minimizing the maximum weighted latency for k robots to serve $\{s_0, \cdots, s_k\}$ has weighted latency $L^* \geq 2L$.*

```
procedure k-ROBOT ASSIGNMENT($\mathcal{W}, L$)
    for every set $W_j \in \mathcal{W}$
        for $t \leftarrow 1$ to $k$
            Run algorithm $\mathcal{A}$ to obtain a $t$-min-max tree cover $\mathcal{C}_t^j$ on $W_j$.
        $q_j \leftarrow$ smallest integer $t$ s.t. the max weight of trees in $\mathcal{C}_t^j$ is $< \beta \cdot 2^j L$
        If there is no such $q_j$ then return FALSE
        $\mathcal{T}(W_j) \leftarrow \mathcal{C}_{q_j}^j$
    Set all robots as "free" robots, i.e., not assigned a depot tree.
    for $j \leftarrow 0$ to $m$                                   ▷ Assign trees to robots
        for every tree $T$ in $\mathcal{T}(W_j)$
            $Q \leftarrow V(T)$
            for every non-free robot $r$
                Let $j'$ be such that $x_{\text{dep}}(r) \in W_{j'}$
                $Q' \leftarrow \{v | v \in Q, d(v, x_{\text{dep}}(r)) \leq k2^{j'} L\}$
                Compute MST($Q'$) and assign it to robot $r$.
                $Q \leftarrow Q \setminus Q'$
            if $Q \neq \emptyset$
                if no free robot
                    Return FALSE.
                else
                    Pick a free robot $r$ and set $T_{\text{dep}}(r) \leftarrow MST(Q)$
                    Pick an arbitrary vertex $x$ in $T_{\text{dep}}(r)$ and set $x_{\text{dep}}(r) \leftarrow x$
    For each robot $r_i$, let $\mathcal{T}(r_i)$ be the collection of trees assigned to $r_i$,
    including its depot tree, and return the collections $\mathcal{T}(r_1), \ldots, \mathcal{T}(r_k)$.
```

Proof. Let speed(r, t) denote the speed of a robot r at time t. Let S be a schedule of latency L^*. The proof proceeds in k rounds. The goal of the p-th round is to change the schedule into a new schedule that has a stationary robot at site s_{p-1}. To keep the latency at L^*, we will increase the speed of some other robots. We will show the following claim.

Claim. After the p-th round we have a schedule of latency L^* such that

1. there is a stationary robot at each of the sites s_i with $i < p$,
2. at any time t we have $\sum_r \text{speed}(r, t) \leq k$, where the sum is overall k robots.

This claim implies that after the $(k-1)$-th round we have a schedule of latency L^* with stationary robots at $s_0, s_1, \cdots, s_{k-2}$, and one robot of maximum speed k serving the sites s_{k-1} and s_k. The distance between these sites is at least kL/w_{k-1}, so the latency L^* of our modified schedule satisfies $L^* \geq 2kL/k = 2L$. This is what is needed in the Lemma.

The proof of the claim is by induction. Suppose the claim holds after the $(p-1)$-th round. Thus we have a stationary robot at each of the sites $s_0, \cdots, s_{p-2}$, and at any time t we have $\sum_r \text{speed}(r, t) \leq k$. Note that for $p = 1$, the required conditions are indeed satisfied. Now consider the site s_{p-1}.

Define $\ell_0, \ell_1, \cdots$ to be the moments in time where there is at least one robot at s_{p-1} and all robots present at s_{p-1} are leaving. In other words, $\ell_0, \ell_1, \cdots$ are the times at which s_{p-1} is about to become unoccupied. If no such time exists then there is always a robot at s_{p-1}, and so we are done. Let $a_1, a_2, \cdots$ be the moments in time where a robot arrives at s_{p-1} while no other robot was present at s_{p-1} just before that time, that is, s_{p-1} becomes occupied. Assuming without loss of generality that $\ell_0 < a_1$, we have

$$\ell_0 \leq a_1 \leq \ell_1 \leq \cdots .$$

Consider an interval (ℓ_i, a_{i+1}). By definition $a_{i+1} - \ell_i \leq L^*/w_{p-1}$. Let r be a robot leaving s_{p-1} at time ℓ_i and suppose r is at position z at time a_{i+1}. Let r' be a robot arriving at s_{p-1} at time a_i. We modify the schedule such that r stays stationary at s_{p-1}, while r' travels to z via s_{p-1}. We increase the speed of r' by adding the speed of r to it, that is, for any $t \in (\ell_i, a_{i+1})$ we change the speed of r' at time t to $\text{speed}(r', t) + \text{speed}(r, t)$. Since r is now stationary at s_{p-1}, this does not increase the sum of the robot speeds. Moreover, with this new speed, r' will reach z at time a_{i+1}. Finally, observe that this modification does not increase the latency. Indeed, the sites $s_0, \cdots, s_{p-2}$ have a stationary robot by the induction hypothesis, and all sites $s_p, \cdots, s_k$ are at distance at least kL/w_{p-1} from s_{p-1} so during (ℓ_i, a_{i+1}) the robots r and r' did not visit any of these sites in the unmodified schedule. □

Lemma 3. *Given L, if k-ROBOT SCHEDULE$(\mathcal{W}, L)$ returns* FALSE *then $L^* \geq L$, where L^* is the optimal maximum weighted latency.*

Proof. There are two cases of the algorithm returning FALSE. We discuss them separately.

In the first case, there is a value j such that the maximum tree weight of a β-approximation of the t-min-max tree cover is larger than $\beta 2^{j-1}L$ for all $1 \leq t \leq k$ (Line 7). It implies that the optimal value λ of k-min-max tree cover is larger than $2^{j-1}L$ for sites in W_j. Since the k-robot solution also cover all the sites in W_j, $\lambda/2^{j-1}$ is also a lower bound of the optimal latency (see [2] for details). Thus, $L^* \geq \lambda/2^{j-1} > 2^{j-1}L/2^{j-1} = L$.

In the second case, there is a tree with vertices that are far away from existing depots and there is no free robot anymore. Notice that there are precisely k depots at this moment. Suppose the depots are $s_0, s_1, \cdots s_{k-1}$ and there is another vertex s_k which is at distance at least kL/w_i from the depot s_i of weight w_i, for $0 \leq i \leq k-1$. Apply Lemma 2, the latency of the optimal schedule visiting only these k sites is at least $2L$, so is the optimal latency L^*. □

Lemma 4. *If k-ROBOT SCHEDULE$(\mathcal{W}, L)$ does not return* FALSE*, each robot is assigned at most $k(m+1)$ trees and a depot site such that*

- *one of the trees is the depot tree T_{dep} which includes a depot x_{dep}. x_{dep} has the highest weight among all sites assigned to this robot;*
- *all other vertices are within distance $kL/\bar{w}$ from the depot, where $\bar{w}$ is the weight of x_{dep};*

– *each tree T has vertices of the same weight w and the sum of tree edge length is at most $\beta L/w$.*

Proof. Most of the claims are straight-forward from the algorithm k-ROBOT SCHEDULE($\mathcal{W}, L$). A tree T assigned to a robot has vertices coming from the vertices of the same tree T' in the min-max tree cover (obtained on Line 4). Thus the vertices have the same weight (say w). These vertices are within distance $kL/\bar{w}$, from the depot x_{dep}, where $\bar{w}$ is the weight of x_{dep}, by Line 15. Further, the tree T is always taken as a minimum spanning tree on its vertices. Thus the sum of the edge length on T is no greater than that of the original tree T' (with potentially more vertices), which is no greater than $\beta L/w$, by Line 5.

It remains to prove that each robot r is assigned at most km trees. Note that the loop of line 9 in the algorithm has $m+1$ iterations and each loop of line 10 has at most k iterations. Moreover, in one iteration of lines 13 to 23 each robot r is assigned at most one tree: it may be assigned a tree in line 16 when it is already non-free, and in line 22 when it was still free. Hence, r is assigned at most $k(m+1)$ trees. □

Now we are ready to present the algorithm for finding the schedule for robot r_i to cover all vertices in the family of trees $\mathcal{T}(r_i)$, as the output of k-ROBOT SCHEDULE($\mathcal{W}, L$). We apply the algorithm in [16, 3] for the patrol problem with one robot, with the only one difference of handling the sites of small weights. The details are presented in the pseudo code SINGLE ROBOT SCHEDULE($\mathcal{T}$) which takes a set $\mathcal{T}$ of h trees. By Lemma 4, there are at most km trees assigned to one robot, i.e., $h \le km$. For a tree T (a path P) we use $|T|$ (resp. $|P|$) as the sum of the length of edges in T (resp. P).

procedure SINGLE-ROBOT-SCHEDULE($\mathcal{T} = \{T_0, T_1, \cdots, T_{h-1}\}$)
 ▷ T_0 is the depot tree and w_0 is the weight of the vertices in T_0. $h \le km$
 $\delta \leftarrow 2kL/w_0$.
 for $i \leftarrow 0$ to $h-1$
  Compute a tour D_i of length at most $2|T_i|$ on the vertices in T_i.
  Partition D_i into a collection $\mathcal{P}^i = \{P^i_0, P^i_1, \cdots\}$ of at most $\lceil 2|T_i|/\delta \rceil$ paths such that $|P^i_j| \le \delta$ for all j.
  $\text{idx}(i) \leftarrow 0$ ▷ $P^i_{\text{idx}(i)}$ is the path in $\mathcal{P}^i$ to be traversed next
 Put the robot on the first vertex of path P^0_0 and set $i \leftarrow 0$
 while TRUE
  Let the robot traverse path $P^i_{\text{idx}(i)}$
  $i' \leftarrow (i+1) \mod h$
  Let the robot move from the end of $P^i_{\text{idx}(i)}$ to the start of $P^{i'}_{\text{idx}(i')}$
  Set $\text{idx}(i) \leftarrow (\text{idx}(i)+1) \mod |\mathcal{P}_i|$ and set $i \leftarrow i'$

Lemma 5. *The* SINGLE ROBOT SCHEDULE*($\mathcal{T} = \{T_0, T_2, \cdots, T_{h-1}\}$), $h \le k(m+1)$, returns a schedule for one robot that covers all sites included in $\mathcal{T}$ such that the maximum weighted latency of the schedule is at most $O(k^2 m \cdot L)$.*

Proof. By Lemma 4 the distance between the depot and any other vertices on tree T_i is at most kL/w_0, where w_0 is the weight of the depot. By triangle inequality, the distance of any two sites (either on the same tree or on different trees) is at most $2kL/w_0 = \delta$. Consider any site s and assume $s \in P_j^i$ for some $P_j^i \in \mathcal{P}^i$. Let w_i be the weight of the vertices in T_i. Note that some path from $\mathcal{P}^i$ is visited once every h iterations of the while loop of line 9 to 13, and that the paths from $\mathcal{P}^i$ are visited in a round-robin fashion. Thus P_j^i (and, hence, site s) is visited once every $h \cdot |\,\mathcal{P}^i\,|$ iterations. In one iteration the robot moves over a distance at most δ in line 10, and over a distance at most δ in line 12. Hence, the total distance traveled by the robot before returning to s is bounded by $h \cdot |\,\mathcal{P}^i\,| \cdot 2\delta$, and so the total weighted latency is bounded by

$$w_i \cdot h \cdot |\,\mathcal{P}^i\,| \cdot 2\delta \leq w_i \cdot h \cdot \lceil 2|T_i|/\delta \rceil \cdot 2\delta$$

There are two cases. If $|T_i| > \delta$, the above term is at most $w_i \cdot |T_i| \cdot h \leq 2L \cdot h$. If $|T_i| \leq \delta$, the above term is at most $w_i \cdot h \cdot 2\delta \leq 2kL \cdot h$. Since $h \leq k(m+1)$, the weighted latency of s is $O(k^2 mL)$. □

To analyze the running time, we use the best known t-min-max tree cover algorithm [30] with running time $O(n^2 t^2 \log n + t^5 \log n)$. In Algorithm k-robot Assignment, from line 2 to line 8 it takes time in the order of $O(mn^2 \log n) \cdot (1^2 + 2^2 + \cdots k^2) = O(mn^2 k^3 \log n)$ (suppose $n \gg k$). From line 9 to line 24, we assign some subset of vertices Q' in each tree to occupied robots. The running time is $O(k(m+1) \cdot n \log n)$, where $O(n \log n)$ is the time to compute the minimum spanning tree for Q' (line 16). The total running time is $O(mn^2 \log n)$ for Algorithm k-robot Assignment. Algorithm Single Robot Schedule takes $O(n)$ time, since a robot is assigned at most n sites. Thus, given a value L, it takes $O(n^2 k^3 m \log n)$ to either generate patrol schedules for k robots with approximation factor $O(k^2 m)$ or confirm that there is no schedule with maximum weighted latency L.

To solve the optimization problem (i.e., finding the minimum L^*) if there are fewer than than k sites, we put one robot per site. Otherwise, we start with parameter L taking the distance between the closest pair of the n sites, and double L whenever the decision problem answers negatively. The number of iterations is bounded by $\log L^*$. Notice that L^* is bounded, e.g., at most $1/k$-th of the traveling salesman tour length.

Theorem 1. *The approximation algorithm for k-robot patrol scheduling for weighted sites in the general metric has running time $O(n^2 k^3 m \log n \log L^*)$ with a $O(k^2 m)$-approximation ratio, where $m = \log \frac{w_{\max}}{w_{\min}}$ with $w_{\max}$ and $w_{\min}$ being the maximum and minimum weight of the sites and L^* is the optimal maximum weighted latency.*

4 Sites in $\mathbb{R}^1$

In this section we consider the case where the sites are points in $\mathbb{R}^1$. For the case where the sites have uniform weights, we present an algorithm that computes an optimal solution. For the case of arbitrary weights, we design a 12-approximation algorithm. Before we do so, we give a simple observation about the case of a single robot. After that we turn our attention to the more interesting case of multiple robots.

We define the schedule of a robot in $\mathbb{R}^1$ to be a *zigzag schedule*, or *zigzag* for short, if the robot moves back and forth along an interval at maximum speed (and only turns at the endpoints of the interval).

Observation 1 *Let P be a collection of n sites in $\mathbb{R}^1$ with arbitrary weights. Then the zigzag schedule where a robot travels back and forth between the leftmost and the rightmost site in P is optimal for a single robot.*

Proof. Let $s_1, \ldots, s_n$ be the sites in P, ordered from left to right, and let w_i denote the weight of s_i. Then the weighted latency of s_i in the zigzag schedule is $w_i \cdot \max(2\ d(s_i, s_1), 2\ d(s_i, s_n))$. Let s_{i^*} be a site whose weighted latency is maximal, and assume without loss of generality that $d(s_{i^*}, s_1) \geq d(s_{i^*}, s_n)$. Clearly the minimum weighted latency of a robot that only has to visit s_1 and s_{i^*} is at most the minimum weighted latency of a robot that must visit all sites in P. The former is equal to $w_{i^*} \cdot 2\ d(s_{i^*}, s_1)$ because the robot must go back and forth between s_1 and s_{i^*}. Since the zigzag on P has latency $w_{i^*} \cdot 2\ d(s_{i^*}, s_1)$ as well, it must thus be optimal. □

4.1 An Optimal Solution for Uniform Weights

We just observed that for a single robot a zigzag schedule is always optimal. Next, we prove a similar result for multiple robots, as long as the sites have uniform weights. More precisely, we show there is an optimal schedule consisting of disjoint zigzags.

Theorem 2. *Let P be a set of n sites in $\mathbb{R}^1$, with uniform weights, and let k be the number of available robots, where $1 \leq k \leq n$. Then there exists an optimal schedule such that each robot follows a zigzag schedule and the intervals covered by these zigzag schedules are disjoint.*

Proof. Let $r_1, \ldots, r_k$ denote the available robots and assume that initially the robots are ordered from left to right with ties broken arbitrarily. Let $f_i(t)$ denote the position of robot r_i at time t. We may assume that this ordering does not change. That is, $f_1(t) \leq f_2(t) \leq \cdots \leq f_k(t)$ at any time t. Indeed, when two robots swap, we can switch their roles so that we keep the original order.

Let a_i and b_i be the leftmost and rightmost site ever visited by r_i, respectively, and define $I_i := [a_i, b_i]$. The order on the robots implies that $a_i \leq a_j$ for $i < j$. Now consider an optimal schedule with the above properties, where we assume without loss of generality that each robot is assigned a non-empty interval, which could be a single point. We will modify this schedule (if necessary) to obtain an optimal schedule consisting of disjoint zigzags. First we ensure that $a_i < a_j$ for all $i < j$. Suppose that $a_i = a_j$ for (one or more) $j > i$. Note that at any time t such that $f_j(t) = a_i$ for some $j > i$, we must also have $f_i(t) = a_i$. Hence, the visits of these robots r_j to a_i are not necessary, and we can modify their schedules so that their leftmost visited sites are the site immediately to the right of a_i. By doing this repeatedly we obtain a schedule such that $a_i < a_j$ for all $i < j$.

We now prove the following statement—note that this statement implies the lemma—by induction on j:

> There is an optimal schedule such that, for any $1 \leq j \leq k$, we have (i) the intervals $I_1, \dots, I_j$ are disjoint from each other and from the intervals $I_{j+1}, \dots, I_k$, and (ii) each of the robot r_i with $1 \leq i \leq j$ follows a zigzag on I_i.

First consider the case $j = 1$. Note that a_1 is the leftmost site in P and that r_1 is the only robot visiting a_1. Since r_1 also visits b_1, the latency of a_1 is at least $2(b_1 - a_1)$, which is achieved if we make r_1 follow a zigzag along I_1. This zigzag guarantees a latency $2(b_1 - a_1)$ for any site in I_1, so there is no need for another robot to visit those sites. Hence, we can ensure that the intervals $I_2, \dots, I_k$ are strictly to the right of I_1, and so the statement is true for $j = 1$.

Now consider the case $j > 1$. Because $a_j < a_i$ for all $i > j$ we know that a_j is not visited by any of the robots r_i with $i > j$. By the induction hypothesis a_j is not visited by any of the robots r_i with $i < j$ either. Hence, r_j is the only robot visiting a_i. Following the same reasoning as in the case $j = 1$ we can thus ensure that r_j follows a zigzag along I_j and that the intervals $I_{j+1}, \dots, I_k$ are disjoint from I_j. Together with the induction hypothesis this proves the statement for j, thus finishing the proof. □

With Theorem 2, the min-max latency problem reduces to the following: Given a set S of n numbers and a parameter k, compute the smallest L such that S can be covered by k intervals of length at most L. When S is stored in sorted order in an array, L can be computed in $O(k^2 \log^2 n)$ time [1, Theorem 14]. If S is not sorted, there is a $\Omega(n \log n)$ lower bound in the algebraic computation tree model [7], since for $k = n - 1$ element uniqueness reduces to this problem.

4.2 Sites of Arbitrary Weights

In general, if the sites have different weights, there may not exist an optimal solution that is composed of disjoint zigzags; see [2] for details. In the following, we describe a 12-approximation algorithm that runs in polynomial time when k, the number of robots, is a constant. The algorithm uses dynamic programming and has running time $((n/w_{\min})^{O(k)})$, where n is the number of sites and the maximum weight is 1 and the minimum weight is $w_{\min}$. In order to get an approximate solution, we will perform a series of relaxations to our problem. First, we introduce the Dyadic Time Window Problem.

Definition 1 (Dyadic Time Window Problem (DTW) and Dyadic Time Window Tour Problem (DTT)). *Let $s_1, \cdots, s_n$ be a collection of n weighted sites in $\mathbb{R}^1$, where w_i denotes the weight of s_i, such that $1 = w_1 \geq w_2 \geq \ldots \geq w_n$ and each weight w_i is of the form $(1/2)^{\alpha(i)}$ for some non-negative integer $\alpha(i)$. Given a parameter Λ called the window length, the decision version of the k-robot*

Dyadic Time Window Problem (DTW) asks whether there exists a schedule of the k robots for the time interval $[0, \Lambda/w_n]$ with the following property: each site s_i is visited at least once during every time interval $[(j-1)\Lambda/w_i, j\Lambda/w_i]$ with $j \in \{1, \ldots, w_i/w_n\}$. In the optimization version of the DTW *problem, we ask for a minimum value of Λ and wish to output a schedule achieving this minimum.*

The k-robot Dyadic Time Window Tour Problem (DTT) is defined similarly, except that we find an infinite schedule in which each site s_i is visited at least once during every time interval $[(j-1)\Lambda/w_i, j\Lambda/w_i]$ for all positive integers j.

The reason we introduce the DTW and DTT problems is that answers to these problems can help to provide constant approximations to the k-robot min-max weighted latency problem.

Lemma 6. *If there is a γ-approximation algorithm that solves DTT problem where sites have weights as powers of two, we can use it to solve the k-robot min-max weighted latency problem, in which site weights are not necessarily powers of two, with an approximation factor 4γ.*

Next, we briefly review how to solve the DTW problem approximately. This is built on the following observations. First, consider an optimal schedule σ^* to the DTW problem with window length Λ^*. Consider the schedule f of a robot r during the smallest interval $I_j = [(j-1)\Lambda^*, j\Lambda^*], j \in \{1, 2, \cdots, 1/w_n\}$. There are two possibilities: 1) the robot visits some sites during this time interval; 2) the robot does not visit any site but is in the middle of moving from one site s_1 to another site s_2 (but both s_1 and s_2 are visited outside the time interval I_j). The later is called a *M-schedule*. For the former, since the sites are on a line, there are four parameters that are important to the schedule $f[I_j]$: the starting position, the ending position, and the leftmost/rightmost point that r travels to – a schedule f' that matches these four parameters will visit all sites visited in f within interval I_j. Thus $f[I_j]$ can be replaced by the schedule traveling the minimum distance and meeting this requirement, called a *P-schedule* with the four parameters. Therefore, we use an exchange argument and assume that the optimal schedule does use a P-schedule (with possibly extra waiting time at the ending position). Further, we can actually limit the four parameters taking only values at site locations, which introduces another factor of three to the approximation ratio.

For an algorithm to solve the decision version of DTW, we run a bottom-up process by a proper concatenation of P-schedules and M-schedules, which have their parameters (the starting position, ending position and leftmost/rightmost points) limited to site locations. The main idea is to first find all possible schedules that cover sites of weight 1 for an interval of duration Λ. Keep all such schedules in a set $\mathbb{Q}_0$. In general, we have the schedules in $\mathbb{Q}_{j-1}$ as the schedules for k-robots for an interval of duration $2^{j-1}\Lambda$, that meet the requirements for all sites of weight at least $1/2^{j-1}$. We take all possible concatenations of pairs of schedules in $\mathbb{Q}_{j-1}$ and keep only those that cover the sites with weight $1/2^j$. If $\mathbb{Q}_m$ is not empty, we answer positively to the DTW. We show that if $\Lambda \geq 3\Lambda^*$ we will answer positively. The details are in [2].

The last step is to find a DTT schedule, using the above solution to the DTW problem. The main idea is to identify all valid DTW solutions for a $\Lambda \geq 3\Lambda^*$ and look for a cycle among them. The details of the algorithm and the analysis are provided in [2].

Theorem 3. *A 12-approximation of the min-max weighted latency for n sites in* $\mathbb{R}^1$ *with k robots, for a constant k, can be found in time* $((n/w_{\min})^{O(k)})$*, where the maximum weight of any site is* 1 *and the minimum weight is* $w_{\min}$*.*

References

1. M. Abrahamsen, M. de Berg, K. Buchin, M. Mehr, and A. D. Mehrabi. Range-clustering queries. In *33rd International Symposium on Computational Geometry (SoCG 2017), 14-17*, pages 1–16, 2017.
2. P. Afshani, M. de Berg, K. Buchin, J. Gao, M. Loffler, A. Nayyeri, B. Raichel, R. Sarkar, H. Wang, and H.-T. Yang. Approximation algorithms for multi-robot patrol-scheduling with min-max latency. https://arxiv.org/abs/2005.02530, 2020.
3. S. Alamdari, E. Fata, and S. L. Smith. Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations. *The International Journal of Robotics Research*, 33(1):138–154, 2014.
4. E. M. Arkin, R. Hassin, and A. Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006.
5. S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.
6. A. B. Asghar, S. L. Smith, and S. Sundaram. Multi-robot routing for persistent monitoring with latency constraints. In *2019 American Control Conference (ACC)*, pages 2620–2625, 2019.
7. M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983.
8. Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 302–308, 2004.
9. N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.
10. J. Czyzowicz, L. Gąsieniec, A. Kosowski, and E. Kranakis. Boundary patrolling by mobile agents with distinct maximal speeds. In *European Symposium on Algorithms*, pages 701–712, 2011.
11. G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.
12. N. Drucker, M. Penn, and O. Strichman. Cyclic routing of unmanned aerial vehicles. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 125–141, 2016.
13. A. Dumitrescu, A. Ghosh, and C. D. Tóth. On fence patrolling by mobile agents. *The Electronic Journal of Combinatorics*, 21(3):P3–4, 2014.
14. Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3-4):293–320, 2009.

15. Y. Elmaliach, A. Shiloni, and G. A. Kaminka. A realistic model of frequency-based multi-robot polyline patrolling. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '08, pages 63–70, Richland, SC, 2008.
16. L. Gąsieniec, R. Klasing, C. Levcopoulos, A. Lingas, J. Min, and T. Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In *SOFSEM 2017: Theory and Practice of Computer Science*, pages 229–240, 2017.
17. B. L. Golden, S. Raghavan, and E. A. Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer Science & Business Media, 2008.
18. L. Iocchi, L. Marchetti, and D. Nardi. Multi-robot patrolling with coordinated behaviours in realistic environments. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2796–2801, Sept 2011.
19. A. Kawamura and M. Soejima. Simple strategies versus optimal schedules in multi-agent patrolling. In *International Conference on Algorithms and Complexity*, pages 261–273, 2015.
20. M. Y. Khachai and E. Neznakhina. A polynomial-time approximation scheme for the Euclidean problem on a cycle cover of a graph. *Proceedings of the Steklov Institute of Mathematics*, 289(1):111–125, 2015.
21. M. Khachay and K. Neznakhina. Polynomial time approximation scheme for the minimum-weight k-size cycle cover problem in Euclidean space of an arbitrary fixed dimension. *IFAC-PapersOnLine*, 49(12):6–10, 2016.
22. M. R. Khani and M. R. Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica*, 69(2):443–460, 2014.
23. K. S. Liu, T. Mayer, H. T. Yang, E. Arkin, J. Gao, M. Goswami, M. P. Johnson, N. Kumar, and S. Lin. Joint sensing duty cycle scheduling for heterogeneous coverage guarantee. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9, 2017.
24. J. S. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-mst, and related problems. *SIAM Journal on computing*, 28(4):1298–1309, 1999.
25. C. H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical computer science*, 4(3):237–244, 1977.
26. D. Portugal, C. Pippin, R. P. Rocha, and H. Christensen. Finding optimal routes for multi-robot patrolling in generic graphs. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 363–369, 2014.
27. D. Portugal and R. P. Rocha. On the performance and scalability of multi-robot patrolling algorithms. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 50–55, Nov 2011.
28. E. Stump and N. Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. In *Automation Science and Engineering (CASE), 2011 IEEE Conference on*, pages 569–575, Aug 2011.
29. P. Toth and D. Vigo. *The vehicle routing problem.* SIAM, 2002.
30. W. Xu, W. Liang, and X. Lin. Approximation algorithms for min-max cycle cover problems. *IEEE Transactions on Computers*, 64(3):600–613, 2013.
31. H.-T. Yang, S.-Y. Tsai, K. S. Liu, S. Lin, and J. Gao. Patrol scheduling against adversaries with varying attack durations. In *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1179–1188, 2019.