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, Aarhus, Denmark
peyman@cs.au.dk
[2] Department of Mathematics and Computer Science, TU Eindhoven, 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, Utrecht, The Netherlands
m.loffler@uu.nl
[5] School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, 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, UK
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 time $(nw_{\max}/w_{\min})^{O(k)}$.

S. M. LaValle et al. (Eds.): WAFR 2020, SPAR 17, pp. 107–123, 2021.
https://doi.org/10.1007/978-3-030-66723-8_7

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 [15,18,23,27,28,31]. 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

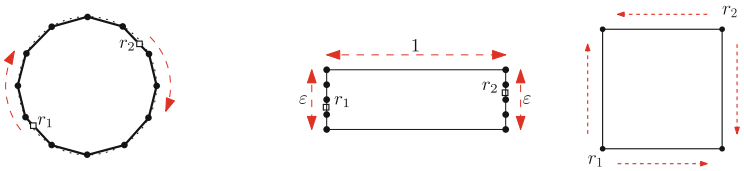

Fig. 1. Left: Two robots with n sites evenly placed on a unit circle. The optimal solution is to place two robots, maximum apart from each other, along the perimeter of a regular n-gon. Middle: Two robots with two clusters of vertices of distance 1 apart. The optimal solution is to have two robots each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

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.

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 Fig. 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 Fig. 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 Fig. 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 any bounds.

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 Fig. 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 Sect. 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 $t \leqslant k$ 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 $L \geqslant L^*$, 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 Sect. 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 this case, we turn the problem into the Time-Window Patrolling Problem, the solution to which is a constant approximation to our patrol problem. Again we round the weights to powers of two. 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. This way we have a 12-approximation solution in time $O((nw_{\max}/w_{\min})^{O(k)})$, where the maximum weight is $w_{\max}$ 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 $f_j : \mathbb{R}^{\geqslant 0} \to C(P, d)$, where $f_j(t)$ specifies the position of r_j at time t. A schedule

must obey the speed constraint, that is, we require $d(f_j(t_1), f_j(t_2)) \leqslant |t_1 - t_2|$ 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 $t_j^* \geqslant 0$ and period length $\tau_j > 0$ such that for any integer $i \geqslant 0$ and any $0 \leqslant t < \tau_j$ 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 $t_R^* \geqslant 0$ and $\tau_R > 0$ such that for any integer $i > 0$ and any $0 \leqslant t < \tau_R$ 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,

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

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 $w_i' = \sup\{2^x | x \in \mathbb{Z} \text{ and } 2^x \geqslant w_i\}$. Clearly, $w_i \leqslant w_i' < 2w_i$. 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, $0 \leqslant j \leqslant m$. Note that $|\mathcal{W}| \leqslant m + 1 = \log \frac{w_{\max}}{w_{\min}} + 1$. 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 $\beta k^2 mL$. 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 $\leqslant L$, 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.

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 $x_{\text{dep}}(r) \in W_{j'}$
      $Q' \leftarrow \{v | v \in Q, d(v, x_{\text{dep}}(r)) \leqslant k 2^{j'} L\}$
      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 $\mathcal{T}(r_1), \ldots, \mathcal{T}(r_k)$.

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 $w_i \geqslant w_j$, 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 $w_0 \geqslant \cdots \geqslant w_k$, 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 $L^* \geqslant 2L$.*

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 $\sum_r \text{speed}(r, t) \leqslant k$, 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 $L^* \geqslant 2kL/k = 2L$. 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 $\sum_r \text{speed}(r, t) \leqslant k$. 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

$$\ell_0 \leqslant a_1 \leqslant \ell_1 \leqslant \cdots .$$

Consider an interval (ℓ_i, a_{i+1}). By definition $a_{i+1} - \ell_i \leqslant L^*/w_{p-1}$. 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. □

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. $h \leqslant km$

1: $\delta \leftarrow 2kL/w_0$.
2: **for** $i \leftarrow 0$ to $h-1$
3: Compute a tour D_i of length at most $2|T_i|$ on the vertices in T_i.
4: Partition D_i into a collection $\mathcal{P}^i = \{P_0^i, P_1^i, \cdots\}$ of at most $\lceil 2|T_i|/\delta \rceil$ paths such that $|P_j^i| \leqslant \delta$ for all j.
5: $\text{idx}(i) \leftarrow 0$ ▷ $P^i_{\text{idx}(i)}$ is the path in $\mathcal{P}^i$ to be traversed next
6: Put the robot on the first vertex of path P_0^0 and set $i \leftarrow 0$
7: **while** TRUE
8: Let the robot traverse path $P^i_{\text{idx}(i)}$
9: $i' \leftarrow (i+1) \mod h$
10: Let the robot move from the end of $P^i_{\text{idx}(i)}$ to the start of $P^{i'}_{\text{idx}(i')}$
11: Set $\text{idx}(i) \leftarrow (\text{idx}(i)+1) \mod |\mathcal{P}_i|$ and set $i \leftarrow i'$

The proofs for the following two Lemmas can be found in [2].

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

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 $\beta L/w$.*

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 [3,16] 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., $h \leqslant km$. For a tree T (a path P) we use $|T|$ (resp. $|P|$) as the sum of the length of edges in T (resp. P).

Lemma 5. *The* Single Robot Schedule*($\mathcal{T} = \{T_0, T_2, \cdots, T_{h-1}\}$), $h \leqslant k(m+1)$, 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 $O(k^2 m \cdot L)$.*

To analyze the running time, we use the best known t-min-max tree cover algorithm [30] with running time $O(n^2t^2 \log n + t^5 \log n)$. In Algorithm k-robot Assignment, from line 2 to line 8 it takes time in the order of $O(mn^2 \log n) \cdot (1^2 + 2^2 + \cdots k^2) = O(mn^2k^3 \log n)$ (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 $O(n^2k^3m \log n)$ to either generate patrol schedules for k robots with approximation factor $O(k^2m)$ 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 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 $O(n^2k^3m \log n \log L^*)$ with a $O(k^2m)$-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 $\mathbb{R}^1$. We start with 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.*

Next, for multiple robots, as long as the sites have uniform weights, we show there is an optimal schedule consisting of disjoint zigzags. Both proofs are in [2].

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 $1 \leqslant k \leqslant n$. Then there exists an optimal schedule such that each robot follows a zigzag schedule and the intervals covered by these zigzag schedules are disjoint.*

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.

We now turn our attention to sites in $\mathbb{R}^1$ with arbitrary weights. In this setting there may not exist an optimal solution that is composed of disjoint zigzags (see [2] for details), which makes it difficult to compute an optimal solution. Hence, we present an approximation algorithm. Let $\rho := w_{\max}/w_{\min}$ be the ratio of the largest and smallest weight of any of the sites. Our algorithm has a 12-approximation ratio and runs in polynomial time when k, the number of robots, is a constant, and ρ is polynomial in n. More precisely, the running time of the algorithm is $O((\rho n)^{O(k)})$.

Instead of solving the k-robot patrol-scheduling problem directly, our algorithm will solve a discretized version that is defined as follows.

- The input is a set $P = \{s_1, \ldots, s_n\}$ of sites in $\mathbb{R}^1$, each with a weight w_i of the form $(1/2)^{\alpha(i)}$ for some non-negative integer $\alpha(i)$ and such that $1 = w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n$.
- Given a value $L > 0$, which we call the *window length*, we say that a k-robot schedule is *valid* if the following holds: each site s_i is visited at least once during every time interval of the form $[(j-1)L/w_i, jL/w_i]$, where j is a positive integer. The goal is to find the smallest value L that admits a valid schedule, and to report the corresponding schedule.

We call this problem the *Time-Window Patrolling Problem*. The following lemma shows that its solution can be used to solve the patrol-scheduling problem. The proof can be found in [2].

Lemma 6. *Suppose we have a γ-approximation algorithm for the k-robot Time-Window Patrolling Problem that runs in $T(n, k, \rho)$ time. Then there is 4γ-approximation algorithm for the k-robot patrol scheduling problem that runs in $O(n \log n + T(n, k, \rho))$ time.*

An Algorithm for the Time-Window Patrolling Problem. We now describe an approximation algorithm for the Time-Window Patrolling Problem. To this end we define a class of so-called *standard schedules*, and we show that the best standard schedule is a good approximation to the optimal schedule. Then we present an algorithm to compute the best standard schedule.

Standard schedules, for a given window length L, have length (that is, duration) L/w_n and they are composed of $1/w_n$ so-called atomic L-schedules.

An *atomic L-schedule* is a schedule θ that specifies the motion of a single robot during a time interval of length L. It is specified by a 6-tuple

$$(s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta), t_{\text{before}}(\theta), t_{\text{after}}(\theta)),$$

where $s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta) \in P \cup \{\text{NIL}\}$ and $t_{\text{before}}(\theta), t_{\text{after}}(\theta) \in \{0, 2L/3, L\}$. Roughly speaking, $s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta)$ denote the first, last, leftmost and rightmost site visited during the time interval, and $t_{\text{before}}(\theta), t_{\text{after}}(\theta)$ indicate how long the robot can spend traveling before arriving at $s_{\text{start}}(\theta)$ resp. after leaving $s_{\text{end}}(\theta)$. Next we define this more precisely.

There are two types of atomic L-schedules. For concreteness we explain the different types of atomic L-schedules for the time interval $[0, L]$, but remember that an atomic L-schedule can be executed during any time interval of length L.

Type I: $s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta) \in P$, and $t_{\text{before}}(\theta) = 0$ and $t_{\text{after}}(\theta) = 2L/3$, where $s_{\text{left}}(\theta)$ and $s_{\text{right}}(\theta)$ are the leftmost and rightmost site among the four sites, respectively. (We allow one or more of these four sites to be identical.) A Type I atomic L-schedule specifies the following movement of the robot.

- At time $t = 0$ the robot is at site $s_{\text{start}}(\theta)$.
- At time $t = L/3$ the robot is at site $s_{\text{end}}(\theta)$.
- The robot will visit $s_{\text{left}}(\theta)$ and $s_{\text{right}}(\theta)$ during the interval $[0, L/3]$ using the shortest possible path, which must have length at most $L/3$. For example, if $s_{\text{left}}(\theta) < s_{\text{start}}(\theta) < s_{\text{end}}(\theta) < s_{\text{right}}(\theta)$, then the robot will use the path $s_{\text{start}}(\theta) \to s_{\text{left}}(\theta) \to s_{\text{right}}(\theta) \to s_{\text{end}}(\theta)$ and we require $|s_{\text{start}}(\theta) - s_{\text{left}}(\theta)| + |s_{\text{right}}(\theta) - s_{\text{left}}(\theta)| + |s_{\text{right}}(\theta) - s_{\text{end}}(\theta)| \leqslant L/3$.
- The robot does not visit any sites during $(L/3, L]$ but is traveling, towards some site to be visited later. In fact, the robot may pass other sites when it is traveling during $(L/3, L]$ but these events are ignored—they are not counted as visits.

Type II: $s_{\text{start}}(\theta) = s_{\text{end}}(\theta) = s_{\text{left}}(\theta) = s_{\text{right}}(\theta) = \text{NIL}$, and $t_{\text{before}}(\theta) = 0$ and $t_{\text{after}}(\theta) = L$. A Type II atomic L-schedule specifies the following movement of the robot.

- The robot does not visit any sites during $[0, L]$ but is traveling. Again, the robot may pass over sites during its movement. One way to interpret Type II atomic schedules is that the robot visits a dummy site at time $t = 0$ and then spends the entire interval $(0, L]$ traveling towards some site to be visited in a later time interval.

Note that $t_{\text{before}}(\theta) = 0$ in both cases. This will no longer be the case, however, when we start concatenating atomic schedules, as explained next.

Consider the concatenation of 2^h atomic L-schedules, for some $h \geqslant 0$, and suppose we execute this concatenated schedule during a time interval I of the form $[(j-1)2^h L, j2^h L]$. How the robots travel exactly during interval I is important for sites of weight more than $1/2^h$, since such sites need to be visited multiple times. But sites of weight at most $1/2^h$ need to be visited at most once during I, and so for those sites it is sufficient to know the leftmost and rightmost visited

site. Thus our algorithm will concatenate atomic L-schedules in a bottom-up manner. This will be done in rounds, where the h-th round will ensure that sites of weight $1/2^h$ are visited. The concatenated schedule will be represented in a similar way as atomic schedules. Next we describe this in detail.

Let $\mathcal{S}(h)$ denote the collection of all feasible concatenations of 2^h atomic L-schedules. Thus $\mathcal{S}(0)$ is simply the collection of all atomic schedules, and $\mathcal{S}(h)$ can be obtained from $\mathcal{S}(h-1)$ by combining pairs of schedules. A schedule $\theta \in \mathcal{S}(h)$ will be represented by a 6-tuple

$$(s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta), t_{\text{before}}(\theta), t_{\text{after}}(\theta)).$$

As before, $s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta)$ denote the first, last, leftmost and rightmost site visited during the time interval. Furthermore, $t_{\text{before}}(\theta)$ indicates how much time the robot can spend traveling from another site before arriving at $s_{\text{start}}(\theta)$, and $t_{\text{after}}(\theta)$ indicates how much time the robot can spend traveling towards another site after leaving at $s_{\text{end}}(\theta)$. The values $t_{\text{before}}(\theta)$ and $t_{\text{after}}(\theta)$ can now take larger values than in an atomic L-schedule. In particular,

$$t_{\text{before}}(\theta), t_{\text{after}}(\theta) \in \{((2/3)+i)L : 0 \leqslant i < 2^h\} \cup \{iL : 0 \leqslant i \leqslant 2^h\},$$

where $t_{\text{before}}(\theta) + t_{\text{after}}(\theta) \leqslant 2^h L$. Note that certain values may only arise in certain situations. For example, we can only have $t_{\text{after}}(\theta) = 2^h L$ for a schedule that is the concatenation of 2^h atomic L-schedules of type II, which means that $s_{\text{start}}(\theta), s_{\text{end}}(\theta), s_{\text{left}}(\theta), s_{\text{right}}(\theta) = \text{NIL}$ and $t_{\text{before}}(\theta) = 0$.

We denote the concatenation of two schedules $\theta, \theta' \in \mathcal{S}(h)$ by $\theta \oplus \theta'$. The representation of $\theta \oplus \theta'$ can be computed from the representations of θ and θ':

$$s_{\text{start}}(\theta \oplus \theta') = \begin{cases} s_{\text{start}}(\theta') & \text{if } s_{\text{start}}(\theta) = \text{NIL} \\ s_{\text{start}}(\theta) & \text{otherwise} \end{cases}$$

$$s_{\text{end}}(\theta \oplus \theta') = \begin{cases} s_{\text{end}}(\theta) & \text{if } s_{\text{end}}(\theta') = \text{NIL} \\ s_{\text{end}}(\theta') & \text{otherwise} \end{cases}$$

Furthermore, we have $s_{\text{left}}(\theta \oplus \theta') = \min(s_{\text{left}}(\theta), s_{\text{left}}(\theta'))$ and $s_{\text{right}}(\theta \oplus \theta') = \max(s_{\text{right}}(\theta), s_{\text{right}}(\theta'))$. Finally,

$$t_{\text{before}}(\theta \oplus \theta') = \begin{cases} t_{\text{after}}(\theta) + t_{\text{before}}(\theta') & \text{if } s_{\text{start}}(\theta) = \text{NIL} \\ t_{\text{before}}(\theta) & \text{otherwise} \end{cases}$$

$$t_{\text{after}}(\theta \oplus \theta') = \begin{cases} t_{\text{after}}(\theta) + t_{\text{after}}(\theta') & \text{if } s_{\text{start}}(\theta') = \text{NIL} \\ t_{\text{after}}(\theta) & \text{otherwise} \end{cases}$$

Note that not any pair of schedules θ, θ' can be combined: it needs to be possible to travel from $s_{\text{end}}(\theta)$ to $s_{\text{start}}(\theta')$ in the available time. More precisely, assuming $s_{\text{end}}(\theta) \neq \text{NIL}$ and $s_{\text{start}}(\theta') \neq \text{NIL}$—otherwise a concatenation is always possible—we need $d(s_{\text{end}}(\theta), s_{\text{start}}(\theta')) \leqslant t_{\text{after}}(\theta) + t_{\text{before}}(\theta')$.

We now define a *standard k-robot schedule for window length L* to be a k-robot schedule with the following properties.

(i) The schedule for each robot belongs to $\mathcal{S}(\log(1/w_n))$, i.e., each robot starts at a site at time $t = 0$, and is the concatenation of $1/w_n$ atomic L-schedules.
(ii) It is a valid k-robot schedule for the Time-Window Patrolling Problem, for the time period $[0, L/w_n]$.

A standard schedule σ can be turned it into an infinite cyclic schedule, by executing σ and its reverse schedule σ^{-1} in an alternating fashion. (In σ^{-1} each robot simply executes its schedule in σ backward.) Note that σ^{-1} is a valid schedule since σ is valid, and so the schedule alternating between σ and σ^{-1} is valid. The following lemma shows that the resulting schedule is a good approximation of an optimal schedule for the Time-Window Patrolling Problem (proof in [2]).

Lemma 7. *Let L^* be the minimum window length that admits a valid schedule for the Time-Window Patrolling Problem, and let L be the minimum window length that admits a valid standard schedule. Then $L \leqslant 3L^*$.*

We now present an algorithm that, given a window length L, decides if a standard schedule of window length L exists. Since such a schedule is the concatenation of $1/w_n$ atomic L-schedules we basically generate all possible concatenated schedules iteratively from $\mathcal{S}(0)$ to $\mathcal{S}(\log(1/w_n))$. Recall that we need to generate a k-robot schedule, that is, a collection of k schedules (one for each robot). We denote by $\mathcal{S}_k(h)$ the set of all k-robots schedules, where each of the schedules is chosen from $\mathcal{S}(h)$, such that each site of weight at least $1/2^h$ is visited at least once by one of the robots. If $\sigma = \langle \theta, \ldots, \theta_k \rangle$ and $\sigma' = \langle \theta'_1, \ldots, \theta'_k \rangle$ are two k-robot schedules, then we use $\sigma \oplus \sigma'$ to denote the k-robot schedule $\langle \theta \oplus \theta'_1, \ldots, \theta_k \oplus \theta'_k \rangle$.

Note that the concatenation of one pair of single-robot schedules may be the same as—or, more precisely, have the same representation as—the concatenation of a different pair of schedules. This may also result in k-robot schedules that are the same. To avoid generating too many k-robot schedules, our algorithm will keep only one schedule of each representation. Our algorithm is now as follows.

CONSTRUCT-SCHEDULE(P, L)
1: $\mathcal{S}(0) \leftarrow$ { all possible atomic L-schedules }
2: $\mathcal{S}_k(0) \leftarrow$ { all possible combinations of k schedules from $\mathcal{S}(0)$ such that all sites of weight 1 are visited by at least one of the schedules }
3: **for** $h \leftarrow 1$ to m ▷ Recall that $w_n = 1/2^m$
4: $\quad \mathcal{S}_k(h) \leftarrow \emptyset$
5: $\quad$ **for** every pair of k-robot schedules $\sigma, \sigma' \in \mathcal{S}_k(h-1)$
6: $\quad\quad$ If σ and σ' can be concatenated and the resulting schedule visits every site of weight $1/2^h$ at least once then add $\sigma \oplus \sigma'$ to $\mathcal{S}_k(h)$.
7: $\quad$ Remove any duplicates from $\mathcal{S}_k(h)$.
8: If $\mathcal{S}_k(m) \neq \emptyset$ then return YES otherwise return NO.

The algorithm above only reports if a standard schedule of window length L exists, but it can easily be modified such that it reports such schedule if it exists. To this end we just need to keep, for each representation in $\mathcal{S}(h)$ for the current value of h, an actual schedule. Doing so will not increase the time-bound of the algorithm. The main theorem in this section is as below, with proof 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 $O(n^{8k+1}(w_{\max}/w_{\min})^{4k}\log(n{\cdot}w_{\max}/w_{\min})) = (nw_{\max}/w_{\min})^{O(k)}$, where the maximum weight of any site is $w_{\max}$ and the minimum weight is $w_{\min}$.*

5 Conclusion and Future Work

This is the first paper that presents approximation algorithms for multi-robot patrol scheduling minimizing maximum weighted latency in a metric space. The obvious open problem is to improve the approximation ratios for both the general metric setting and the 1D setting.

Acknowledgement. Gao, Wang and Yang would like to acknowledge supports from NSF CNS-1618391, DMS-1737812, OAC-1939459. Raichel would like acknowledge support from NSF CAREER Award 1750780.

References

1. Abrahamsen, M., de Berg, M., Buchin, K., Mehr, M., Mehrabi, A.D.: Range-clustering queries. In: 33rd International Symposium on Computational Geometry (SoCG 2017), pp. 1–16, 14–17 (2017)
2. Afshani, P., de Berg, M., Buchin, K., Gao, J., Loffler, M., Nayyeri, A., Raichel, B., Sarkar, R., Wang, H., Yang, H.-T.: Approximation algorithms for multi-robot patrol-scheduling with min-max latency 2020. https://arxiv.org/abs/2005.02530
3. Alamdari, S., Fata, E., Smith, S.L.: Persistent monitoring in discrete environments: minimizing the maximum weighted latency between observations. Int. J. Robot. Res. **33**(1), 138–154 (2014)
4. Arkin, E.M., Hassin, R., Levin, A.: Approximations for minimum and min-max vehicle routing problems. J. Algorithms **59**(1), 1–18 (2006)
5. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM (JACM) **45**(5), 753–782 (1998)
6. Asghar, A.B., Smith, S.L., Sundaram, S.: Multi-robot routing for persistent monitoring with latency constraints. In: 2019 American Control Conference (ACC), pp. 2620–2625 (2019)
7. Ben-Or, M.: Lower bounds for algebraic computation trees. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, pp. 80–86 (1983)
8. Chevaleyre, Y.: Theoretical analysis of the multi-agent patrolling problem. In: IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 302–308 (2004)
9. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report, Carnegie-Mellon University (1976)
10. Czyzowicz, J., Gąsieniec, L., Kosowski, A., Kranakis, E.: Boundary patrolling by mobile agents with distinct maximal speeds. In: European Symposium on Algorithms, pp. 701–712 (2011)
11. Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Manage. Sci. **6**(1), 80–91 (1959)

12. Drucker, N., Penn, M., Strichman, O.: Cyclic routing of unmanned aerial vehicles. In: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 125–141 (2016)
13. Dumitrescu, A., Ghosh, A., Tóth, C.D.: On fence patrolling by mobile agents. Electron. J. Comb. **21**(3), 3–4 (2014)
14. Elmaliach, Y., Agmon, N., Kaminka, G.A.: Multi-robot area patrol under frequency constraints. Ann. Math. Artif. Intell. **57**(3–4), 293–320 (2009)
15. Elmaliach, Y., Shiloni, A., Kaminka, G.A.: A realistic model of frequency-based multi-robot polyline patrolling. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 63–70 (2008)
16. Gąsieniec, L., Klasing, R., Levcopoulos, C., Lingas, A., Min, J., Radzik, T.: Bamboo garden trimming problem (perpetual maintenance of machines with different attendance urgency factors). In: SOFSEM 2017: Theory and Practice of Computer Science, pp. 229–240 (2017)
17. Golden, B.L., Raghavan, S., Wasil, E.A.: The Vehicle Routing Problem: Latest Advances and New Challenges. Springer Science & Business Media, New York (2008)
18. Iocchi, L., Marchetti, L., Nardi, D.: Multi-robot patrolling with coordinated behaviours in realistic environments. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2796–2801, September 2011
19. Kawamura, A., Soejima, M.: Simple strategies versus optimal schedules in multi-agent patrolling. In: International Conference on Algorithms and Complexity, pp. 261–273 (2015)
20. Khachai, M.Y., Neznakhina, E.: A polynomial-time approximation scheme for the Euclidean problem on a cycle cover of a graph. Proc. Steklov Inst. Math. **289**(1), 111–125 (2015)
21. Khachay, M., Neznakhina, K.: Polynomial time approximation scheme for the minimum-weight k-size cycle cover problem in Euclidean space of an arbitrary fixed dimension. IFAC-Papers OnLine **49**(12), 6–10 (2016)
22. Khani, M.R., Salavatipour, M.R.: Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. Algorithmica **69**(2), 443–460 (2014)
23. Liu, K.S., Mayer, T., Yang, H.T., Arkin, E., Gao, J., Goswami, M., Johnson, M.P., Kumar, N., Lin, S.: Joint sensing duty cycle scheduling for heterogeneous coverage guarantee. INFOCOM **2017**, 1–9 (2017)
24. Mitchell, J.S.: Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-mst, and related problems. SIAM J. Comput. **28**(4), 1298–1309 (1999)
25. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. Theoretical Comput. Sci. **4**(3), 237–244 (1977)
26. Portugal, D., Pippin, C., Rocha, R.P., Christensen, H.: Finding optimal routes for multi-robot patrolling in generic graphs. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 363–369 (2014)
27. Portugal, D., Rocha, R.P.: On the performance and scalability of multi-robot patrolling algorithms. In: 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, pp. 50–55, November 2011
28. Stump, E., Michael, N.: Multi-robot persistent surveillance planning as a vehicle routing problem. In: 2011 IEEE Conference on Automation Science and Engineering (CASE), pp. 569–575, August 2011

29. Toth, P., Vigo, D.: The vehicle routing problem. SIAM (2002)
30. Xu, W., Liang, W., Lin, X.: Approximation algorithms for min-max cycle cover problems. IEEE Trans. Comput. **64**(3), 600–613 (2013)
31. Yang, H.-T., Tsai, S.-Y., Liu, K.S., Lin, S., Gao J.: Patrol scheduling against adversaries with varying attack durations. In: Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems, pp. 1179–1188 (2019)