Min-Max Latency Walks: Approximation Algorithms for Monitoring Vertex-Weighted Graphs

Soroush Alamdari, Elaheh Fata, and Stephen L. Smith

Abstract In this paper, we consider the problem of planning a path for a robot to monitor a known set of features of interest in an environment. We represent the environment as a vertex- and edge-weighted graph, where vertices represent features or regions of interest. The edge weights give travel times between regions, and the vertex weights give the importance of each region. If the robot repeatedly performs a closed walk on the graph, then we can define the latency of a vertex to be the maximum time between visits to that vertex, weighted by the importance (vertex weight) of that vertex. Our goal in this paper is to find the closed walk that minimizes the maximum weighted latency of any vertex. We show that there does not always exist an optimal walk of polynomial size. We then prove that for any graph there exist a constant approximation walk of size $O(n^2)$, where n is the number of vertices. We provide two approximation algorithms; an $O(\log n)$-approximation and an $O(\log \rho)$-approximation, where ρ is the ratio between the maximum and minimum vertex weight. We provide simulation results which demonstrate that our algorithms can be applied to problems consisting of thousands of vertices.

1 Introduction

An emerging application area for robotics is in performing long-term monitoring tasks. Some example problems in monitoring include 1) environmental monitoring tasks such as ocean sampling [15], where autonomous underwater vehicles sense the ocean to detect the onset of algae blooms; 2) surveillance tasks [12], where robots repeatedly visit vantage points in order to detect events or threats, and; 3) infrastructure inspection tasks such as power-line or manhole cover inspection [18], where

Soroush Alamdari
Cheriton School of Computer Science, University of Waterloo, Waterloo ON N2L 3G1, Canada. e-mail: s26hosse@uwaterloo.ca

Elaheh Fata and Stephen L. Smith
Department of Electrical and Computer Engineering, University of Waterloo, Waterloo ON N2L 3G1, Canada. e-mail: efata@uwaterloo.ca, stephen.smith@uwaterloo.ca

spatially distributed infrastructure must be repeatedly inspected for the presence of failures. For such tasks, a key problem is to plan robot paths that visit different parts of the environment so as to efficiently perform the monitoring task. Since some parts of the environment may be more important than others (e.g., in ocean sampling, some regions are more likely to experience an algae bloom than others), the planned path should visit regions with a frequency proportional to their importance.

In this paper, we cast such long-term monitoring tasks as an optimization problem on a vertex- and edge-weighted graph: the *min-max latency walk problem*. The vertices represent features or regions of interest. The edge weights give travel times between regions, and the vertex weights give the importance of each region. Given a robot walk on the graph, the *latency* of a vertex is the maximum time between visits to that vertex, weighted by the importance (vertex weight) of that vertex. We then seek to find a walk that minimizes the maximum latency over all vertices. In an ocean sampling task, this would be akin to minimizing the expected number of algae blooms that occur in any region prior to a robot visit.

Prior work: The min-max latency walk problem generalizes our earlier work [16], where we considered the problem for features distributed in a Euclidean space according to a known probability distribution. Under this setup, constant factor approximation algorithms were developed for the limiting case of large numbers of vertices. However, the algorithms have no performance guarantees for general input graphs that may have non-Euclidean edge weights and smaller numbers of vertices.

In [18], the authors consider a preventative maintenance problem in which the input is the same as in the min-max latency walk problem, but the output is a walk which visits each vertex exactly once. More important vertices (i.e., those that are more likely to fail) should be visited earlier in the path. The authors find a path by solving a mixed-integer program. The min-max latency walk problem can be thought of as a generalization of this problem, where the maintenance and inspection should continually be performed.

The problem considered in this paper is also a more general version of sweep coverage [5], where a robot must move through the environment so as to cover the entire region with its sensor. Variants of this problem include on-line coverage [9], where the robot has no *a priori* information about the environment, and dynamic coverage [10], where each point in the environment requires a pre-specified "amount" of coverage. In [17], a dynamic coverage problem is considered where sensor continually surveys regions of interest by moving according to a Markov Chain. In [3] a similar approach to continuous coverage is taken and a Markov chain is used to achieve a desired visit-frequency distribution over a set of features.

Another related problem is patrolling [4, 8, 14], a region must be continually surveyed by a group of robots. Existing work has considered the case of minimizing the time between visits to each point in space. A variant of patrolling is considered in [2] for continual target surveillance. The persistent monitoring problem considered in this paper extends the work on patrolling in that different points change at different rates, and the change between visits must be minimized.

Finally, the min-max latency walk problem is related to vehicle routing and dynamic vehicle routing (DVR) problems [1]. One example is the period routing problem [6], where each customer must be visited a specified number of times per week. A solution consists of an assignment of customers to days of the week, and a set of routes for the vehicles on each day.

Contributions: The contribution of this paper are threefold. First, we introduce the general min-max latency walk problem and show that it is well-posed and that it is APX-hard. Second, we provide results on the existence of optimal and approximation algorithms for the problem. We showed that in general, the optimal walk can be very long—it's length can be non-polynomial in the size of the input graph, and thus there cannot exist a polynomial time algorithm for the problem. We then show that there always exists a constant factor approximation solution that consists of a walk of length $O(n^2)$, where n is the number of vertices in the input graph. Third, and finally, we provide two approximation algorithms for the problem. Defining ρ_G to be the ratio between the maximum and minimum vertex weight in the input graph G, we give a $O(\log \rho_G)$ approximation algorithm. Thus, when ρ_G is independent of n, we have a constant factor approximation. We also provide an $O(\log n)$ approximation which is independent of the value of ρ. The algorithms rely on relaxing the vertex weights to be powers of 2, and then planning paths through "batches" of vertices with the same relaxed weights.

Organization: This paper is organized as follows. In Section 2 we give some background on graphs and formalize the min-max latency walk problem. In Section 3 we present a relaxation of graph weights which allows for the design of approximation algorithms. In Section 4 we present results on the existence of constant factor approximations and some negative results on the required length of the walk. In Section 5 we present two approximation algorithms for the problem. In Section 6 we present large scale simulation data for standard TSP test-cases and in Section 7 we present conclusions and future directions.

2 Background and Problem Statement

In this section, we review graph terminology and define the problem considered in this paper.

2.1 Background on Graphs

The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$ respectively, where $E(G)$ consists of two element subsets of $V(G)$. We write an edge in $E(G)$ as $\{v_i, v_j\}$ or $v_i v_j$. An edge-weighted graph G associates a weight $w(e) \geq 0$ to each edge $e \in E(G)$. A vertex-weighted graph G associates a weight $\phi(v) \in [0,1]$ to each vertex $v \in V(G)$. Throughout this paper, all referenced graphs are both vertex-weighted and edge-weighted and therefore we omit the explicit reference. Also, without loss of generality, we assume that there is at least one vertex in $V(G)$ with weight 1, as in our applications weights can be scaled so that this is true. We define ρ_G to be the ratio between the maximum and minimum vertex weight: $\rho_G := \max_{v_i, v_j \in V(G)} \{\phi(v_i)/\phi(v_j)\}$. Given a graph G and a set $V' \subseteq V(G)$, the graph

$G[V']$ is the graph obtained from G by removing the vertices of G that are not in V' and all edges incident to a vertex in $V(G) \setminus V'$.

A walk of *length* k in a graph G is a sequence of vertices, $(v_1, v_2, \ldots, v_{k+1})$, such that there exists an edge $v_i v_{i+1} \in E(G)$ for $1 \leq i \leq k$. The *weight* of a walk W, denoted by weight(W), is the sum of the weights of edges of that walk. A walk is closed if its beginning and end are the same vertex. Given a walk $W = (v_1, \ldots, v_k)$, and integers $i \leq j \leq k$, the *sub-walk* $W(i, j)$ is defined as the subsequence of W given by $W(i, j) = (v_i, v_{i+1}, \ldots, v_j)$. Given the walks $W_1, W_2, \ldots, W_k$, the walk $W = [W_1, W_2, \ldots, W_k]$ is the result of concatenation of W_1 through W_k, while preserving order.

An *infinite walk* is a sequence of vertices, $(v_1, v_2, \ldots)$, such that there exists an edge $v_i v_{i+1} \in E(G)$ for $i \in \mathbb{N}$. We say that a closed walk W expands to an infinite walk $\Delta(W)$, when $\Delta(W)$ is constructed by an infinite number of copies of W appended together: $\Delta(W) = [W, W, \ldots]$. It can be seen that for any closed walk, there exists a unique expansion to an infinite walk. The *kernel* of an infinite walk W, denoted by $\delta(W)$, is the shortest closed walk such that W is the *expansion* of $\delta(W)$. It is easy to observe that there are infinite walks for which a kernel does not exist. For such an infinite walk W, we define $\delta(W)$ to be W itself.

2.2 The Min-Max Latency Walk Problem

Let G be a weighted graph and W be an infinite walk in G. We define the *latency* of vertex v on walk W, denoted by $L(W, v)$, as the maximum weight of the sub-walk between any two consecutive visits to v on W. Then, we can define the cost of a vertex $v \in V(G)$ on the walk W to be

$$c(W, v) := \phi(v) L(W, v).$$

The cost of an infinite walk W, denoted by $c(W)$ is

$$c(W) = \max_{v \in V(G)} c(W, v).$$

Then, the min-max latency walk problem can be stated as follows.

The min-max latency walk problem. Find an infinite walk W that minimizes the cost $c(W)$.

2.3 Well-Posedness of the Problem

Finding an infinite walk is computationally infeasible. Instead, we will try to find the kernel of the minimum cost infinite walk. The first question, however, is whether or not there always exists a minimum cost walk.

Lemma 1. *For any graph G, there exists a walk of minimum cost.*

Proof. Let W be a walk in G that covers $V(G)$. Let c be the (necessarily finite) cost of W. There are finite walks in G with cost less than c. The reason is that for any vertex $v \in V(G)$, there are finite closed walks beginning and ending in v with weight less than $c/\phi(v)$. Hence there are finite possible costs so that v can induce to a walk of cost less than $c(W)$. In other words, there are finite numbers $c' < c$ that can be the cost of some walk in G. □

We define OPT_G to be the minimum cost among all infinite walks on G. By Lemma 1, such a number always exists. Let S be the set of kernels of all infinite walks of cost OPT_G in G. We define $\tau(G)$ to be the length of the shortest kernels in S. Next we will show that the problem of min-max latency is APX-hard, implying that there is no polynomial-time approximation scheme (PTAS) for it, unless P=NP.

Theorem 1. *The min-max latency problem is APX-hard.*

Proof. The reduction is from the metric Traveling Salesman Problem (TSP). TSP is the problem of finding the smallest closed walk that visits all vertices exactly once. Such walk is referred to as the TSP tour. It is known that finding the TSP tour is APX-hard in metric graphs [13], and it is approximable within a factor of 1.5. We show a reduction that preserves the hardness of approximation.

Let G be the input of the metric TSP. Assign weight 1 to all vertices of G. Let W be a minimum cost infinite walk in G. Let c be the cost of W and v be the vertex with $c(W,v) = c$. Let i and j be the indices of two consecutive instances of v with weight$(W(i,j)) = c$. It is easy to see that all vertices of G appear in $W(i,j)$, otherwise, there is another vertex u, with $c(W,v) < c(W,u)$. Let M be a closed walk that is an optimal solution for TSP in G, we prove weight$(M) = c$. Let weight$(M) = c'$. It is easy to observe that the cost of $\Delta(M)$ is also c'. Therefore, c' cannot be less than c, because this would contradict the the fact that W has minimum cost. Also, c' can not be greater than c, since in that case, the spanning closed walk $W(i,j)$ with cost $c < c'$ would imply existence of a better solution for TSP than M. It is well known that in metric graphs with a closed walk T, there is a cycle T' with weight$(T) \geq$ weight(T') that visits the same set of vertices and each vertex exactly once by shortcutting the repetitive vertices in T. Note that we showed that the size of the solution for the two problems are equal, hence the reduction is gap preserving and the APX-hardness carries over. □

We focus on solving the min-max latency problem only for complete metric graphs. The reason is that for any graph G and any $u,v \in V(G)$ we can create a graph G' with the same set of vertices such that the uv edge in G' has weight equal to the shortest-path distance from u to v in G, $d(u,v)$. Then to construct a walk for G based on a walk in G', we can replace each uv edge with the shortest path connecting u and v in G. Since $\mathrm{OPT}_G = \mathrm{OPT}_{G'}$ and any walk in G' corresponds to a walk of lower or equal cost in G, any approximation in G' carries over to G. In the literature, the graph G' is refereed to as the *metric closure* of G.

3 Relaxations and Simple Bounds

In this section, we present a relaxation of the problem and two simple bounds based on the weights of the edges of the input graph.

3.1 Relaxation of Vertex Weights

Here, we define a relaxation of the problem so that all weights are of the form $1/2^x$, where x is an integer.

Definition 1 (Weight Relaxation). We say weights of vertices are relaxed, if for any vertex $v \in V(G)$, we update its weight to $\phi'(v)$, where $\phi'(v) = \frac{1}{2^x}$ with the property that x is the smallest integer so that $\frac{1}{2^x} \leq \phi(v)$ holds.

Lemma 2 (Relaxed vertex weights). *Let G' be obtained by relaxing the weights of G. Let W be a walk with cost c in G and cost c' in G'. Then $c' \leq c < 2c'$. Consequently, $\mathrm{OPT}_{G'} \leq \mathrm{OPT}_G < 2\mathrm{OPT}_{G'}$.*

Proof. The weight of each vertex in G' is less than or equal to the weight of that vertex in G. Therefore, $c' \leq c$. Also, since the weight of each vertex in G' is more than half of the weight of the same vertex in G, we have that $c < 2c'$. □

3.2 Simple Bounds on Optimal Cost

It is easy to observe that no vertex can be too far away from a vertex with weight one, as this distance will bound the cost of the optimal solution.

Lemma 3. *Let G be a graph with $\mathrm{OPT}_G = c$. For any vertex $v \in V(G)$ with weight 1 and any $u \in V(G)$, $d(u,v) \leq c/2$.*

Proof. For the sake of contradiction, assume that $d(u,v) > c/2$ for some $v \in V(G)$ with $\phi(v) = 1$ and $u \in V(G)$. Let u_i be an occurrence of u in W. Let v_j and v_k be the two consecutive occurrences of v on W with $j < i < k$. It is obvious that the sub-walk of W that lies between the two visits to v has length more than c. Since $\phi(v) = 1$, this contradicts the assumption that W has cost c. □

Corollary 1. *Let G be a metric graph with $\mathrm{OPT}_G = c$. Then the maximum edge weight in G is at most c.*

4 Properties of Min-Max Latency Walks

In this section, we characterize the optimal and approximate solutions of the min-max latency problem.

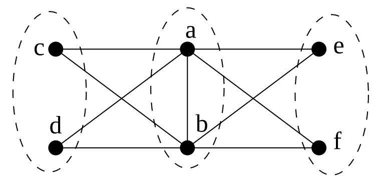

Fig. 1 The graph G as in proof of Lemma 4 with $n = 6$, $s = 2$, $V_1 = \{a,b\}$, $V_2 = \{c,d\}$ and $V_3 = \{e, f\}$. The walk that Algorithm 1 constructs would be $[[[a,b],c,[a,b],d],[a,b],e,[[a,b],c,[a,b],d],[a,b], f]$

4.1 Bounds on the Length of Kernel of an Optimal Walk

Here, we first show that the optimal solution for the min-max latency problem can be very large with respect to the size of the input graph.

Lemma 4. *There are infinitely many graphs for which any optimal walk has a kernel that is non-polynomial in the size of G.*

Proof. For any constant integer k and any multiple of it $n = sk$, we construct a graph G with unit weight edges and $|V(G)| = n$ and prove $\tau(G)$ to be in $\Omega(n^{k-1})$. Let $V_1,\ldots,V_k$ be a partition of $V(G)$ into k sets each having size s. Let there be a unit weight uv edge for any $u \in V_1$ and $v \in V_i$, where $i \in \{1,2,\ldots,k\}$. For each $v \in V_i$ where $1 \le i \le k$, let $\phi(v) = \frac{1}{(s+1)^i}$. We first prove $\mathrm{OPT}_G \le 1$. Let W be a walk constructed by Algorithm 1. It is easy to see that cost of $\Delta(W)$ is 1. The reason is that each vertex in V_i for $i \in \{1,2,\ldots,k-1\}$ has weight $\frac{1}{(s+1)^i}$ and is visited in $\Delta(W)$ every other $(s+1)^i$ steps. $i \in \{1,2,\ldots,k\}$. Also the vertices in V_k have weight $\frac{1}{(s+1)^k}$ and are visited every other $(s+1)^k-(s+1)^{k-1}$ steps. Therefore $c(\Delta(W),v)$ for any vertex v is 1.

We have proved $\mathrm{OPT}_G \le 1$. It remains to prove any infinite walk M in G with cost less or equal to 1 has a kernel of size $\Omega(n^{k-1})$. Let M_1 be a sub-walk of length s of M. Then all vertices of V_1 appear in M_1, otherwise there is a vertex v in V_1 that does not appear in M_1 (note that both cost of M and $|V_1|$ are equal to s). Therefore, $c(M,v) \ge (s+2) \times \frac{1}{s+1} > 1$. This means that after each visit to a member of V_i with $i > 1$, the next s vertices that are visited in M all belong to V_1.

Now we need to show that at most a single instance of vertices in $\bigcup_{j>i} V_j$ appears in any sub-walk of M of length $(s+1)^{i-1}-1$. To prove this we use induction on i. Let M' be a sub-walk of M with length $(s+1)^{i-1}-1$. We can partition the elements of M' into $s+1$ disjoint sub-walks of length $(s+1)^{i-2}-1$. By the induction hypothesis, we know that each part of this partition has at most a single instance of vertices in $\bigcup_{j>i-1} V_j$. Also, we know that all vertices of V_i appear in M', or else the vertex $v \in V_i$ that is not visited in M' would have cost $c(M,v) > 1$. Since there are s vertices in V_i and $s+1$ visits to vertices of $\bigcup_{j>i-1} V_j$ in M', there is at most a single visit to a vertex in $\bigcup_{j>i} V_j$ in M'. Since all vertices in V_k appear in the kernel of M, this means that the kernel of M has length at least $(s+1)^{k-1}-1$ which is in $\Omega(n^{k-1})$ since k is a constant. □

Corollary 2. *There is no polynomial time algorithm for the min-max latency problem.*

Algorithm 1 WalkMaker($\{V_1,\dots,V_{i-1},V_i\}$)

```
if i < 1 then
    return ∅
else
    W ← ∅
    for j = 1 → |V_i| do
        W ← [W, WalkMaker({V_1, ..., V_{i-1}})],
        W ← [W, WalkMaker({V_1, ..., V_{i-2}})],
        W ← [W, v]; where v is the j-th element in V_i
    end for
    return W
end if
```

Corollary 2 does not show exactly how hard the problem is. In fact, any algorithm that checks all possible walks to find the optimal solution will have complexity $\Omega(c^{e(|V(G)|)})$, where $c > 1$ and e grows faster than any polynomial function.

4.2 Binary Walks

We showed that any exact algorithm is not scalable with respect to the size of the input graph. Therefore, we turn our attention to finding walks that approximate OPT. We show there always exists a walk that has polynomial size and has a cost within a constant factor of the optimal walk. To obtain this result, we first need to define a special structure. Here we define a class of walks, and show that there are walks in this class that provide constant factor approximations.

Definition 2 (Binary Walks and Decompositions). Let G' be a relaxed graph and V_i be the set of vertices with weight $1/2^i$ in G'. Let S be the walk $[S_1,S_2,\dots,S_t]$, where $t = 2^{log_2 \rho_{G'}+1}$. We say S is a *binary* walk if for any $v \in V_i$ and $0 \le j < t/2^i$, v appears exactly once in $S_{j2^i+1}, S_{j2^i+2}, \dots, S_{(j+1)2^i}$, i.e., in each 2^i consecutive S_l's starting from S_{j2^i+1}, v appears exactly once. Also, we say that the tuple of walks $(S_1,S_2,\dots,S_t)$ is a *binary decomposition* of S.

It is easy to see that $t \le 2\rho_{G'}$. Also, each vertex appears in each S_i at most once, therefore length of each S_i is bounded by n. This means that S has length bounded by $2n\rho_{G'}$.

Lemma 5. *Let G' be a graph with relaxed weights. There is a binary walk W in G' of cost less than or equal to $2.5 \times \mathrm{OPT}_{G'}$. Moreover, since this walk is binary, it has length bounded by $2n\rho_{G'}$.*

Proof. Let $M' = (m'_1, m'_2, \dots)$ be an infinite walk of cost c in G'. Note that for any infinite walk, we can remove any prefix of it without increasing the cost of it. Let $M = (m_1, m_2, \dots)$ be an infinite walk of cost c obtained by removing some prefix of M' such that $\phi(m_1) = 1$. Based on M, we construct a binary walk W, such that the cost of $\Delta(W)$ is at most $2c$ as follows: Let a_0 be 0 and S_i be the sub-walk

$M(a_i+1, a_{i+1})$ such that a_{i+1} is the maximal index satisfying weight$(M(1,a_{i+1})) \leq ic$. Each S_i is a walk of weight at most c, such that the union of S_i's partitions M.

Now we modify the walks $S_1, S_2, \dots$ by omitting some of the instances of vertices in them. Let V_i be the set of vertices with weight $1/2^i$ in G'. Let $t = 2^{log_2 \rho_{G'}+1}$ as in definition of binary walks. For any vertex $u \in V_i$ and any number $0 \leq j < t/2^i$, omit all but one of the instances of u that appear in $S_{j2^i+1}, S_{j2^i+2}, \dots, S_{(j+1)2^i}$. There exists at least one such instance, otherwise a vertex u with weight $1/2^i$ exists that is not visited in an interval of weight larger than $c \times 2^i$, implying $c(M,u) > c$.

Let $S'_1, S'_2, \dots$ be the result of this modification, note that weight$(S_i) \geq$ weight(S'_i). Let S be $[S'_1, S'_2, \dots, S'_t]$. We claim that $\Delta(S)$ has cost at most $2c$. Let $u \in V_i$ be a vertex of G'. Then we know that u appears exactly once in $[S_{j2^i+1}, S_{j2^i+1}, \dots, S_{(j+1)2^i}]$, for any $0 < j$. Also, by the construction we have that for any j, k with $0 < j \leq k$, $[S'_j, S'_{j+1}, \dots, S'_k]$ has weight at most $c(k-j+1)$. Also since $\phi(m_1) = 1$, by Lemma 3 we know that for any $0 \leq k \leq j$, $[S'_j, S'_{j+1}, \dots S'_t, S'_1, S'_2, \dots, S'_k]$ has length at most $c((t-j+1)+0.5+k)$. This means weight$(\Delta(S)(a,b)) < 2^{i+1}c + 0.5c \leq (2.5)c2^i$, for any a and b that are the indices of two consecutive visits to u in $\Delta(S)$. Consequently, the cost $c(\Delta(S), u) \leq 2.5c$. ☐

Theorem 2. *In any graph G, there exists a closed walk W of length $O(n^2)$, where the cost of $\Delta(W)$ is less or equal to* $6 \times \mathrm{OPT}_G$.

Proof. Let G' be the relaxation of G and $U = \{u_1, u_2, \dots, u_{|U|}\}$ be the set of vertices in $V(G')$ with weights less than $1/2^{\lfloor \log n \rfloor + 2}$. Graph G'' is obtained by removing vertices in U from G'. Note that G'' is also a complete metric graph. Therefore $\rho_{G''} \leq 2^{\lfloor \log n \rfloor + 2} \leq 4n$. Let S be a binary walk in G'', with cost at most $2.5\mathrm{OPT}_{G''}$ as described in Lemma 5. Since $\rho_{G''} \leq 4n$, length of S is bounded by $2\rho_{G''} n \leq 8n^2$.

Now, we add the vertices in U to S in order to obtain a walk W that covers all vertices of G'. Let $v \in V(G')$ be a vertex with $\phi(v) = 1$. Let $(S_1, S_2, \dots, S_t)$ where $t = 2^{\lfloor \log n \rfloor + 2}$ be the binary decomposition of S. Let v_i be the i-th instance of v in S. Note that $t > 2n$, and thus v appears in S at least $2n$ times. For each $1 \leq i \leq |U|$ modify S by duplicating v_{2i} and inserting an instance of u_i between the two copies of v_{2i}. Let W be the resulting walk. We claim that the cost of $\Delta(W)$ is at most $2\mathrm{OPT}_{G'} + \mathrm{OPT}_G$.

Let $\phi(u)$ be $1/2^i \geq 1/2^{\lfloor \log n \rfloor + 2}$ in G'. Let a and b be the indices of two consecutive visits to u in $\Delta(W)$. Then there are at most 2^i instances of vertices in U in $W(a,b)$. This follows from the proof of Lemma 5, where we showed that $W(a,b)$ intersects at most 2^{i+1} members of $(S_1, S_2, \dots, S_t)$. Of these 2^{i+1} walks, at least half of them were not changed in W. Therefore, at most 2^i vertices of U lie between indices a and b of W. Also, we inserted the visits to the vertices of U at visits to v with $\phi(v) = 1$. Therefore, by Lemma 3 each of these new detours made to visit a member of U has weight at most $2(\mathrm{OPT}_G/2) = \mathrm{OPT}_G$. Also, by Lemma 5 we already know that $c(\Delta(S), u) \leq 2.5\mathrm{OPT}_{G'}$. Therefore, we already have that:

$$c(\Delta(W), u) < 2.5\mathrm{OPT}_{G'} + \mathrm{OPT}_G \tag{1}$$

Note that the extra 0.5 factor in Lemma 5 is due to the distance of the last vertex of S_t to the first element of S_1. However, this extra cost can be treated as one of the detours to vertices of U, as we avoided adding one of these detours to S_1 and S_t. This means that we have already accounted for this extra cost in the second part of the righthand side of the inequality 1. Consequently, we have $c(\Delta(W),u) < 2\mathrm{OPT}_{G'} + \mathrm{OPT}_G$. By Lemma 2 we have $\mathrm{OPT}_{G'} \leq \mathrm{OPT}_G$, therefore $c(\Delta(W),u) < 3\mathrm{OPT}_G$. Also by Lemma 2 we know that the cost of W in G would be less than $6\mathrm{OPT}_G$, concluding the proof. □

Lemma 6. *Any algorithm with guaranteed output size $O(n^2/k)$ has approximation factor $\Omega(k)$.*

Proof. Let ε be a very small positive number. Let the graph G be as follows:

- There are $n/2$ vertices of weight 1, called heavy vertices,
- There are $n/2$ vertices of weight ε, called light vertices,
- The heavy vertices are in a clique with edges of weight ε,
- There is an edge of weight 1 connecting any light vertex to any heavy vertex.

In G, any minimum cost infinite walk visits all heavy vertices between visits to two light vertices. This means that each heavy vertex is repeated $n/2$ times in any walk that expands into a minimum cost infinite walk. So far we have shown that any optimum solution has size $\Omega(n^2)$. Note that to reduce the size of the output by a factor k, we would need to visit at least k light vertices between two consecutive visits to some heavy vertex v. This means that a walk of length smaller than $\frac{n^2}{4k}$, has cost at least k, which is $k/2$ times the optimal solution $2 + (\varepsilon \times O(n^2))$. Therefore, any solution for the min-max latency in G with size $O(n^2/k)$ has approximation factor of $\Omega(k)$. This concludes the lemma. □

Lemma 6 directly gives that there is no constant factor approximation algorithm with guaranteed output size of $o(n^2)$. Note that this implies that Theorem 2 is tight in the sense that the size of the constructed kernel, can not be reduced except for a constant factor, maybe.

5 Approximation Algorithms for the Min-Max Latency Walk

In this section, we present two polynomial time approximation algorithms for the min-max latency problem. The approximation factor in the first algorithm is a function of the ratio of the maximum weight to the minimum weight among vertices. The approximation ratio of the second algorithm however, solely relies on the number of vertices in the graph.

5.1 An $O(\log \rho_G)$-Approximation Algorithm

A crucial requirement for our algorithms is a useful property regarding binary walks.

Lemma 7. *(Binary property) Let G' be a graph with relaxed weights. Let S be a binary walk in G' with the binary decomposition $(S_1, S_2, \ldots, S_t)$. Assume we know that* $\max_{1\le i\le t}(\text{weight}(S_i)) \le c$ *and for some vertex v, each S_i begins and ends in v. Then the cost of S is at most $2c$.*

Proof. Let S_j be $S_{(j \bmod t)}$. Let V_i be the set of vertices $u \in V(G')$ of weight $1/2^i$. Let $u \in V_i$ be a vertex of G'. Then we know that u appears exactly once in $[S_{j2^i+1}, S_{j2^i+1}, \ldots, S_{(j+1)2^i}]$ for any $0 < j$. Also, by the construction and Corollary 1, we have that for any $0 < j \le k$, $[S_j, S_{j+1}, \ldots, S_k]$ has weight at most $c(k-j+1)$. This means $\text{weight}(\Delta(S)(a,b)) \le 2c2^i$, for any a and b that are the indices of two consecutive visits to u in $\Delta(S)$. Consequently, the cost $c(\Delta(S), u) < 2c$. □

Here we define a useful tool. Let the function $\text{Partition}(W,k)$ be a function that gets a walk W and an integer k as input and returns a set of k walks $\{W_1, W_2, \ldots, W_k\}$ such that these walks partition vertices of W and also $\text{weight}(W_i) \le \text{weight}(W)/k$ for all $1 \le i \le k$. It is easy to see this is always doable in linear time.

Given a graph G, our first algorithm is guaranteed to find a solution within a factor of $O(log(1/\varepsilon))$ of the optimal solution, where ε is the smallest weight among the vertices.

Algorithm 2 $\text{BrutePartitionAlg}(G)$

Let V_i be the set of vertices of weight $\frac{1}{2^{i+1}} < w(u) \le \frac{1}{2^i}$ for $0 \le i \le \log_2 \rho_G$
Let t be $2^{\lfloor \log_2 \rho_G \rfloor + 1}$
$S_1, S_1, \ldots, S_t \leftarrow \emptyset$
for $i = 0 \to \lfloor \log_2 \rho_G \rfloor$ **do**
    $\{W_1, \ldots, W_{2^i}\} \leftarrow \text{Partition}(TSP(G[V_i]), 2^i)$
    for $j = 1 \to t$ **do**
        $S_j \leftarrow [S_j, W_x]$; where x is $j \bmod 2^i$,
    end for
end for
$S \leftarrow \emptyset$
for $i = 1 \to t$ **do**
    $S \leftarrow [S, S_i]$
end for
return S

Theorem 3. *Given a graph G, Algorithm 2 constructs a walk of length $O(n\rho_G)$ that is within $O(\log \rho_G)$ factor of the* OPT_G.

Proof. Let G' be the result of relaxing the weights of G. Let v be a vertex in G' with $\phi(v) = 1$. Let V_i be the set of vertices $u \in V(G')$ of weight $\frac{1}{2^i}$. Let t be the smallest power of two that is larger than ρ_G. Algorithm 2 constructs a binary walk $S = [S_1, S_1, \ldots, S_t]$ such that all S_i begin and end in v and $max_{1\le i\le t}\text{weight}(S_i) < 2(\lceil \log \rho_G \rceil)\text{OPT}_G$.

Assume addition and subtraction on the index of S_i is modulo t (e.g., S_{t+4} is the same as S_4). Here, in addition to the constraints defining a binary walk, we will be trying to satisfy another constraint: Each vertex in V_i appears in S_j through S_{j+2^i-1} exactly once. This condition will force a better behavior of S as it guarantees vertices to be visited more uniformly.

For minimizing the maximum weight S_j, we look at each V_i separately and try to minimize the maximum contribution of V_i to each S_j. Since there are at most $\log\rho_G$ sets V_i, this will give us an overhead approximation factor of $\log\rho_G$.

Let us look at V_i. We will construct 2^i closed walks beginning and ending in v, such that they cover V_i. Let W be the TSP tour of V_i. We showed that the best solution for min-max latency problem in graphs with uniform weight is the same as the TSP tour. Therefore $\mathrm{weight}(W)/2^i \leq \mathrm{OPT}_{G'}$. Let $W_1, W_2, \dots, W_{2^i}$ be a set of 2^i paths partitioning W such that the maximum of them is smaller than $\mathrm{OPT}_{G'}$. Construct W'_j by adding v to the both ends of W_j. By Lemma 3, this increases the weight of each W_j by at most $2(\mathrm{OPT}_{G'}/2) = \mathrm{OPT}_{G'}$. Therefore, each W'_j has weight at most $2\mathrm{OPT}_{G'}$. Appending each W'_j to S_{2^i+j} for all $0 \leq i \leq t$ will construct our desired solution. Note that since all W'_j end in v, we do not need to worry about concatenation of these walks. In the end, there will be $\lceil\log\rho_G\rceil$ closed walks in S_j each of weight at most $2\mathrm{OPT}_{G'}$. Therefore $\max_{1\leq j\leq t}(\mathrm{weight}(S_j)) \leq 2\lceil\log\rho_G\rceil\mathrm{OPT}_{G'}$. By Lemma 7 this means that S has cost $4\lceil\log\rho_G\rceil\mathrm{OPT}_{G'}$. Hence by Lemma 2 S has cost within $8\lceil\log\rho_G\rceil$ factor of the optimal solution for G. □

5.2 An $O(\log n)$-Approximation Algorithm

In many applications, the value ρ_G is independent of n. For example, in a monitoring scenario, there may be only a finite number of importance levels that can be assigned to a point of interest. In this case we have a constant factor algorithm. However, the ratio between largest and smallest weights ρ_G does not directly depend on the size of the input graph. For even a small graph, ρ_G can be very large. Therefore, in such cases we need an algorithm with an approximation guarantee that is bounded by a function of the size of the graph. Next we present an approximation algorithm for min-max latency problem that is guaranteed to find a solution within logarithmic factor of the optimal solution.

Theorem 4. *Given a graph G, Algorithm 3 constructs a walk of length $O(n^2)$ that is within $O(\log n)$ factor of the* OPT_G.

Proof. The idea is to remove the vertices of small weight so that we can use Algorithm 2 as a subroutine. Let G' be the result of relaxing the weights of G and U be the set of vertices of G' with weight at most $1/2^{\lfloor\log n\rfloor+1}$. Let G'' be the result of removing vertices in U from G'. Assume $S = [S_1, S_1, \dots, S_t]$ is the result of running Algorithm 2 on G'' with $\rho_{G''} = 2^{\lfloor\log n\rfloor+2} < 4n$. Add the i-th vertex of U at the end of S_{2i}. Note that since $|U| < n-1$ and $2n < t$, this is possible. Let $S' = [S'_1, S'_1, \dots, S'_t]$ be the result of this modifications. Each walk S_i begins and ends in v, where $\phi(v) = 1$. Therefore, by Lemma 3 each detour to a vertex in U has weight bounded by OPT_G. Also, by the proof of Theorem 3, each S_i has weight at most $(2\log n + 2)\mathrm{OPT}_{G'}$.

This means that each S'_i has weight at most $(2\log n+3)\mathrm{OPT}_G$. By Lemmas 2 and 7, this means that the cost of $\Delta(S')$ in G is bounded by $(8\log n+12)\mathrm{OPT}_G$. □

Algorithm 3 SmartPartitionAlg(G)

Let V_i be the set of vertices of weight $\frac{1}{2^{i+1}} < w(u) \leq \frac{1}{2^i}$ for $0 \leq i \leq \log \rho_G$
Let v be an element with weight 1
$U \leftarrow \cup_{i>\lfloor \log n \rfloor} V_i$
$S \leftarrow$ BrutePartitionAlg($G[V \setminus U]$)
$i \leftarrow 0$
for all $u \in V_k$ where $k > \lfloor \log n \rfloor$ **do**
    Insert u after the $(2i)$-th instance of v in S
    Increment i
end for
return S

6 Simulations

In this section, we present simulation results for the two approximation algorithms presented in Section 5. As Algorithm 3 always performs better than Algorithm 2 both in runtime and approximation factor, we will be studying the performance of Algorithm 3.

For the simulations, we use test data that are standard benchmarks for testing performance of heuristics for calculating TSP tour. The data sets used here are taken from [7]. Each data set represents a set of locations in a country. We construct a graph by placing a vertex for each locations and letting the distance of any pair of vertices be the Euclidean distance of the corresponding points. Unfortunately, no information regarding each individual location was available. Such information could be used to assign weights of the vertices of the graph. For example, if the population of each city was also available, it would have made a meaningful measure for the weights of the vertices.

In many applications of the min-max latency problem—such as monitoring or inspection—the likelihood of a vertex with very high weight is low. In other words, majority of vertices have low priority, while few vertices need to be visited more frequently. To simulate this behavior, we use a distribution that has the following exponential property:

$$\mathrm{P}[(1/2)^{k+1} < \phi(v) \leq (1/2)^k] = 1/B \qquad (2)$$

for $k < B$ where B is a fixed integer. If we assign to a vertex v the weight $(1/2)^B \leq \phi(v) \leq 1$ with probability $f(\phi(v)) = (\phi(v)B\ln 2)^{-1}$ the exponential property holds.

Here, we compare our algorithms to the simple algorithm of following a TSP tour through all vertices in G. For finding an approximate solution for TSP we used an implementation of the Lin-Kernighan algorithm [11] available at [7]. Relative

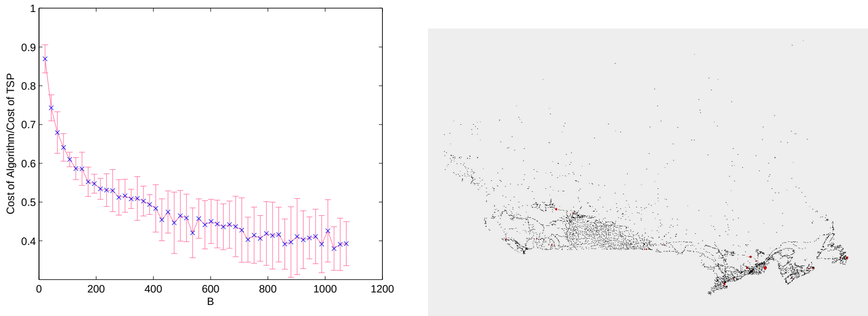

Fig. 2 (Left) The ratio of the cost of the walk produced by Algorithm 3 to the cost of the TSP for different values of B. (Right) The 4663 vertex graph used for all tests corresponding to all cities in Canada [7].

to other heuristics we test for the min-max latency problem, the cost of TSP tour is low when the weights are distributed uniformly. One of the reasons for this is that when weights are uniform, ρ_G grows proportional to $\log n$. This means that by rounding all weights to ε and calculating the TSP we can obtain a solution of expected approximation factor of $\log n$. However, it is easy to construct a graph G in which the cost of the TSP tour of G can be $\Omega(n) \times \mathrm{OPT}_G$.

6.1 Performance with respect to Vertex Weight Distribution

An important aspect of an environment is the ratio of weight of the elements, therefore it is natural to test our algorithm with respect to ρ_G. Note that $\rho_G > (1/2)^B$. Therefore, we consider different values of B to assess the performance of the algorithm for different ranges of weights on the same graph (see Figure 2). It is easy to see that if $B = 1$, then Algorithm 3 returns the TSP tour of the graph. Also, if $B < \log n$, then Algorithms 3 and 2 behave the same. Figure 2 depicts the behavior of Algorithm 3 on a graph induced by 4663 cities in Canada with different values for B. It can be seen that for larger B our algorithm outperforms the TSP tour by a greater factor.

6.2 Performance with respect to Input Graph Size

Here, we use graphs of different sizes to evaluate the performance of our algorithms. Again, the cost is compared to that of a simple TSP tour that visits each vertex in the graph once. Figure 3 depicts the ratio of the cost of the walk constructed by Algorithm 3 to the cost of the TSP tour, on 27 different graphs each corresponding to a set of locations in a different country. Here B is fixed. It can be seen that the ratio of the cost of the TSP tour to the cost of the walk produced by our algorithm increases as the size increases.

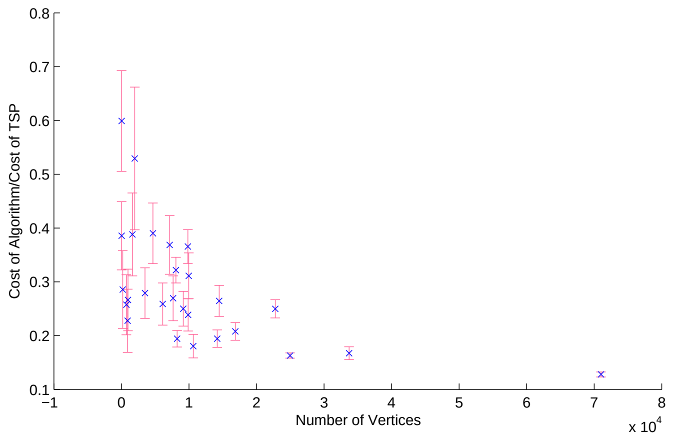

Fig. 3 The ratio of the cost of Algorithm 3 to the cost of the TSP on the 27 test graphs in [7].

Also, the time complexity of the algorithm is $O(n^2 + \beta(n))$ where $\beta(n)$ is the running time of the algorithm used for finding the TSP tour. For the test data corresponding to 71009 locations in China, our Java implementation of Algorithm 3 constructs an approximate solution in 20 seconds using a regular laptop with a 2.50 GHz CPU and 3 GB RAM.

7 Conclusions and Future Work

In this paper, we considered the problem of planning a path for a robot to monitor a known set of features of interest in an environment. We represent the environment as a vertex- and edge-weighted graph and we addressed the problem of finding a closed walk that minimizes the maximum weighted latency of any vertex. We showed several results on the existence and non-existence of optimal and constant factor approximation solutions. We then provided two approximation algorithms; an $O(\log n)$-approximation and an $O(\log \rho_G)$-approximation, where ρ_G is the ratio between the maximum and minimum vertex weights. We also showed via simulation that our algorithms scale to very large problems consisting of thousands of vertices.

For future work there are several directions. We continue to seek a constant factor approximation algorithm, independent of ρ_G. We also believe that by adding some heuristic optimizations to the walks produced by our algorithms, we could significantly improve their performance in practice. Finally, we are currently looking at ways to extend our results to multiple robots. One approach we are pursuing is to equitably partition the graph such that the single robot solution can be utilized for each partition.

Acknowledgements. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Bullo, F., Frazzoli, E., Pavone, M., Savla, K., Smith, S.L.: Dynamic vehicle routing for robotic systems. Proceedings of the IEEE **99**(9), 1482–1504 (2011)
2. Caffarelli, L., Crespi, V., Cybenko, G., Gamba, I., Rus, D.: Stochastic distributed algorithms for target surveillance. In: Intelligent Systems and Design Applications, pp. 137–148. Tulsa, OK (2003)
3. Cannata, G., Sgorbissa, A.: A minimalist algorithm for multirobot continuous coverage. IEEE Transactions on Robotics **27**(2), 297–312 (2011)
4. Chevaleyre, Y.: Theoretical analysis of the multi-agent patrolling problem. In: IEEE/WIC/ACM Int. Conf. Intelligent Agent Technology, pp. 302–308. Beijing, China (2004)
5. Choset, H.: Coverage for robotics – A survey of recent results. Annals of Mathematics and Artificial Intelligence **31**(1-4), 113–126 (2001)
6. Christofides, N., Beasley, J.E.: The period routing problem. Networks **14**(2), 237–256 (1984)
7. Cook, W.: National travelling salesman problem (2009). Available at `http://www.tsp.gatech.edu/index.html`
8. Elmaliach, Y., Agmon, N., Kaminka, G.A.: Multi-robot area patrol under frequency constraints. In: IEEE Int. Conf. on Robotics and Automation, pp. 385–390. Roma, Italy (2007)
9. Gabriely, Y., Rimon, E.: Competitive on-line coverage of grid environments by a mobile robot. Computational Geometry: Theory and Applications **24**(3), 197–224 (2003)
10. Hussein, I.I., Stipanoviċ, D.M.: Effective coverage control for mobile sensor networks with guaranteed collision avoidance. IEEE Transactions on Control Systems Technology **15**(4), 642–657 (2007)
11. Lin, S., Kernighan, B.: Effective heuristic algorithm for the traveling salesman problem. Operations Research **21**, 498–516 (1973)
12. Michael, N., Stump, E., Mohta, K.: Persistent surveillance with a team of mavs. In: IEEE/RSJ Int. Conf. on Intelligent Robots & Systems, pp. 2708–2714. San Francisco, CA (2011)
13. Papadimitriou, C.H., Yannakakis, M.: The traveling salesman problem with distances one and two. Math. Oper. Res. **18**, 1–11 (1993). DOI 10.1287/moor.18.1.1. URL http://dl.acm.org/citation.cfm?id=154540.154541
14. Pasqualetti, F., Franchi, A., Bullo, F.: On optimal cooperative patrolling. In: IEEE Conf. on Decision and Control, pp. 7153–7158. Atlanta, GA, USA (2010)
15. Smith, R.N., Schwager, M., Smith, S.L., Rus, D., Sukhatme, G.S.: Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. Journal of Field Robotics **28**(5), 714–741 (2011)
16. Smith, S.L., Rus, D.: Multi-robot monitoring in dynamic environments with guaranteed currency of observations. In: IEEE Conf. on Decision and Control, pp. 514–521. Atlanta, GA (2010)
17. Tiwari, A., Jun, M., Jeffcoat, D.E., Murray, R.M.: Analysis of dynamic sensor coverage problem using Kalman filters for estimation. In: IFAC World Congress. Prague, Czech Republic (2005)
18. Tulabandhula, T., Rudin, C., Jaillet, P.: Machine learning and the traveling repairman (2011). Available at `http://http://arxiv.org/abs/1104.5061`