The International Journal of
Robotics Research
2014, Vol 33(1) 138–154
© The Author(s) 2013
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364913504011
ijr.sagepub.com
SAGE

Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations

Soroush Alamdari[1], Elaheh Fata[2] and Stephen L Smith[2]

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 graph with vertex weights and edge lengths. The vertices represent regions of interest, edge lengths give travel times between regions and the vertex weights give the importance of each region. As the robot repeatedly performs a closed walk on the graph, we define the weighted 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 is to find a closed walk that minimizes the maximum weighted latency of any vertex. We show that there does not exist a polynomial time algorithm for the problem. We then provide two approximation algorithms; an $O(\log n)$-approximation algorithm and an $O(\log \rho_G)$-approximation algorithm, where ρ_G is the ratio between the maximum and minimum vertex weights. We provide simulation results which demonstrate that our algorithms can be applied to problems consisting of thousands of vertices and a case study for patrolling a city for crime.

1. Introduction

An emerging application area for robotics is in performing long-term monitoring tasks. Some example monitoring tasks include (a) environmental monitoring such as ocean sampling (Smith et al., 2011), where autonomous underwater vehicles sense the ocean to detect the onset of algae blooms; (b) surveillance (Michael et al., 2011), where robots repeatedly visit vantage points in order to detect events or threats and (c) infrastructure inspection such as power-line or manhole cover inspection (Tulabandhula et al., 2011), where spatially distributed infrastructure must be repeatedly inspected for the presence of failures. For such tasks, a key problem is the high-level path planning problem of determining 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 graph with vertex weights and edge lengths: the *min–max weighted latency walk problem*. The vertices represent regions (or features) of interest that must be repeatedly observed by a robot. The edge lengths give travel times between regions and the vertex weights give the importance of each region. A vertex is observed by the robot once it is reached. Given a robot *walk*[1] on the graph, the *weighted 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 weighted latency over all vertices. In an infrastructure task, this would be akin to minimizing the expected number of failures that occur in any region prior to a robot visit.

Prior work: The problem of continuously (or persistently) monitoring an environment using mobile robots has been studied in the literature under several names, including continuous sweep coverage; patrolling; persistent surveillance; and persistent monitoring. The basis of all of these problems is sweep coverage (Choset, 2001), where a robot must move through the environment so as to cover the entire region with its sensor. Variants of sweep coverage include on-line coverage (Gabriely and Rimon, 2003), where the robot has no a priori information about the environment and dynamic coverage (Hussein and Stipanovìc, 2007), where

[1] Department of Computer Science, Cornell University, Ithaca, NY, USA
[2] Department of Electrical and Computer Engineering, University of Waterloo, Canada

Corresponding author:
Stephen L Smith, Department of Electrical and Computer Engineering, University of Waterloo, 200 University Avenue, West Waterloo, ON N2L 3G1, Canada.
Email: stephen.smith@uwaterloo.ca

each point in the environment requires a pre-specified "amount" of coverage.

One approach to persistent monitoring has been to focus on randomized approaches in discrete environments. These works frequently use the name of continuous sweep coverage. In Tiwari et al. (2005), a continuous coverage problem is considered where a sensor must continually survey regions of interest by moving according to a Markov chain. In Cannata and Sgorbissa (2011), 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. In Arvelo et al. (2012), the authors look at robots modeled by controlled Markov chains and seek to persistently monitor regions while avoiding forbidden regions.

The other main approach to persistent monitoring is to cast the problem as one in combinatorial optimization. This research typically falls under the name of patrolling or persistent surveillance. In the most basic problem, a robot seeks to minimize the time between visits to each point in space. For this problem, both centralized approaches (Chevaleyre, 2004; Elmaliach et al., 2007; Nigram and Kroo, 2008) and distributed algorithms for multiple robots (Pasqualetti et al., 2012b) have been proposed. Recently, there has been work on minimizing the weighted time between visits to each region of interest, where the weight of a region captures its priority relative to other regions (Pasqualetti et al., 2012a; Smith et al., 2012). These works focus on controlling robots along predefined paths. In Smith et al. (2012), the authors consider velocity controllers for persistent monitoring along fixed tours, while in Pasqualetti et al. (2012a), the authors focus on coordination issues for multiple robots on fixed tours. An optimal control formulation for persistent monitoring in one-dimensional spaces is given in Cassandras et al. (2011).

In our prior work (Smith and Rus, 2010), we considered a specific case of the min–max weighted latency walk problem on Euclidean graphs, where the graphs are constructed by distributing vertices in a Euclidean space according to a known probability distribution. Under these assumptions, constant factor approximation algorithms are developed for the limiting case when the number of vertices becomes very large.

The combinatorial approaches taken in persistent monitoring problems often draw from solutions to vehicle routing (Christofides and Beasley, 1984; Laporte, 2009) and dynamic vehicle routing (DVR) problems (Bullo et al., 2011). Stump and Michael (2011) make the connection between multi-robot persistent surveillance and the vehicle routing problem with time windows (Laporte, 2009). In Tulabandhula et al. (2011), the authors consider a preventative maintenance problem in which the input is a vertex and edge-weighted graph, as in the min–max weighted latency walk problem, but the output is a path 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 weighted latency walk problem can be thought of as a generalization of preventative maintenance, where the maintenance and inspection should continually be performed.

Other work in persistent monitoring and surveillance includes Bethke, How et al. (2008) and Bethke, Redding et al. (2010), where a persistent task is defined as one whose completion takes much longer than the life of a robot. The authors focus on issues of battery management and recharging. Such issues have also been recently considered in the context of persistent monitoring in Mathew et al. (2013).

Contributions: There are four main contributions of this paper. First, we introduce the general min–max weighted 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 algorithms and approximation algorithms for the problem. We show that in general, the optimal walk can be very long—its size can be exponential 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 size in $O(n^2)$, where n is the number of vertices in the input graph. Third, we provide two approximation algorithms for the problem. Defining ρ_G to be the ratio between the maximum and minimum vertex weights in the input graph G, we give an $O(\log \rho_G)$ 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_G. 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. Fourth and finally, we show in simulation that we can solve large problems consisting of thousands of vertices and we demonstrate our algorithm on a case study of patrolling the city of San Francisco, California, for crime.

A preliminary version of this paper appeared as (Almadari et al., 2012). Compared to the conference version, this version presents detailed proofs of all statements, new results on the existence of optimal finite walks, additional remarks and illustrative examples and a new case study on patrolling in the simulations section.

Organization: This paper is organized as follows. In Section 2, we give the essential background on graphs and formalize the min–max weighted 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 size 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 perform a case study in patrolling for crime. Conclusions and future directions are presented in Section 7.

2. Background and problem statement

In this section, we review graph terminology and define the min–max latency walk problem.

2.1. Background on graphs

Vertex-weighted and edge-weighted graphs: The vertex set and edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively, such that $E(G) \subseteq V(G) \times V(G)$. The number of vertices in a graph G, i.e. $|V(G)|$, is called the *size of graph* G and is denoted by n. An edge in $E(G)$ is referred to as (v_i, v_j) or v_iv_j. We consider only *undirected* graphs, meaning $(v_i, v_j) \in E(G)$ if and only if $(v_j, v_i) \in E(G)$. An edge-weighted graph G associates a finite weight $l(e) > 0$ to each edge $e \in E(G)$. We will refer to the weight of an edge as its *length*. A vertex-weighted graph G associates a positive finite *weight* $\phi(v) \in (0, 1]$ to each vertex $v \in V(G)$. 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 $V(G) \setminus V'$. 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 weights, i.e.

$$\rho_G = \max_{v_i, v_j \in V(G)} \frac{\phi(v_i)}{\phi(v_j)}$$

Walks in graphs: A walk of *size* k in a graph G is a sequence of vertices $(v_1, v_2, \dots, v_k)$ such that for any $1 \leq i \leq k$ we have $v_i \in V(G)$ (with the possibility that $v_i = v_j$ for some $1 \leq i, j \leq k$) and there exists an edge $v_jv_{j+1} \in E(G)$ for each $1 \leq j < k$. The graphs we study are assumed to be *connected*, meaning that for any pair of vertices s and t there is a walk from s to t. A walk is *closed* if its beginning and end vertices are the same. The *length* of a walk $W = (v_1, \dots, v_k)$, denoted by $l(W)$, is defined as the sum of the length of edges of graph G that appear in that walk, i.e.

$$l(W) = \sum_{i=1}^{k-1} l(v_iv_{i+1})$$

Given a walk $W = (v_1, \dots, v_k)$ and integers $1 \leq i \leq j \leq k$, the *sub-walk* $W(i, j)$ is defined as the subsequence of W given by $W(i, j) = (v_i, v_{i+1}, \dots, v_j)$. Given the walks $W_1, W_2, \dots, W_l$, the walk $W = [W_1, W_2, \dots, W_l]$ is the result of *concatenation* of W_1 through W_l, while preserving order.

The traveling salesman problem: A *tour* of a graph G is a closed walk $T = (v_1, v_2, \dots, v_{n+1})$ starting and ending at some vertex $v_1 = v_{n+1}$ that visits all n vertices in G. Thus, a tour visits each vertex exactly once and then returns to its start vertex. The *Traveling Salesman Problem* (TSP) is to find a tour in G of minimum length. We refer to a solution of the TSP as a *TSP tour*. We denote one such TSP tour by TSP(G). Also, we denote the first n vertices of TSP(G) by TSP-Path(G); that is, the tour TSP(G) with the last edge dropped. Thus, TSP-Path(G) is a walk that visits each vertex exactly once (note, however, that it is not necessarily the shortest walk and thus should not be confused with the minimum Hamiltonian path). The length of TSP-Path(G) is upper-bounded by the length of TSP(G).

Infinite walks in graphs: An *infinite walk* is a sequence of vertices, $(v_1, v_2, \dots)$, such that there exists an edge $v_iv_{i+1} \in E(G)$ for each $i \in \mathbb{N}$. We say that a walk W *expands* to an infinite walk $\Delta(W)$ if $\Delta(W)$ is constructed by an infinite number of copies of W concatenated together, i.e. $\Delta(W) = [W, W, \dots]$. It can be seen that for any walk W, there exists a unique expansion to an infinite walk. The *kernel* of an infinite walk W, denoted by $\delta(W)$, is the shortest walk such that W is the *expansion* of $\delta(W)$. It is easy to observe that there are infinite walks for which a finite size 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 $G = (V, E)$ be a graph with edge lengths l and vertex weights ϕ. In a robotic monitoring application, the graph G can be obtained as a discrete abstraction of the environment, with vertices corresponding to regions of the environment and edge lengths corresponding to travel distances (or times) between regions. The vertex weights on the graph give the relative importance of the regions for the monitoring task. Given an infinite walk W for the robot in G, we define the *latency* of vertex v on walk W, denoted by $L(W, v)$, as the maximum length of the sub-walk between any two consecutive visits to v on W. The latency of a vertex v in walk W corresponds to the maximum time between observations of the region represented by v.

Then, we can define the weighted latency, or *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, is then

$$C(W) := \max_{v \in V(G)} C(W, v)$$

Therefore, the cost of a robot walk on a graph is the maximum weighted latency over all vertices in the graph. This corresponds to the maximum importance-weighted time between observations of any region. The min–max weighted latency walk problem can be stated as follows.

The min–max weighted latency walk problem: Find an infinite walk W that minimizes the cost $C(W)$.

For brevity, we will refer to this problem as the *min–max latency walk problem* in the rest of the paper.

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 there always exists a minimum cost walk. We define OPT_G to be the minimum cost among all infinite walks on G. An infinite walk W is an optimal walk if $C(W) = \mathrm{OPT}_G$. In Lemma 2.1, we first show that such an infinite walk always exists. We then show that there always exists a finite size kernel realizing OPT_G; that is, there is a walk with cost equal to OPT_G that consists of an infinite number of repetitions of a finite size walk.

Lemma 2.1. *For any graph G, the following two claims hold:*

(i) There exists a walk of minimum cost.
(ii) There is a walk of minimum cost that has a finite size kernel.

Proof. (i) Suppose that there does not exist a walk of minimum cost in G. There are two ways in which this could happen. Either every walk in G has infinite cost, or there exists an infinite sequence of walks $W_1, W_2, \dots$ with costs $C(W_1) \geq C(W_2) \geq \cdots$, such that $\lim_{i\to\infty} C(W_i) = c^*$, but there is no walk in G attaining the cost c^*. Thus, to prove the result we will eliminate each of these cases.

First, let $\overline{W}$ be any walk of size n in G that visits all vertices in $V(G)$. By connectivity of G we know that such $\overline{W}$ always exists and hence the cost $C(\Delta(\overline{W}))$ is necessarily finite. Next, we show that there are only a finite number of different values $c' < C(\Delta(\overline{W}))$ that can be costs of walks in G. Since the length of each edge is positive, for any vertex $v \in V(G)$ there are a finite number of walks beginning in v with length less than $C(\Delta(\overline{W}))/\phi(v)$. Therefore, there are a finite number of possible values for the latency of v that are less than $C(\Delta(\overline{W}))/\phi(v)$. Hence, there are a finite number of possible costs for vertex v on a walk of cost less than $C(\Delta(\overline{W}))$. Moreover, since there are n vertices in G, a walk in G can only have a finite number of different costs. As a result, there exists a walk of minimum cost for graph G.

(ii) We now prove that the second claim also holds utilizing the first claim in the lemma. Assume W is an optimal walk with cost OPT_G. Note that W is an infinite walk. Let v be a vertex in $V(G)$. Let W_1 be a sub-walk of W starting at v with length larger than $\mathrm{OPT}_G \times \rho_G$. Since $l(W_1) > \mathrm{OPT}_G \times \rho_G$, every vertex of G is visited at least once in W_1; otherwise $C(W) \geq l(W_1)/\rho_G > \mathrm{OPT}_G$. Let U be the set of all possible walks in G starting at v with lengths between $l(W_1)$ and $l(W_1) + \max_{e\in E(G)} l(e)$. Since the edge lengths are positive and finite, the size of U is finite.

Let i be the index of a visit to v in W. Then, for every walk $W(i,j)$ of length at least $l(W_1)$, there exists a sub-walk starting at $W(i,i)$ that is in U. Due to the fact that W is an infinite walk and hence v appears an infinite number of times in W, there exists a walk $W' \in U$ that appears at least twice in W. Let W'' be such that $[W', W'', W']$ is a sub-walk of W. Note that $[W', W'']$ is a finite walk and so $\Delta([W', W''])$ is a walk with a finite size kernel. We now claim that $\Delta([W', W''])$ is also an optimal walk for G. Consider any two consecutive instances of a vertex $u \in V(G)$ in $\Delta([W', W''])$. For these two instances of u, one of the following two cases occurs: (a) the two instances are in the same copy of $[W', W'']$, or (b) the two instances are in consecutive copies of $[W', W'']$. However, since $[W', W'', W']$ is a sub-walk of W, both cases occur in optimal walk W, and thus $\Delta([W', W''])$ is also an optimal walk for G. □

Next we will show that the problem of min–max latency walk is APX-hard, implying that there is no polynomial-time approximation scheme (PTAS) for it, unless P = NP. However, before that we need to introduce the following definitions.

A *complete graph* is a graph in which each pair of its vertices are connected by an edge. A graph is called a *metric graph* if (a) it is a complete undirected graph and (b) for any three vertices $u, v, w \in V(G)$ we have $l(uw) \leq l(uv) + l(vw)$ (*triangle inequality*) (Vazirani, 2004).

Theorem 2.2. *The min–max latency walk problem is APX-hard.*

Proof. The reduction is from the *metric Traveling Salesman Problem (TSP)*. Recall that the TSP is the problem of finding the shortest closed walk that visits all vertices exactly once (except for its beginning vertex). Such a walk is referred to as a TSP tour. The problem of finding TSP tours in metric graphs is called the metric TSP. It is known that the metric TSP is APX-hard [22] and it is approximable within a factor of 1.5. Here we show a reduction from the metric TSP to the min–max latency walk problem that preserves the hardness of approximation.

Let G be the input of the metric TSP. Assign weight 1 to all vertices of G. Assume W is an infinite walk with optimal cost OPT_G in G. Let M be a closed walk that is an optimal solution for TSP in G with $l(M) = c'$. We prove $c' = \mathrm{OPT}_G$. Since each vertex is visited exactly once in M, the cost of $\Delta(M)$ is c'. However, due to optimality of W we have $\mathrm{OPT}_G \leq c'$. It remains to show that $c' \leq \mathrm{OPT}_G$.

Let $v \in V(G)$ be a vertex with $C(W, v) = \mathrm{OPT}_G$ and i and j be the indices of two consecutive instances of v with $l\big(W(i,j)\big) = \mathrm{OPT}_G$. Since the weight of every vertex is 1 and $l\big(W(i,j)\big) = \mathrm{OPT}_G$, all vertices of G appear in $W(i,j)$; otherwise $C(W) > \mathrm{OPT}_G$. Consider the spanning tour T that is obtained from $W(i,j)$ by removing all but one of the instances of each vertex. Since we only remove vertices and due to the triangle inequality we have $l(T) \leq l\big(W(i,j)\big)$. Since T visits each vertex of G exactly once, it is a candidate solution for TSP and hence we have $l(M) \leq l(T)$. Therefore, $c' = l(M) \leq l\big(W(i,j)\big) = \text{OPT}_G$. 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. □

For a graph G and two vertices $u, v \in V(G)$, the shortest-path distance between u and v is denoted by $d(u, v)$. We focus on solving the min–max latency walk problem only for 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 edge uv in G' has length equal to the shortest-path distance from u to v in G, i.e. $l(uv) = d(u, v)$. Then, we construct a walk for G based on a walk in G' by replacing each edge uv with the shortest path connecting u and v in G. Since $\text{OPT}_G = \text{OPT}_{G'}$ 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 (Vazirani, 2004). It should be noted that to aid the presentation in some examples we show non-complete graphs with the understanding that we are referring to their metric closures.

In the proof of Theorem 2.2, we gave a reduction from the TSP to the min–max latency walk problem. However, in general the TSP tour is not a good approximation for the min–max latency walk problem.

Lemma 2.3. *The cost of a TSP tour of G can be larger than* $(n-1)\,\text{OPT}_G$.

Proof. Let G be a graph of n vertices constructed as follows:

- There is a vertex v with weight 1.
- There are $n-1$ vertices, $v_1, v_2, \ldots, v_{n-1}$, each having weight $1/n$.
- There exists an edge connecting v_i to v with length 1 for any $1 \leq i < n$.
- There exists an edge connecting v_i to v_{i+1} with length 2 for any $1 \leq i < n-1$ (see Figure 1).

It is easy to see that the triangle inequality holds for edge lengths in G. The TSP tour has length $2n-2$ and hence the cost of the TSP tour is $2n-2$. However, the cost of the walk that only uses edges of unit weight and visits v at every other index would be 2. This means that the cost of TSP can be as bad as $(n-1)$ times the cost of the optimal walk. □

In the following sections we seek better approximation algorithms for the min–max latency walk problem.

3. Relaxations and simple bounds

In this section, we present a relaxation of the min–max latency walk problem and two simple bounds based on the lengths of the edges of the input graph.

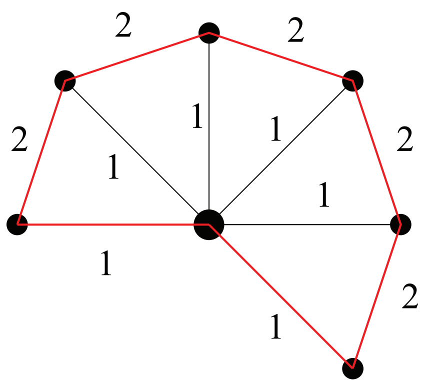


Fig. 1. The graph G as in Lemma 2.3 with $n = 7$. The cost of the TSP tour (red thick edges) in this graph is $2n - 2 = 12$. Note that if the weight of all vertices except the middle vertex is small, there is a walk that has cost 2.

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. A similar relaxation has been used in both Gørtz et al. (2011) and Smith and Rus (2010).

Definition 3.1 (Weight relaxation)**.** We say weights of vertices of graph G are *relaxed*, if for any vertex $v \in V(G)$ we update its weight $\phi(v)$ to $\phi'(v) = \frac{1}{2^x}$ such that x is the smallest integer for which $\frac{1}{2^x} \leq \phi(v)$ holds.

Lemma 3.2. *(Relaxed vertex weights) For a graph G' obtained by relaxing the weights of vertices of G the following statements hold:*

(i) If a walk W has cost c in G and cost c' in G', then $c' \leq c < 2c'$.

(ii) $\text{OPT}_{G'} \leq \text{OPT}_G < 2\text{OPT}_{G'}$.

Proof. (i) The weight of each vertex in G' is less than or equal to the weight of that vertex in G, while the lengths of the corresponding edges are the same. Hence, for costs of W in G and G' we have $c' \leq c$. Moreover, the weight of each vertex in G' is more than half of the weight of the same vertex in G. This results in $c < 2c'$. Consequently, we have $c' \leq c < 2c'$.

(ii) Let W and W' be optimal walks in G and G', respectively. For cost of W in G', denoted by c, we have $\text{OPT}_{G'} \leq c$. Also, (i) results in $c \leq \text{OPT}_G < 2c$. Consequently, we have $\text{OPT}_{G'} \leq \text{OPT}_G$. Similarly, for cost of W' in G, denoted by c', we have $\text{OPT}_G \leq c'$. Moreover, by (i) it follows that $\text{OPT}_{G'} \leq c' < 2\text{OPT}_{G'}$ and hence $\text{OPT}_G < 2\text{OPT}_{G'}$. Therefore, we have $\text{OPT}_{G'} \leq \text{OPT}_G < 2\text{OPT}_{G'}$. □

The reason for considering this relaxation is as follows. Given a relaxed graph G', we can define V_i to be all vertices in G' with weight $1/2^i$. Then, in order for the vertices in V_i and V_{i+1} to have the same weighted latency, each vertex in

V_i should be visited twice as often as each vertex in V_{i+1} in a walk on G'. This observation gives us some structure that we can exploit in our search for approximation algorithms for walks on G'. By Lemma 3.2 (ii), an α-approximation algorithm on G' would yield a 2α-approximation algorithm on the unrelaxed graph G.

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 1, as this distance will bound the cost of the optimal solution.

Lemma 3.3. *Let G be a metric graph. The following two properties hold for the edge lengths in G:*

(i) For any edge uv in which v has weight 1, we have $l(uv) \leq \mathrm{OPT}_G/2$.
(ii) The maximum edge length in G is at most OPT_G.

Proof. (i) By way of contradiction, assume that $l(uv) > \mathrm{OPT}_G/2$ for some $u, v \in V(G)$ such that $\phi(v) = 1$. Let W be an optimal walk in G, i.e. $C(W) = \mathrm{OPT}_G$. Let u_i be an occurrence of u in W. Let v_j and v_k be the two consecutive occurrences of v preceding and succeeding u_i in W, respectively. Since G is metric, by the triangle inequality we have that the sub-walk of W that lies between v_j and u_i has length at least $l(uv)$. A similar argument holds for the sub-walk of W that lies between u_i and v_k. Therefore, the sub-walk of W that lies between v_j and v_k has length at least $2l(uv)$ which is greater than OPT_G. However, since $\phi(v) = 1$, this contradicts the assumption that W has cost OPT_G.

(ii) Consider an edge uv with both $\phi(u)$ and $\phi(v)$ less than unit (if $\phi(u)$ or $\phi(v)$ is equal to one, then the edge length is bounded by $\mathrm{OPT}_G/2$, by part (i)). Let w be a vertex in $V(G)$ such that $\phi(w) = 1$. Note that since G is a metric graph, w is connected to both u and v. From part (i) of the lemma we have that $l(uw), l(vw) \leq \mathrm{OPT}_G/2$. Moreover, since G is a metric graph it can be concluded that $l(uv) \leq l(uw) + l(vw) \leq \mathrm{OPT}_G$. □

4. Properties of min–max latency walks

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

4.1. Bounds on size of kernel of an optimal walk

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

Lemma 4.1. *There are infinitely many graphs for which any optimal walk has a kernel that is at least exponential in the size of G.*

Proof. For any constant integer k and any integer multiple of it $n = sk$, we construct a graph G with unit length edges

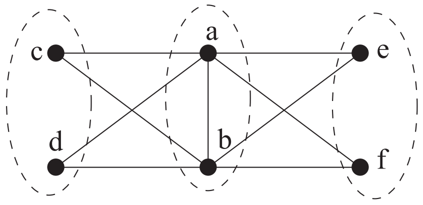


Fig. 2. The graph G as in proof of Lemma 4.1 with $n = 6$, $s = 2$, $V_1 = \{a, b\}$, $V_2 = \{c, d\}$ and $V_3 = \{e, f\}$. The walk that Algorithm 1 constructs would be $[[[a, b], c, [a, b], d], [a, b], e, [[a, b], c, [a, b], d], [a, b], f]$, where brackets show recursive calls in Algorithm 1.

and $|V(G)| = n$ and prove that the smallest kernel of any optimal solution has size 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 length edge uv for any $u \in V_1$ and $v \in V_i$, where $i \in \{1, 2, \ldots, k\}$. For each $v \in V_i$ with $1 \leq i \leq k$, let $\phi(v) = \frac{1}{(s+1)^i}$. We first prove that $\mathrm{OPT}_G \leq 1$.

Let W be a walk constructed by visiting all vertices in the sets $V_1, V_2, \ldots, V_{i-1}$ recursively between any two consecutive visits to members of V_i (see Algorithm 1). It is easy to see that cost of $\Delta(W)$ is at most 1. The reason is that each vertex in V_i for $i \in \{1, 2, \ldots, k\}$ has weight $\frac{1}{(s+1)^i}$ and is visited in $\Delta(W)$ at least once every other $(s+1)^i$ steps by the construction (see Figure 2). Therefore $C(\Delta(W), v)$ is bounded by 1 for any vertex v.

We have proved $\mathrm{OPT}_G \leq 1$. It remains to prove any infinite walk M in G with cost less than or equal to 1 has a kernel of size $\Omega(n^{k-1})$. Let M_1 be a sub-walk of size $s+1$ of M. Then all vertices of V_1 appear in M_1; otherwise, the vertex v in V_1 that does not appear in M_1 would induce a cost larger than 1 to M, that is, $C(M, v) \geq (s+2) \times \frac{1}{s+1} > 1$. 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 vertex in $\bigcup_{j>i} V_j$ appears in any sub-walk of M of size $(s+1)^{i-1}$. To prove this we use induction on i. Let M' be a sub-walk of M with size $(s+1)^{i-1}$. We can partition the elements of M' into $s+1$ disjoint sub-walks of size $(s+1)^{i-2}$. 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 size at least $(s+1)^{k-1}$. Since k is a constant and $n = sk$, therefore $(s+1)^{k-1} \in \Omega(n^{k-1})$. Hence, the kernel of any optimal walk is at least exponential in the size of G. □

Lemma 4.1 implies that there does not exist a polynomial time algorithm for the min–max latency walk problem even if P = NP. In fact, any algorithm that searches the

Algorithm 1 WalkMaker($\{V_1, \dots, V_{i-1}, V_i\}$)
```
1: if $i < 1$ then
2:     return $\emptyset$
3: else
4:     $W \leftarrow \emptyset$
5:     for $j = 1 \rightarrow |V_i|$ do
6:         $W \leftarrow [W, \text{WalkMaker}(\{V_1, \dots, V_{i-1}\})]$,
7:         $W \leftarrow [W, v]$; where $v$ is the $j$th element in $V_i$
8:     end for
9:     return $W$
10: end if
```

solution space to find the optimal solution will have doubly exponential time complexity.

4.2. Binary walks

In Section 4.1, 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 the optimal cost of the graph. We show there always exists a polynomial size walk that has a cost within a constant factor of the optimal cost. To obtain this result, we first need to define a special class of walks and show that there are walks in this class that provide constant factor approximations.

Definition 4.2 (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'. A walk S is a *binary walk* if it can be written as $[S_1, S_2, \dots, S_t]$, where $t = 2\rho_{G'}$, such that for any $v \in V_i$ and any $0 \le j < t/2^i$, vertex v appears exactly once in $[S_{j2^i+1}, S_{j2^i+2}, \dots, S_{(j+1)2^i}]$. In other words, in each 2^i consecutive S_l's starting from S_{j2^i+1}, vertex 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.

By Definition 4.2, each vertex appears in each S_l at most once. Therefore, the size of each S_l is bounded by n, where $n = |V(G')|$. This means that S has size bounded by $2n\rho_{G'}$. Consider a binary walk $S = [S_1, \dots, S_t]$, its expansion $\Delta(S)$ and a member of the binary walk $S_l = (v_1, \dots, v_k)$ for some $1 \le l \le t$. We say that a sub-walk $\Delta(S)(i,j)$ of $\Delta(S)$ *intersects* S_l if either $(v_1, \dots, v_{i'})$ or $(v_{i'}, \dots, v_k)$ appears in $\Delta(S)(i,j)$ for some $1 \le i' \le k$.

Lemma 4.3. *Let G' be a relaxed graph and let V_i denote the set of all vertices of G' with weight $1/2^i$. Let $S = [S_1, S_2, \dots, S_t]$ be a binary walk in G'. If a and b are indices of two consecutive visits to a vertex $v \in V_i$ in $\Delta(S)$, then $\Delta(S)(a,b)$ intersects at most 2^{i+1} members of $(S_1, S_2, \dots, S_t)$.*

Proof. Since $\Delta(S)$ is constructed by the concatenation of an infinite number of copies of walk S, the two following cases can arise for indices a and b: (a) indices a and b refer to two visits of vertices in the same copy of S, or (b) indices a and b refer to two visits such that they are in two consecutive copies of S. Note that since, by Definition 4.2, every vertex is visited at least once in a binary walk S, no other case is possible. For case (a), by Definition 4.2 we have that $\Delta(S)(a,b)$ intersects $2^i + 1$ members of $(S_1, S_2, \dots, S_t)$. For case (b), let $a' \ge a$ be the maximum index of a vertex in $\Delta(S)$ such that visits a and a' are in the same copy of S. Similarly, let $b' \le b$ be the minimum index of a vertex in $\Delta(S)$ such that visits b' and b are in the same copy of S. Since there is one visit to v in both $\Delta(S)(a,a')$ and $\Delta(S)(b',b)$, each of these two sub-walks intersects at most 2^i members of $(S_1, S_2, \dots, S_t)$. Consequently, $\Delta(S)(a,b)$ intersects at most 2^{i+1} members of $(S_1, S_2, \dots, S_t)$. □

Lemma 4.4. *Let G' be a graph with relaxed weights. There is a binary walk S in G' with cost at most $2.5 \times \mathrm{OPT}_{G'}$ and size bounded by $2n\rho_{G'}$, where $n = |V(G')|$.*

Proof. Let $M = (m_1, m_2, \dots)$ be an optimal infinite walk in G' with cost $c = \mathrm{OPT}_{G'}$. Since M is an infinite walk, we can begin the walk at any vertex m_i and obtain the cost c. Therefore, we assume without loss of generality that m_1 is such that $\phi(m_1) = 1$. Based on M, we construct a binary walk S such that the cost of $\Delta(S)$ is at most $2.5c$ 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 $l\big(M(1, a_{i+1})\big) \le ic$. Therefore, we have $l\big(M(1, a_i + 1)\big) > (i-1)c$ and hence $l\big(M(a_i + 1, a_{i+1})\big) \le c$. Consequently, each S_i is a walk of length 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 $t = 2\rho_{G'}$ as in Definition 4.2. For every vertex $u \in V_i$ and any number $0 \le j < t/2^i$, 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 length greater than $c \times 2^i$, implying $C(M,u) > c$. Let $S'_1, S'_2, \dots$ be the result of this modification; note that $l(S'_i) \le l(S_i)$ for each $1 \le i \le t$.

Let S be $[S'_1, S'_2, \dots, S'_t]$. We claim that $\Delta(S)$ has cost at most $2.5c$. For $u \in V_i$ a vertex of G', we know that u appears exactly once in each 2^i consecutive S'_l's, i.e. $[S'_{j2^i+1}, S'_{j2^i+1}, \dots, S'_{(j+1)2^i}]$ for any $0 \le j < t/2^i$. Therefore by Definition 4.2, S is a binary walk. By Lemma 4.3, two consecutive visits to a vertex $u \in V_i$ in $\Delta(S)$ intersects at most 2^{i+1} members of $(S_1, S_2, \dots, S_t)$.

By the construction, we have that for any j, k with $0 < j \le k$, walk $[S'_j, S'_{j+1}, \dots, S'_k]$ has length at most $(k-j+1)c$. Also since $\phi(m_1) = 1$, by Lemma 3.3 we know that for any j, k with $0 \le k \le j$, walk $[S'_j, S'_{j+1}, \dots S'_t, S'_1, S'_2, \dots, S'_k]$ has length at most $((t-j+1)+0.5+k)c$. Therefore, if a and b are indices of two consecutive visits to $u \in V_i$ in $\Delta(S)$, then

$$l\big(\Delta(S)(a,b)\big) < 2^{i+1}c + 0.5c \le 2^i \times (2.5c)$$

Consequently, each vertex $u \in V(G')$ has cost $C(\Delta(S), u) \leq 2.5c$ and hence $C\big(\Delta(S)\big) \leq 2.5c$. Moreover, since S is a binary walk, its size is bounded by $2n\rho_{G'}$. □

At times we will require that our graph contains a vertex of a specific weight $\varphi \leq 1$. The following lemma shows that if our graph does not contain such a vertex, then we can add one without altering the properties of any optimal or approximate walk. We do this as follows.

Definition 4.5 (Dummy Vertex with Weight φ)**.** Let v be a vertex of weight 1 in G. A *dummy vertex* with weight φ is an additional vertex that has an edge of length 0 to vertex v and an edge of length $l(v, u)$ to any other vertex u. Note that adding the dummy vertex to a graph preserves the state of being metric.

Lemma 4.6. *Given any graph G and number $\varphi \leq 1$, let G^* be the graph obtained by adding a dummy vertex of weight φ as described above. Then, the graphs G^* and G have the same optimal cost. Moreover, the cost of any walk W^* in G^*, is greater than or equal to the cost of the walk W in G, obtained by removing all instances of the dummy vertex from W^*.*

Proof. Let W^* be a walk of cost c in G^*. By removing all instances of the dummy vertex from W we can obtain a walk W of cost at most c in G, as the latency of the remaining vertices can only decrease.

By applying this argument to the optimal walk in G^*, we get $\mathrm{OPT}_G \leq \mathrm{OPT}_{G^*}$. Now we need to show that $\mathrm{OPT}_G \geq \mathrm{OPT}_{G^*}$. Let W be an optimal walk in G. We construct W^* by adding a detour of length 0 to the dummy vertex each time v is visited in W. This does not change the cost of other vertices and since the weight of the dummy vertex is smaller than the weight of v, the dummy vertex will not be inducing the maximum cost in W^*. Therefore the cost of W^* in G^* is equal to the cost of W in G and $\mathrm{OPT}_{G^*} = \mathrm{OPT}_G$. □

Theorem 4.7. *In any graph G with $n = |V(G)|$, there exists a walk W of size $O(n^2)$ such that the cost of $\Delta(W)$ is less than or equal to $6 \times \mathrm{OPT}_G$.*

Proof. First we add a dummy vertex of weight $1/2^{\lfloor \log_2 n \rfloor + 1}$ to G, so that we can later assume a vertex with such weight exists in the graph. By Lemma 4.6, doing so does not change the cost of the optimal walk and we can search for walks in the modified graph instead of the original graph. Let G' be the relaxation of G and let V_i be the set of vertices $u \in V(G')$ of weight $1/2^i$. The set of vertices in $V(G')$ with weights less than $1/2^{\lfloor \log_2 n \rfloor + 1}$ is denoted by $U = \{u_1, u_2, \ldots, u_{|U|}\}$. Let G'' be the graph obtained by removing vertices in U from G'. We need a walk that covers vertices of G'' with small cost and is long enough that we can insert the vertices of U into it without significantly increasing the cost. Let S be a binary walk in G'' with cost at most $2.5\mathrm{OPT}_{G''}$ as described in Lemma 4.4. Since $\rho_{G''} \leq 2n$, the size of S is bounded by $2\rho_{G''}n \leq 4n^2$.

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 $(S_1, S_2, \ldots, S_t)$ be the binary decomposition of S, where $t = 2^{\lfloor \log_2 n \rfloor + 2}$. Let v_i be the ith instance of v in S. Note that $t > 2n$; thus v appears at least $2n$ times in S (see Definition 4.2 for $i = 0$). 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}. Since $|U| < n$, this operation is possible. Let W be the resulting walk. Note that the size of W is in $O(n^2)$. We claim that the cost of $\Delta(W)$ in G' is at most $2\mathrm{OPT}_{G'} + \mathrm{OPT}_G$.

Let $w \in V_i$ be a vertex in G' such that $\phi(w) \geq 1/2^{\lfloor \log_2 n \rfloor + 1}$. Let a and b be the indices of two consecutive visits to w in $\Delta(S)$. Let a' and b' be the indices of the corresponding visits to w in $\Delta(W)$. By Lemma 4.3, sub-walk $\Delta(S)(a, b)$ intersects at most 2^{i+1} members of $(S_1, S_2, \ldots, S_t)$. Each S_i has initially one instance of v. Also, we modified only every other instance of v by duplicating and inserting an element of U. Therefore, at most 2^i vertices of U lie between indices a' and b' of $\Delta(W)$.

Furthermore, we inserted the visits to the vertices of U at visits to v with $\phi(v) = 1$. Therefore, by Lemma 3.3, each of these new detours made to visit a member of U has length at most $2(\mathrm{OPT}_G/2) = \mathrm{OPT}_G$. Also, by Lemma 4.4, we already know that $C(\Delta(S), w) \leq 2.5\mathrm{OPT}_{G'}$. Hence, we have

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

Note that the extra 0.5 factor in Lemma 4.4 is due to the distance of the last vertex of S_t to the first vertex 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 inequality (1). Consequently, we have $C(\Delta(W), w) < 2\mathrm{OPT}_{G'} + \mathrm{OPT}_G$. By Lemma 3.2 (ii), we have $\mathrm{OPT}_{G'} \leq \mathrm{OPT}_G$, therefore $C(\Delta(W), w) < 3\mathrm{OPT}_G$. Also, recall that this cost is calculated for $\Delta(W)$ in the relaxed graph G'. Therefore, by Lemma 3.2 (i), the cost of $\Delta(W)$ in G would be less than $6\mathrm{OPT}_G$. Consequently, we have $C\big(\Delta(W)\big) < 6\mathrm{OPT}_G$. □

In the following, we show that Theorem 4.7 is almost tight with respect to the size of the output.

Lemma 4.8. *Any algorithm for the min–max latency walk problem with guaranteed output size in $O(n^2/k)$ has approximation factor in $\Omega(k)$.*

Proof. Let ε be a very small positive number and let us construct a graph G as follows. □

- There are $n/2$ vertices of weight 1, called heavy vertices.
- There are $n/2$ vertices of weight ε, called light vertices.

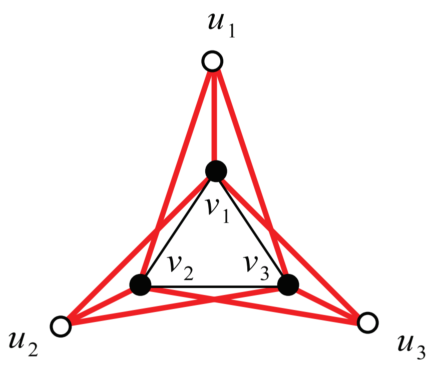


Fig. 3. Graph G as in Lemma 4.8 with $n = 6$. Vertices v_1, v_2, v_3 are the heavy vertices and u_1, u_2, u_3 are the light vertices. The red thick edges have unit weights and the weight of the black edges is small. We have $C(\Delta(W)) = 2 + 2\varepsilon$ for $W = (v_1, v_2, v_3, u_1, v_1, v_2, v_3, u_2, v_1, v_2, v_3, u_3)$ with size 12. On the other hand, $C(\Delta(W')) = 4 + 2\varepsilon$, where $W' = (v_1, u_1, v_2, u_2, v_3, v_1, u_3, v_2, v_3)$ has size 9. Here, we have $k = 2$.

- Any two heavy vertices are connected to each other by an edge of length ε.
- There is an edge of length 1 connecting any light vertex to any heavy vertex (see Figure 3).

Any minimum cost infinite walk in G visits all heavy vertices between visits to any 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. Therefore, any optimum solution has size in $\Omega(n^2)$ and its cost is upper-bounded by $2 + \varepsilon \times O(n)$. The value of ε can be chosen to be small enough so that the cost of optimal walk is close to 2.

To reduce the size of the output walk by a factor of k, we need to visit at least k light vertices between two consecutive visits to a heavy vertex v. This means that a walk of size smaller than $\frac{n^2}{4k}$ has cost at least $2k$, which is k times the optimal cost. Therefore, any solution for the min–max latency walk problem in G with size in $O(n^2/k)$ has approximation factor in $\Omega(k)$. $\square$

Lemma 4.8 directly gives that there is no constant factor approximation algorithm with guaranteed output size in $o(n^2)$. Note that this implies that Theorem 4.7 is tight in the sense that the size of the constructed kernel can be reduced by at most a constant factor.

5. Approximation algorithms for the min–max latency walk problem

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

Algorithm 2 BRUTEPARTITIONALG(G)

```
1:  Let $V_i$ be the set of vertices of weight $\frac{1}{2^i} \le \phi(u) < \frac{1}{2^{i-1}}$
    for $0 \le i \le \lceil \log_2 \rho_G \rceil$
2:  Let $t$ be $2^{\lceil \log_2 \rho_G \rceil + 1}$
3:  $S_1, S_2, \ldots, S_t \leftarrow \emptyset$
4:  for $i = 0 \to \lceil \log_2 \rho_G \rceil$ do
5:      $\{W_{i,0}, \ldots, W_{i,2^i-1}\} \leftarrow$ Partition (TSP-Path($G[V_i]$), $2^i$)
6:      for $k = 1 \to t$ do
7:          $S_k \leftarrow [S_k, W_{i,j_i}]$; where $j_i$ is $k$ mod $2^i$,
8:      end for
9:  end for
10: $S \leftarrow [S_1, \ldots, S_t]$
11: return $S$
```

5.1. *An $O(\log \rho_G)$-approximation algorithm*

A crucial requirement for our algorithms is a useful property regarding binary walks. This property is discussed in the following lemma.

Lemma 5.1. *(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 that*

(i) for some $c \in \mathbb{R}_+$, $\max_{1 \le i \le t} l(S_i) \le c$; and
(ii) each S_i begins in a vertex $v \in V(G')$, where $\phi(v) = 1$.

Then the cost of $\Delta(S)$ in G' is at most $2c + \mathrm{OPT}_{G'}$.

Proof. Let V_i be the set of vertices of weight $1/2^i$ in $V(G')$. Let $u \in V_i$ be a vertex of G'. By Lemma 4.3, for any a and b that are the indices of two consecutive visits to u in $\Delta(S)$, we have that $\Delta(S)(a, b)$ intersects at most 2^{i+1} members of $(S_1, S_2, \ldots, S_t)$. Also, by condition (ii) and Lemma 3.3 we know that the lengths of edges connecting two consecutive S_l's in $\Delta(S)$ are at most $\mathrm{OPT}_{G'}/2$. Hence, we have $l(\Delta(S)(a, b)) \le 2^{i+1}(c + \mathrm{OPT}_{G'}/2)$. Consequently, each vertex $u \in V(G')$ has cost $C(\Delta(S), u) \le 2c + \mathrm{OPT}_{G'}$. Hence, we have $C(\Delta(S)) \le 2c + \mathrm{OPT}_{G'}$. $\square$

Here, we define a tool that will be useful in our approximation algorithms. Let Partition(W, k) be a function that takes as input a walk W and an integer k and returns a set of k walks $\{W_1, W_2, \ldots, W_k\}$ that partitions the vertices of W such that $l(W_i) \le l(W)/k$, for all $1 \le i \le k$. It is easy to see this can always be computed in linear time by a single traversal of W. Note that in the case that $k > |W|$, Partition(W, k) returns a set $\{W_1, W_2, \ldots, W_k\}$ in which $W_i = W(i, i)$, for $1 \le i \le |W|$ and $W_i = \emptyset$ for $|W| < i \le k$. Also, recall from Section 2.1 that TSP-Path(G) is a walk that visits each vertex in G exactly once.

Given a graph G, our first approximation algorithm, shown in Algorithm 2, is guaranteed to find a solution with cost within a factor of $O(\log \rho_G)$ of the optimal cost, where ρ_G is the ratio of the maximum vertex weight to the minimum vertex weight in G. The main idea in Algorithm 2

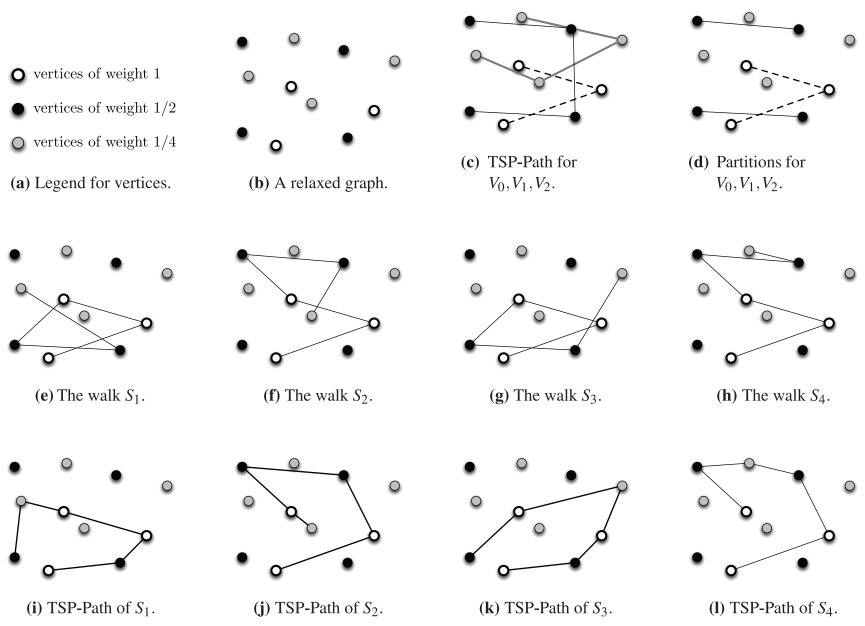


Fig. 4. An illustration of the BRUTEPARTITIONALG in Algorithm 2. Figure (b) shows a relaxed graph consisting of vertices of weight 1, 1/2 and 1/4. The edge lengths are given by the Euclidean distance between vertices. Figures (c)–(d) show the partitioning of the vertices that occurs in line 5. Figures (e)–(h) show the four walks obtained in line 7. Figures (i)–(l) shows how the walks can be heuristically improved by recomputing the TSP-Path through each of them, as discussed in Remark 5.3.

is to construct a binary walk $S = [S_1, S_2, \dots, S_t]$ that satisfies the binary property discussed in Lemma 5.1. Recall that in Lemma 5.1 it was shown that if $\max_{1 \le k \le t} l(S_k) < c$ for some c and each S_k begins in v for some vertex $v \in V(G)$ with $\phi(v) = 1$, then cost of S is at most $2c + \mathrm{OPT}_{G'}$. Here, we obtain a method that constructs a binary walk $S = [S_1, S_2, \dots, S_t]$ such that for each $1 \le k \le t$, S_k starts with a unit weight vertex v and the length of S_k is in $O(\log \rho_G)\,\mathrm{OPT}_{G'}$. An overview of Algorithm 2 is as follows. First graph G is relaxed. Then a TSP-Path is calculated for each set of vertices V_i, where V_i denotes the set of all vertices with relaxed weight of $1/2^i$. The TSP-Path through vertices in V_i is partitioned into 2^i walks. These walks are concatenated to create each S_k. An illustration of Algorithm 2 is given in Figure 4. The final walk S is obtained by repeatedly performing the four walks shown in Figure 4(e) through to Figure 4(h).

Theorem 5.2. *Given a graph G, Algorithm 2 constructs a walk S of size in $O(n\rho_G)$ such that $\Delta(S)$ is an $O(\log \rho_G)$-approximation for the min–max latency walk problem in G.*

Proof. Let G' be the result of relaxing the weights of G. The set of vertices $u \in V(G')$ of weight $\frac{1}{2^i}$ is denoted by V_i. Let v denote a vertex in V_0, i.e. $\phi(v) = 1$, such that v is the first vertex that appears in TSP-Path($G[V_0]$). Let t be the smallest power of two that is not smaller than $2\rho_G$; that is, $t = 2\rho_{G'}$. We will show that Algorithm 2 constructs a binary walk $S = [S_1, S_2, \dots, S_t]$ such that each S_k begins in v and $\max_{1 \le k \le t} l(S_k) < 2(\log_2 \rho_{G'})\,\mathrm{OPT}_{G'} + \mathrm{OPT}_{G'}$.

Let us look at the set V_i for some $0 \le i \le \log_2 \rho_{G'}$. Algorithm 2 constructs 2^i walks such that each vertex in V_i appears in exactly one of these walks. Let W_i be the output of TSP-Path($G[V_i]$). We claim that $l(W_i)/2^i \le \mathrm{OPT}_{G'}$. It is clear that $\mathrm{OPT}_{G'[V_i]} \le \mathrm{OPT}_{G'}$ as there are less vertices in $G'[V_i]$ that need to be visited. In the proof of Theorem 2.2, we showed that in graphs with uniform weights a TSP tour is an optimal solution for the min–max latency walk problem. This means that any TSP tour on V_i has cost $\mathrm{OPT}_{G'[V_i]}$ that is no larger than $\mathrm{OPT}_{G'}$. Considering that the weight of each vertex in V_i is $1/2^i$ in G', this means that a TSP tour of $G'[V_i]$ has length at most $2^i\mathrm{OPT}_{G'}$. Also, the length of TSP tour is an upper-bound for the length of TSP-Path. Therefore, $l(W_i)/2^i \le \mathrm{OPT}_{G'}$.

As in line 5 of Algorithm 2, the output of Partition($W_i, 2^i$) is $\{W_{i,0}, W_{i,2}, \dots, W_{i,2^i-1}\}$. Hence, we have $l(W_{i,j}) \le l(W_i)/2^i \le \mathrm{OPT}_{G'}$ for all $0 \le j < 2^i$. Observe that the output of Partition($W_0, 1$) is $\{W_{0,0}\}$, where $W_{0,0} = W_0$. A walk $S = [S_1, S_2, \dots, S_t]$ is constructed using each $W_{i,j}$ as follows.

In line 6 of Algorithm 2, for each $1 \le k \le t$, walk S_k is constructed by concatenating the walks $W_{0,j_0}, W_{1,j_1}, \dots, W_{\log_2 \rho_{G'}, j_{\log_2 \rho_{G'}}}$ while preserving order, where $k \equiv j_i \pmod{2^i}$ for each $0 \le i \le \log_2 \rho_{G'}$ (see Figure 5). Observe that for $1 \le k \le t$, each S_k starts with $W_{0,0}$. Therefore, the first vertex of each S_k is vertex v, where v is the first vertex that appears in TSP-Path($G[V_0]$). Note that by Lemma 3.3, the length of edges connecting two $W_{i,j}$'s is at most $\mathrm{OPT}_{G'}$. In the end, there will be $\log_2 \rho_{G'} + 1$ walks in each S_k, for $1 \le k \le t$, each

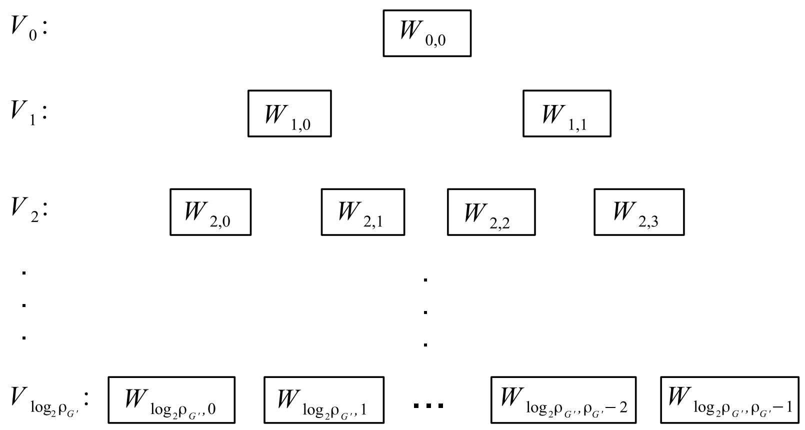


Fig. 5. The construction of S_k's from walks $W_{i,j}$'s is depicted. For $1 \le k \le t$, each S_k is constructed by concatenation of the walks $W_{0,j_0}, W_{1,j_1}, \ldots, W_{\log_2 \rho_{G'}, j_{\log_2 \rho_{G'}}}$ while preserving order, where $k \equiv j_i \pmod{2^i}$ for each $0 \le i \le \log_2 \rho_{G'}$.

of length at most $\mathrm{OPT}_{G'}$, connected to each other using $\log_2 \rho_{G'}$ edges, each with length at most $\mathrm{OPT}_{G'}$. Therefore $\max_{1\le k\le t} l(S_k) \le 2\log_2 \rho_{G'}\mathrm{OPT}_{G'} + \mathrm{OPT}_{G'}$.

By Lemma 5.1, walk S has cost at most $4\log_2 \rho_{G'}\mathrm{OPT}_{G'} + 3\mathrm{OPT}_{G'}$ in G'. Using Lemma 3.2 (ii), it follows that cost of S in G' is at most $4\log_2 \rho_{G'}\mathrm{OPT}_{G} + 3\mathrm{OPT}_{G}$. Hence, by Lemma 3.2 (i), walk S is an $(8\log_2 \rho_{G'} + 6)$-factor approximate solution for G. Note that since $\rho_{G'} < 2\rho_G$, we have that $8\log_2 \rho_{G'} + 6$ is in $O(\log \rho_G)$. Therefore, $\Delta(S)$ is an $O(\log \rho_G)$-approximation for the min–max latency walk problem in G. Finally, since each vertex $u \in V_i$ appears exactly once in W_i and by the construction of S_k's, for $1 \le k \le t$, we have that S is a binary walk. Therefore, the size of S is in $O(n\rho_G)$. □

The following remark discusses a heuristic improvement that should be used in practice.

Remark 5.3 (Implementation of Algorithm 2). In practice, we can recompute the TSP-Path through each walk $S_1, \ldots, S_t$. We simply require that each walk TSP-Path(S_i) starts at the same vertex. In recomputing the TSP-Paths, the bounds remain unchanged. However, in practice the performance is improved. An example of the improvement is shown in Figure 4. Thus, the final infinite walk is obtained by repeatedly performing the walks in Figure 4(i) through to Figure 4(l).

5.2. *An $O(\log n)$-approximation algorithm*

In many applications, the value of ρ_G is independent of n. For example, in a robotic 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 approximation algorithm. However, the ratio between the largest and the smallest weights 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.

Algorithm 3 SMARTPARTITIONALG(G)

```
1:  Add a dummy vertex of weight $1/2^{\lfloor \log_2 n \rfloor + 1}$ to $G$
2:  Let $V_i$ be the set of vertices of weight $\frac{1}{2^i} \le \phi(u) < \frac{1}{2^{i-1}}$
    for $0 \le i \le \lceil \log_2 \rho_G \rceil$
3:  $U \leftarrow \cup_{i > \lfloor \log_2 n \rfloor + 1} V_i$
4:  $S \leftarrow$ BRUTEPARTITIONALG($G[V \setminus U]$)
5:  $k \leftarrow 1$
6:  for all $u \in V_i$ where $i > \lfloor \log_2 n \rfloor + 1$ do
7:      Insert $u$ at the end of $S_{2k}$
8:      Increment $k$
9:  end for
10: Remove the instances of the dummy vertex from $S$
11: return $S$
```

Our second approximation algorithm is shown in Algorithm 3. This algorithm is guaranteed to find a solution with cost within a logarithmic factor of the optimal cost. The main idea in Algorithm 3 is to construct a graph G'' from the input graph G by relaxing weights and removing vertices with very low weights from G. The result is that $\rho_{G''}$ can be a function of n. Performing Algorithm 2 on G'' results in a binary walk S such that $\Delta(S)$ is an $O(\log n)$-factor approximation for the min–max latency walk problem in G''. We then insert the vertices in $V(G) \setminus V(G'')$ into S in such a way that we maintain the $O(\log n)$-factor approximation on the input graph G.

Theorem 5.4. *Given a graph G, Algorithm 3 constructs a walk S of size in $O(n^2)$ such that $\Delta(S)$ is an $O(\log n)$-approximation for the min–max latency walk problem in G.*

Proof. The idea of Algorithm 3 is to remove the vertices of small weight so that we can use Algorithm 2 as a subroutine. In line 1 of Algorithm 3, we add a dummy vertex of weight $1/2^{\lfloor \log n \rfloor + 1}$ to G that will be later removed from the final solution. By Lemma 4.6, this dummy vertex does not alter the approximation factor of the algorithm: we add the dummy vertex to G to ensure that S contains enough walks, $S_1, \ldots, S_t$, such that we can reinsert just one low weight vertex into every other S_k (as described in line 7).

Let G' be the result of relaxing the weights of G and U be the set of vertices of G' with weight less than $1/2^{\lfloor \log_2 n \rfloor + 1}$, as in line 3 of Algorithm 3. Let G'' be the result of removing vertices in U from G'. We have $\rho_{G''} = 2^{\lfloor \log_2 n \rfloor + 1}$, and thus $n < \rho_{G''} \leq 2n$. From Line 4 of Algorithm 3, the walk $S = [S_1, S_2, \ldots, S_t]$ is the result of running Algorithm 2 on G'' and by Definition 4.2, we have $t = 2\rho_{G''}$. Recall that in Algorithm 2, for $1 \leq k \leq t$ each S_k starts with the same vertex v with $\phi(v) = 1$. Moreover, by the proof of Theorem 5.2, since $\rho_{G''} \leq 2n$, each S_k has length at most $2 \log_2 \rho_{G''} \mathrm{OPT}_{G''} + \mathrm{OPT}_{G''} \leq (2 \log_2 n + 3) \mathrm{OPT}_{G''}$. Also, since $\mathrm{OPT}_{G''} \leq \mathrm{OPT}_{G'}$, we have that $(2 \log_2 n + 3) \mathrm{OPT}_{G'}$ is an upper-bound for length of each S_k.

In line 7 of Algorithm 3, the kth vertex of U is inserted at the end of walk S_{2k}. Note that since $|U| < n$ and $t = 2\rho_{G''} > 2n$, this is possible. Since each walk S_k begins in vertex v with $\phi(v) = 1$, by Lemma 3.3, each detour to a vertex in U has length bounded by $\frac{3}{2}\mathrm{OPT}_{G'}$. Consequently, after inserting vertices in U into S, each S_k has length at most $(2 \log_2 n + \frac{9}{2}) \mathrm{OPT}_{G'}$ in G'. Hence, by Lemma 5.1, the cost of $\Delta(S)$ in G' is at most $(4 \log_2 n + 10) \mathrm{OPT}_{G'}$. Furthermore, the cost of $\Delta(S)$ in G is bounded by $(8 \log_2 n + 20) \mathrm{OPT}_G$ using Lemma 3.2. As a result, $\Delta(S)$ is an $O(\log n)$-approximation for the min–max latency walk problem in G.

Moreover, since S is a binary walk, its size is bounded by $2n\rho_{G''} \leq 4n^2$. Since the number of vertices added to S during its modification is in $O(n)$, the size of the constructed walk S is in $O(n^2)$. □

As mentioned in Remark 5.3, we can improve the performance of Algorithm 3 by computing TSP-Paths through each of the modified walks in S, making sure that all paths start at the same vertex. The following result shows that Algorithm 3 always achieves a better approximation factor than Algorithm 2.

Corollary 5.5. *The approximation ratio of Algorithm 3 is* $O(\log \min\{n, \rho_G\})$, *which is always less than or equal to that of Algorithm 2.*

Proof. In Algorithm 3, if for the input graph G we have $\rho_G \leq 2n$, then the set U is empty and hence the approximation guarantees of both Algorithms 2 and 3 are $O(\log \rho_G)$. On the other hand, if $\rho_G > 2n$ then the approximation factor of Algorithm 3, i.e. $O(\log n)$, is smaller than the approximation factor of Algorithm 2, which is $O(\log \rho_G)$. □

6. Simulations

In this section, we present two sets of simulations. The first shows the scalability of our algorithms with respect to the size of the graph and compares the performance to a simple TSP-based algorithm. The second set shows a case study for patrolling for crime in the city of San Francisco, California.

6.1. Scalability of approximation algorithms

In this section, we study the scalability of Algorithm 3. Recall that by Corollary 5.5, Algorithm 3 always performs better than Algorithm 2, both in runtime and approximation factor.

For the simulations, we use test data that are standard benchmarks for testing the performance of heuristic algorithms for the TSP. The data sets used here are taken from (Cook, 2009). Each data set represents a set of locations in a country. We construct a graph by placing a vertex for each location and letting the length of the edge connecting each pair of vertices be the Euclidean distance of the corresponding points.

What remains is to assign weights to each vertex in the test cases. In many persistent monitoring applications, 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:

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

for $k < B$, where B is a fixed integer. To create such a distribution, we assign to each vertex v a weight $(1/2)^B \leq \phi(v) \leq 1$ with probability $f(\phi(v)) = (\phi(v) B \ln 2)^{-1}$.

Here, we compare our algorithms to the simple algorithm of finding a TSP tour through all vertices in G. For finding an approximate solution to the TSP we use an implementation of the Lin-Kernighan algorithm (Lin and Kernighan, 1973) available in Cook (2009). Recall that from Lemma 2.3, the walk obtained from the TSP can have a cost that is n times larger than the optimal cost. However, when all vertex weights are equal, the TSP yields the optimal walk. In simulation, when the vertex weights are uniformly distributed, the TSP appears to provide a fairly good approximation for the min–max latency walk problem. One of the reasons for this is that when the vertex weights are distributed uniformly, we expect that half of the vertices will have weights in $[0.5, 1]$. Thus $O(n)$ of the vertices must be visited very frequently and not much can be gained by visiting vertices at different frequencies.

Performance with respect to vertex weight distribution: An important aspect of an environment is the ratio of weight of its elements, therefore it is natural to test our algorithm with respect to ρ_G. Note that by Equation 2, $\rho_G > (1/2)^B$. 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 6). It is easy to see that if $B <$

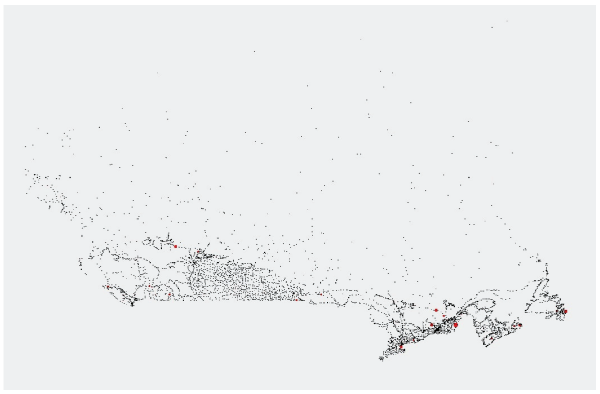


Fig. 6. The 4663 vertex graph used for all tests corresponding to all cities in Canada (Cook, 2009).

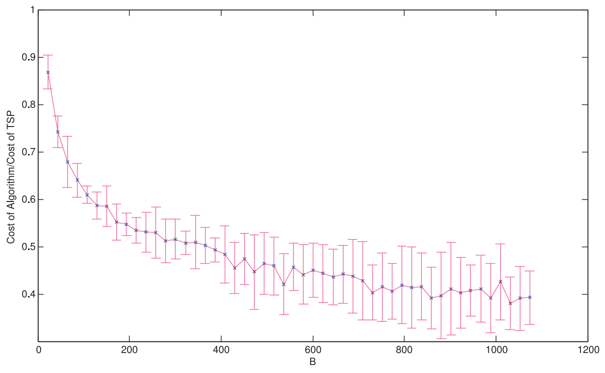


Fig. 7. The ratio of the cost of the walk produced by Algorithm 3 to the cost of the TSP tour as a function of B in (2).

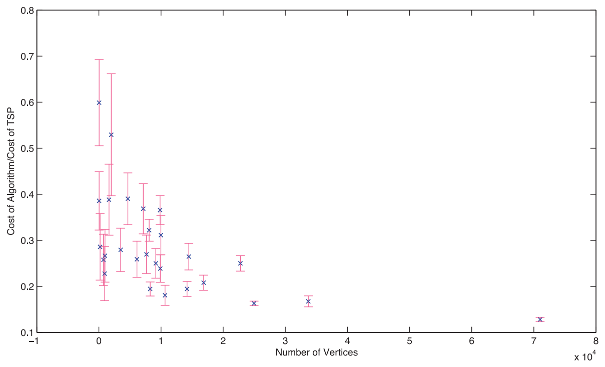


Fig. 8. The ratio of the cost of Algorithm 3 to the cost of the TSP tour on the 27 test graphs in Cook (2009).

$\log_2 n$, then Algorithms 3 and 2 produce the same output. Figure 7 depicts the behavior of Algorithm 3 on a graph induced by 4663 cities in Canada, shown in Figure 6, with different values for B. It can be seen that as B increases, our algorithm outperforms the TSP by a greater factor.

Performance with respect to input graph size: We use graphs of different sizes to evaluate the performance and scalability of our algorithms. Again, the cost is compared to that of a simple TSP tour that visits each vertex in the graph exactly once before returning to its starting vertex. Figure 8 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 the value of B is fixed to 1000. It can be seen that the ratio of the cost of the walk produced by our algorithm to the cost of the TSP tour decreases as the size of graph increases. Hence, as the number of vertices in graphs increases, the performance of Algorithm 3 relative to the TSP improves.

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 TSP tours. For the test data corresponding to 71,009 locations in China, our Java implementation of Algorithm 3 constructs an approximate solution in 20 seconds using a laptop with a 2.50 GHz CPU and 3 GB RAM.

6.2. A case study in patrolling for crime

Here, we study the problem of planning a route for a robot (or vehicle) to patrol a city. Assume we want to plan a route that patrols a set of intersections in a city and the goal is to minimize the maximum expected number of crimes that occur at any of the intersections between two consecutive visits. If the weight of each intersection is given by the average crime rate in that intersection, then, since expectation is a linear function, this problem will exactly translate to the min–max latency walk problem.

We look at twelve intersections in the central district of the San Francisco police department (see Table 1 and Figure 9). The number of crimes that occurred in August of 2012[2] in the vicinity of these intersections is used as the weight function ϕ. The weight of each intersection approximates the average rate of crime happening in vicinity of that intersection. Table 2 shows the pairwise travel times (for a road vehicle) of these intersections in seconds. Note that these travel times are not, in general, symmetric as some streets are one-way. For $l(uv)$, we use the average between the travel time from u to v and the travel time from v to u.

Let us step through Algorithm 3. Since $\log_2 \rho_G < \lfloor \log_2 n \rfloor + 1$, Algorithm 3 simply returns the output of Algorithm 2. We have $V_0 = \{A\}$, $V_1 = \{B, C, D, E, F, G\}$ and $V_2 = \{H, I, J, K, L\}$. Then we find the TSP-Paths, which are given by $W_0 = (A)$, $W_1 = (C, G, D, F, E, B)$ and $W_2 = (I, K, L, J, H)$. We then partition these into $W_{0,0} = (A)$, $W_{1,0} = (C, G, D)$, $W_{1,1} = (F, E, B)$, $W_{2,0} = (I)$, $W_{2,1} = (K, L)$, $W_{2,2} = (J)$ and $W_{2,3} = (H)$. Then, by concatenation of these walks and finding the TSP-Path of the results we get $S_1 = (A, C, G, D, I)$, $S_2 = (A, B, L, K, F, E)$, $S_3 = (A, C, J, D, G)$ and $S_4 = (A, B, E, F, H)$. The walks are shown in Figure 10. Note that the TSP-Path is shown instead of the TSP tour, and thus the final edge that returns to intersection A is omitted. This is done so that the output is a binary walk, i.e. each walk S_i contains at most one instance of each intersection.

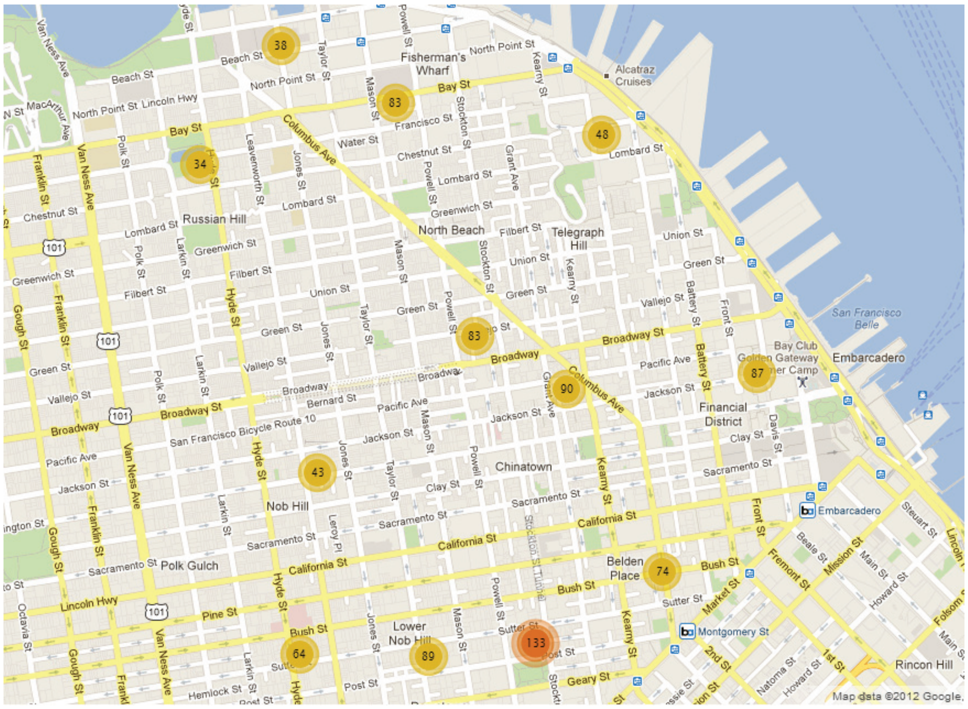


Fig. 9. The twelve intersections of Table 1 in the north east of San Francisco. Map obtained from San Francisco Police Department CrimeMAPS program.

Table 1. Twelve locations in the central district of San Francisco police department with the number of recorded criminal acts in the vicinity of each location.

Index	# of crimes in August 2012	Approximate address
A	133	Sutter St & Stockton St, San Francisco, CA 94108, USA
B	90	Pacific Ave & Grant Ave, San Francisco, CA 94133, USA
C	89	Post St & Taylor St, San Francisco, CA 94142, USA
D	87	Jackson St & Front St, San Francisco, CA 94111, USA
E	83	Vallejo St & Powell St, San Francisco, CA 94133, USA
F	83	Bay St & Mason St, San Francisco, CA 94133, USA
G	74	Bush St & Montgomery St, San Francisco, CA 94104, USA
H	64	Bush St & Hyde St, San Francisco, CA 94109, USA
I	48	Chestnut St & Montgomery St, San Francisco, CA 94111, USA
J	43	Washington St & Leavenworth St, San Francisco, CA 94109, USA
K	38	Jones St & Beach St, San Francisco, CA 94133, USA
L	34	Hyde St & Francisco St, San Francisco, CA 94109, USA

The walk $\Delta(S)$ with $S = [S_1, S_2, S_3, S_4]$ is our final patrolling route. The latencies and costs induced by the intersections are shown in Table 3. The expected number of crimes that occur between two consecutive visits to any intersection is bounded by $C(\Delta(S)) = 0.102$.

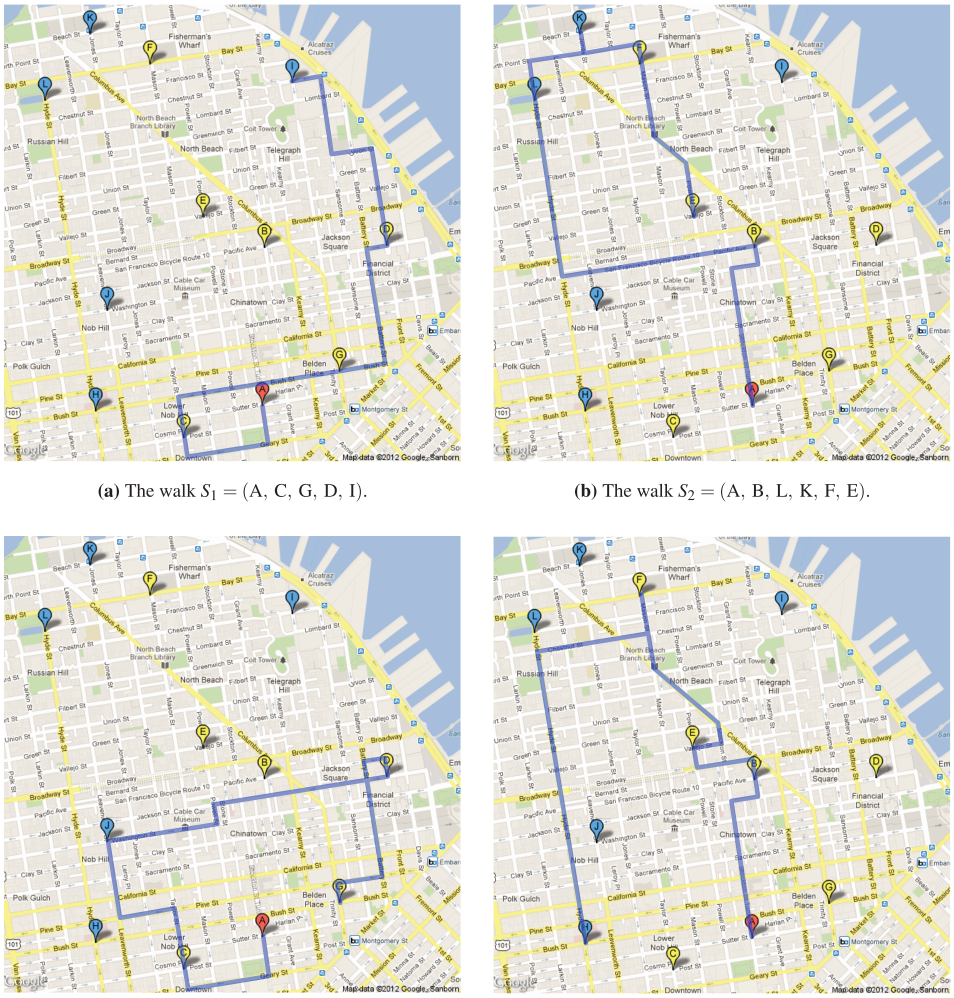
7. Conclusions and future work

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 represented the environment as a graph with vertex weights and edge lengths and we addressed the problem of finding a 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 algorithm and an $O(\log \rho_G)$-approximation algorithm, where ρ_G is the ratio between the maximum and minimum vertex weights. We also showed via simulations that our algorithms scale to very large problems consisting of thousands of vertices.

There are several directions for the future work . We continue to seek a constant factor approximation algorithm, independent of ρ_G. We also believe that by adding some

Table 2. The pairwise by-car travel times in seconds corresponding to the locations in Table 1, queried from Google maps.

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	141	121	293	209	329	134	250	406	199	358	344
B	141	0	271	200	105	226	201	299	297	169	254	274
C	127	291	0	368	311	433	153	198	491	219	461	362
D	304	207	417	0	253	309	226	387	249	358	337	384
E	210	147	340	244	0	180	244	268	342	164	209	230
F	330	216	460	244	175	0	313	370	126	311	61	163
G	90	246	162	244	310	369	0	271	400	292	397	427
H	147	293	105	370	338	412	154	0	492	153	406	287
I	426	324	539	203	343	226	348	509	0	448	299	389
J	201	170	231	322	164	290	279	159	415	0	283	164
K	354	240	474	337	199	105	337	332	226	273	0	125
L	334	220	354	316	179	121	317	212	246	153	114	0



(a) The walk $S_1 = (A, C, G, D, I)$.

(b) The walk $S_2 = (A, B, L, K, F, E)$.

(c) The walk $S_3 = (A, C, J, D, G)$.

(d) The walk $S_4 = (A, B, E, F, H)$.

Fig. 10. The four walks for patrolling San Francisco, corresponding to Table 1. Location A (red) is V_0. Locations B, C, D, E, F, G (yellow) correspond to V_1. Locations H, I, J, K, L (blue) correspond to V_2. The final monitoring walk is given by $[S_1, S_2, S_3, S_4, S_1, \ldots]$.

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.

Table 3. The latencies and costs induced by *S* for the crime case-study. The maximum cost is attained at intersection H.

Index	Latency (seconds)	Cost (crimes per visit)
A	1159	0.059
B	2193	0.075
C	2136	0.072
D	2309	0.076
E	2694	0.085
F	2339	0.074
G	2779	0.078
H	**4206**	**0.102**
I	4206	0.077
J	4206	0.069
K	4206	0.061
L	4206	0.054

Acknowledgement

A preliminary version of this paper appeared in the *Proceedings of the 2012 Workshop on the Algorithmic Foundations of Robotics* (Almadari et al., 2012).

Funding

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

Notes

1. We refer to a robot path as a walk to be consistent with the terminology in graph theory (Bondy and Murty, 2008), where a path is a sequence of unique vertices, while a walk may repeat vertices.
2. Crime data for San Francisco was obtained from the San Francisco Police Department CrimeMAPS website: http://www.sf-police.org/.

References

Alamdari S, Fata E and Smith SL (2013) Min–max latency walks: Approximation algorithms for monitoring vertex-weighted graphs. *Springer Tracts in Advanced Robotics* 86: 139–155.

Arvelo E, Kim E and Martins NC (2012) Memoryless control design for persistent surveillance under safety constraints. arXiv.org. Submitted: 26 September 2012. Available at: http://arxiv.org/abs/1209.5805.

Bethke B, How JP and Vian J (2008) Group health management of UAV teams with applications to persistent surveillance. In: *American control conference*, Seattle, WA, 11–13 June 2008, pp. 3145–3150.

Bethke B, Redding J, How JP, Vavrina MA and Vian J (2010) Agent capability in persistent mission planning using approximate dynamic programming. In: *American control conference*, Baltimore, MD, 30 June–2 July 2010, pp. 1623–1628.

Bondy JA and Murty USR (2008) *Graph Theory* (*Graduate Texts in Mathematics*, Vol. 244). New York: Springer.

Bullo F, Frazzoli E, Pavone M, Savla K and Smith SL (2011) Dynamic vehicle routing for robotic systems. *Proceedings of the IEEE* 99(9): 1482–1504.

Cannata G and Sgorbissa A (2011) A minimalist algorithm for multirobot continuous coverage. *IEEE Transactions on Robotics* 27(2): 297–312.

Cassandras CG, Ding XC and Lin X (2011) An optimal control approach for the persistent monitoring problem. In: *IEEE Conference on decision and control and European control conference*, Orlando, FL, December 2011, pp. 2907–2912.

Chevaleyre Y (2004) Theoretical analysis of the multi-agent patrolling problem. In: *IEEE/WIC/ACM international conference intelligent agent technology*, Beijing, China, 20–24 September 2004, pp. 302–308.

Choset H (2001) Coverage for robotics—A survey of recent results. *Annals of Mathematics and Artificial Intelligence* 31(1–4): 113–126.

Christofides N and Beasley JE (1984) The period routing problem. *Networks* 14(2): 237–256.

Cook WJ (2009) National travelling salesman problems. Available at: http://www.tsp.gatech.edu/index.html.

Elmaliach Y, Agmon N and Kaminka GA (2007) Multi-robot area patrol under frequency constraints. In: *IEEE International conference on robotics and automation*, Roma, Italy, 10–14 April 2007, pp. 385–390.

Gabriely Y and Rimon E (2003) Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications* 24(3): 197–224.

I Gørtz, Molinaro M, Nagarajan V and Ravi R (2011) Capacitated vehicle routing with non-uniform speeds. *Proceedings of the 15th international conference on integer programming and combinatoral optimization*, Armonk, NY, 15–17 June 2011, pp. 235–247.

Hussein II and Stipanoviè DM (2007) Effective coverage control for mobile sensor networks with guaranteed collision avoidance. *IEEE Transactions on Control Systems Technology*, 15(4): 642–657.

Laporte G (2009) Fifty years of vehicle routing. *Transportation Science* 43(4): 408–416.

Lin S and Kernighan BW (1973) Effective heuristic algorithm for the traveling salesman problem. *Operations Research* 21: 498–516.

Mathew N, Smith SL and Waslander SL (2013) A graph-based approach to multi-robot rendezvous for recharging in persistent tasks. In: *IEEE international conference on robotics and automation*, Karlsruhe, Germany, 6–10 May 2013, pp. 3482–3487.

Michael N, Stump E and Mohta K (2011) Persistent surveillance with a team of MAVs. In: *IEEE/RSJ international conference on intelligent robots and systems*, San Francisco, CA, 25–30 September 2011, pp. 2708–2714.

Nigram N and Kroo I (2008) Persistent surveillance using multiple unmanned air vehicles. In: *IEEE Aerospace Conference*, Big Sky, MT, 1–8 March 2008, pp. 1–14.

Papadimitriou CH and Yannakakis M (1993) The traveling salesman problem with distances one and two. *Mathematics of Operations Research* 18: 1–11.

Pasqualetti F, Durham JW and Bullo F (2012a) Cooperative patrolling via weighted tours: Performance analysis and distributed algorithms. *IEEE Transactions on Robotics* 28(5): 1181–1188.

Pasqualetti F, Franchi A and Bullo F (2012b) On cooperative patrolling: Optimal trajectories, complexity analysis and approximation algorithms. *IEEE Transactions on Robotics* 28(3): 592–606.

Smith RN, Schwager M, Smith SL, Rus D and Sukhatme GS (2011) Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics* 28(5): 714–741.

Smith SL and Rus D (2010) Multi-robot monitoring in dynamic environments with guaranteed currency of observations. In: *IEEE Conference on Decision and Control*, Atlanta, GA, 15–17 December 2010, pp.514–521.

Smith SL, Schwager M and Rus D (2012) Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics* 28(2): 410–426.

Stump E and Michael N (2011) Multi-robot persistent surveillance planning as a vehicle routing problem. In: *IEEE Conference on Automation Science and Engineering*, Trieste, Italy, 24–27 August 2011, pp.569–575.

Tiwari A, Jun M, Jeffcoat DE and Murray RM (2005) Analysis of dynamic sensor coverage problem using Kalman filters for estimation. In: *IFAC World Congress*, Prague, Czech Republic, 4–8 July 2005, Vol. 16, Part I, p.360.

Tulabandhula T, Rudin C and Jaillet P (2011) Machine learning and the traveling repairman. Available at: http://arxiv.org/abs/1104.5061.

Vazirani VV (2004) *Approximation Algorithms*. Berlin, New York: Springer.