Computing an Approximation of the 1-Center Problem on Weighted Terrain Surfaces

MARK A. LANTHIER, DORON NUSSBAUM, and TSUO-JUNG WANG
Carleton University

1.3

In this article, we discuss the problem of determining a meeting point of a set of scattered robots $R = \{r_1, r_2, \ldots, r_s\}$ in a weighted terrain $\mathcal{P}$, which has $n > s$ triangular faces. Our algorithmic approach is to produce a discretization of $\mathcal{P}$ by producing a graph $G = \{V^G, E^G\}$, which lies on the surface of $\mathcal{P}$. For a chosen vertex $p' \in V^G$, we define $\|\Pi(r_i, p')\|$ as the minimum weight cost of traveling from r_i to p'. We show that $\min_{p' \in V^G}\{\max_{1 \le i \le s}\{\|\Pi(r_i, p')\|\}\} \le \min_{p^* \in \mathcal{P}}\{\max_{1 \le i \le s}\{\|\Pi(r_i, p^*)\|\}\} + 2W|L|$, where L is the longest edge of $\mathcal{P}$, W is the maximum cost weight of a face of $\mathcal{P}$, and p^* is the optimal solution. Our algorithm requires $O(snm \log(snm) + snm^2)$ time to run, where $m = n$ in the Euclidean metric and $m = n^2$ in the weighted metric. However, we show, through experimentation, that only a constant value of m is required (e.g., m = 8) in order to produce very accurate solutions (<1% error). Hence, for typical terrain data, the expected running time of our algorithm is $O(sn \log(sn))$. Also, as part of our experiments, we show that by using geometrical subsets (i.e., 2D/3D convex hulls, 2D/3D bounding boxes, and random selection) of the robots we can improve the running time for finding p', with minimal or no additional accuracy error when comparing p' to p^*.

Categories and Subject Descriptors: I.3.5 [**Computational Geometry and Object Modeling**]: Robotics

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: 1-Center, algorithms, approximation, meeting point, robots, shortest path, terrain, weighted

ACM Reference Format:

Lanthier, M. A., Nussbaum, D., and Wang, T.-J. 2008. Computing an approximation of the 1-center problem on weighted terrain surfaces. ACM J. Exp. Algor. 13, Article 1.3 (Sept. 2008), 29 pages. DOI = 10.1145/1412228.1412231 http://doi.acm.org/10.1145/1412228.1412231

Research supported, in part, by NSERC.
Authors' addresses: Mark A. Lanthier, Doron Nussbaum, and Tsuo-Jung Wang, Carleton University, Ottawa, Ontario, Canada; email: lanthier@scs.carleton.ca.

1. INTRODUCTION

In this article, we discuss the problem of determining an efficient meeting point of a set of scattered robots in a weighted terrain $\mathcal{P}$. A terrain $\mathcal{P} = \{f_1, f_2, \ldots, f_n\}$ is a continuous 2.5D polyhedral surface made of triangular faces, f_i, $1 \le i \le n$, which is formed by taking a 2D continuous triangular subdivision and assigning a height value to each unique vertex of the triangulation. Each face f_i is associated with a height function F_i such that for a particular point $q = (x, y)$ in f_i, then $F_i(q)$ is the height of q.

The problem addressed here can be defined as the weighted 1-center problem on polyhedral surfaces. Let $R = \{r_1, r_2, \ldots, r_s\}$ be a set of robots that are scattered throughout a polyhedral surface $\mathcal{P}$ (it is assumed that $n > s$). The weighted 1-center problem is defined as finding a point $p^* \in \mathcal{P}$ such that

$$\max_{1 \le i \le s}\{d(r_i, p^*)\} = \min_{p \in \mathcal{P}} \left\{ \max_{1 \le i \le s}\{d(r_i, p)\} \right\}$$

where $d(r_i, p^*)$ is an objective function (e.g., distance, time energy, etc...) between robot r_i and p^*.

There has been much work investigating the 1-center problem (also known as the facility location problem) in the Euclidean setting, L_2 metric, in both 2D and 2.5D and in L_1 metric in 2D and 3D. In the plane under the Euclidean metric, the optimal solution to this problem is the center of the smallest enclosing circle, which can be computed in time linear in the number of source points [Megiddo 1983b; Welzl 1991]. In fact, there is a close connection between the 1-center problem and the furthest-site Voronoi diagram in that the solution to the 1-center problem must lie on a vertex or edge of the diagram. It appears, however, that some of the 2D furthest-site Voronoi diagram properties are different for polyhedral surfaces and the combinatorial complexity of the diagram is $\Theta(sn^2)$ for s source points on an n-face terrain [van Oostrum 1999]. The work of van Trigt [1995] presents an algorithm to solve the facility location problem on a polyhedral terrain in $O(s^4 n^3 \log n)$ time. Aronov et al. [1998] improved on this with a near-optimal algorithm that computes the furthest-site Voronoi diagram and finds the facility center in $O(sn^2 \log^2 s \log n)$ time. Sharir [1996] provided a solution to the 2-center problem in 2D. In the L_1 metric, Drezner [1984] gave a linear time algorithm for solving the 1-center problem. Sharir and Welzl [1996] presented $O(n \log n)$ time and $O(n \log^5 n)$ algorithms for the 4- and 5-center problems in the plane, respectively. Nussbaum [1997] presented an $O(n \log n)$ for the 4- and 5-center problems in 2D. The results of Nussbaum can be extended to solve the weighted 4- and 5-center problems in the plane in $O(n \log^2 n)$ using the parametric search technique of Megiddo [1983a].

Despite the previous work in the Euclidean metric, the problem of computing the weighted 1-center problem is hard and has lacked sufficient research. This may be because of the fact that even the simpler problem of computing an approximation to a single weighted shortest path in 2D has an unpleasantly high runtime of $O(n^8 \log n)$ time [Mitchell and Papadimitriou 1991]. To our knowledge the work presented here is the first algorithm to provide an approximation to the weighted 1-center problem on polyhedral surfaces (e.g., terrains).

In addition to the work on the 1-center problem, there has been much work in computing shortest paths on polyhedral surfaces in both the Euclidean and weighted L_2 metric. Although we are solving a different problem, we briefly mention some of this previous work, since the techniques of polyhedral discretization are similar. Several research articles, including surveys, have been written presenting the state-of-the-art in this active field; we refer the interested reader to those Mitchell [1997, 2000]. Lanthier et al. [2001] apply a transformation technique from a continuous to a discrete problem, by adding Steiner points to $\mathcal{P}$ and then connecting the Steiner points to form a graph G. Once G is constructed, a single-source shortest path is computed in G. The work presented here is an extension of their work and it is applied to the weighted facility location problem on terrains. Aleksandrov et al. [1998; 2003] apply a different Steiner placement scheme to achieve an ϵ-approximation for shortest paths on $\mathcal{P}$. Their graph construction technique can be easily modified to solve the 1-center problem. More recently, Sun and Reif [2003] provided an ϵ-approximate solution to computing weighted shortest paths that does not depend on terrain parameters using an algorithm known as a *BUSHWHACK* algorithm.

By using this strategy of transforming the terrain surface into a graph, we can apply network-based solutions to the 1-center problem to find a solution which then maps directly back onto the terrain surface. One such algorithm is that of Kariv and Hakimi [1979] that finds the absolute 1-center of a vertex-weighted graph in $O(|E|n \log n)$ time for an n-vertex graph with $|E|$ edges and $O(|E|n+n^2 \log n)$ time for a vertex-unweighted graph. Their algorithm requires, as input, a distance matrix that contains distance/cost information between every pair of vertices in the graph. The algorithm can be modified and used to provide a solution to our problem, subject to a preprocessing step that computes the required distance matrix. In this case, the distance matrix would be of size $|R| \times |V^G|$, where R is the set of robots and V^G is the vertex set of the approximation graph. Our algorithm, on the other hand, although computing some distance information between the robots and the vertices, does not require computation of the complete distance matrix as part of the input. A further distinction between the algorithm presented here and that of Kariv and Hakimi [1979] is that the time complexity of their algorithm is not sensitive to the location of the robots, whereas our algorithm is input sensitive with respect to the initial robot locations. That is, if the robots are in close proximity to one other (e.g., clustered) then it is expected that the algorithm will complete its execution faster than if the robots were positioned farther apart on the terrain.

Letting $V^{\mathcal{P}}$ be the set of vertices of $\mathcal{P}$, the approximation algorithm presented here assumes that each robot is positioned on a vertex of $\mathcal{P}$ ($r_i \in V^{\mathcal{P}}, 1 \leq i \leq s$). Note that minor adjustments can be made to overcome this restriction that do not affect the runtime analysis. Our algorithmic approach is to produce a discretization of $\mathcal{P}$ by producing a graph $G = \{V^G, E^G\}$ that lies on the surface of $\mathcal{P}$. For a vertex $p' \in V^G$, we define $\Pi'(r_i, p')$ to represent the weighted shortest path in G from r_i to p'. Thus, $\|\Pi'(r_i, p')\|$ will represent the approximate

Table I. Terrains Used in Our Experimental Testing

Name	# Faces	# Vert.
America	9,788	5,000
Sanbern	15,710	8,000
Madagascar	29,582	15,000

minimum weighted cost of traveling on $\mathcal{P}$ from r_i to p'. We are, therefore, interested in computing $p' \in V^G$ such that

$$\max_{1 \le i \le s}\{\|\Pi'(r_i, p')\|\} = \min_{p \in V^G}\left\{\max_{1 \le i \le s}\{\|\Pi'(r_i, p)\|\right\}.$$

Let $\|\Pi(r_i, p^*)\|$ represent the minimum cost of travel on $\mathcal{P}$ from r_i to p^*, where p^* is the optimal meeting point in $\mathcal{P}$. If L is the longest edge of $\mathcal{P}$ and W is the maximum cost weight of a face of $\mathcal{P}$, we show here that

$$\max_{1 \le i \le s}\{\|\Pi'(r_i, p')\|\} \le \max_{1 \le i \le s}\{\|\Pi(r_i, p^*)\|\} + 2W|L|.$$

The simplicity and the practicality of our technique makes it attractive for computing an approximated solution to the weighted 1-center problem that minimizes the maximum cost for any particular robot using a variety of objective functions to represent the cost. Such objective functions include distance, time, or energy consumption. In addition, the algorithm is easily extended to find an approximated point $p' \in V^G$ that minimizes the cumulative cost of all of robots:

$$\sum_{i=1}^{s} d(r_i, p') = \min_{p \in V^G}\left\{\sum_{i=1}^{s} d(r_i, p)\right\}$$

This may be a more desirable solution if, for example, the objective is minimizing the fuel consumption of all robots and thus maximizing the amount of energy (e.g., fuel) remaining to complete the tasks.

In addition to theoretical bounds, we also present experimental results and show the practicality and accuracy of our solution. We ran tests on a variety of terrains that have different sizes as shown in Table I. Our tests were run in both the Euclidean and weighted metrics. We also tested our algorithm on terrains with different height characteristics. Thus, in addition to their normal heights, the terrains were *stretched* by multiplying the heights of all vertices by a factor of five and *flattened* by setting vertex heights to zero.

The theoretical worst-case running time of our algorithm is $O(snm\log(snm)+snm^2)$, where $m = n$ in the Euclidean metric and $m = n^2$ in the weighted metric. However, we show, through experimentation, that only a constant value of m is required (e.g., m = 8) in order to produce very accurate solutions (which result in $<1\%$ error). Hence, for typical terrain data, the expected running time of our algorithm is $O(sn\log(sn))$. Also, as part of our experiments, we show that by using geometrical subsets of the robots, we can improve the running time for finding p' with minimal or no additional accuracy error when comparing p' to

p^*. Examples of subsets are the 2D/3D convex hulls of R, 2D/3D bounding box of R and randomized subset selection.

2. OUR ALGORITHM

As mentioned, our algorithm begins with a discretization of $\mathcal{P}$ through the construction of a graph $G = \{V^G, E^G\}$, which is spatial network approximation of $\mathcal{P}$. In the following subsection, we describe the construction of a set of subgraphs (one subgraph per face of $\mathcal{P}$) whose union forms G. It is well known that a Euclidean shortest path $\pi(v_a, v_b)$ on $\mathcal{P}$ between vertices v_a and v_b of $\mathcal{P}$ is piecewise linear (i.e., consecutively joined straight line segments)[Sharir and Schorr 1986]. Moreover, such a path only *bends* (i.e., changes direction) at edges of $\mathcal{P}$ (see Sharir and Schorr [1986]). Similarly, Mitchell and Papadimitriou [1991] show that a weighted shortest path $\Pi(v_a, v_b)$ also exhibits similar characteristics.

Our graph is formed such that each edge in E^G corresponds to a line segment either crossing a single face of $\mathcal{P}$ or lying along an edge of $\mathcal{P}$. We, therefore, transform the problem of computing robot paths on the surface of $\mathcal{P}$ to that of computing robot paths in the approximating graph G. This allows us to compute a solution to the meeting point problem by running a variation of Dijkstra's graph shortest path algorithm on G. The resulting paths in G map directly onto paths lying on the surface of $\mathcal{P}$. The approximate meeting point solution is found by running the modified Dijkstra algorithm and observing the first vertex in G that is reached from all robots in R. Section 2.1 describes the construction of G. Section 2.2 then describes the modified multisource Dijkstra algorithm that is used to determine the meeting point.

2.1 Constructing Graph G

In this section we discuss the construction of a graph $G = (V^G, E^G)$ that corresponds to a terrain $\mathcal{P}$. Let $\mathcal{P} = \{f_1, f_2, \ldots, f_n\}$ be a terrain, which is composed of triangular faces, $f_i, 1 \le i \le n$. The vertex set V^G is constructed by first placing m evenly spaced Steiner points (called *edge Steiners*) along each edge $e \in E^{\mathcal{P}}$ (see Figure 1(a)). For each vertex of $\mathcal{P}$ and each edge Steiner, we create a corresponding vertex in V^G.

The creation of the edge set E^G is explained here by constructing subgraphs and then adding the edges of the subgraphs to E^G, although, in practice, E^G is constructed by connecting the appropriate vertices of V^G directly. The subgraphs are: a set of three chains each connecting m+1 edges, a set of three complete bipartite graphs $K_{1,m}$ and a set of three complete bipartite graphs $K_{m,m}$. Next, we describe how the edges of E^G are assembled:

1. Adding edges of chains—in this step, we construct a set of chain graphs as follows: for each edge $e = (u, v) \in E^{\mathcal{P}}$, we create a set of vertices $V^e = \{v_0, v_1, \ldots, v_{m+1}\}$ such that $v_0 = u$, v_1 is the edge Steiner on e that is closest to u, and v_2 is the edge Steiner on e that is second closest to $u, \ldots,$ and $v_{m+1} = v$. The chain is formed by adding edges $(v_i, v_{i+1}), 0 \le i \le m$ (see Figure 1a). Once constructed, the edges are added to E^G.

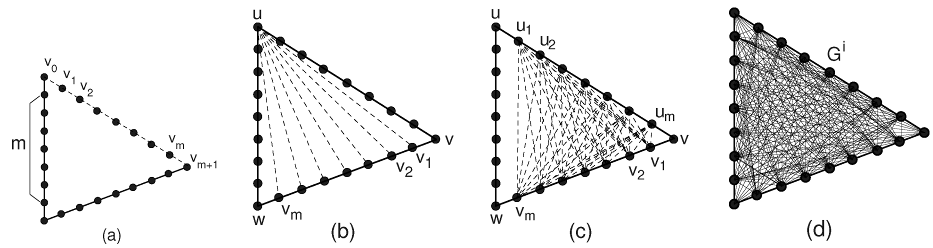

Fig. 1. (a) Steiner points and *edge Steiners* along edges of $\mathcal{P}$, (b) edges from a $K_{1,m}$ graph of face f_i, (c) edges from a $K_{m,m}$ graph of face f_i, and (d) the subgraph G_i pertaining to face f_i.

2. Adding edges of $K_{1,m}$—in this step, we create for each face $f_i \in \mathcal{P}$ three complete bipartite $K_{1,m}$ graphs. Let u, v, w be the three vertices that correspond to the vertices of f_i and let $\{v_1, \ldots, v_m\}$ be the edge Steiners along edge (v, w). We construct a complete bipartite graph $K_{1,m}$ by adding edges $(u, v_i), 1 \leq i \leq m$ (see Figure 1b). Once constructed, we add the edges of the constructed graph to E^G. Similarly, we create two other $K_{1,m}$ graphs between v and the edge Steiners along (u, w), and between w and the edge Steiners along (u, v).
3. Adding edges of $K_{m,m}$—in this step, we create for each face $f_i \in \mathcal{P}$ three complete bipartite graphs $K_{m,m}$. Let u, v, w be the three vertices that correspond to the vertices of f_i, let $\{u_1, \ldots, u_m\}$ be the edge Steiners along edge (u, v), and let $\{v_1, \ldots, v_m\}$ be the edge Steiners along edge (v, w). We construct a complete bipartite graphs $K_{m,m}$ by adding edges $(u_i, v_j), 1 \leq i, j \leq m$ (see Figure 1c). Once constructed, we add the edges of the constructed graph to E^G. Similarly, we create two other $K_{m,m}$ graphs between the edge Steiners along (u, v) and the edge Steiners along (u, w), and between the edge Steiners along (u, w) and the edge Steiners along (v, w).

Let G_i denote the portion of G that corresponds to face $f_i \in \mathcal{P}, 1 \leq i \leq n$ (see Figure 1d). In the following lemma, we show the bound on the size of G when m Steiner points are added on each edge of $\mathcal{P}$.

Lemma 2.1. *Graph $G = (V^G, E^G)$ has $V^G = O(nm)$ vertices and $E^G = O(m^2 n)$ edges, where $m \geq 1$.*

Proof. The set V^G includes the vertices of $\mathcal{P}$ as well as vertices that correspond to edge Steiners, thus, $|V^G| = |V^{\mathcal{P}}| + m|E^{\mathcal{P}}|$. Each face $f_i \in \mathcal{P}, 1 \leq i \leq n$ has three edges, thus, $|E^{\mathcal{P}}| \leq 3n$. Similarly, the number of faces in $\mathcal{P}$ is bounded yielding $|V^{\mathcal{P}}| \leq 3n$. This leads to $|V^G| = |V^{\mathcal{P}}| + m|E^{\mathcal{P}}| \leq 3n + 3nm = O(nm)$. The set E^G consists of edges that are the result of constructing chains as well as complete bipartite graphs $K_{1,m}$ and $K_{m,m}$. Each chain contributes $m + 1$ edges to E^G and, since we add one chain per edge of $\mathcal{P}$, we, therefore, add $|E^{\mathcal{P}}|(m + 1)$ total chain edges to E^G. For each face $f_i \in \mathcal{P}$, we add $3m$ edges for the three complete bipartite graphs of type $K_{1,m}$ and $3m^2$ edges for the three bipartite graphs of type $K_{m,m}$. Since $\mathcal{P}$ has n faces and at most $3n$ edges, we can conclude that $|E^G| = |E^{\mathcal{P}}|(m+1) + (3m^2 + 3m)n \leq 3n(m+1) + 3nm(m+1) = O(m^2 n)$. □

The following lemma explains the time required to construct graph G when m Steiner points are added per edge of $\mathcal{P}$.

LEMMA 2.2. *Let $\mathcal{P}$ be a terrain represented as a triangulation irregular network with n faces. Graph $G = \{V^G, E^G\}$ can be constructed in $O(nm)$ time and space, where $m \geq 1$.*

PROOF. For each edge $e \in E^{\mathcal{P}}$, we maintain an m-size array of the Steiner points that were placed along e. Thus, the vertices of G are stored in $O(n)$ arrays of size m. Since the Steiner points of e are placed evenly along e, we can compute their locations and create the arrays in $O(nm)$ time, requiring $O(nm)$ space. For each vertex, we store a pointer to the edge $e \in E^{\mathcal{P}}$ on which it lies. Each subgraph G_i of G represents a well-defined graph and, therefore, G can be constructed without explicitly creating and storing its edges. Assuming that $\mathcal{P}$ is stored as a quad edge data structure [Guibas and Stolfi 1985], then, in $O(1)$ time, each edge $e \in E^{\mathcal{P}}$ can be associated with the two faces (say f_j and f_k) that share e. Although not stored directly, we can compute and construct any edge, say $(u, v) \in E^G$ incident to a given vertex $u \in V^G$, where u lies on some edge $e \in E^{\mathcal{P}}$, within $O(1)$ time, as follows. If v also lies on e, then v is stored in the same array as u and, thus, (u, v) can be computed and constructed in constant time. If v does not lie on e, then we can determine the (up to four) edges of $\mathcal{P}$ that share a face with e in $O(1)$ time by using available pointers in the quad edge data structure of $\mathcal{P}$. We can then obtain the array containing v in $O(1)$ time, since the location of v is well-defined along one of these four edges. □

2.2 The Modified Dijkstra's Algorithm

After the construction of $G = \{V^G, E^G\}$, our algorithm searches for a vertex $p' \in V^G$ by invoking a graph shortest-path algorithm from each of the source vertices $R = r_1, r_2, \ldots, r_s \in V^G$ on which the robots are positioned. We modified the multiple-source single-target variation of Dijkstra's shortest-path algorithm [Dijkstra 1959] such that our algorithm stops at vertex p', which is the approximating meeting point. Intuitively, the strategy is to propagate outwards, in a wavefront fashion, from each source r_i. The best approximating meeting point p' within the graph G is the first point to be reached (processed) by all robots during their propagation.

Our implementation of the algorithm uses a min-heap Q as the priority queue to ensure that all wavefronts of the robots propagate at an equal rate. For each vertex $v \in V^G$, we maintain an array that holds the minimum traveling cost from each source point to v, which we denote as: $costs(v)[1], costs(v)[2], \ldots, costs(v)[s]$, where $costs(v)[i]$, $1 \leq i \leq s$, represents $|\Pi'(r_i, v)|$. Initially, these values are set to ∞ for each vertex, except for the vertices that represent the initial robot locations whose cost from their own starting location is set to 0 (i.e., $costs(r_i)[i] \leftarrow 0, 1 \leq i \leq s$).

For each vertex $v \in V^G$, we also keep a local min-heap, denoted local heap (v), that contains the indices of the robots. The local heap has size s and is organized by the cost (i.e., $costs(v)[i]$) of reaching v from the various sources. Thus, the top element of the local heap contains the index of the robot that has minimum cost in the array $costs(v)[i]$, $1 \leq i \leq s$. Let $minCostInd$ be the index of the robot at the top of the local heap (i.e., $minCostInd = [i \mid \min_{1 \leq i \leq s}\{costs(v)[i]\}]$). The min-heap Q, that is used by the algorithm to determine which vertex to process next, is ordered in ascending order of $costs(v)[minCostInd]$. Thus, at any time during the algorithm, the top element of the heap represents the vertex/source pair with the global minimum cost. The remainder of the algorithm is similar to Dijkstra's algorithm in that during the relaxation stage the heap Q is updated accordingly. We also maintain for each vertex, an array $parent(v)[i]$ that stores the vertex preceding it in the shortest-path $\Pi'(r_i, v)$. This allows us to trace backward from the meeting point and obtain the actual shortest paths in G from each r_i to p'.

The algorithm halts when a vertex v is extracted exactly s times from Q, indicating that v was the first vertex reached by the wavefronts of all robots $r_i, 1 \leq i \leq s$. A more complete description of the algorithm is given in the pseudocode in Algorithm 1, which takes as input an approximating graph $G = (V^G, E^G)$ of terrain $\mathcal{P}$ using m Steiner points and a set of source vertices $R = r_1, r_2, \ldots, r_s$. It returns a vertex $p' \in V^G$ representing the meeting-point solution.

Before we discuss the time and space complexity of our algorithm, we show, in the following lemma, that the algorithm is correct.

Lemma 2.3. *Let $G = (V^G, E^G)$ be an approximating graph of terrain $\mathcal{P} = \{V^{\mathcal{P}}, E^{\mathcal{P}}\}$ with n faces using $m \geq 1$ Steiner points per edge of $\mathcal{P}$ and let $R = r_1, r_2, \ldots, r_s$ be a set of robots that require to meet at a point. The meeting point p', which Algorithm 1 returns when using G as the approximating graph of $\mathcal{P}$, is the best approximation of the optimal meeting point p^*.*

Proof. Each time a vertex v is removed from Q, line 22 of Algorithm 1 ensures that the top element is removed from local heap (v), indicating that this vertex has been reached by one of the robot's wavefronts, say r_j. At this point the shortest path $\Pi(r_j, v)$ is known, since all weights of G are positive, implying that during the relaxation step (i.e., lines 27–29) the cost associated with v in Q can only increase. After its initial construction, local heap (v) never grows during the algorithm and shrinks by one in size exactly s times (i.e., once for each robot). Hence, when local heap (v) is empty, this indicates that all shortest paths $\Pi(r_i, v)$, $1 \leq i \leq s$ to v have been determined. The first such vertex $v = p'$, whose local heap becomes empty, therefore, has all shortest paths to it computed and is returned from Algorithm 1 in lines 23 through 24. Let $C^{p'} = \max_{1 \leq i \leq s}(|\Pi'(r_i, p')|)$ be the largest of these path costs (i.e., representing the robot furthest from the meeting point). Since Q is sorted by ascending order of costs for each source point, any vertex, say u, whose local heap (u) becomes empty at a later time, must necessarily have cost $C^u \geq C^{p'}$, and thus represent a meeting point that does not minimize $\max_{1 \leq i \leq s}(|\Pi'(r_i, p')|)$. $\square$

Algorithm 1. FindMeetingPoint(*G, R*)

```
for each vertex v of G do
    for i ← 1 to s do
        costs(v)[i] ← ∞
        parent(v)[i] ← ∅
        insert i into local-heap(v)
    end for
    minCostInd ← top(local-heap(v))
    insert v into Q using costs(v)[minCostInd] as the key
end for
for i ← 1 to s do
    v ← vertex at which r_i is positioned
    costs(v)[i] ← 0
    minCostInd ← i
    update local-heap(v)
    update v in Q using costs(v)[minCostInd]
end for
while TRUE do
    v ← top(Q)
    remove v from Q
    minCostInd ← top(local-heap(v))
    costV ← costs(v)[minCostInd]
    remove top element from local-heap(v)
    if local-heap(v) is empty then {All sources were processed}
        return(v)
    end if
    for each vertex u adjacent to v do
        if costs(u)[minCostInd] > (costV + weightedCost(v, u)) then
            costs(u)[minCostInd] ← costV + weightedCost(v, u)
            parent(u)[minCostInd] ← v
            update local-heap(u)
            if minCostInd = top(local-heap(u)) then {minimum cost of u has changed}
                update Q using costs(u)[minCostInd] as key
            end if
        end if
    end for
end while
```

The following lemma describes the theoretical worst-case runtime requirements for this phase of our algorithm.

LEMMA 2.4. *Let $G = (V^G, E^G)$ be an approximating graph of terrain $\mathcal{P} = \{V^{\mathcal{P}}, E^{\mathcal{P}}\}$ with n faces using $m \geq 1$ Steiner points per edge of $\mathcal{P}$, and let $R = r_1, r_2, \ldots, r_s$ be a set of robots that require to meet at a point. Algorithm 1 requires, in the worst case, $O(snm^2 \log(snm))$ time, to return the best approximating meeting point $p' \in V^G$.*

Proof. Lemma 2.1 ensures that $|V^G| = O(nm)$ and so the initialization phase of the algorithm (i.e., lines 1–16) takes only $O(snm + s\log(snm))$ time. The runtime bound of our algorithm is, therefore, determined by the WHILE loop of lines 17–36 of Algorithm 1. In the worst case under the weighted metric, the meeting point can be one of the last vertices that are removed from the heap. Thus, we assume that the algorithm executes until the heap Q is empty. Each vertex is removed from the Q exactly once for each source point. Again using Lemma 2.1, we may remove vertices from the heap up to $O(snm)$ times, where each removal (i.e., line 19) costs $O(\log(nm))$. in addition, removing the element from the local heap in line 22 requires $O(\log s)$ time. Similar to Dijkstra's single-source shortest paths algorithm, the body of the "FOR" loop (i.e., lines 27–34) is executed, at most, twice for each edge in E^G, and lines 28–33 are executed, at most, once per edge, where the cost of execution is $O(\log(nm) + \log(s))$. Since Lemma 2.1 ensures that $|E^G| = O(nm^2)$, the "FOR" loop body may require $O(nm^2(\log(nm) + \log(s)))$ time. Applying this analysis for all s sources, lines (i.e., lines 27–34) contribute $O(snm^2(\log(nm) + \log(s)))$ time to the overall runtime cost. Therefore the overall cost of the algorithm is $O(snm(\log(snm)) + snm^2(\log(snm)))$. □

The space requirements of the algorithm are greatly impacted by the fact that the edges of the approximating graph G are stored implicitly. The implicit storage, which does not affect the time complexity of the algorithm, is achieved because of the symmetric and systematic construction of G.

Lemma 2.5. *Let $G = (V^G, E^G)$ be an approximating graph of terrain $\mathcal{P} = \{V^{\mathcal{P}}, E^{\mathcal{P}}\}$ with n faces using $m \geq 1$ Steiner points per edge of $\mathcal{P}$, and let $R = r_1, r_2, \ldots, r_s$ be a set of robots that require to meet at a point. Algorithm 1 requires $O(snm)$ space during its execution.*

Proof. Algorithm 1 maintains, during its execution, two types of priority queues that are min-heaps: a global min-heap Q and some local heaps. The global queue Q maintains an entry for each vertex in V^G and thus it requires $O(nm)$ space. During the execution, each vertex $v \in V^G$ is assigned a local heap that maintains, in priority order, the robot that must be processed next. Since all robots may pass through v, all the local heap s of all vertices require $O(snm)$ space. Similarly, each vertex $v \in V^G$ maintains an array of size s storing the travel cost of each robot that traverses through v, which requires an additional $O(snm)$ space. □

Using the results of Lemmas 2.4 and 2.5, we can present the following theorem.

Theorem 2.6. *Let $G = (V^G, E^G)$ be an approximating graph of terrain $\mathcal{P} = \{V^{\mathcal{P}}, E^{\mathcal{P}}\}$ with n faces using $m \geq 1$ Steiner points per edge of $\mathcal{P}$ and let $R = r_1, r_2, \ldots, r_s$ be a set of robots that require to meet at a point. Algorithm 1 requires, in the worst case, $O(snm^2\log(snm))$ time and $O(snm)$ space to return the best approximating meeting point $p' \in V^G$.*

Fredman and Tarjan [1987] and Driscoll et al. [1988] have shown that a modification to Dijkstra's algorithm using Fibonacci [Fredman and Tarjan 1987]

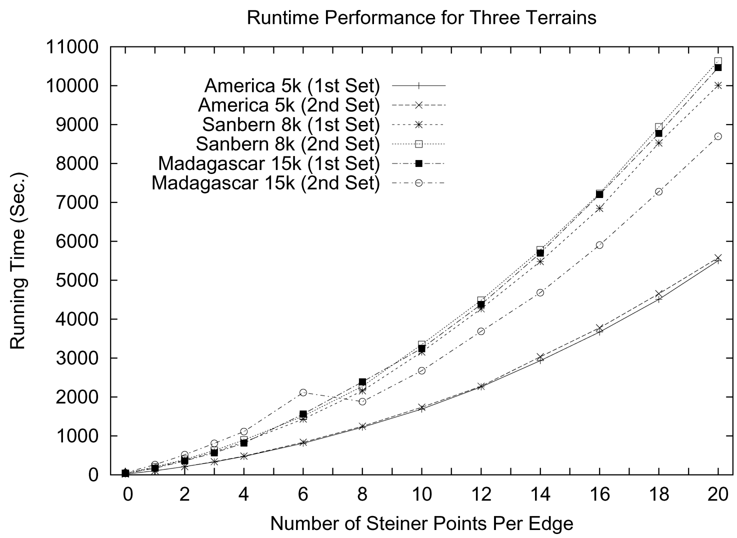

Fig. 2. Graph showing runtime performance for three terrains as the number of Steiner points per edge increases.

or relaxed heaps [Driscoll et al. 1988] can reduce the running time for finding a shortest paths in a graph to $O(|V^G|\log|V^G| + |E^G|)$ by amortizing the costs of updating the heap structure. A similar argument holds for updating the local heap of each vertex $v \in V^G$.[1] By applying their results, we can further reduce the time complexity, as shown in the following corollary.

Corollary 2.7. *Let $G = (V^G, E^G)$ be an approximating graph, which uses $m \geq 1$ Steiner points per edge of $\mathcal{P}$ to approximate terrain $\mathcal{P} = (V^{\mathcal{P}}, E^{\mathcal{P}})$ with n faces and let $R = r_1, r_2, \ldots, r_s$ be a set of robots that require to meet at a point. Algorithm 1 requires, in the worst case, $O(snm\log(snm)+snm^2)$ time and $O(snm)$ space to return the best approximating meeting point $p' \in V^G$.*

Through our experiments, we were able to observe the quadratic nature of the runtime performance with respect to the value of m. Figure 2 shows the runtime performance obtained from tests on three different terrains, using two different source point sets for R.[2] Notice the significance that both n and m play in the runtime performance, since the larger terrains require much more computation time. As will be seen later, the effect of $s = |R|$ is also quite significant. We will show how reducing s can dramatically effect the runtime performance of our algorithm.

[1]Our implementation did not use Fibonacci or relaxed heaps.

[2]The first two graphs result from tests on a Sparc Ultra II (dual 400-MHz 32-bit processors) with 512 M of memory. Because of the large terrain size of the Madagascar terrain, the third graph shows results from running on an Itanium II (dual 900-MHz 64-bit processors) with 2 GB of memory.

3. QUALITY OF THE APPROXIMATION

In this section, we discuss the approximation bound for the algorithm. Let $\mathcal{P}$ be a terrain with n faces and let $R = r_1, r_2, ..., r_s$ be a set of robots lying on vertices of $\mathcal{P}$. Let p^* be the optimal meeting point of R. Assume that p^* lies within some face f_j of $\mathcal{P}$, $1 \leq j \leq n$. From the construction of G^j, we notice a simple property that can be used in analyzing path costs passing through face f_j. The following property holds:

PROPERTY 3.1. *If p_0 and p_{m+1} are the endpoints of an edge $e \in E^{\mathcal{P}}$ and p_i and p_{i+1} are two adjacent points on e (either a Steiner point or endpoint of e) where $1 \leq i \leq m$, then $|\overline{p_i p_{i+1}}| = \frac{|e|}{m+1}$.*

The following are two fundamental properties of shortest paths on polyhedral surfaces that we will need later.

PROPERTY 3.2 ([SHARIR AND SCHORR 1986]). *A Euclidean shortest-path $\pi(r_i, p^*)$ on $\mathcal{P}$ may not pass through more than n faces and so it may have $\theta(n)$ segments.*

PROPERTY 3.3 ([MITCHELL AND PAPADIMITRIOU 1991]). *A weighted shortest paths $\Pi(r_i, p^*)$ on $\mathcal{P}$ may cross a face $\theta(n)$ times and so it may have $\theta(n^2)$ segments.*

To begin our approximation quality analysis, we first examine the cost of an approximated path $\Pi'(r_i, p_i')$ in G from a single-robot location r_i to an approximate meeting point $p_i' \in V^G$, where $1 \leq i \leq s$. We show that this path is bounded with respect to the actual shortest paths $\Pi(r_i, p_i')$ on $\mathcal{P}$ from r_i to p_i'. We can then compare the bound $\max_{1 \leq i \leq s}\{\|\Pi(r_i, p_i')\|\}$ with $\max_{1 \leq i \leq s}\{\|\Pi(r_i, p^*)\|\}$ to obtain a meaningful estimate of the accuracy.

Recall from the algorithm description that $\Pi'(r_i, p_i')$ corresponds to a path in G whose edges also happen to lie on $\mathcal{P}$. For the purposes of simplifying the analysis, we will assume that $\Pi'(r_i, p_i')$ passes through the same edge sequence (and hence faces) as $\Pi(r_i, p_i')$. We will bound $\Pi'(r_i, p_i')$ under this assumption, although, in practice, the search phase of our algorithm may produce a shorter path in G, thereby improving the accuracy stated here.

First we show how to bound the cost of a single shortest-path segment within a face. Let $s_j = \overline{ab}$ be a segment of $\Pi(r_i, p_i')$ that passes through a particular face f_j of $\mathcal{P}$ and let a and b lie on edges e_a and e_b of f_j, respectively. Let $s_j' = \overline{uv}$ be a segment such that u is the Steiner point on e_a that is closest to a and v is the Steiner point on e_b that is closest to b (see Figure 3). Assuming that f_j has weight w_{f_j}, then the following lemma bounds the weighted cost $\|\overline{uv}\|$ w.r.t. $\|\overline{ab}\|$:

LEMMA 3.4. $\|\overline{uv}\| \leq \|\overline{ab}\| + w_{f_j} \cdot \frac{\max\{|e_a|,|e_b|\}}{m+1}$

PROOF. Without loss of generality, assume that $e_a \neq e_b$ (although the lemma also holds when $e_a = e_b$). Since u and v were chosen as the closest of the two Steiner points adjacent to a and b, respectively, then by Property 3.1 we obtain $|\overline{ua}| \leq \frac{1}{2}\frac{|e_a|}{m+1}$ and $|\overline{bv}| \leq \frac{1}{2}\frac{|e_b|}{m+1}$. The triangle inequality ensures that

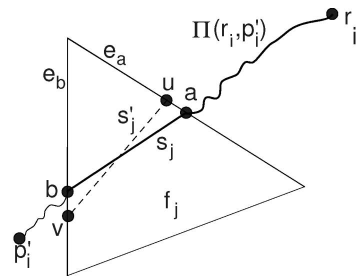

Fig. 3. Approximating a segment $\overline{ab}$ with a segment $\overline{uv}$.

$|\overline{uv}| \leq |\overline{ua}| + |ab| + |\overline{bv}|$. Hence,

$$|\overline{uv}| \leq |\overline{ab}| + \frac{|e_a|}{2(m+1)} + \frac{|e_b|}{2(m+1)} \leq |\overline{ab}| + \frac{\max\{|e_a|, |e_b|\}}{m+1} \tag{1}$$

In the weighted metric, the cost of travel through f_j is w_{f_j}, which can then be applied to Equation (1) as follows:

$$\|\overline{uv}\| = w_{f_j} \cdot |\overline{uv}| \leq w_{f_j} \left(|\overline{ab}| + \frac{\max\{|e_a|, |e_b|\}}{m+1} \right) = \|\overline{ab}\| + w_{f_j} \cdot \frac{\max\{|e_a|, |e_b|\}}{m+1} \tag{2}$$

Note that we are using an upper bound that assigns a weight of w_{f_j} to both $\overline{ua}$ and $\overline{bv}$. If the faces adjacent to f_j have a smaller weight, then the weights on $\overline{ua}$ and $\overline{bv}$ would be reduced and the bound on $\|\overline{uv}\|$ would be better than the one stated herein. □

Next, we show how to bound the entire k-segment shortest paths $\Pi(r_i, p_i')$. We compute the bound by constructing a corresponding bounded-path $\Pi'(r_i, p_i')$ in G. Each segment $\overline{ab}$ of $\Pi(r_i, p_i')$ is approximated with a segment $\overline{uv}$ of $\Pi'(r_i, p_i')$ and we use Lemma 3.4 to bound the cost of all such approximating segments. Let L be the longest edge of $\mathcal{P}$ and $W = \max_{1 \leq j \leq n}\{w_{f_j}\}$ be the maximum weight of any face of $\mathcal{P}$. We now introduce the following lemma:

LEMMA 3.5. *Given two vertices $v_a, v_b \in V^G$, there exists a path $\Pi'(v_a, v_b)$ in G such that $\|\Pi'(v_a, v_b)\| \leq \|\Pi(v_a, v_b)\| + k \cdot \frac{W|L|}{m+1}$, where k is the number of segments of $\Pi(v_a, v_b)$.*

PROOF. Let $\Pi(v_a, v_b) = \{s_1, s_2, \ldots, s_k\}$ and $\Pi'(v_a, v_b) = \{s_1', s_2', \ldots, s_{k'}'\}$. We consider an approximate path that passes through the same edge sequence (i.e., $k = k'$), although the algorithm may produce a better path. For each s_i, $1 \leq i \leq k$, we choose s_i' as described in Lemma 3.4, such that $\|s_i'\| \leq \|s_i\| + \frac{w_{f_{s_i}}|L_{f_{s_i}}|}{m+1}$, where $w_{f_{s_i}}$ and $L_{f_{s_i}}$ are the weight and longest edge of the face through which s_i crosses. In the special case where s_i travels along an edge e of $\mathcal{P}$, then $w_{f_{s_i}}$ and $L_{f_{s_i}}$ are the largest weight and longest edge of the two faces sharing e. By applying this bound to each segment s_i' of $\Pi'(v_a, v_b)$, we obtain:

$$\sum_{i=1}^{k} \|s_i'\| \leq \sum_{i=1}^{k} \left(\|s_i\| + w_{f_{s_i}} \cdot \frac{|L_{f_{s_i}}|}{m+1} \right)$$

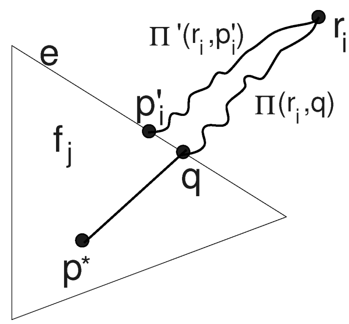

Fig. 4. Selection of p_i' for path $\Pi'(r_i, p_i')$.

This can be rewritten as:

$$\|\Pi'(v_a, v_b)\| \leq \|\Pi(v_a, v_b)\| + \frac{1}{m+1} \cdot \sum_{i=1}^{k} \left(w_{f_{s_i}} |L_{f_{s_i}}|\right)$$

Since $w_{f_{s_i}} \leq W$ and $|L_{f_{s_i}}| \leq |L|$ for all f_{s_i} by definition, then we obtain the bounds stated in the Lemma. □

Until now, we have bounded paths on $\mathcal{P}$ between vertices in G. However, the likelihood that the optimal meeting point p^* would lie on a vertex of G is very low and, therefore, we adjust our bound on each individual robot's approximating path in cases where p^* lies interior to a face $\mathcal{P}$. Assume that p^* lies in face f_j and let $p' \in V^{G^j}$ be a vertex of G^j. Since $p', r_i \in V^G$, there exists a shortest path $\Pi'(r_i, p')$ in G. We will show a bound for the solution p', although it should be noted that the searching phase of our algorithm (i.e., Dijkstra's shortest path search in G) may produce a better solution.

Lemma 3.6. *Given $r_i \in V^G, 1 \leq i \leq s$, there exists a point $p_i' \in V^G$ such that $\|\Pi'(r_i, p_i')\| \leq \|\Pi(r_i, p^*)\| + \frac{W|L|(1+2k)}{2(m+1)}$, where k is the number of segments of $\Pi(r_i, p^*)$.*

Proof. Let f_j be the face containing p^* and let q be the point on an edge e of f_j through which $\Pi(r_i, p^*)$ enters f_j. We assume that r_i lies outside f_j (although the lemma also holds when r_i is a vertex of f_j). Let p_i' be the Steiner point on e that is closest to q (see Figure 4). In the degenerate case, q is one of the vertices of f_j.
Then by Lemma 3.5, we can bound the approximate cost of $\Pi'(r_i, p_i')$ as follows:

$$\|\Pi'(r_i, p_i')\| \leq \|\Pi(r_i, p_i')\| + \frac{W|L|k}{m+1} \tag{3}$$

From the definition of a shortest path, we are ensured that:

$$\|\Pi(r_i, p_i')\| \leq \|\Pi(r_i, q)\| + \|\Pi(q, p_i')\| \leq \|\Pi(r_i, q)\| + W|\overline{qp_i'}| \tag{4}$$

and so

$$\|\Pi(r_i, p_i')\| \leq \|\Pi(r_i, p^*)\| \; + W|\overline{qp_i'}| \tag{5}$$

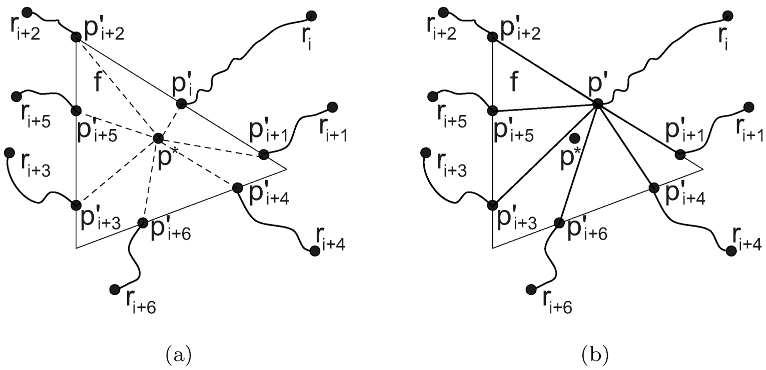

Fig. 5. (a) Various values of p'_i where $1 \le i \le s$. (b) The choice of p' and the resulting path approximations.

Substituting Equation (5) into (3) results in:

$$\|\Pi'(r_i, p'_i)\| \le \|\Pi(r_i, p^*)\| + W|\overline{qp'_i}| + \frac{W|L|k}{m+1} \tag{6}$$

Using Property 3.1 along with the fact that p'_i was chosen as the closest Steiner to q, we can deduce that $|\overline{qp'_i}| \le \frac{|L|}{2(m+1)}$ and apply this to Equation (6) to obtain

$$\|\Pi'(r_i, p'_i)\| \le \|\Pi(r_i, p^*)\| + \frac{W|L|(1+2k)}{2(m+1)}. \quad \square$$

LEMMA 3.7. *There exists a point $p' \in V^G$ such that for every $r_i \in R$, $1 \le i \le s$, $\|\Pi'(r_i, p')\| \le \|\Pi(r_i, p^*)\| + \frac{W|L|(2m+2k+3)}{2(m+1)}$.*

PROOF. Assume that p^* lies in face f. Lemma 3.6 ensures that for each robot r_i, $1 \le i \le s$ there exists a Steiner point (or vertex) p'_i along an edge of f such that $\|\Pi'(r_i, p'_i)\| \le \|\Pi(r_i, p^*)\| + \frac{W|L|(1+2k)}{2(m+1)}$. In general, however, the p'_i may not be the same for different values of i, as shown in Figure 5(a). Assume that $p' = p'_i$ as shown in Figure 5(b). For all $j \ne i$, $1 \le j \le s$, assume that the path $\Pi'(r_j, p') = \Pi'(r_j, p'_j) + \overline{p'_j p'}$ (although a better path may exist). Clearly, $|\overline{p'_j p'}| \le L$ and also $||\overline{p'_j p'}|| \le W|L|$. Therefore, $\|\Pi'(r_j, p')\| \le \|\Pi(r_j, p^*)\| + \frac{W|L|(1+2k)}{2(m+1)} + W|L| = \|\Pi(r_j, p^*)\| + \frac{W|L|(2m+2k+3)}{2(m+1)}$. $\square$

We now introduce the main theorem that bounds the runtime and accuracy of our algorithm.

THEOREM 3.8. *Let L be the longest edge of $\mathcal{P}$ and W be the maximum weight among all faces of $\mathcal{P}$. Let p^* be the optimal meeting point of a set of robots initially placed at vertices $r_1, r_2, \ldots, r_s$ of $\mathcal{P}$. Algorithm 1 produces, in $O(sn^5)$ time, an approximation p' of p^* such that*

$$\max_{1\le i\le s}\{\|\Pi(r_i, p')\|\} - \max_{1\le i\le s}\{\|\Pi(r_i, p^*)\|\} \le 2W|L|$$

PROOF. From Lemma 3.7, there exists a point $p' \in \{p'_j \mid 1 \le j \le s\}$, such that $\|\Pi'(r_i, p')\| \le \|\Pi(r_i, p^*)\| + \frac{W|L|(2m+2k+3)}{2(m+1)}$. The approximate path $\Pi'(r_i, p^*)$

that we have been analyzing was assumed to travel through the same faces as $\Pi(r_i, p^*)$. Therefore, if we set $m = k$ from Lemma 3.7, we obtain

$$\|\Pi'(r_i, p')\| \leq \|\Pi(r_i, p^*)\| + \frac{W|L|(4k+3)}{2k+2} \leq \|\Pi(r_i, p^*)\| + 2W|L|$$

Since $\|\Pi(r_i, p')\| \leq \|\Pi'(r_i, p')\|$, then $\|\Pi(r_i, p')\| \leq \|\Pi(r_i, p^*)\| + 2W|L|$. For every r_i, $1 < i < s$ it is easily seen that

$$\|\Pi(r_i, p')\| \leq \max_{1 \leq j \leq s}\{\|\Pi(r_j, p^*)\|\} + 2W|L|$$

and since this holds true for all i, then

$$\max_{1 \leq i \leq s}\{\|\Pi(r_i, p')\|\} - \max_{1 \leq i \leq s}\{\|\Pi(r_i, p^*)\|\} \leq 2W|L|$$

To analyze the time complexity of the algorithm, we will assume that $s < n$. Corollary 2.7 shows that our algorithm requires $O(snm \log(nm) + snm^2)$ running time. From Property 3.3, it follows that $\Pi(r_i, p^*)$ may have in the worst-case $\Theta(n^2)$ segments, so $m = O(n^2)$ from our assumption. Therefore, in the worst-case, the algorithm requires $m = n^2$ Steiner points per edge of $\mathcal{P}$ to obtain the given approximation bound with a running time of $O(sn^5)$. □

Corollary 3.9. *Let L be the longest edge of $\mathcal{P}$. Let p^* be the optimal meeting point of a set of robots initially placed at vertices $r_1, r_2, \ldots, r_s$ of $\mathcal{P}$. Algorithm 1 produces in $O(sn^3)$ time, an approximation p' of p^* such that*

$$\max_{1 \leq i \leq s} |\pi'(r_i, p')| - \max_{1 \leq i \leq s} |\pi(r_i, p^*)| \leq 2|L|$$

Proof. The path accuracy follows from Theorem 3.8 where $W = 1$. As for the time complexity, Property 3.2 indicates that $\pi(r_i, p^*)$ may have, at most, $\Theta(n)$ segments. Thus, only $m = n$ Steiner points are required per edge of $\mathcal{P}$ to ensure the desired accuracy. Corollary 2.7 shows that the running time is, therefore, $O(sn^3)$. □

Claim 3.10. *In practice (i.e., for a typical terrain), Algorithm 1 requires only a constant number of Steiner points per edge to obtain a very accurate solution. Hence, the algorithm has $O(sn \log(sn))$ expected running time.*

4. EXPERIMENTAL ANALYSIS OF PATH ACCURACY

To verify Claim 3.10, we tested the effect of increasing the number of Steiner points per edge of $\mathcal{P}$ on the quality of the solutions. Ideally, to assess our algorithm's solution accuracy, we would like to show that the difference between the approximate solution with that of the optimal solution is small. That is,

$$\max_{1 \leq i \leq s}\{\|\Pi(r_i, p')\|\} - \max_{1 \leq i \leq s}\{\|\Pi(r_i, p^*)\|\} \approx 0 \tag{7}$$

However, for a given robot r_i, $1 \leq i \leq s$, our algorithm does not compute the path cost $\|\Pi(r_i, p')\|$, but instead computes $\|\Pi'(r_i, p')\|$, which approximates the shortest path cost from r_i to the approximated solution p'.

As the number of Steiner points added per edge of $\mathcal{P}$ increases (i.e., $m \to \infty$), it can be easily proved that $\|\Pi'(r_i, p')\| \to \|\Pi(r_i, p')\|$ for all $1 \leq i \leq s$. In our experiments, we define

$$MaxCost = \max_{1 \leq i \leq s}\{\|\Pi'(r_i, p')\|\}$$

and we measure the algorithm's accuracy by examining the improvement of *MaxCost* as $m \to \infty$. Letting $MaxCost_m$ denote the *MaxCost* when m Steiner points are added per edge of $\mathcal{P}$, we show that $|MaxCost_m - MaxCost_{m+1}| \to 0$ as $m \to \infty$. In the unlikely case in which p^* falls on an edge of $\mathcal{P}$, this method of measuring solution accuracy shows that we approach the optimal solution as $m \to \infty$. When p^* lies within a face of $\mathcal{P}$, we would need to add Steiner points to the interior faces of $\mathcal{P}$ to be able to indicate how close the solution is to optimal. We ran some preliminary experiments in which we did place Steiner points interior to faces of $\mathcal{P}$ in order to assess the significance of the final path links. The improvement to the overall cost of the path was negligible, because only the interior points, where p^* is located, contributed to the path cost. Thus, the large increase in computation did not merit the negligible improvement and makes this option impractical. For the special case using the Euclidean metric on flat terrains, we can compute $\max_{1 \leq i \leq s}\{\|\Pi(r_i, p^*)\|\}$ as the center of the smallest circle enclosing the points in R and substitute this solution into Equation (7) above in order to compare the algorithm's accuracy with the optimal solution.

4.1 Effects on MaxCost When Increasing m

We first examined the effects that increasing m had on *MaxCost* for both the Euclidean and weighted metrics. Figure 6 shows six graphs pertaining to tests that were conducted on our three terrains with normal and flattened height values using the Euclidean metric. Each graph shows two data sets representing two completely different sets of robots on the terrain. Each subset R was chosen as a random uniform distribution of 100 points with respect to the x and y dimensions across the terrain surface. Although no meaningful comparison can be made between these two data sets, they are shown on the same graph to save space. Here, *MaxCost* represents the maximum cost that any robot would need to travel to reach the meeting point (the units of *MaxCost* depend on the objective function $d(r_i, p^*)$ that is used. In our tests, we used the Euclidean length multiplied by the face slope). From the graphs on the left side, notice how *MaxCost* decreases as the number of Steiner points per edge of $\mathcal{P}$ increases. Also notice that this cost converges quickly with very little improvement after 10 Steiner points are used.

Notice that for small values of m, the convergence is not smooth. For example, for the flattened Sandbern terrain, the cost for $m = 2$ is greater than the cost for $m = 1$ for the first dataset. This is because of the fact that the placement of a single Steiner point along the center of the edges resulted in a better overall path than when two Steiner points were placed a third and two thirds of the way along the edges (i.e., shifted to different positions along the edge). As more Steiner points are added, however, this *shifting* along the edge has less of an effect of the overall path. For example, when placing one Steiner point, the Steiner point is placed at the center of the edge. When placing two

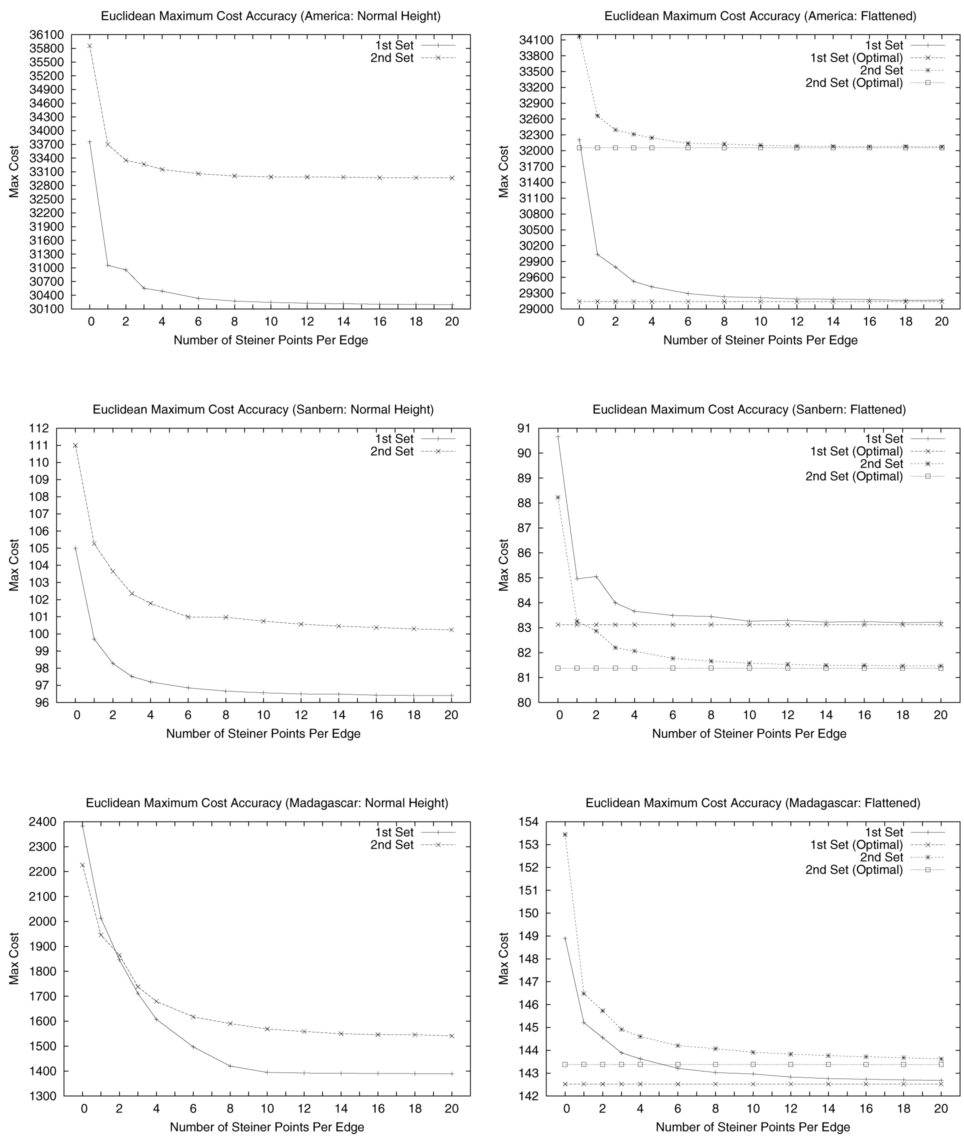

Fig. 6. Graphs showing how solution accuracy increases with the number of Steiner points for normal and flattened terrains.

Steiner points they are placed at locations 1/3 and 2/3 along the edge length. However, when placing three Steiner points the points are placed at 1/4, 1/2, and 3/4 and, therefore, the result is at least as good as placing a single Steiner point.

The graphs on the right side of Figure 6 show results pertaining to tests conducted on flattened terrains (height values of zero). Since the terrain is flat for these tests, we were able to compare the algorithm output with the optimal solution, which is the center of the smallest enclosing circle of all source points.

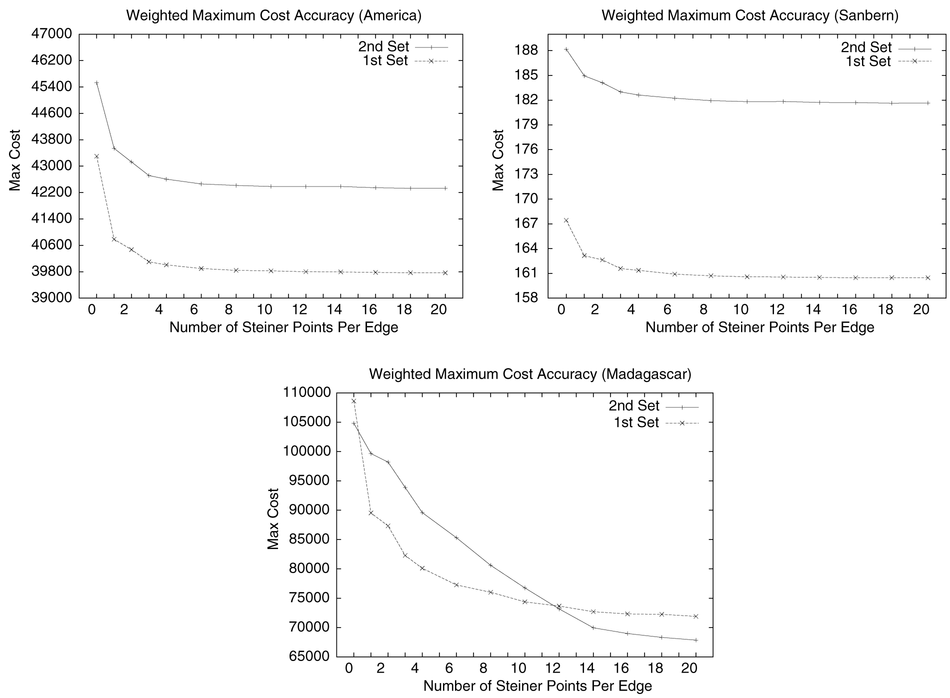

Fig. 7. Graphs showing MaxCost accuracy for weighted terrains.

Notice that our algorithm does indeed converge very quickly to a near-optimal solution.

In the weighted setting, we conducted tests in which weights were assigned to the terrain faces as a function of their slopes (i.e., steeper faces are usually considered more costly to traverse). However, it should be noted that our algorithm applies to terrains with arbitrarily chosen weights (e.g., based on terrain features such as water, forest, mountainous, and desert). The results of these tests are shown in Figure 7. Notice that similar convergence behavior is obtained as in the Euclidean metric, although the more spiky Madagascar terrain (that has overly exaggerated heights) has slower convergence in the weighted setting. It would be interesting to compare the results of an exact solution to this problem, however, in the weighted setting, we are unaware of any implemented algorithm for determining the optimal solution to the weighted one-center problem. We do conjecture that for typical terrains, our algorithm does converge close to the optimal solution when using only a constant number of Steiner points.

4.2 Iterative Usage of Steiner Points

Our algorithm can be used in an iterative manner by progressively adding Steiner points along edges of $\mathcal{P}$ as needed. In practice, the user may wish to compute a meeting point first using only a small number of Steiner points, say, m per edge, and then recompute again using $m+1$ per edge. If m is kept small, the solution is computed quickly. Depending on the accuracy improvement

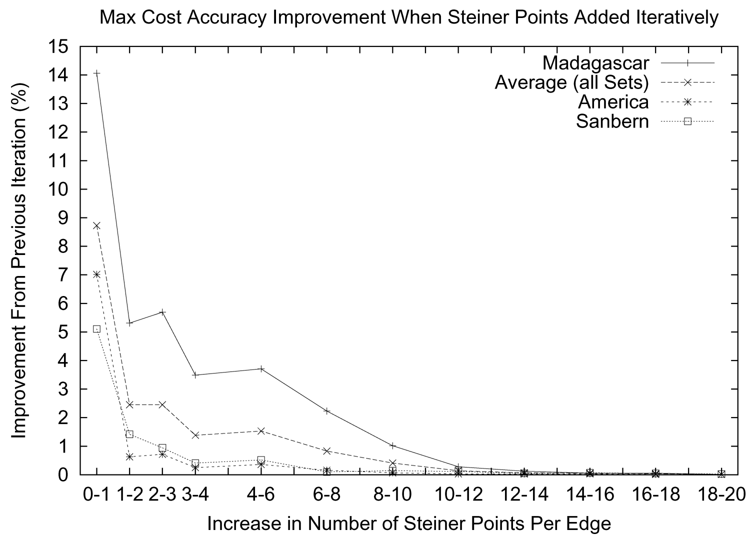

Fig. 8. Graph showing the improvement of *MaxCost* accuracy when iteratively adding Steiner points.

achieved when advancing from m to $m+1$ points, the user can decide whether or not it would be beneficial to add more Steiner points. If the accuracy improvement is insignificant then the user may elect to refrain from further computation. In many situations, it may be much quicker to compute two solutions using m and $m+1$ Steiner points per edge rather than running a single test using $M >> m$ per edge (since often a large value of m results in the need to use swap memory, in practice, which drastically slows down computation time).

The graph in Figure 8 shows the results of iteratively adding Steiner points to the terrain edges. We compute the improvement from m to $m+1$ Steiner points as

$$(|MaxCost_m - MaxCost_{m+1}|/MaxCost_m * 100)\%$$

Notice that the largest relative improvement occurs between the first few Steiner points (2–4 Steiner points per edge). When approximately 8 Steiner points are used, the additional use of Steiner points results only in a minor improvement of less than 1% and there is negligible improvement in solution accuracy after 14 Steiner points are used. This shows experimentally that Claim 3.10 is valid.

4.3 Fine-Tuning the Path Accuracy

Since we compute only "approximate" paths $\|\Pi'(r_i, p')\|$, $1 \le i \le s$, our values of *MaxCost* reflect the "approximate" cost of travel from each r_i, not the actual cost of travel. Hence, there is a measure of imprecision in our estimate of meeting point accuracy since the solution is based only on "approximate" maximum traveling costs. One way of reducing such imprecision is to try and fine-tune each path $\Pi'(r_i, p'), 1 \le i \le s$ so that it more closely resembles $\Pi(r_i, p')$.

Although this would not affect the solution (i.e., p' remains as computed), we can obtain a more precise value of *MaxCost*, which would help in assessing the solution accuracy as well as the final route of each robot to the destination.

Currently, there are no known algorithms for computing exact weighted shortest paths on terrains, so it is difficult to fine-tune our approximate paths. We can however, obtain a better estimate of each path cost $\|\Pi(r_i, p')\|$, $1 \leq i \leq s$ by applying the technique used by Lanthier et al. [2001]. In the Euclidean case, they unfold the sleeve containing $\pi'(r_i, p')$ into two dimensions and then compute the exact shortest paths from r_i to p' that remains within the unfolded sleeve of triangles using the algorithm of Lee and Preparata [1984]. Lanthier et al. [2001] showed that for typical terrain data, optimal paths are actually computed using this approach between 40 and 80% of the time. We apply the same unfolding technique to compute a path $\pi''(r_i, p')$ that represents the optimal shortest paths from r_i to p' that passes through the same edge sequence as $\pi'(r_i, p')$. As with Lanthier et al., it is likely that $|\pi''(r_i, p')| = |\pi(r_i, p')|$ for most paths.

Our experiments using this sleeve-refinement technique have shown that $\pi''(r_i, p')$ improves the traveling cost in comparison to $\pi'(r_i, p')$. Figure 9 shows the sleeve-refinement results of our three terrains with normal height for both point sets tested earlier. Notice that the convergence is quicker in all cases and, in three tests, we obtain a very good path accuracy through the use of only two or three Steiner points per edge.

With respect to the effect on runtime performance, we found that the cost to apply this additional sleeve-refinement, in practice, is insignificant. In fact, if the combined runtime of the algorithm along with the sleeve refinement was shown on the graphs of Figure 2, there would be no noticeable difference. In all cases, this additional step in the algorithm took less than 1 seconds to perform.

5. REDUCING THE RUNNING TIME

As stated earlier, our algorithm's runtime performance is largely based on the size of the approximation graph G, the number of Steiner points m used on each edge of $\mathcal{P}$, and the number of robots. We have verified Claim 3.10 through experimental analysis by showing that the number of Steiner points required per edge needs to be only a small constant in practice. Thus, the typical running time required for an accurate solution is $O(sn \log(sn))$. Although m is constant in practice, it still has a noticeable effect on the runtime performance.

One strategy in reducing the algorithm runtime is to reduce the number of robots $s = |R|$, hence reducing the linear effect that R has on the overall runtime complexity. Of course, by omitting some of the robots in R (i.e., using only a subset of R), we may obtain a different solution, which can be far off when compared to the solution obtained using R. Hence, there is a tradeoff between reduced running time and solution accuracy. The key is to reduce $|R|$ by selectively eliminating robots that may have little or no effect on the solution. In order to investigate the impact of the solution on accuracy when reducing $|R|$, we experimented with three strategies for replacing R with a subset of R and then compared the solution accuracy:

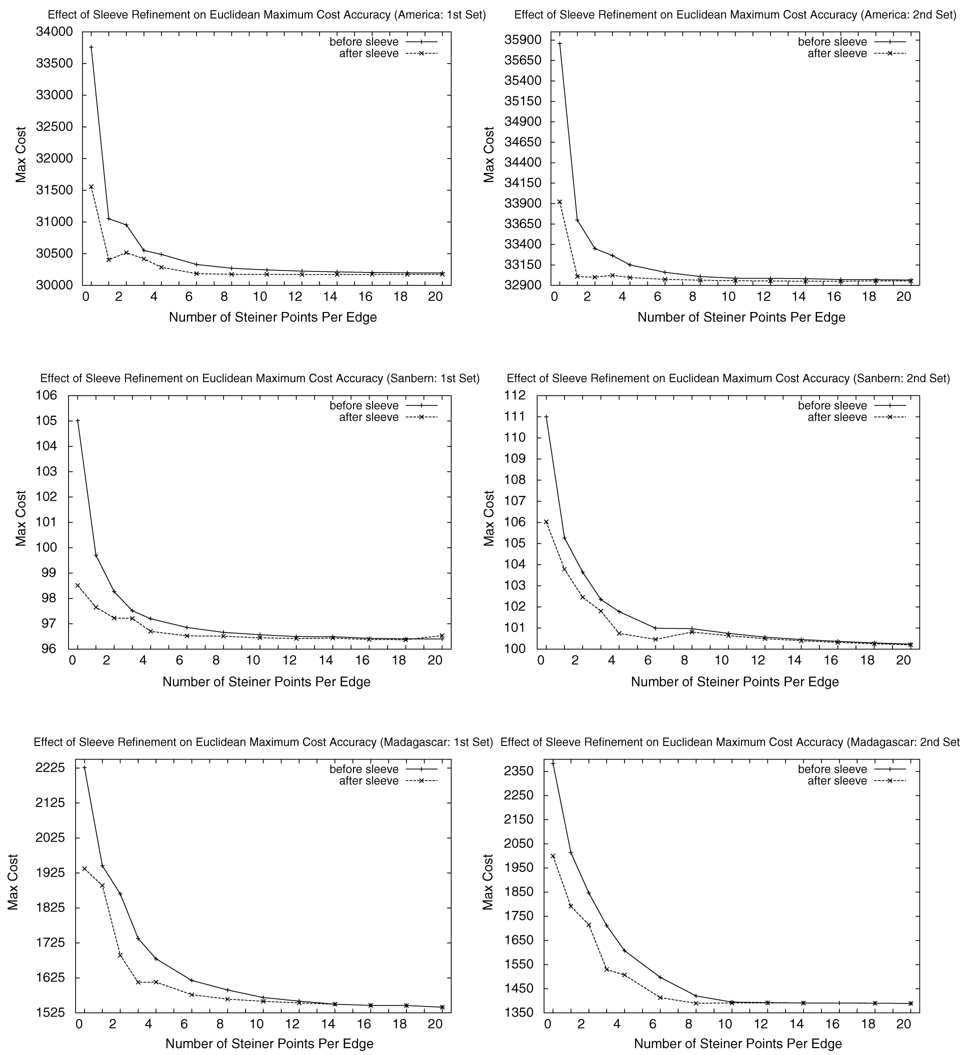

Fig. 9. Graphs showing how the sleeve refinement stage causes quicker convergence.

Random: a constant number of randomly chosen robots from R
Convex Hull: robots on the 2D/3D convex hull of R
Bounding Box: robots on the 2D/3D bounding box of R

In the subsections to follow, we explain the selection of each of these subsets in detail and present experimental results that show their effect on running time and solution accuracy.

5.1 Randomized Subsets

If all initial robot positions have equal effect on the solution, then the simplest strategy for reducing $|R|$ is to choose a random subset of R. In our experiments we first randomly chose 10 robots from an initial set of $|R| = 100$. To account for bad choices, we chose three random subsets of 10 and averaged the results. The robots were chosen as random selections of indexes from an array with the complete set of robots; we denote this averaged set as $R'_{RAND_{ind}}$. Since the indexes were used to select the robots randomly, the subset does not represent a random spatial selection. We, therefore, applied two additional random selection strategies. We computed a random x and a random y coordinate within the terrain range. We then selected a robot from R whose location was closest to (x, y), ignoring the height dimension. We did this ten times to produce a subset denoted as $R'_{RAND_{src}}$. Last, by again selecting a random coordinate (x, y) ten times, we chose the closest terrain vertex in place of the closest robot location and used these as the subset denoted as $R'_{RAND_{ver}}$. In our graphs, we show the average of the results (labeled as "Random Average") obtained from the three subsets $R'_{RAND_{ind}}$, $R'_{RAND_{src}}$, and $R'_{RAND_{ver}}$.

5.2 Convex Hull Subsets

The second strategy for selecting a subset of R, which is shown to be superior to the other two strategies, was to select robots that, with high likelihood, are furthest away from the required meeting point. This is because these robots should, with high probability, have the greatest effect on the solution. When using the Euclidean metric, on a completely flat terrain, the furthest robots would lie on the 2D convex hull of the point set formed by R. Since most traversable terrain is relatively flat, it is likely that by choosing robots on the convex hull of R, when R is projected on the XY plane, we would obtain a "good" set of robots that will have the greatest effect on the meeting point solution. As a terrain becomes more and more uneven (i.e, very spiky), there is a chance that some of the non-2D convex hull points will have a greater effect on the solution. However, in this case, we choose points along the 3D convex hull of R. In the weighted metric however, some of the more "inner" robots may require a long time to traverse costly terrain to reach the meeting point. In such cases, the 2D/3D hulls will not contain these more significant source points (robots). Nevertheless, in typical data, we believe that the effects of the nonconvex hull points on the solution will be minimal.

To compute the 2D convex hull subset R'_{CH2D}, we ignore the z coordinate from the points in R. Hence, we can use any of the well-known 2D convex hull algorithms for a set of points in the plane. One example is the $O(s \log s)$ time *Graham's scan*, as introduced in [Graham 1972]. We do this as a preprocessing phase of our algorithm before the graph G is constructed. On a flat terrain when using the Euclidean metric, it can be easily shown that p^* is identical whether R or R'_{CH2D} is used. We conjecture, since most terrains are close to being flat, that this will be true for typical terrain data.

As the terrain becomes more uneven and spiky, the solution using R'_{CH2D} can become more and more inaccurate. A better solution would be to use the points on the 3D convex hull so that robot positions near mountain peaks, for example, will also have their intended effect on the solution. We also compute the 3D convex hull of R, denoted as R'_{CH3D} and show experimentally that for uneven terrains, the solution obtained improves upon the 2D hull subset solution.

5.3 Bounding-Box Subsets

Even though the use of R'_{CH3D} reduces the runtime cost, it may still be that $|R'_{CH3D}| = O(s)$. That is, many robots may lie on the convex hull. In this worst case scenario, there would be no improvement in runtime. Perhaps it would always be better to select a constant number of robots so that s has no significant runtime effect on the solution. Our randomized schemes represent one attempt at ensuring that s was kept constant. We did, however, try one more strategy by selecting subsets of robots that lie on the 2D and 3D bounding boxes of R, which we denote as R'_{BB2D} and R'_{BB3D}, respectively. These bounding boxes are represented by robots with extreme x, y, and z coordinates and so we are certain that $|R'_{BB2D}| \leq 4$ and $|R'_{BB3D}| \leq 6$. Although this strategy may ignore some robots that significantly effect the solution, it does provide the tradeoff of keeping s small and, moreover, these subsets are very fast and easy to compute.

5.4 Experimental Results

Using the same terrains and point sets as in Section 4, we investigated the tradeoff between runtime performance and solution accuracy when using the subsets just described. Given a subset R' of R, we compute the meeting point solution p'' as

$$\min_{p'' \in V^{\mathcal{P}}} \left\{ \max_{r_i \in R'} \{ \| \Pi'(r_i, p'') \| \} \right\}$$

One way of assessing the accuracy of the subset solution is to compute the following relative error:

$$MaxCostError = \frac{\max_{r_i \in R} \{ \| \Pi'(r_i, p') \| \} - \{ \max_{r_i \in R'} \{ \| \Pi'(r_i, p'') \| \}}{\max_{r_i \in R} \{ \| \Pi'(r_i, p') \| \}} * 100\%$$

The graphs in Figure 10 depict the results of using each of our subset schemes, showing the maximum cost error for each terrain and point set. To obtain these results, we first obtained p'' by running our graph search using R'. We then reran the test using the full set R, but changed the stopping condition of our algorithm to be that in which all robots $r_i \in R$ reached p''. This is analogous to the one-to-all shortest paths problem. The graphs show the difference in maximum cost error, calculated as a percentage.

As expected, the largest error is achieved when using random subsets, since some of the more important "outer" robots are not necessarily selected when forming the subset. Notice that for the first two terrains, the R'_{CH2D} and R'_{CH3D} solutions produced the exact same solution (i.e., 0% error) as those in which R was used. The advantage of using R'_{CH3D} over R'_{CH2D} is visible only in the graphs for the spiky Madagascar terrain. Here we can see that the R'_{CH2D} solution can result in up to 10.5% error while the R'_{CH3D} solution remains at 0% error. A similar behavior is observed when comparing R'_{BB2D} and R'_{BB3D} in which R'_{BB3D} almost always outperforms R'_{BB2D}. In fact, for the spiky Madagascar terrain, the R'_{BB3D} solution performs as well as the R'_{CH3D} solution when over ten Steiner points per edge are used.

To get a better understanding of the relative accuracy for the solutions, the graph to the left of Figure 11 depicts the average of these results for all terrains

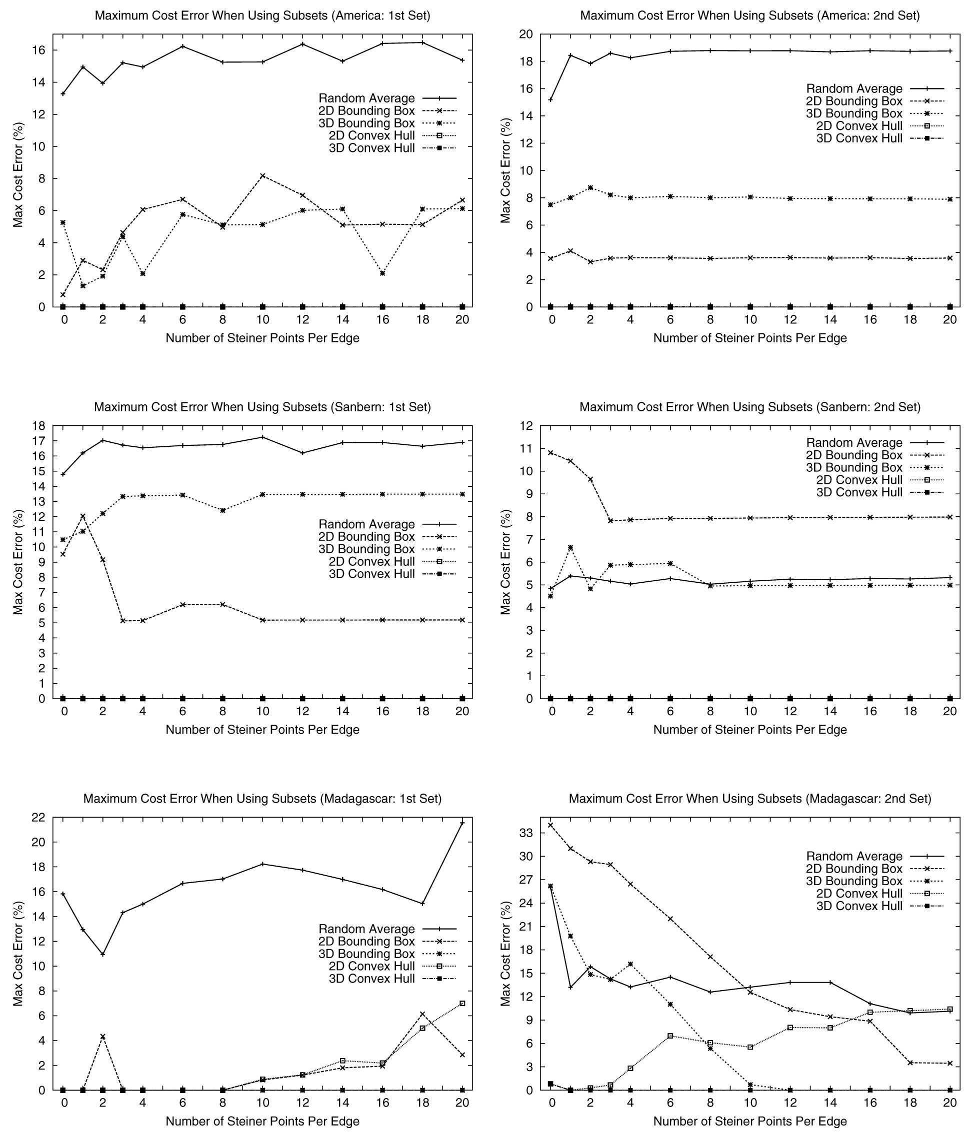

Fig. 10. Graphs showing the maximum cost error associated with the various subset selections for each terrain and point set.

and point sets tested when using the Euclidean metric. Notice that the best subset scheme is R'_{CH3D} with a 0% error followed by R'_{CH2D}, which provides less than one-half the error of the bounding box schemes. The random subset schemes were the worst with up to 16% error. The graph on the right side of the figure represents the average of the results when using the weighted metric. Note that the relative error of the subset schemes is similar, except that the R'_{CH2D} solution performs very poorly. This is because of the fact that some of the robot locations of the original set are near mountainous peaks, which have a high cost

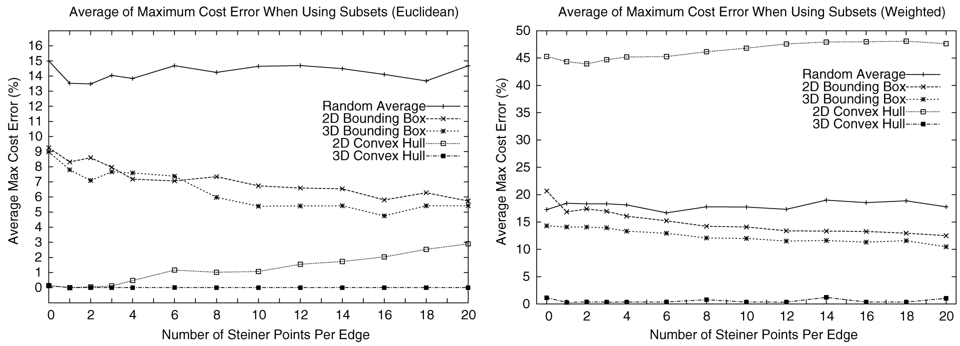

Fig. 11. Graphs showing the average maximum cost error across all three terrains using both point sets for Euclidean and weighted metrics.

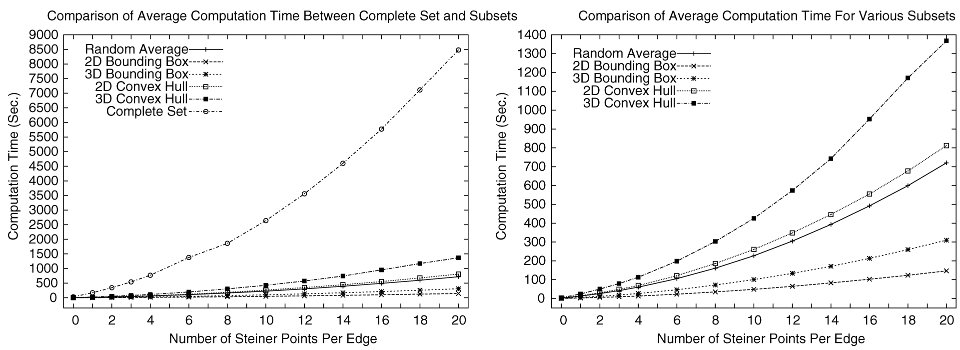

Fig. 12. Graphs showing the average Euclidean running time across all three terrains using both point sets.

of travel as a result of their steep slopes, causing the meeting point to be higher up the mountains, while R'_{CH2D} typically does not include these robots, causing the meeting point to be further down the mountain. This effect is also noticeable by the fact that the R'_{BB3D} solution provides the second best overall solution.

From the observations just made, we see that the R'_{CH3D} solution performs best by matching the optimal solution in the Euclidean setting and by producing a less than 2% error when using the weighted metric. Let us examine now the effect on runtime performance. The graphs in Figure 12 show the runtime costs for each of these subset schemes; the left graph includes the complete set R and the right graph shows a "zoomed-in" version for comparing the subset schemes. Notice the significant decrease in computation time for all subset schemes when compared to using the complete set R. The slowest of the subset schemes still takes only 16.1% of the runtime required for the complete set and the quickest takes only 1.7% of the time. When comparing the subset schemes with each other, the right graph shows that the convex hull schemes take longer to compute, while the bounding-box schemes are quickest. The main factor here is in the size of the subsets. In our tests, the size of the subsets of R associated with the bounding box schemes was always four in the case of 2D or six in the case of 3D. The size of the subset associated with the random scheme

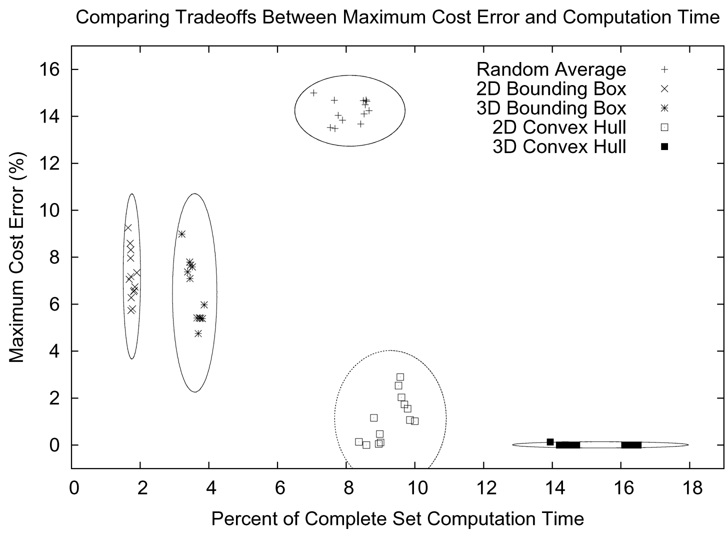

Fig. 13. Graph showing the tradeoffs between maximum cost error and percentage of computation time for each subset.

was of size 10 and the typical size of the subsets associated with convex hull schemes was 10 or 11 for the 2D hull and 24 for the 3D hull.

It is clear that the 3D convex hull subset scheme provides the best accuracy for a very reasonable runtime performance. Figure 13 shows a graph that compares the tradeoffs between the various subset schemes in terms of accuracy and runtime. Each subset is represented by a set of 13 data points representing the solution variation for the number of Steiner points used per edge. Schemes closest to the bottom produced more accurate solutions while those closer to the left have a faster computation time in terms of the percentage of computation time required for the complete set R.

6. CONCLUSIONS AND FUTURE WORK

In this work, we presented the first practical algorithm for solving the meeting point (1-center) for a set of robots placed in a weighted terrain. Given a set of scattered robots $R = \{r_1, r_2, \ldots, r_s\}$ in a weighted terrain $\mathcal{P}$ with n faces, our algorithm finds an approximation point p' to the optimal meeting point p^* of these robots such that:

$$\max_{1\le i\le s}\{\|\Pi(r_i, p')\|\} \le \max_{1\le i\le s}\{\|\Pi(r_i, p^*)\|\} + 2W|L|$$

Moreover, we have shown that such an approximation can be obtained in $O(snm\log(snm)+snm^2)$ time, where $m = n$ in the Euclidean metric and $m = n^2$ in the weighted metric. Our experiments have shown, however, that a constant value for m is sufficient (e.g., $m = 8$–12) in order to produce very accurate solutions. Through experimentation, we also show that the running time may also be significantly reduced by selecting a subset of R. This work compared and contrasted the effect of the various ways for selecting subsets of R with respect to runtime performance and solution accuracy. When the terrain was relatively flat, it was shown that some subsets such as R'_{CH2D} and R'_{CH3D} of R, corresponding to the 2D and 3D convex hulls, respectively, produced very good results while reducing the running time by 98.3%.[3] For the more spiky terrains the runtime was still decreased by 83.9%, although with a more significant degradation of solution accuracy when using R'_{CH2D}, while R'_{CH3D} maintained its high accuracy.

We have taken the approach of Lanthier et al. [2001] to bound the approximate cost and running times. It should be noted, however, that there are other methods of applying Steiner points to $\mathcal{P}$ that achieve different bounds and accuracies. For instance, the Steiner placement strategies presented by Aleksandrov et al. [2003; 1998] can be applied to produce G and obtain an ϵ-approximation for p' that can be made to be arbitrarily close to p^*, at a cost of increased running time. Although this ϵ-approximation work may provide a better theoretical bound, the simpler and more practical algorithm for placing Steiner points evenly on the edges of $\mathcal{P}$ was presented in this article. The work of Lanthier et al. [2001] also made use of graph spanners to reduce the graph G, and, hence, also reduce the running time. Likewise, our algorithm can easily be extended to use graph spanners to achieve a reduced running time.

It is possible that other metrics may also be applied such as the anisotropic path model presented by Lanthier et al. [1999], which takes into account the direction of travel across terrain faces. Of course, in such a model, it is possible that no meeting point solution is obtainable. Also, the location of Steiner points on edges of $\mathcal{P}$ becomes critical, so the construction of G would change accordingly. We are currently working on an extension to this work, involving a secondary heap, that determines a meeting point such that the total accumulated cost of all robots is minimized (e.g., combined fuel/battery usage is kept as small as possible). There is also the possibility for future extensions of this algorithm in the parallel setting. Lanthier et al. [2003] have shown that the graph G can be efficiently partitioned across multiple processors to obtain speedups of up to 60% for shortest paths computations. We are confident that similar speedups can be obtained for our algorithm.

REFERENCES

Aleksandrov, L., Lanthier, M., Maheshwari, A., and Sack, J.-R. 1998. An epsilon-approximation algorithm for weighted shortest paths s on polyhedral surfaces. In *Proceedings SWAT '98 (LNCS 1432)*. Stockholm, Sweden. 11–22.

Aleksandrov, L., Maheshwari, A., and Sack, J.-R. 2003. An improved approximation algorithm for computing geometric shortest paths. In *Proceedings 14th Annual Symposium on the Fundamentals of Computation Theory*. Malmo, Sweden. 246–257.

Aronov, B., van Kreveld, M., van Oostrum, R., and Varadarajan, K. 1998. Facility location on terrains. In *Proceedings of the 9th International Symposium of Algorithms and Computation (LNCS 1533)*. Taejon, Korea. 19–28.

Dijkstra, R. 1959. A note on two problems in connection with graphs. *Numerical Math. 1*, 269–271.

Drezner, Z. 1984. The *p*-center problem: Heuristic and optimal algorithms. *J. Operational Res. Soc. 35*, 741–748.

[3] Percentage corresponds to one of the test results that are incorporated in the average run time shown in Figure 12.

DRISCOLL, J., GABOW, H., SHRAIRMAN, R., AND TARJAN, R. 1988. Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation. *Commun. ACM 31,* 11, 1343–1354.

FREDMAN, M. AND TARJAN, R. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM 34,* 3, 596–615.

GRAHAM, R. L. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett. 1*, 132–133.

GUIBAS, L. AND STOLFI, J. 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graphics 4,* 2, 75–123.

KARIV, O. AND HAKIMI, S. 1979. An algorithmic approach to network location problems I: The p-centers. *Siam J. Appl. Math. 37,* 3, 513–538.

LANTHIER, M., MAHESHWARI, A., AND SACK, J.-R. 1999. Shortest anisotropic paths on terrains. In *Proceedings ICALP 99 (LNCS 1644)*. Prague, Czech Republic. 524–533.

LANTHIER, M., MAHESHWARI, A., AND SACK, J.-R. 2001. Approximating weighted shortest paths on polyhedral surfaces. *Algorithmica 30,* 4, 527–562.

LANTHIER, M., NUSSBAUM, D., AND SACK, J.-R. 2003. Parallel implementation of geometric shortest path algorithms. *Parallel Comput. 29*, 1445–1479.

LEE, D. AND PREPARATA, F. 1984. Euclidean shortest paths in the presence of rectilinear barriers. *Networks 14*, 393–410.

MEGIDDO, N. 1983a. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM 30*, 852–865.

MEGIDDO, N. 1983b. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comput. 12*, 759–776.

MITCHELL, J. 1997. In *Handbook of Discrete and Computational Geometry*, J. Goodman and J. O'Rourke, Eds. CRC Press LLC, Boca Raton, FL. 445–466.

MITCHELL, J. 2000. In *Handbook on Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science, Amsterdam. 633–701.

MITCHELL, J. AND PAPADIMITRIOU, C. 1991. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *J. ACM 38*, 18–73.

NUSSBAUM, D. 1997. Rectilinear p-piercing problems. In *ISSAC'97 International Symposium on Symbolic and Algebraic Computation*. Maui, Hawaii.

SHARIR, M. 1996. A near-linear algorithm for the planar 2-center problem. In *Proceedings 12th ACM Symposium on Computational Geometry*. Philadelphia, Pennsylvania. 110–121.

SHARIR, M. AND SCHORR, A. 1986. On shortest paths in polyhedral spaces. *SIAM J. Comput. 15*, 193–215.

SHARIR, M. AND WELZL, E. 1996. Rectilinear and polygonal p-piercing and p-center problems. In *Proceedings of the 12th ACM Symposium on Computational Geometry*. Philadelphia, Pennsylvania. 122–132.

SUN, Z. AND REIF, J. 2003. Adaptive and compact discretization for weighted region optimal path finding. In *Proceedings of the 14th International Symposium on Fundamentals of Computation Theory (FCT2003) (LNCS 2751)*. Malmö, Sweden. 258–270.

VAN OOSTRUM, R. 1999. Geometric algorithms for geographic information systems. Ph.D. thesis, Department of Computer Science, Utrecht University.

VAN TRIGT, M. 1995. Proximity problems on polyhedral terrains. M.S. thesis, Department of Computer Science, Utrecht University.

WELZL, E. 1991. *Lecture Notes in Computer Science: New Results and New Trends in Computer Science*, H. Maurer, Eds. Vol. 555. Springer-Verlag, New York. 359–370.

Received November 2004; revised January 2008; accepted April 2008