BUSHWHACK: An Approximation Algorithm for Minimal Paths Through Pseudo-Euclidean Spaces*

Zheng Sun and John Reif

Department of Computer Science, Duke University, Durham, NC 27708, USA
{sunz, reif}@cs.duke.edu

Abstract. In this paper we define *piecewise pseudo-Euclidean optimal path problems*, where each region has a distinct cost metric of a class we call pseudo-Euclidean, that allows the path cost to possibly vary within the region in a predictable and efficiently computable way. This pseudo-Euclidean class of costs allows us to model a wide variety of various geographical features. We provide an approximation algorithm named *BUSHWHACK* that efficiently solves these piecewise pseudo-Euclidean optimal path problems. BUSHWHACK uses a previously known technique of dynamically generating a discretization in progress. However, it combines with this technique a "lazy" and best-first path propagation scheme so that fewer edges need to be added into the discretization. We show both analytically and experimentally that BUSHWHACK is more efficient than approximation algorithms based on Dijkstra's algorithm.

1 Introduction

In many applications, such as robotic motion planning and geographical information systems, there arise *optimal path problems* where each problem is to find a minimal cost path in the plane. One common assumption made by many previous studies on these problems is that, for any s and t, an optimal path from s to t is the straight line segment $\overline{st}$ if $\overline{st}$ lies entirely inside the free space.

In recent years there has been an increasing attention and motivation on path planning problems with various non-Euclidean metrics. If the free space consists of multiple regions and the metric is not the same for all regions, the straight line segment $\overline{st}$ may no longer be an optimal path from s to t, even if $\overline{st}$ lies in the free space. Therefore, many techniques developed in previous motion planning works are no longer valid.

In the weighted region optimal path problem ([5, 4, 3, 1, 6, 2]), the entire free space is divided into polygonal regions each of which is associated with a unit weight. The cost of a path p is defined to be the weighted sum of the lengths of the segments of p inside each region. Another example is the flow problem ([7]),

* Supported by NSF ITR EIA-0086015, NSF-IRI-9619647, NSF CCR-9725021, SEGR Award NSF-11S-01-94604, Office of Naval Research Contract N00014-99-1-0406.

where inside each region there is a flow defined by a vector, and the cost of path p is the total travel time on p by robot with a fixed maximum velocity.

To solve the weighted region optimal path problem, a number of previous works ([3, 1, 2]) used a discretization of the problem based on edge subdivision, and Dijkstra's algorithm to find an optimal path in the graph induced by discretization. Aleksandrov et al [1] proposed a logarithmic discretization that guarantees an ϵ-short approximation, where $m = O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ Steiner points are placed on each boundary edge. Their algorithm then applies a "pruned" version of Dijkstra's algorithm to find an optimal path in the discrete graph in $O(\frac{n}{\epsilon}(\frac{1}{\sqrt{\epsilon}} + \log n)\log\frac{1}{\epsilon})$ time, where n is the number of all boundary edges.

The algorithm BUSHWHACK presented here uses a subgraph to render an optimal path in the discrete graph by dynamically adding edges. This technique has been used by a number of prior precedents, but our BUSHWHACK algorithm, by adopting a "lazy" and best-first path propagation scheme, is able to use fewer edges to compute an optimal path in the discrete graph.

Some of the key features of the BUSHWHACK algorithm were used in an approximation algorithm of Reif and Sun [6] introduced to approximately solve the weighted region optimal path problem; that algorithm finds an optimal path in the discrete graph in time $O(nm\log nm)$. The generalized algorithm BUSHWHACK can be used to compute approximate optimal paths in $2\mathcal{D}$ spaces that satisfy a wide range of possible geometric properties. In Section 2 we define a class of spaces that we call *Piecewise Pseudo-Euclidean Space*. We show that the BUSHWHACK algorithm can be applied to an optimal path problem in any space in this class. An immediate application is path planning in an area with various geographical features such as plains (regions with low unit costs), swamps (regions with high unit costs), and rivers and tides (regions with flows).

The focus of this paper is on the generality of the BUSHWHACK algorithm and the characterizations of the spaces to which BUSHWHACK can be applied. However, we feel that it is important to make a brief digression here on its efficiency. For the weighted region optimal path problem, by applying BUSHWHACK to the logarithmic discretization scheme proposed by [2], we can have an algorithm that computes an ϵ-short approximate optimal path in $O(nm\log nm) = O(\frac{n}{\epsilon}(\log\frac{1}{\epsilon} + \log n)\log\frac{1}{\epsilon})$ time. This improves on all other approximation algorithms, including those mentioned above.

2 Preliminaries

A convex polygonal region r is said to be a *pseudo-Euclidean region* if it satisfies the following two properties.

Property 1 *Region r is associated with a cost function $d_r : (\mathcal{R}^2, \mathcal{R}^2) \to \mathcal{R}^+ \bigcup \{0\}$ so that, for any two points x and y in r, the cost of the straight line path $\overline{xy}$ is $d_r(x, y)$. $d_r(x, y) = 0$ if and only if $x = y$. The cost function d_r has the property that the path with the least cost, among all paths from x to y that lie completely inside r, is the straight line segment $\overline{xy}$.*

Property 2 *Letting x_0 be a point in region r (including the boundary) and letting $e = \overline{v_0 v_1}$ be an edge of r that is not incident to x_0, then there are only a small number of local extrema for function $g_{x_0,e} : [0,1] \rightarrow \mathcal{R}^+$, where $g_{x_0,e}(\lambda) \equiv d_r(x_0, (1-\lambda)v_0 + \lambda v_1)$. These local extrema can be computed efficiently.*

In the following discussion, we will refer to $d_r(x,y)$ as the *region distance*, or *region cost*, from x to y. A space is said to be a *piecewise pseudo-Euclidean space* if it consists of a finite number of pseudo-Euclidean regions. For an arbitrary path p in the space, if p can be divided into segments $p_1, p_2, \cdots, p_m$ so that each $p_i, 1 \leq i \leq m$, lies entirely inside a region r, the cost of p is defined to be the sum of the costs of all segments, each of which is determined by the respective region cost function. In case a segment $\overline{xy}$ lies on a boundary edge e, we define its cost $d_e(x,y)$ to be $\min\{d_r(x,y), d_{r'}(x,y)\}$, where r and r' are the neighboring regions of e. A *piecewise pseudo-Euclidean optimal path problem* is to find an optimal path (i.e., the path with the least cost) from a source point s to a destination point t in a piecewise pseudo-Euclidean space. Here s and t are both vertices of some pseudo-Euclidean regions.

According to Property 1, an optimal path in a piecewise pseudo-Euclidean space is piecewise linear. A segment of a path is said to be "edge-crawling" if it lies on an edge; or "face-crossing" if it cuts through a region. We call a path a "face-crossing" ("edge-crawling") path if the last segment of the path is "face-crossing" ("edge-crawling", respectively). Although a piecewise pseudo-Euclidean space may consist of hybrid regions whose cost functions can be completely different, inside each region it is a Euclidean-like space because the shortest path between two points inside the region is still the straight line segment.

At a later point we will explain the importance of Property 2 to our algorithm; we will also show how "efficient" the computation of local extrema needs to be. Although this property may seem to be restrictive, in practice many optimal path planning problems satisfy this property. Examples are the weighted region problem and flow problem mentioned in the previous section.

Here we first introduce some notations that will be used in the rest of the paper. We let S be a polygonal decomposition of the planar space and let V be the set of vertices in S. We use E to denote the set of all boundary edges in S and let $n = |E|$. Without loss of generality, we assume that each region is a triangle. For any path p, we let $d(p)$ denote the cost of p. For two paths p_1 and p_2, we let $p_1 + p_2$ denote the concatenation of p_1 and p_2. If $p = p_1 + p_2$, we say p is an *extension* of p_1. In particular, if $p = p_1 + \overline{v_1 v_2}$, we say p is a *one-segment extension* of p_1. For any two points x and y, we use $p(x,y)$ to denote a path from x to y and use $p_{opt}(x,y)$ to denote an optimal path from x to y. We define the "distance" from s to t, $d_{opt}(s,t)$, to be the cost of $p_{opt}(s,t)$. At any time during the search of an optimal path from s to t, a point v is said to be *discovered* if and only if $d_{opt}(s,v)$ is determined.

A natural approach to these problems is to discretize the $2\mathcal{D}$ space by introducing Steiner points. For each boundary edge $e \in E$, we add m Steiner points on e for some positive integer m. Let V_s be the set of Steiner points and let $V' = V \bigcup V_s$. A directed discrete graph $G(V', E')$ is constructed by intercon-

necting points in V' that are on the boundary of the same region. Each edge (x, y) in G is assigned a weight $w(x, y)$ where $w(x, y)$ is defined to be $d_e(x, y)$ if $\overline{xy}$ is on edge e; or $d_r(x, y)$ if $\overline{xy}$ crosses region r of S.

By constructing G, the original path planning problem in a continuous space is transformed to the problem of finding a minimum path in the discrete graph. The latter problem can be solved by Dijkstra's algorithm. A path in G is called a *discrete path*. An optimal discrete path found from s to t is then used to approximate an optimal path in the original continuous space. The more Steiner points we place on each edge of S, the more accurate the approximation will be. Aleksandrov et al [2] showed that there exists a discretization, with $m = O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ Steiner points inserted on each boundary edge, that guarantees an ϵ-short approximation of an optimal path from s to t.

It takes $O(|E'| + |V'| \log |V'|) = O(nm^2 + nm \log nm)$ time to find an optimal discrete path in G using Dijkstra's algorithm. Observe that, when m is large, the dominant part of the time complexity is $O(nm^2) = O(|E'|)$. To reduce the cost of processing edges of G, Aleksandrov et al [2] proposed a "pruned" version of Dijkstra's algorithm. By exploiting the fact that the in-angle and out-angle of an optimal path at a bending point obey "Snell's Law," their algorithm only uses a sparse subgraph $G'(V', E'')$ of G which still yields an optimal discrete path from s to t in G. The number of edges included in subgraph G', $|E''|$, is $O(n/\epsilon^{\frac{3}{2}} \log \frac{1}{\epsilon}) = O(\sqrt{\epsilon}/\log \frac{1}{\epsilon} \cdot |E'|)$. The total time complexity of the algorithm is therefore reduced from $O(\frac{n}{\epsilon}(\frac{1}{\epsilon} \log \frac{1}{\epsilon} + \log n) \log \frac{1}{\epsilon})$ to $O(\frac{n}{\epsilon}(\frac{1}{\sqrt{\epsilon}} + \log n) \log \frac{1}{\epsilon})$.

Our BUSHWHACK algorithm follows the same discretization approach. However, by maintaining a collection of data structures called *intervals*, on average BUSHWHACK only needs to evaluate for each point v the costs of $O(\log m)$ adjacent edges of v, as we will show at the end of Section 4. The total number of edges accessed by the algorithm is thus $O(nm \log m) = O(\epsilon \cdot |E'|)$. Our BUSHWHACK algorithm can therefore find an optimal discrete path efficiently in $O(nm \log nm) = O(\frac{n}{\epsilon}(\log \frac{1}{\epsilon} + \log n) \log \frac{1}{\epsilon})$ time. More importantly, compared to the "pruned" Dijkstra's algorithms [1, 2], BUSHWHACK makes weaker assumptions on the metric inside each region and thus can also be applied to other piecewise pseudo-Euclidean optimal path problems.

As the goal of our algorithm is to find the exact optimal discrete path, in the following discussion, wherever we refer to an "optimal path," we mean an optimal discrete path in G unless specified otherwise. We let $p'_{opt}(s, t)$ denote an optimal discrete path from s to t and let $d'_{opt}(s, t)$ be the cost of $p'_{opt}(s, t)$.

3 Intervals

The BUSHWHACK algorithm works similarly to Dijkstra's algorithm. It keeps a sorted list QLIST of candidate optimal paths. At each step, the candidate optimal path p_{min} with the minimum cost is extracted from QLIST. Consequently, a number of candidate optimal paths are inserted into QLIST that are one-segment extensions of p_{min}. (We call this process "propagation.") The iteration continues until the destination point is reached.

Dijkstra's algorithm can be used to compute an optimal path in an arbitrary weighted graph, while the aforementioned discrete graph G is derived from a piecewise pseudo-Euclidean space with certain geometric properties. Therefore, directly applying Dijkstra's algorithm to G does not fully utilize the underlying geometric properties. In particular, Property 1 implies the following lemma:

Lemma 1 *Any two optimal paths in G with the same source point cannot intersect in the interior of any region.*

For example, let p_1 and p_2 be two paths originated from the same source point s that intersect at point a inside region r. Dijkstra's algorithm will consider both paths as potential optimal paths for their respective destination points. However, in a piecewise pseudo-Euclidean space, it is not possible that two optimal paths intersect inside a region, as stated by Lemma 1. Suppose $\overline{v_{i,1}v_{i,2}}$ is the segment of p_i that contains a. The fact that line segments $\overline{v_{1,1}v_{1,2}}$ and $\overline{v_{2,1}v_{2,2}}$ intersect inside region r indicates that these two line segments can not both contribute to optimal paths originated from s. If we can identify "useful" line segments (i.e., those that may contribute to optimal paths) from "useless" line segments, we will be able to avoid an explicit construction of the entire discrete graph G.

To keep track of useful line segments, we introduce a data structure named *interval*. Let r be a region of S and let e, e' be two boundary edges of r. Let v be any discovered point on e that is not incident to e'. Interval $I_{v,e,e'}$ is defined to be $I_{v,e,e'} = \left\{v^* \in e' \mid d_r(v, v^*) + d'_{opt}(s, v) \leq d_r(v', v^*) + d'_{opt}(s, v') \forall\ v' \in \text{PLIST}_e\right\}$, where PLIST_e is the list that includes all discovered points on e. That is, for any point v^* on e', $v^* \in I_{v,e,e'}$ if and only if path $p'_{opt}(s, v) + \overline{vv^*}$ is the least costly path among all paths from s to v^* that are one-segment extensions of optimal paths from s to discovered points on e. For any edge e and e' that share a region r, we use $\text{ILIST}_{e,e'}$ to denote the list of intervals $I_{v,e,e'}$ for all $v \in \text{PLIST}_e$.

From this definition of intervals we can conclude that, for any discovered point v on e and any point v^* on e', $\overline{vv^*}$ can not be part of any optimal path originated from s that enters region r through point v if $v^* \notin I_{v,e,e'}$. Therefore, by maintaining $\text{ILIST}_{e,e'}$ for each e and e' that share a region, the BUSHWHACK algorithm is able to avoid accessing most of the edges in G. Observe that each interval $I_{v,e,e'}$ is a dynamic set of points on e'. It is first created when v is discovered. When more points on e are discovered, PLIST_e will contain more points and thus $I_{v,e,e'}$ may also be changed, according to the definition.

Lemma 1 implies that each interval is composed of consecutive points on e' (which leads us to name this data structure "interval"). Further, an interval $I_{v,e,e'}$ is located to the left (right) of another interval $I_{v',e,e'}$ on e' if and only if v is located to the left (right, respectively) of v' on e. Figure 1.a shows how points on edge e' are partitioned into intervals corresponding to discovered points on e. We claim that the two end points of an interval $I_{v,e,e'}$ can be computed efficiently (in $O(\log m)$ time) when it is initially created.

For each $v^* \in I_{v,e,e'}$, the face-crossing path $p'_{opt}(s, v) + \overline{vv^*}$ needs to be considered as a candidate optimal path from s to v^*. We call such a path a *direct interval path* associated with $I_{v,e,e'}$. One strategy is to insert all these paths into QLIST simultaneously when v is discovered. However, this may not be efficient as

both $\text{ILIST}_{e,e'}$ and intervals in $\text{ILIST}_{e,e'}$ are dynamic data structures. Whenever a new point on e is discovered, a new interval (although possibly empty) will be created and inserted into $\text{ILIST}_{e,e'}$. If the new interval is not empty, the ranges of the two neighboring intervals will be adjusted.

As shown in 1.b, even though a point v^* originally is in $I_{v_2,e,e'}$, after a new point $v_{new} \in e$ is discovered, v^* may fall into the range of the new interval $I_{v_{new},e,e'}$. If this is the case, path $p'_{opt}(s,v_2) + \overline{v_2v^*}$ no longer needs to be considered as an optimal path from s to v^* as it is more costly than $p'_{opt}(s,v_{new}) + \overline{v_{new}v^*}$, according to the definition of "interval."

A more efficient strategy is to insert direct interval paths in a "lazy" and best-first manner. Interval paths associated with $\text{ILIST}_{e,e'}$ are sorted in the increasing order of path cost. A path $p'_{opt}(s,v) + \overline{vv^*}$ is inserted into QLIST only when the previous path is extracted from the list, and only if v^* is still in $I_{v,e,e'}$. This strategy will avoid inserting a path $p'_{opt}(s,v) + \overline{vv^*}$ into QLIST if v^* is later "switched" to another interval.

To achieve this, we need to sort efficiently the direct interval paths by path cost. Since these paths are all one-segment extensions of $p'_{opt}(s,v)$, we only need to sort $d_r(v,v^*)$ for all $v^* \in I_{v,e,e'}$. According to Property 2, the region distance function from v to points on e' has a constant number of local extrema. Thus, $I_{v,e,e'}$ can be divided into a constant number of parts by these local extrema so that the region distance from v to points in each part is monotonically increasing or decreasing. We create a *monotonic interval* for each monotonic part of $I_{v,e,e'}$ and replace $I_{v,e,e'}$ by these intervals in $\text{ILIST}_{e,e'}$. Points in each such interval are already sorted by region distance to v. In the following discussion, we always assume that each interval is monotonic. For the weighted region optimal path problem, each region is a Euclidean space and thus each interval $I_{v,e,e'}$ will at most be split into two monotonic intervals by the perpendicular point of v on e', as illustrated in Figure 2.a. The same is true for a region with a uniform flow, although computing the split point is more complicated as shown in [7].

Suppose interval $I_{v,e,e'}$ contains points $v_1^*, v_2^*, \cdots, v_d^*$ when it is initially created (i.e., when v is discovered), as shown in Figure 2.b. Here $v_1^*, v_2^*, \cdots, v_d^*$ are consecutive points on e' and v_1^* and v_d^* are the two end points of the interval. W.L.O.G, we assume $d_r(v,v_1^*) \leq d_r(v,v_d^*)$. As interval $I_{v,e,e'}$ is monotonic, we have $d_r(v,v_1^*) \leq d_r(v,v_2^*) \leq \cdots \leq d_r(v,v_d^*)$, where r is the region incident to both e and e'. Let $p_1, p_2, \cdots, p_d$ be direct interval paths associated with $I_{v,e,e'}$, where $p_i = p'_{opt}(s,v) + \overline{vv_i^*}$ for $1 \leq i \leq d$. For each v_i^*, let $P_i = \{p_{j,i} = p_j + \overline{v_j^*v_i^*} \mid 1 \leq j \leq d\}$. Observe that $p_i \in P_i$ as $p_i = p_i + \overline{v_i^*v_i^*}$. All paths in P_i except p_i are *extended interval paths* that are one-segment extensions of direct interval paths associated with $I_{v,e,e'}$. We call both direct interval paths and extended interval paths *interval paths*. P_i is the set of all interval paths associated with $I_{v,e,e'}$ that connect s and v_i^*.

We say interval path $p' \in P_i$ is *locally optimal* if $d(p') = \min\{d(p) \mid p \in P_i\}$. For each v_i^*, we want to insert into QLIST only one locally optimal interval path p_i^* that connects s and v_i^*. Initially, interval path $p_1^* = p_1$ is inserted into QLIST. Iteratively, an interval path p_i^* from s to v_i^* is added into QLIST when

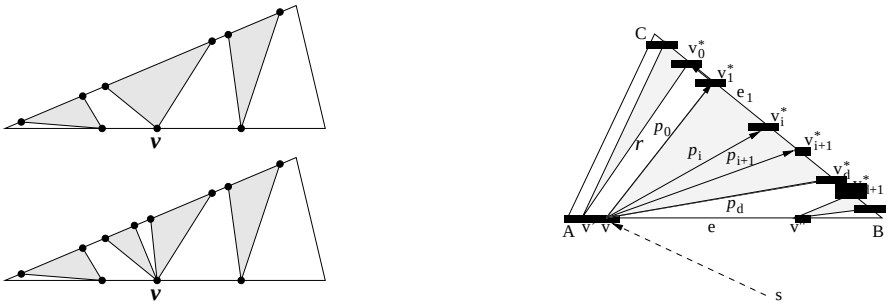
(a) Intervals

(b) Inserting an Interval

Fig. 1. Intervals and Inserting an Interval

the interval path p^*_{i-1} from s to v^*_{i-1} is extracted from QLIST. p^*_i is defined to be the less costly path between two paths, p_i and $p^*_{i-1} + \overline{v^*_{i-1}v^*_i}$. That is, the interval path for v^*_i can be constructed by either extending $p'_{opt}(s,v)$ by line segment $\overline{vv^*_i}$, or extending p^*_{i-1} by line segment $\overline{v^*_{i-1}v^*_i}$, whichever is less costly. The propagation process terminates when all points in $I_{v,e,e'}$ are reached by such interval paths. Observe that this process may be terminated before p^*_d is generated and inserted into QLIST. This would occur when another interval $I_{v',e,e'}$ is created that re-adjust the range of $I_{v,e,e'}$. We can establish the following theorem (we include the proof in the full version of this paper):

Theorem 1 *Each p^*_i is locally optimal.*



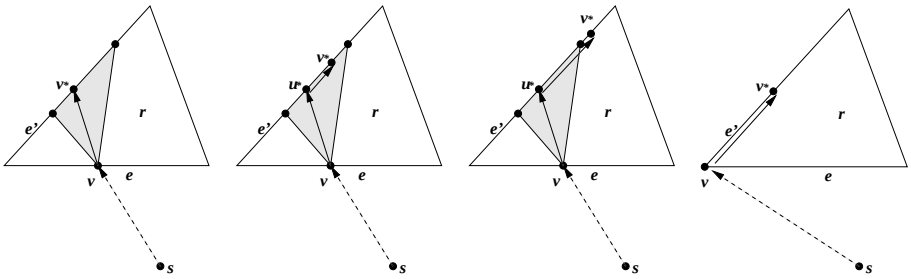
(a) Splitting an Interval

(b) Propagating Interval Paths

Fig. 2. Operations on Intervals

By the BUSHWHACK algorithm, each interval will generate no more than d locally optimal interval paths, where d is the number of points inside the interval. Thus, for each interval list $\text{ILIST}_{e,e'}$, only $m+2$ interval paths are inserted into QLIST, where m is the number of Steiner points on e'. The total number of interval paths, therefore, is $O(mn)$.

We need to show that this propagation scheme can find an optimal discrete path from s to v^* for any $v^* \in V'$. Following the notations used previously, we let e and e' be two boundary edges of region r. Let $p'_{opt}(s, v^*)$ be an optimal discrete path from s to point $v^* \in e'$ that enters region r through point $v \in e$. As shown in Figure 3, $p'_{opt}(s, v^*)$ can be categorized into one of the following four types: (1) $p'_{opt}(s, v^*) = p'_{opt}(s, v) + \overline{vv^*}$ is a face-crossing path; (2) $p'_{opt}(s, v^*) = p'_{opt}(s, v) + \overline{vu^*} + \overline{u^*v^*}$ is an edge-crawling path where $u^*, v^* \in I_{v,e,e'}$; (3) $p'_{opt}(s, v^*) = p'_{opt}(s, v) + \overline{vu^*} + \overline{u^*v^*}$ is an edge-crawling path where $u^* \in I_{v,e,e'}$ and $v^* \notin I_{v,e,e'}$; and (4) $p'_{opt}(s, v^*) = p'_{opt}(s, v) + \overline{vv^*}$ is an edge-crawling path where v is the joint end point of e' and e.



(a) Type 1 (b) Type 2 (c) Type 3 (d) Type 4

Fig. 3. Four Types of Optimal Paths

It is easy to see that this propagation scheme can find an optimal path $p'_{opt}(s, v^*)$ if $p'_{opt}(s, v^*)$ is of Type 1 or 2, as a Type 1 optimal path is a direct interval path and a Type 2 optimal path is an extended interval path. To capture an optimal path of Type 3, we need to create two other paths for each interval. Let v_0^* be the point next to v_1^* outside interval $I_{v,e,e'}$, and let v' be the discovered point on e whose interval $I_{v',e,e'}$ contains v_0^*. We insert path $p_0^* = p_1^* + \overline{v_1^* v_0^*}$ into QLIST when $I_{v,e,e'}$ is created, if $d'_{opt}(s, v) + d_r(v, v_1^*) + d_e(v_1^*, v_0^*) < d'_{opt}(s, v') + d_r(v', v_0^*)$. Similarly, we will add path $p_{d+1}^* = p_d^* + \overline{v_d^* v_{d+1}^*}$, if necessary, after all interval paths associated with this interval are inserted. Here v_{d+1}^* is the point next to v_d^* outside $I_{v,e,e'}$. We call these two paths *non-interval paths* as they are not associated with any interval. These non-interval paths will also be propagated when they are removed from QLIST, as we will show later in this

paper. It will also be clear in the next section how our algorithm finds optimal paths of Type 4.

4 Algorithm

The BUSHWHACK algorithm maintains three types of dynamic lists: (a) QLIST, the list of candidate optimal paths sorted by path cost; (b) PLIST$_e$, the list of discovered points on edge e; and (c) ILIST$_{e,e'}$, the list of intervals for edge e and e' that are on the boundary of the same region.

As we mentioned previously, all paths can be divided into two categories, interval paths and non-interval paths. All face-crossing paths along with some edge-crawling paths are interval paths. We have shown how two non-interval paths are generated for each interval. This section will explain how other non-interval paths are created by extending these paths.

The main body of the BUSHWHACK algorithm is a loop. Each iteration, the candidate optimal path p_{min} in QLIST with the minimum cost is extracted from the list. Let v be the ending point of path p. If v is not a discovered point (i.e., the distance from s to v is not yet decided), we claim that path p is an optimal path from s to v, and $d(p)$ is the distance from s to v.

```
FindPath(s,t)
    insert path p'_opt(s,s) into QLIST                                  [1]
    while t is not reached                                              [2]
        extract the least costly path p(s,v) from QLIST                 [3]
        if v is a not a discovered point                                [4]
            p'_opt(s,v) = p(s,v); d'_opt(s,v) = d(p(s,v))               [5]
            HandleNewDiscovery(v)                                       [6]
        Propagate(v,p)                                                  [7]
```

Function HandleNewDiscovery(v) creates new intervals for the newly discovered point v, and then inserts into QLIST an interval path associated with each of these intervals.

```
HandleNewDiscovery(v)
    if v is a Steiner point on an edge e                                [1]
        for each region r incident to e                                 [2]
            for each edge e' of r that is not e                         [3]
                create interval I_{v,e,e'}                              [4]
    else (v is a vertex in S)                                           [5]
        for each edge e incident to v                                   [6]
            let v_next be the neighboring Steiner point of v on e       [7]
            insert path p_new = p + v v_next into QLIST                 [8]
            for each region r incident to e                             [9]
                for each edge e' of r that is not e                     [10]
                    create interval I_{v,e,e'}                          [11]
    for each newly created interval I_{v,e,e'}                          [12]
        split I_{v,e,e'} into monotonic intervals I^1_{v,e,e'}, I^2_{v,e,e'}, ..., I^i_{v,e,e'}   [13]
        for each monotonic interval I^j_{v,e,e'}, 1 <= j <= i           [14]
            add the first non-interval path for I_{v,e,e'} if necessary [15]
            add the first interval path for I^j_{v,e,e'}                [16]
```

Whether or not v is a newly discovered point, Function Propagate(v, p) creates candidate optimal paths by propagating p in a constant number of ways and inserts these paths into QLIST.

```
Propagate(v, p)
    if p is an interval path associated with I_{v',e',e}                          [1]
        if p is still valid                                                        [2]
            if v is the last point in I_{v',e',e'}                                 [3]
                add the second non-interval path for I_{v',e',e'} if necessary     [4]
            else                                                                   [5]
                add the next interval path for I_{v',e',e}                         [6]
    else (p is an edge-crawling path whose last segment is on edge e)              [7]
        if v is not an end point of e                                              [8]
            let v_prev be the previous point of v on path p                        [9]
            let v_next be the neighboring point of v that is not between v_prev and v   [10]
            if there has not already been a path that extends to v_next from v     [11]
                insert path p_new = p + vv_next into QLIST                         [12]
```

We have explained previously how paths are propagated inside intervals. Observe that the task of handling interval paths is accomplished by the combination of the procedures Propagate and HandleNewDiscovery. For example, intervals are created in the procedure HandleNewDiscovery when a point v is discovered. At the same time, the first interval path associated with each new interval is inserted into QLIST (line 16 of the procedure HandleNewDiscovery). The propagation of interval paths for each interval is accomplished in the procedure Propagate (line 6). Generating the two non-interval paths for each interval is handled through HandleNewDiscovery (line 16) and Propagate (line 4).

Each interval also generates two non-interval paths, one when the interval is created (line 16 of the procedure HandleNewDiscovery) and the other when the last interval path of that interval is extracted from QLIST (line 4 of the procedure Propagate). Another situation that generates non-interval paths is that the newly discovered point v is an end point of an edge e. In this case, a non-interval path is inserted into QLIST that extends $p'_{opt}(s, v)$ to the neighboring Steiner point of v along edge e, as indicated by line 8 of HandleNewDiscovery.

All the non-interval paths are edge-crawling paths. According to the procedure Propagate (line 8 to line 12), when an non-interval path p from s to v is extracted from QLIST, we may insert an extension of this path into QLIST. Suppose $\overline{v_{prev}v}$ is the last segment of path p. Since p is edge-crawling, v_{prev} is on the same edge e as v. Let v_{next} be the adjacent Steiner point of v on e that is on the other side of v_{prev}. We insert path $p + \overline{vv_{next}}$ into QLIST, if there has not been another path $p' + \overline{vv_{next}}$ inserted into QLIST. The propagation of non-interval paths guarantees that any optimal path of Type 3 or 4 will not be missed by our algorithm.

For each Steiner point $v \in V_s$, there will be at most two non-interval paths from s to v inserted into QLIST, one approaching v from left and one approaching v from right. Similarly, for any vertex $v \in V$, there will be at most $d(v)$ non-interval paths that connect s and v, one from each edge incident to v. Here $d(v)$ is the number of incident boundary edges of v in the original triangular de-

composition. Thus, the total number of non-interval paths is bounded by $O(mn)$, and therefore the total number of all paths inserted into QLIST is $O(mn)$.

To show that the BUSHWHACK algorithm is correct, it is sufficient to prove the following theorem: (we include the proof in the full version of this paper):

Theorem 2 *When path $p(s,v)$ is extracted from QLIST, if v is not yet discovered, p is an optimal path from s to v in discrete graph G.*

The complexity of the BUSHWHACK algorithm depends on three factors: (a) the cost of maintaining QLIST which is $O(mn \log mn)$, as at most $O(mn)$ candidate optimal paths are inserted into QLIST; (b) the cost of maintaining all discovered point lists PLIST_e which is $O(nm \log m)$; (c) the cost of maintaining all interval lists $\text{ILIST}_{e,e'}$ which is $O(nm \log m)$. The complexity of the algorithm, therefore, is $O(nm \log nm)$.

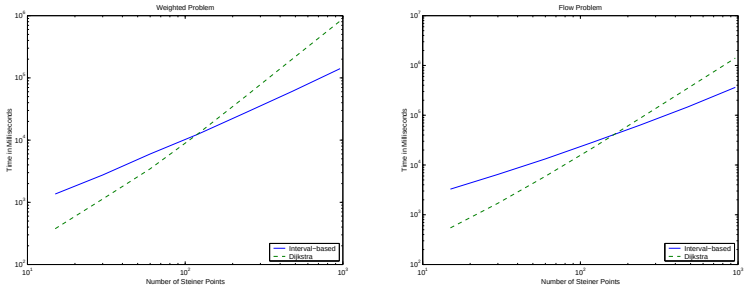
In the Introduction section we claimed that, in average, for each Steiner point v BUSHWHACK needs to evaluate the costs of only $O(\log m)$ adjacent edges of v. Even though inside each region only $O(m)$ edges are ever used by candidate optimal paths inserted into QLIST, that is, $O(1)$ edges per Steiner point in the region, BUSHWHACK has to evaluate the costs of additional edges in order to maintain the intervals. To decide the boundary of a new interval $I_{v,e,e'}$, BUSHWHACK needs to take a binary search of $O(\log m)$ steps. At each step, BUSHWHACK has to compare the cost of $\overline{vv^*}$ for some $v^* \in e'$ with the cost of $\overline{v'v^*}$, where v' is one of the two neighboring discovered points of v on e. As a result, $O(\log m)$ edges are evaluated for each Steiner point.

When a new interval $I_{v,e,e'}$ is created, we need to divide it into monotonic intervals. To do that, we need to compute the local extrema of the region distance function $g_{v,e}$ from v to points on e'. By Property 2 a pseudo-Euclidean region requires that these local extrema can be computed "efficiently." But we have not yet specified how efficient the computation needs to be. As only one splitting is performed for each interval, as long as the cost of computing local extrema (and thus the cost of splitting) for each interval does not exceed $O(\log m)$, the total cost of splitting will be bounded by $O(nm \log m)$ and will not affect the total complexity of $O(nm \log nm)$. It should be noted that we only need to find among m values (corresponding to m Steiner points) the values closest to the local extrema. For a weighted region or a flow region, such local extrema can be computed in constant time. In the full version of the paper, we will show that, as long as $g_{v,e}$ itself is a constant degree polynomial function or computing the local extrema of $g_{v,e}$ can be converted to computing the roots of a constant degree polynomial function, the local extrema of $g_{v,e}$ can be computed in $O(\log m)$ time.

5 Preliminary Experimental Results

In this section we report on some preliminary experimental results produced by *IntervalPathFinder*, a Java implementation of the BUSHWHACK algorithm. We compared our algorithm against Dijkstra's algorithm by running experiments on the same group of artificially generated datasets for both algorithms. All experiments are performed on a Windows 2000 workstation with 256MB memory

and a 550MHz Pentium III processor. For simplicity we used uniform discretization and each edge has equal number of Steiner points. The results of these experiments, as shown in Figure 4, are consistent with our complexity analysis. Although our interval-based BUSHWHACK is slower when m is small due to the high cost of maintaining various complex data structures, its efficiency quickly becomes evident when m is increased to 256 and above.



(a) Weighted Problem (b) Flow Problem

Fig. 4. Experimental Results

References

1. L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces. *Lecture Notes in Computer Science*, 1432:11–22, 1998.
2. L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Approximation algorithms for geometric shortest path problems. In *ACM Symposium on Theory of Computing*, 2000.
3. M. Lanthier, A. Maheshwari, and J. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings of the 13th International Annual Symposium on Computational Geometry*, pages 274–283. 4–6 June 1997.
4. C. Mata and J. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions. In *Proceedings of the 13th International Annual Symposium on Computational Geometry*, pages 264–273. June4–6 1997.
5. J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem. In *Proceedings of the 3rd International Annual Symposium on Computational Geometry*, pages 30–38. June 8–10 1987.
6. J. Reif and Z. Sun. An efficient approximation algorithm for weighted region shortest path problem. In *Proceedings of the 4th Workshop on Algorithmic Foundations of Robotics*, Mar. 16–18 2000.
7. J. Reif and Z. Sun. Movement planning in the presence of flows. *Lecture Notes in Computer Science*, 2125:450–461, 2001.