k-Link Rectilinear Shortest Paths Among Rectilinear Obstacles in the Plane

Valentin Polishchuk* Joseph S. B. Mitchell†

Abstract

We present an algorithm for computing k-link rectilinear shortest paths among rectilinear obstacles in the plane. We extend the "continuous Dijkstra" paradigm to store the link distance information associated with each propagating "wavefront". Our algorithm runs in time $O(kn \log^2 n)$ and space $O(kn)$, where n is the number of vertices of the obstacles. Previous algorithms for the problem had worst-case time complexity $O(kn^2)$.

Our algorithm builds a j-link shortest path map, rooted at a given source s, for each $j \leq k$. A shortest path query from s to a query point t can then be answered in time $O(\log n + j)$.

1 Introduction

We consider the bi-criteria rectilinear two-dimensional shortest path problem: Determine a rectilinear path of minimum (L_1) length, having at most k links, from s to t that avoids the interiors of a set of disjoint simple rectilinear obstacles having a total of n vertices. The bi-criteria rectilinear path problem naturally arises in certain wire-routing applications in which one is interested in finding a shortest rectilinear path for a wire that avoids a given set of components and is constrained to have at most k links.

Related Work

Bi-criteria path problems have received considerable attention in the computational geometry literature; see, e.g., [10, 11, 13]. Many problems are known to be NP-hard [1], but often only weakly so. For the special case of two criteria consisting of *length* (e.g., L_1 or L_2) and *link distance*, several polynomial-time algorithms are known for basic geometric optimal paths. See [6] for a survey specific to rectilinear paths and rectilinear obstacles, as studied here.

It is challenging to compute Euclidean shortest paths having at most k links, since there is no simple discrete graph that is "path preserving" for optimal paths. The special case of k-link paths in simple polygons and some approximation algorithms for more general cases are considered in [12].

In rectilinear polygonal domains, efficient algorithms are known for the bi-criteria path problem that combines *rectilinear* link distance and L_1 length. One can achieve worst-case time $O(kn^2)$ (more precisely, $O(k(m + n \log n))$, where m is a number of crossings in an arrangement, and is worst-case $\Theta(n^2)$); see [6, 14]. More efficient algorithms, running in nearly linear time, are known for optimal paths in a "combined metric"; see [2, 15][1]. Results are also known in higher dimensions for optimal paths in a combined metric, if the obstacles are given as a set of axis-parallel boxes [4].

Our results give an $O(kn \log^2 n)$ time algorithm, improving the $O(kn^2)$ bound by roughly a factor of n. We construct a family of planar subdivisions (link-restricted shortest path maps), one for each $j = 1, \ldots, k$, which gives a decomposition of the plane into cells according to the combinatorial type of a j-link shortest rectilinear path from the source s. For any query point t, the length of a shortest j-link path from s to t is determined by locating t in the jth map.

2 Overview of the Algorithm

We apply the "continuous Dijkstra" paradigm [7], which has been applied successfully to solving many optimal path problems in geometry. Since our new algorithm is based on a variant of the continuous Dijkstra algorithm of [9], we begin with a review of that method and then describe the changes necessary to extend it to our problem.

The algorithm considers the effects of sweeping an advancing "wavefront" from a source point s to all points of free space $\mathcal{F}$. (The *wavefront* at distance D is the set of points p of $\mathcal{F}$ for which the shortest path length from s to p is D.) In order to simulate the advancement

*Department of Applied Mathematics and Statistics, Stony Brook University, valentin.polishchuk@stonybrook.edu

†Department of Applied Mathematics and Statistics, Stony Brook University, jsbm@ams.sunysb.edu. J. Mitchell is partially supported by grant No. 2000160 from the U.S.-Israel Binational Science Foundation, NASA Ames Research (NAG2-1620), the National Science Foundation (CCR-0098172, ACI-0328930, CCF-0431030), and Metron Aviation.

[1] Some of these nearly-linear time results are said to apply to optimizing any nondecreasing function $f(l, j)$ of the L_1 length l and the number of links j; however, we suspect there is a misunderstanding, since one could seemingly apply this result to the function $f(l, j)$ that is l if $j \leq k$ and ∞ otherwise, giving a nearly-linear time algorithm (independent of k) for the k-link shortest rectilinear path. The algorithms are based on applying Dijkstra's algorithm in a single-criterion weighted graph, rather than a dynamic program (e.g., Bellman-Ford) that searches for shortest j-link paths.

of wavefronts correctly, the following information associated with each segment $\overline{qq'}$ of the wavefront is stored in a priority queue (called the *event queue*):

(a). its orientation, which will always be either northwest (NW), southwest (SW), southeast (SE) or northeast (NE) in the case of the L_1 metric;

(b). its endpoints q and q', which are the positions of the segment's endpoints at the moment the segment is first instantiated, before it starts being "dragged";

(c). its left and right *track rays* — these are the rays along which q and q' must be dragged, and they may be horizontal or vertical rays through free space or rays containing obstacle edges;

(d). the *stop points* L and R of the left and right track rays — these are the first obstacle points "hit" by the left and right track rays. (If the track rays intersect each other at point u before they hit obstacles, then $L = R = u$, where u is the inside corner of the corresponding segment dragging query.);

(e). its *root* r, which is an obstacle vertex that is responsible for propagating the portion of the wavefront to which the segment belongs;

(f). its *contact list*, which is the set of obstacle edges that the dragged segment touches (including the obstacle edges on which its endpoints may be sliding);

(g). its *event position* $\overline{q_e q'_e}$, which is the next position of the segment at which the contact list changes;

(h). its *event point* p, which is the point that is responsible for the change in the contact list when the segment reaches its event position. The event point p must lie on the boundary of an obstacle, and it will either be a stop point or a vertex;

(i). the *event distance*, which is the distance from s at which the event point is encountered by the segment.

The segments in the event queue are ordered according to their event distances. The *next event* is the dragged segment whose event distance is minimum and is obtained by popping the queue. In the case of ties, we can order the event distances by the lexicographic ordering of the x- and y-coordinates of their roots.

Each obstacle vertex u has associated with it a sorted list, the *SE-hit list*, $\mathcal{R}^{SE}(u) = \{r_1,\ldots,r_N\}$ of roots r_i of dragged segments that are southeast of u and are such that the dragged segment has "hit" point u (i.e., u has been an event point for a dragged segment rooted at r_i, and this event has already occurred). Similar definitions apply to the hit lists $\mathcal{R}^{NE}(u)$, $\mathcal{R}^{NW}(u)$, and $\mathcal{R}^{SW}(u)$ of the roots of the segments which have hit u from southwest, southeast and northeast respectively. The total size of all lists is bounded above by $O(n \log n)$.

Also associated with each obstacle vertex u is a *permanent label*, $d(u)$, which, at the conclusion of the algorithm, gives the length $\ell(s, u)$ of shortest path from s to u. Initially, $d(u) = +\infty$ for all u. We say that u has been *permanently labeled* if $d(u) < +\infty$. We say that a *non*-vertex point x has been permanently labeled if it lies in the region swept out by some dragged segment. Each vertex u also has a pointer, parent(u), which, at the conclusion of the algorithm, points to the parent of u in the shortest path tree SPT(s). Initially, parent(u) =NIL.

There are three types of events:

(I). the event point p is one of the stop points;

(II). p is interior to the dragged segment in its event position; and,

(III). p is a vertex encountered by an endpoint of the dragged segment.

The event queue is updated at each event so as to simulate the wavefront propagation correctly. Determining events in the continuous Dijkstra method involves answering *segment dragging queries* of special forms; see [9] for details.

Modifications to account for link distance. In order to modify the above algorithm for the k-link path problem, we extend the continuous Dijkstra algorithm to store the (rectilinear) link distance from s to any point u on the wavefront. In particular, we distinguish between s-u paths ending with a vertical link and s-u paths ending with a horizontal link. To this end we associate with each wavefront segment one or two additional segments: a horizontal segment, called a *v-source* and/or a vertical segment, called an *h-source*. With each v-source v we store its link number, $l.v$, which is the link distance from s to v, and pointer to a predecessor h-source, $pred.v$. Then the shortest $(l.v + 1)$-link s-u path with last link vertical may go from s through $pred.v$ to v and then to u. We store similar information with each of the h-sources. Refer to Figure 1.

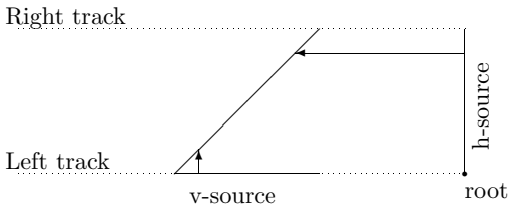

Figure 1: The information associated with a wavefront segment.

Each obstacle vertex u has now associated with it k SE-hit lists. For $j \in \{1 \ldots k\}$, the hit list ${}^j\mathcal{R}^{SE}(u)$ contains the roots and sources of dragged segments that are southeast of u and are such that the dragged segment has "hit" point u *and* the link distance from s to the source is less than j. (Clearly, ${}^1\mathcal{R}^{SE}(u) \subseteq {}^2\mathcal{R}^{SE}(u) \subseteq \cdots \subseteq {}^k\mathcal{R}^{SE}(u)$, so we only store the corresponding set differences). We similarly define hit lists for other hit directions.

Initially, s is permanently labeled with 0. Four

dragged segments rooted at s are inserted along with their distance labels into the event queue: NE, NW, SW and SE segments with the tracks being horizontal and vertical rays from s.

Events. The propagation of the wavefront involves doing different things depending on whether the next event is of Type I, II, or III and on whether or not the event point p has already been permanently labeled. The cases are illustrated in Figures 2, 3, 4 and 5 for a segment dragged northeast. Processing the events for the segments propagating in other directions is similar. The details of the events processing are mostly the same as in [9]. It is important, though, to modify the clipping of wavefronts in order that the only wavefronts that are permitted to continue (and not be clipped appropriately) are those corresponding to Pareto-optimal solution paths.

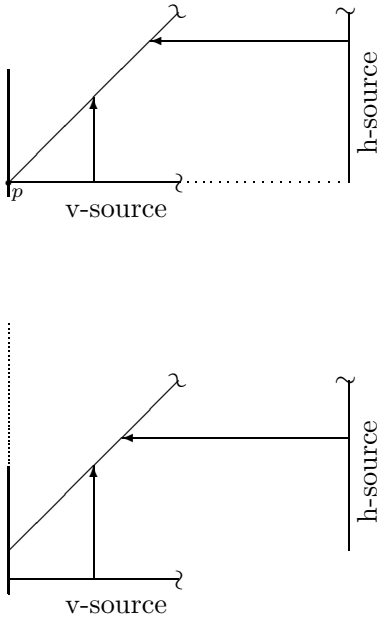

Figure 2: Type I event.

Complexity of the algorithm. Since we are propagating up to k different wavefronts (corresponding to up to k different link distances to the points on wavefronts), the complexity of the algorithm in [9] goes up by a factor of k and becomes $O(kn \log^2 n)$.

The proof of correctness is based on an induction argument, establishing that each point t reached (swept over) by the ith event (with associated L_1 distance d_i) have been reached by a dragged segment corresponding to each of the link distances $j = d_L(s,t), \ldots, l(t)$, where $d_L(s,t)$ is the rectilinear link distance from s to t and $l(t)$ is the (maximum) rectilinear link length of a shortest L_1 path from s to t.

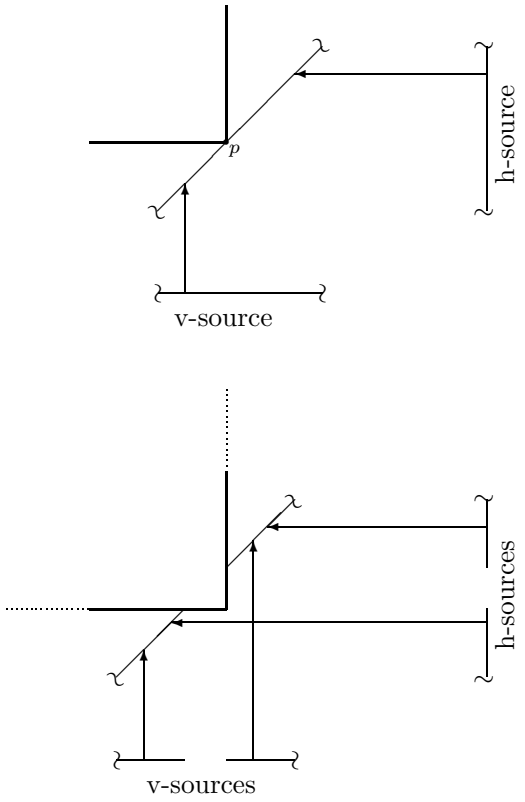

Figure 3: Type II event.

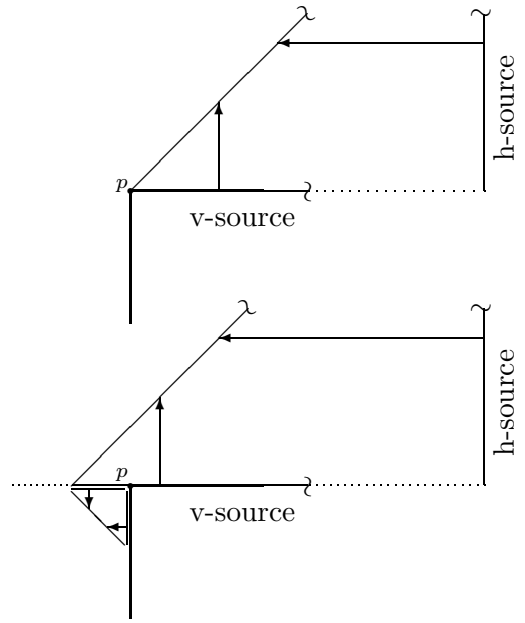

Figure 4: Type III event.

3 Conclusion

We expect that our results can be extended and improved. First, we expect that our method applies

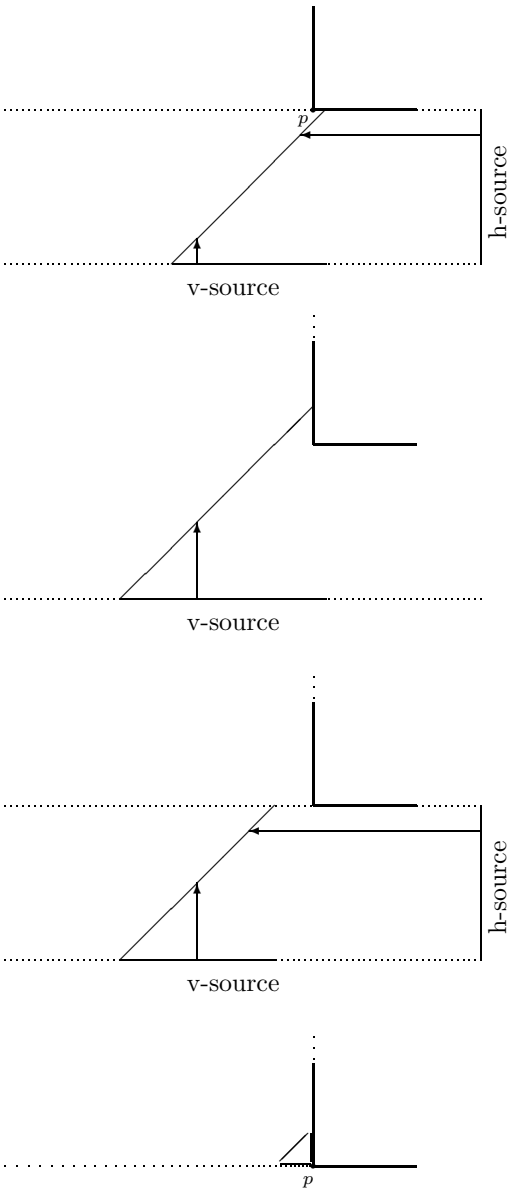

Figure 5: Another Type III event.

to computing rectilinear paths among general (non-rectilinear) polygonal obstacles. Also, it should be possible to reduce one factor of $\log n$ from the time complexity by applying a more careful accounting scheme, which has led to continuous Dijkstra algorithms with complexity $O(n \log n)$ for L_1 shortest paths among obstacles [5, 8]. Further, by applying the algorithm to multiple fixed orientations of path links, and possibly to weighted subdivisions, one could address more general versions of the k-link shortest path problem, as has been recently studied in [3].

References

[1] E. M. Arkin, J. S. B. Mitchell, and C. D. Piatko. Bicriteria shortest path problems in the plane. In *Proc. 3rd Canad. Conf. Comput. Geom.*, pages 153–156, 1991.

[2] D. Z. Chen, O. Daescu, and K. S. Klenk. On geometric path query problems. *Internat. J. Comput. Geom. Appl.*, 11(6):617–645, 2001.

[3] O. Daescu, J. S. Mitchell, S. Ntafos, J. D. Palmer, and C. K. Yap. k-link shortest paths in weighted subdivisions. In *Proc. 9th Workshop Algorithms Data Struct.*, page to appear, 2005.

[4] M. de Berg, M. van Kreveld, B. J. Nilsson, and M. H. Overmars. Shortest path queries in rectilinear worlds. *Internat. J. Comput. Geom. Appl.*, 2(3):287–309, 1992.

[5] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Comput.*, 28:2215–2256, 1999.

[6] D. T. Lee, C. D. Yang, and C. K. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Appl. Math.*, 70:185–215, 1996.

[7] J. S. B. Mitchell. *Planning shortest paths.* Ph.D. thesis, Stanford Univ., Stanford, CA, 1986.

[8] J. S. B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. In *Abstracts 1st Canad. Conf. Comput. Geom.*, page 22, 1989.

[9] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

[10] J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[11] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry (2nd Edition)*, chapter 27, pages 607–641. Chapman & Hall/CRC, Boca Raton, FL, 2004.

[12] J. S. B. Mitchell, C. Piatko, and E. M. Arkin. Computing a shortest k-link path in a polygon. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 573–582, 1992.

[13] C. D. Piatko. *Geometric Bicriteria Optimal Path Problems.* Ph.D. thesis, Cornell University, 1993.

[14] C. D. Yang, D. T. Lee, and C. K. Wong. On bends and lengths of rectilinear paths: a graph theoretic approach. *Internat. J. Comput. Geom. Appl.*, 2(1):61–74, 1992.

[15] C. D. Yang, D. T. Lee, and C. K. Wong. Rectilinear paths problems among rectilinear obstacles revisited. *SIAM J. Comput.*, 24:457–472, 1995.