

# All Farthest Neighbors in the Presence of Highways and Obstacles<sup>\*</sup>

Sang Won Bae<sup>1</sup>, Matias Korman<sup>2</sup>, and Takeshi Tokuyama<sup>2</sup>

<sup>1</sup> Division of Computer Science, KAIST, Korea. [swbae@tclab.kaist.ac.kr](mailto:swbae@tclab.kaist.ac.kr)

<sup>2</sup> Graduate School of Information Sciences, Tohoku University, Sendai, 980-8579 Japan. [{mati,tokuyama}@dais.is.tohoku.ac.jp](mailto:{mati,tokuyama}@dais.is.tohoku.ac.jp)

**Abstract.** We consider the problem of computing all farthest neighbors (and the diameter) of a given set of  $n$  points in the presence of highways and obstacles in the plane. When traveling on the plane, travelers may use highways for faster movement and must avoid all obstacles. We present an efficient solution to this problem based on knowledge from earlier research on shortest path computation. Our algorithms run in  $O(nm(\log m + \log^2 n))$  time using  $O(m + n)$  space, where the  $m$  is the combinatorial complexity of the environment consisting of highways and obstacles.

## 1 Introduction

Given a set  $S$  of  $n$  points in a space with a metric  $d$ , the *farthest neighbor*  $f(s)$  of  $s \in S$  is defined by  $\arg \max_{p \in S} d(s, p)$ . The value  $\max_{s \in S} d(s, f(s))$  is called the *diameter* of  $S$ . The problem of computing the farthest point for every point of  $S$  is called the *all farthest neighbors problem* (AFNP).

The AFNP and diameter computation are classical problems in computational geometry, and it is clear that the diameter problem can be solved once we solve the AFNP. If we can evaluate the distance between given two points in  $O(1)$  time, we can compute all farthest neighbors in  $O(n^2)$  time in a naïve way. Also, if the distance is defined by the shortest path distance of a graph with  $n$  vertices, the all farthest neighbors problem can be solved by first solving the all-pairs-shortest-path problem and spending  $O(n^2)$  additional time.

However, we can do better in several cases: If the space is the Euclidean plane, the diameter and also AFNP are computed in  $O(n \log n)$  time by constructing the farthest Voronoi diagram. The diameter problem in the 3-dimensional space has also been well-studied, and can be solved in  $O(n \log n)$  time [9].

In this paper, we consider the AFNP in metric spaces modeling urban transportation systems. The underlying space of our environment is the plane with the  $L_1$  metric, also known as the Manhattan metric. In addition, we are given a set  $\mathcal{O}$  of obstacles and a set  $\mathcal{H}$  of vertical and horizontal highways in the plane.

---

<sup>\*</sup> Work by S.W. Bae was supported by the Brain Korea 21 Project. Work by M. Korman was supported by MEXT scholarship and CERIES GCOE project, MEXT Japan.

We assume that a traveler can use the highways to move faster and that (s)he should avoid the obstacles during traveling. In this situation, we want to compute paths of shortest travel time. This setting well reflects a city scene with a transportation system: consider a road system in a city. Areas that cars cannot trespass are considered as obstacles. We drive on avenues and streets, which are vertical and horizontal, respectively. We formally define this environment, shortest travel time paths, and a metric  $d$  induced by shortest paths, called the *generalized city metric*.

The generalized city metric space is a natural extension of two known realistic models; the plane with either polygonal obstacles or with highways. Shortest path computation under either of the models has been extensively studied: in the presence of polygonal obstacles, the optimal algorithm was introduced by Mitchell [10, 11] applying the *continuous Dijkstra method*, and is extended to compute the Voronoi diagram with respect to the shortest path distance. The case in which only highways exist (named the *city metric*) was considered by Aichholzer *et al.* [3]. Also, the shortest path and the Voronoi diagram in this case can be computed in optimal time [5].

However, in the literature, only few results about farthest neighbors in the presence of highways or obstacles can be found. A brute-force way to solve the AFNP examines all the pairs of given points, spending quadratic time in the number of given points. As in the Euclidean case, the farthest Voronoi diagram can be used to solve the AFNP efficiently: once the diagram is computed, the farthest neighbor  $f(s)$  of each  $s \in S$  can be found in logarithmic time. When the obstacles are all axis-parallel rectangles on the  $L_1$  plane, an implicit structure of the farthest Voronoi diagram can be constructed in  $O(mn \log(m+n))$  time, where  $m$  is the total complexity of the given obstacles [6]. In the presence of  $m$  highways and no obstacle, an  $O(nm \log^3(n+m))$  time algorithm for computing the farthest Voronoi diagram has been recently introduced [4]. Memory space for any algorithm that uses the FVD is  $\Omega(nm)$  since it is known that the diagram can have  $\Omega(nm)$  complexity [4]. To the best of our knowledge, there is no known algorithm to solve the AFNP in the presence of highways and obstacles, simultaneously.

In this paper, we adopt a different approach using the *shortest path map* and *segment dragging queries*, and solve the AFNP without building the farthest Voronoi diagram. Our algorithm runs in  $O(nm(\log m + \log^2 n))$  time and uses linear space (i.e.:  $O(n+m)$ ), which improves the previously best known  $O(nm \log^3(n+m))$  running time for the city metric, reduces the total required memory and works in a more general environment.

We mention an application of our AFNP algorithm: the diameter is an important criterion to measure efficiency of a transportation system, and can be used for extending any such system. Ahn *et al.* [2] and Cardinal *et al.* [7] considered the problem of finding the optimal location of a highway in order to minimize the diameter of a given set of points in an empty transportation system (i.e.: an environment with no highways or obstacles). Our AFNP algorithm combined with some optimization techniques allow us to generalize such results to locate a

highway in a city with existing highways and obstacles (full details of such method are explained in a companion paper).

The precise definition of the generalized city metric and some preliminaries are given in Section 2. We first present an algorithm to solve the AFNP for the city metric in Section 3, and then consider the generalized city metric in Section 4. Finally, Section 5 concludes this paper with some remarks and open issues.

## 2 Preliminaries

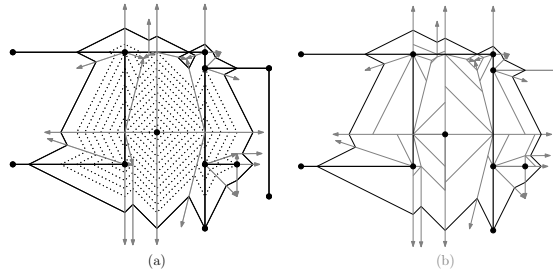
In this section we introduce the formal definition of the generalized city metric and the Shortest Path Maps and Segment Dragging Queries, key elements of our AFNP algorithm.

### 2.1 City Metric and Generalized City Metric

A *highway* is a facility supporting faster movement. We represent a highway by a line segment on the plane. Each highway  $h$  is associated with a *speed*  $\nu(h) > 1$ . A traveler moves at speed  $\nu(h)$  when moving along  $h$ , while the speed when not using any highway is 1. We deal with two kinds of highways; one is called a *freeway* allowing access through any point on it, and the other a *turnpike* accessible only through its two endpoints. An *obstacle* is a region that a traveler is not allowed to cross, and represented by a simple polygon.

Let  $\mathcal{H}$  be a set of highways and  $\mathcal{O}$  a set of disjoint obstacles in the  $L_1$  plane. A feasible path is a rectilinear path avoiding all obstacles in  $\mathcal{O}$ . We let  $\mathcal{F} := \mathbb{R}^2 \setminus \bigcup \mathcal{O}$  be the *free space*. For any feasible path  $\pi$  between two points in  $\mathcal{F}$ , we can measure the travel time of  $\pi$  since we know the speed of movement at any point on  $\pi$  and the length of any piece of  $\pi$ . We call a feasible path  $\pi$  connecting  $s, t \in \mathcal{F}$  a *shortest (travel time) path* if  $\pi$  minimizes the travel time between  $s$  and  $t$  among all feasible  $s$ - $t$  paths. We let  $d(s, t)$  be the travel time of a shortest path between  $s$  and  $t$ . Then,  $d$  is a metric on  $\mathcal{F}$  since  $d$  is based on shortest paths on  $\mathcal{F}$ . We call  $d$  the *generalized city metric induced by  $\mathcal{H}$  and  $\mathcal{O}$* . Through this paper, let  $m$  be the combinatorial complexity of the set of highways and obstacles.

One can find earlier research related to the generalized city metric: If  $\mathcal{H} = \emptyset$ , we have polygonal obstacles in the  $L_1$  plane [10]. The case  $\mathcal{O} = \emptyset$  and all highways are freeways was considered by Aichholzer *et al.* [3] and Bae *et al.* [5] (this metric is called the *city metric*). The case where we are given only turnpikes was considered by Ostrovsky-Berman [12]. Thus, by the generalized city metric, we deal with all the three kinds of objects in one environment. For simplicity in the explanation, we deal with only axis-parallel highways of equal speed  $\nu$  but allow obstacle edges to have any orientation. Note that one can allow a constant number of speeds for the highways without much modification of our algorithms.



**Fig. 1.** (a) The wavefront propagation in the presence of freeways of speed 2. The wavefront  $W(\delta)$  is divided into wavelets dragged by track rays (gray arrows). (b) The (partial) resulting shortest path map.

## 2.2 The Continuous Dijkstra Method and Shortest Path Maps

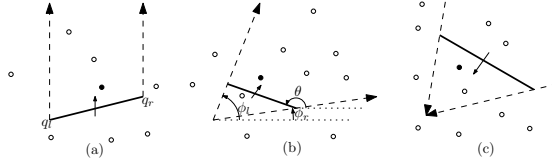
The *continuous Dijkstra method* is a conceptual algorithmic method to compute shortest paths from a given source  $s \in \mathbb{R}^2$  to *every* other point. The output of the continuous Dijkstra method is called the *shortest path map* of a given source point  $s \in \mathbb{R}^2$ . For a fixed point  $s$ , a shortest path map for  $s$  is a subdivision of the plane into cells each of which is a set of points from which shortest paths to  $s$  are combinatorially equivalent. An implementation of the continuous Dijkstra method simulates the *wavefront propagation* from  $s$ : the *wavefront*  $W(\delta)$  is defined as the set  $\{p \in \mathbb{R}^2 \mid d(p, s) = \delta\}$  for any positive  $\delta$ . In particular, under the  $L_1$  metric, the wavefront  $W(\delta)$  is expressed by a set of line segments, called *wavelets*.

In this environment, the wavelets have eight possible inclinations:  $\pi/4$ ,  $3\pi/4$ ,  $\beta$ ,  $\pi - \beta$ ,  $\pi/2 + \beta$  and  $\pi/2 - \beta$ , where  $\beta = \tan^{-1} 1/\nu$  [5]. Each wavelet is propagated in a certain direction along two *track rays* as  $\delta$  increases. Each track ray of a wavelet is the locus of the endpoints of the wavelet and traces an edge of the resulting shortest path map. Figure 1 illustrates the wavefront propagation and how related the wavelets are to the resulting shortest path map. For more details, we refer to several technical papers implementing the continuous Dijkstra method [5, 10–12].

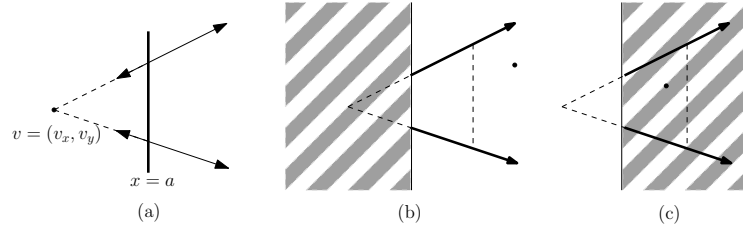
## 2.3 Segment Dragging Queries

The segment dragging query problem is formulated as follows: Determine the next point “hit” by a query segment  $qq'$  when it is “dragged” along two rays. More formally, given three orientations  $\theta$ ,  $\phi_l$ , and  $\phi_r$ , we want to preprocess a set  $S$  of points for determining the first point hit by a query segment  $q_lq_r$  of orientation  $\theta$  when  $q_l$  slides in direction  $\phi_l$  and  $q_r$  in direction  $\phi_r$  in such a way that the segment being dragged remains parallel to  $\theta$ . We call the locus of the endpoints of the dragged segment the *track rays*.

This problem was first considered by Chazelle [8] in the particular case in which track rays were parallel. The generalization to three different types (par-



**Fig. 2.** Three types of segment dragging: parallel, out of a corner and into a corner



**Fig. 3.** (a) Queries  $out(a, C)$  and  $into(a, C)$  (b)(c) Oracle construction: after performing an out of the corner query we know in which side our solution lies. Depending on whether or not the reported point is inside the query range, we can discard either the right or the left halfplane, respectively.

allel, dragging *out* of a corner and dragging *into* a corner, see Figure 2) was considered in [10]. For simplicity, we call each kind of query type (a), (b) or (c).

The first two cases can be handled in optimal time and space:

**Lemma 1 (Chazelle [8] and Mitchell [10]).** *One can preprocess a set  $S$  of  $n$  points into a data structure of  $O(n)$  size in  $O(n \log n)$  time that answers type (a) and (b) segment dragging queries in  $O(\log n)$  time.*

To the authors' knowledge, no optimal way to handle type (c) queries is known. Simple techniques for each particular problem have been used in the literature to avoid those queries [10, 5]. Here, we introduce a simple way to handle them with an additional logarithmic factor in the query time:

**Lemma 2.** *One can preprocess a set  $S$  of  $n$  points into a data structure of size  $O(n)$  in  $O(n \log n)$  time which answers into a corner segment dragging queries in  $O(\log^2 n)$  time.*

*Proof.* Let  $\theta$ ,  $\phi_l$ , and  $\phi_r$  be three line orientations. Let  $C$  be the wedge with apex  $v = (v_x, v_y)$  and bounding rays of orientations  $\phi_l$  and  $\phi_r$ . Let  $out(a, C)$  and  $into(a, C)$  be the dragging out of and into dragging queries with respect to the vertical query segment  $C \cap \{x \geq a\}$  and  $C \cap \{x \leq a\}$ , respectively, (see Figure 3(a)). Without loss of generality, we can assume that the query segment is vertical (i.e.:  $\theta = \pi/2$ ) and  $v_x < a$ .

We will give an oracle that, given  $x_0 \in \mathbb{R}$ , computes whether or not the point to report (if it exists) lies in the halfplane  $\{x \leq x_0\}$  in  $O(\log n)$  time. Combining the oracle with a binary search (in the  $x$ -coordinates of the points in  $S$ ) allow us

to answer type-(c) queries in  $O(\log^2 n)$  time. Note that the only preprocessing needed is the sorted list of points and the structure to allow type-(b) range queries.

The oracle performs a single type (b) dragging query  $out(x_0, C)$ . If a point whose  $x$  coordinate is less than  $a$  is found (i.e.: the reported point is inside the wedge  $C \cup \{x \geq x_0\}$ ), then the solution of  $into(a, C)$  has an  $x$ -coordinate higher than  $x_0$ . Otherwise, the solution of  $into(a, C)$  has an  $x$ -value smaller than  $x_0$ . (See Figure 3(b) and (c).)

### 3 AFNP in the presence of Freeways

In this section, we first consider the AFNP under the city metric induced by a set of freeways. Let  $S \subset \mathbb{R}^2$  be a set of  $n$  points and  $\mathcal{H}$  be the set of  $m$  axis aligned freeways of speed  $\nu > 1$ . Bae *et al.* [5] obtained the first optimal algorithm for computing the shortest path map of a fixed point  $s$  by applying the continuous Dijkstra method:

**Lemma 3 (Bae *et al.* [5]).** *Given  $m$  freeways, the shortest path map  $\mathcal{SPM}_s$  for a given source  $s \in \mathbb{R}^2$  under the city metric can be computed in  $O(m \log m)$  time with  $O(m)$  space.*

They also showed several properties of the shortest path map  $\mathcal{SPM}_s$  obtained by their algorithm. We can rephrase them as follows:

**Lemma 4 (Bae *et al.* [5]).** *Let  $\Theta$  be the set of all possible inclinations for wavelets and  $\Phi$  be the set of all possible orientations of track rays of wavelets under a city metric. Then, we have  $|\Theta| = 6$ ,  $|\Phi| \leq 24$ . Moreover, the orientation of any edge of  $\mathcal{SPM}_s$  is in  $\Phi$  and each cell of  $\mathcal{SPM}_s$  is  $x$ -monotone or  $y$ -monotone.*

These properties of  $\mathcal{SPM}_s$  allow us to find the farthest neighbor  $f(s)$  of  $s \in S$  efficiently:

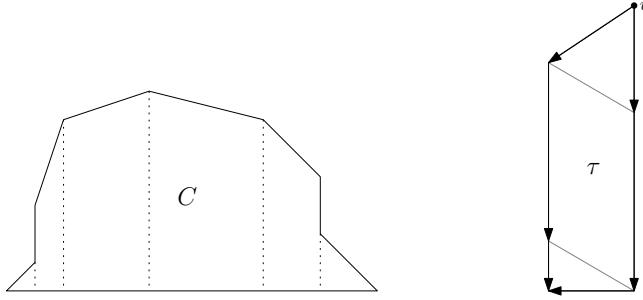
**Theorem 1.** *Given  $m$  axis parallel freeways of equal speed and  $n$  points, all farthest neighbors and the diameter among  $n$  points under the city metric can be computed in  $O(nm(\log m + \log^2 n))$  time using  $O(n)$  space.*

```

Preprocess  $S$  for segment dragging queries
for each  $s \in S$  do
  compute  $\mathcal{SPM}_s$  and decompose it into trapezoids
  for each trapezoid  $\tau$  do
    Find  $f_\tau(s) = \arg \max_{p \in S \cap \tau} d(s, p)$  by segment dragging queries
  end for
end for

```

**Fig. 4.** AFNP pseudo-code



**Fig. 5.** A cell  $C$  of  $\mathcal{SPM}_p$  is divided into trapezoids and each trapezoid  $\tau$  is swept by three consecutive queries.

*Proof.* The pseudo-code of our algorithm can be seen in Figure 4: after preprocessing  $P$  to allow segment dragging queries for  $\theta \in \Theta$  and  $\phi_l, \phi_r \in \Phi$ , we compute  $f(s)$  for each  $s \in S$  independently as follows: we construct the shortest path map  $\mathcal{SPM}_s$  using the algorithm of Bae *et al.* [5]. Also, we divide each cell  $C$  of  $\mathcal{SPM}_s$  into trapezoids: if  $C$  is  $x$ -monotone (resp.  $y$ -monotone) we cut  $C$  by vertical (resp. horizontal) lines through each vertex on the boundary of  $C$ . Consider the resulting subdivision: each cell is a trapezoid whose edges have inclinations in  $\Phi$  by Lemma 4 and our construction of the trapezoids.

Let  $\tau$  be any such trapezoid; without loss of generality we can assume that  $\tau$  comes from an  $x$ -monotone cell  $C$  of  $\mathcal{SPM}_s$ . Now, we describe how to find the point  $f_\tau(s) \in S \cap \tau$  that maximizes the distance  $\max_{p \in S \cap \tau} d(s, p)$ : consider the set  $W_\tau(\delta) := \{q \in \tau \mid d(s, q) = \delta\}$  (that is, the intersection of the wavefront  $W(\delta)$  and  $\tau$ ): since  $\tau$  is completely included in a cell  $C$  of  $\mathcal{SPM}_s$ , the shortest path topology to  $s$  is the same for any point  $q \in \tau$ . Thus  $W_\tau(\delta)$  is either a line segment or an empty set for any  $\delta > 0$ . Moreover, if  $W_\tau(\delta)$  is a segment, its slope must be in  $\Theta$ , since it is a portion of a wavelet by Lemma 4.

Now, consider sweeping  $\tau$  by  $W_\tau(\delta)$  as  $\delta$  decreases. Let  $p_\tau \in S \cap \tau$  be the first point hit by  $W_\tau(\delta)$ , if  $S \cap \tau \neq \emptyset$ . Any other point  $q \in S \cap \tau$  will have *smaller* distance to  $s$  and therefore can be ignored, and thus  $f_\tau(s) = p_\tau$ . We sweep  $\tau$  by three consecutive segment dragging queries: first find the  $v$  of  $\tau$  that is farthest away from  $s$ . We then perform a type (b) dragging query that originates from that vertex  $v$ . The point reported by the query either is  $p_\tau$  or lies out of  $\tau$ . If the point reported is outside  $\tau$ , we perform a type (a) segment dragging query along the vertical sides of  $\tau$ . Similarly, we perform a type (c) dragging query if no point has been found in the second query. Figure 5 shows these three consecutive segment dragging queries. We repeat this procedure for all trapezoids  $\tau$  and then use that  $f(s) = \max_\tau f_\tau(s)$ .

Preprocessing takes  $O(n \log n)$  time and needs  $O(n)$  space since  $\Theta$  and  $\Phi$  are of constant size by Lemma 4. Building  $\mathcal{SPM}_s$  (and its further decomposition into trapezoids) can be done  $O(m \log m)$  time and  $O(m)$  space using the algorithm of Bae *et al.* [5]. At most three segment dragging queries are needed to report  $f_\tau(s)$ ,

and iterating for all the trapezoids of the  $\mathcal{SPM}_s$  takes  $O(m \log^2 n)$  time in total. Since we do this procedure for all  $s \in S$ , the total time is  $O(nm(\log m + \log^2 n))$ .

## 4 Algorithm for the Generalized City Metric

In this section we will generalize the previous algorithm to work in a generalized city metric. First we will consider the case in which only obstacles exist and then focus in the general case.

### 4.1 All Farthest Neighbors in the Presence of Obstacles

For simplicity in the explanation, we assume the set  $\mathcal{O}$  are mutually disjoint simple polygons with total complexity  $m$ . Throughout this section, we let  $V$  be the set of vertices of obstacles in  $\mathcal{O}$ . Mitchell [10] gave an optimal algorithm to construct  $\mathcal{SPM}_s$  in the presence of obstacles on the  $L_1$  plane:

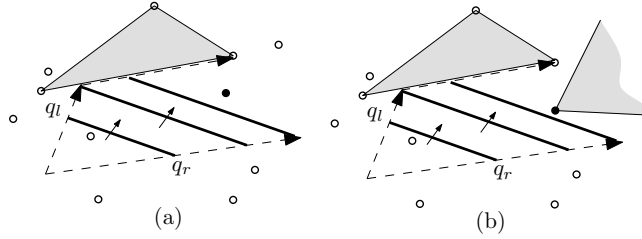
**Lemma 5 (Mitchell [10]).** *Given a set  $\mathcal{O}$  of polygonal obstacles, the  $\mathcal{SPM}_s$  of a source  $s \in \mathcal{F}$  under  $d$  induced by  $\mathcal{O}$  on the  $L_1$  plane can be computed in  $O(m \log m)$  time and  $O(m)$  space. Moreover,  $\mathcal{SPM}_s$  fulfills the following properties:*

- Any obstacle vertex  $v \in V$  is a vertex of the map  $\mathcal{SPM}_s$ .
- Each edge  $e$  of  $\mathcal{SPM}_s$  is a portion of an edge of an obstacle in  $\mathcal{O}$  or has slope  $\phi \in \Phi$ , where  $\Phi$  is defined in Lemma 4.
- Each cell of  $\mathcal{SPM}_s$  is  $x$ -monotone or  $y$ -monotone.
- Let  $v$  be a vertex on the boundary of a cell  $C$  of  $\mathcal{SPM}_s$  minimizing  $d(v, s)$  among points in  $C$ . Then, one of two incident edges to  $v$  is vertical or horizontal.

This algorithm is also based on the continuous Dijkstra paradigm. In the presence of obstacles, the inclinations of wavelets are  $\pi/4$  or  $3\pi/4$ . However, in this case we do not have a constant bound on the number of directions for the track rays of wavelets due to the fact that obstacles have  $m$  edges with free orientations. Mitchell adapted the segment dragging query method in order to cope with the growth of directions; as before, track directions are fixed but if the dragged segment encounters an obstacle edge, it changes the colliding track ray to slide along that edge (see Figure 6).

**Lemma 6 (Mitchell [10]).** *Given a set  $\mathcal{O}$  of obstacles with  $m$  vertices  $V$  and a set  $S$  of  $n$  points in the free space, one can preprocess  $\mathcal{O}$  and  $S$  into a data structure of  $O(m + n)$  size in  $O((m + n) \log(m + n))$  time that answers the segment dragging query of type (a) or (b) in  $O(\log(m + n))$  time to report the point in  $V \cup S$  hit first by the dragged segment, if any.*

Using this modified segment dragging query, we can solve the AFNP using the same approach as before:



**Fig. 6.** Segment dragging queries in the presence of obstacles. The small circles depict candidate points to be reported (the dark points are the ones to report by the query segment  $q_l q_r$ ). The reported point is either (a) a point in  $S$  or (b) an obstacle vertex in  $V$ .

**Theorem 2.** *Given obstacles  $\mathcal{O}$  with  $m$  vertices  $V$  and a set  $S$  of  $n$  points in the free space  $\mathcal{F}$ , all farthest neighbors and the diameter among  $S$  can be computed in  $O(nm(\log m + \log^2 n))$  time with  $O(n + m)$  space.*

*Proof.* First, we preprocess  $\mathcal{O}$  and  $S$  for segment dragging queries of type (a) and (b) by Lemma 6, and preprocess  $S$  for type (c) queries by Lemma 2. For  $s \in S$ , we compute the shortest path map  $\mathcal{SPM}_s$  in  $O(m \log m)$  time and divide each cell  $C$  of  $\mathcal{SPM}_s$  into trapezoids by the third and the fourth properties in Lemma 5: let  $v$  be the nearest vertex to  $s$  among all vertices on the boundary of  $C$ . If there is a vertical (or horizontal) edge incident to  $v$ , we cut  $C$  by horizontal (resp. vertical) lines through each vertex on the boundary of  $C$ . As done in Section 3, we find the farthest point  $f_\tau(s) \in S \cap \tau$  to  $s$  among the obstacles  $\mathcal{O}$  for each such trapezoid  $\tau$ .

Consider a trapezoid  $\tau$ . Without loss of generality, we assume that  $\tau$  comes from a cell of  $\mathcal{SPM}_s$  cut by vertical lines. As in the freeway case we perform three consecutive queries of type (b), (a) and (c) in order as in Figure 5. However, each segment dragging query may end with an obstacle vertex  $v \in V$  before encountering a point  $p \in S$  or a vertex of  $\tau$  since we have preprocessed  $S$  plus  $V$  for segment dragging queries in the presence of obstacles. In that case, we ignore  $v$  and do another segment dragging after  $v$ . Since the first two queries (of type (a) and (b)) sweep the interior of  $\tau$ , vertices  $v \in V$  encountered during these two queries always lie on the boundary of  $\tau$  by the first property in Lemma 5. Thus, these two queries in  $\tau$  are performed in time  $O(k \log(m + n))$ , where  $k$  is the number of obstacle vertices on the boundary of  $\tau$ . Observe that the trapezoidal decomposition of  $\mathcal{SPM}_s$  is indeed a complete subdivision of  $\mathcal{F}$ . This implies that summing  $k$  for all trapezoids  $\tau$  is at most twice the number of obstacle vertices. Therefore, the cost of the first two queries is bounded by  $O(m \log(m + n))$ .

The query of type (c) is performed at last. By the fourth property of Lemma 5 and our construction of trapezoids, one track ray is horizontal and the other is in  $\Phi$ . Since we have processed only  $S$  for segment dragging queries of type (c), during this query we do not encounter any obstacle vertex.

## 4.2 Combining Freeways, Turnpikes and Obstacles

We are now ready for combining all three kinds of transportation objects. The combination of freeways and obstacles was considered by Bae *et al.* [5] in the  $\mathcal{SPM}$  computation: the shortest path map can be computed in  $O(m \log m)$  time where freeways and obstacles with complexity  $m$  are given. Thus, Lemma 3 extends to the combined environment by freeways and obstacles. Moreover, the algorithm can in general compute the Voronoi diagram of  $k$  weighted points in  $O((m+k) \log(m+k))$  time and  $O(m+k)$  space [5]. (The distance of any  $q \in \mathbb{R}^2$  to a weighted point  $p$  is measured as  $d(q, p) + w(p)$ , where  $w(p)$  denotes the weight of  $p$ ). Furthermore, the resulting diagram has information about shortest paths to the nearest point in such a way that each region is subdivided into cells, where all the points in each cell have the combinatorially equivalent shortest paths; That is, the resulting diagram is a multi-source shortest path map.

**Lemma 7.** *The  $\mathcal{SPM}_s$  for any  $s \in \mathcal{F}$  under the generalized city metric induced by a set  $\mathcal{H}$  of highways and a set  $\mathcal{O}$  of obstacles can be computed in  $O(m \log m)$  time using  $O(m)$  space.*

*Proof.* Let  $\mathcal{H}_{\text{free}} \subseteq \mathcal{H}$  be the set of freeways in  $\mathcal{H}$ , and  $d'$  be the generalized city metric induced by the freeways  $\mathcal{H}_{\text{free}}$  and the obstacles  $\mathcal{O}$ . We fix a source  $s \in \mathcal{F}$ . Let  $P$  be the set of all endpoints of the turnpikes in  $\mathcal{H}_{\text{free}}$ , and  $w(p) := d(p, s)$  be the weight of each  $p \in P$ . Also, let  $P_s := P \cup \{s\}$  and  $w(s) = 0$ . Now, consider the Voronoi diagram  $\mathcal{V}_{d'}(P_s)$  of weighted points  $P_s$  under  $d'$ .

We show that  $\mathcal{V}_{d'}(P_s)$  coincides with a shortest path map  $\mathcal{SPM}_s$  for  $s$  under  $d$ . Take any shortest path  $\pi$  from  $a \in \mathcal{F}$  to  $s$ . If  $\pi$  uses no turnpike, we simply have  $d(a, s) = d'(a, s)$ . Otherwise, let  $p \in P$  be the first entrance of a turnpike used by  $\pi$ . Then, we have  $d(a, s) = d(a, p) + d(p, s) = d'(a, p) + d(p, s) = d'(a, p) + w(p)$ . Hence, in general,  $d(a, s) = d'(a, p) + w(p)$  for some  $p \in P_s$ . For any point  $a$  in the Voronoi region of  $p \in P_s$  and any other  $q \in P_s$ , we have  $d'(a, p) + w(p) \leq d'(a, q) + w(q)$  and thus  $d(a, s) = d'(a, p) + w(p)$ .

Therefore, we are done by computing  $\mathcal{V}_{d'}(P_s)$ . However, we do not know the value  $w(p) = d(p, s)$  at the beginning. Here, we introduce a trick to resolve this problem by *lazy evaluation*. The algorithm computing  $\mathcal{V}_{d'}(P_s)$  also applies the continuous Dijkstra method, and simulates the wavefront propagation, where the wavefront  $W(\delta)$  is defined as  $W(\delta) := \{a \in \mathcal{F} \mid \min_{p \in P_s} \{d'(a, p) + w(p)\} = \delta\}$ . We let  $P_s(\delta) := \{p \in P_s \mid d(p, s) \leq \delta\}$  and  $W'(\delta) := \{a \in \mathcal{F} \mid \min_{p \in P_s(\delta)} \{d'(a, p) + d(p, s)\} = \delta\}$ . Observe that  $W(\delta) = W'(\delta)$  and  $d(a, p) = \min_{p \in P_s(\delta)} \{d'(a, p) + d(p, s)\}$  for any  $a \in \mathcal{F}$  by the above argument.

When  $\delta < d(p, s)$ , no wavelets from  $p$  is propagated out. We do the following when the wavefront  $W'(\delta)$  hits an endpoint  $p$  of a turnpike  $h$  whose other endpoint is  $p'$ : if  $p$  has not been assigned its weight, we assign the weight  $w(p) = \delta$ . Similarly, if  $p'$  has not been assigned its weight, we assign the weight  $w(p') = \delta + \frac{l}{\nu(h)}$ , where  $l$  and  $\nu(h)$  are the length and the speed of  $h$ , respectively. Note that the value of  $w(p')$  might not be the same as  $d(p', s)$  but we have  $w(p') \geq d(p', s)$ . If  $w(p') < d(p', s)$ , the wavelet  $W'(\delta')$  will hit  $p'$  at  $\delta' = d(p', s)$  before  $w(p')$  thus the wrong weight assignment will be detected. Note that this

corresponds to the case in which highway  $h$  can be ignored (i.e.: the Voronoi regions of  $p$  and  $p'$  are empty, therefore, we can ignore  $p'$ ). Hence, by our lazy evaluation of the weight, whether it is correct or not, the wavefront  $W(\delta)$  is maintained correctly and the algorithm builds  $\mathcal{V}_{d'}(P_s)$  properly.

As shown in the proof of Lemma 7, the shortest path map  $\mathcal{SPM}_s$  under  $d$  coincides with a Voronoi diagram in the presence of freeways and obstacles only. Hence,  $\mathcal{SPM}_s$  inherits the properties of shortest path maps shown in the previous sections:

**Lemma 8.** *The  $\mathcal{SPM}_s$  computed by the algorithm in the proof of Lemma 7 fulfills the following properties:*

- Any obstacle vertex  $v \in V$  is a vertex of  $\mathcal{SPM}_s$ .
- Each edge  $e$  of  $\mathcal{SPM}_s$  is a portion of an edge of an obstacle in  $\mathcal{O}$  or has slope  $\phi \in \Phi$ , where  $\Phi$  is defined in Lemma 4.
- Each cell  $C$  of  $\mathcal{SPM}_s$  is  $x$ -monotone or  $y$ -monotone.
- Let  $C$  be a cell of  $\mathcal{SPM}_s$  and  $v$  be a vertex on the boundary of  $C$  minimizing  $d(v, s)$  among points in  $C$ . Then, one of two incident edges to  $v$  is vertical (or horizontal) and  $C$  is  $y$ -monotone (resp.  $x$ -monotone).

Finally, we can prove the general case:

**Theorem 3.** *All farthest neighbors and the diameter among  $n$  points in the free space  $\mathcal{F}$  under the generalized city metric induced by a set of highways and obstacles can be computed in  $O(nm(\log m + \log^2 n))$  time using  $O(n + m)$  space.*

*Proof.* The algorithm is almost the same as those introduced in Theorems 1 and 2. Let  $S$  be the given set of  $n$  points. We preprocess  $S$  and the obstacle vertices  $V$  for segment dragging queries as in Lemmas 6 and 2. Then, we compute the shortest path map  $\mathcal{SPM}_s$  for  $s \in S$  using Lemma 7 and subdivide each cell of  $\mathcal{SPM}_s$  into trapezoids as done in Theorems 1 and 2. For each trapezoid  $\tau$ , we perform consecutive segment dragging queries to find  $f_\tau(s)$ , the farthest point from  $s$  in  $S \cap \tau$ . As in Theorem 2, the number of segment dragging queries is bounded by  $O(m)$  in total, thus the theorem is shown.

## 5 Concluding Remarks

We considered the problem of finding the farthest neighbor of each point in a set  $S$  of  $n$  points under metrics defined by shortest paths in the plane. Under conventional metrics like the Euclidean or the  $L_1$  plane, this problem is easy as discussed in the beginning: In the  $L_1$  plane, the problem can be solved even in linear time. The problem we considered poses the additional difficulty that evaluating the distance between two points constant time computable and is no known nice geometry like the convex hull. Thus, finding a lower bound of such problem is an interesting open issue. As a progress on this question, Cardinal *et al.* [7] proved an  $\Omega(n \log n)$  lower bound in computing a diametral pair on the

$L_1$  plane with one turnpike. We conjecture that  $\Omega(nm \log(n + m))$  is the right bound for the generalized city metric, since it is known that the farthest Voronoi diagram can have  $\Omega(nm)$  combinatorial complexity [4, 6].

Constructing an optimal structure for the type (c) segment dragging queries is another challenge. In our algorithms, the segment dragging query is the most frequently called subroutine. Thus, improving its would automatically improve our algorithms. The basic idea of our approach can be extended to any metric where the wavefront is a set of line segments (such as the  $L_\infty$  metric or the fixed orientation metric [10]). This methodology can be generalized to other metrics, provided that there is a way to perform dragging queries of the wavefront shape. For example, under the Euclidean metric, the segments transform into arcs of circles and thus we need “circular-arc” dragging queries. Those queries can be performed in  $O(n^{1/2+\epsilon})$  time using  $O(n)$  space, after  $O(n \log n)$  preprocessing [1]. Thus, given a set of obstacles and turnpikes, we can solve the AFNP in the Euclidean plane  $O(nm(n^{1/2+\epsilon} + \log m))$  time using the Ostrovsky-Berman algorithm [12] for computing  $\mathcal{SPM}$ .

## References

1. P. Agarwal and J. Matousek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11(1):393–418, 1994.
2. H.-K. Ahn, H. Alt, T. Asano, S. W. Bae, P. Brass, O. Cheong, C. Knauer, H.-S. Na, C.-S. Shin, and A. Wolff. Constructing optimal highways. In *Proc. 13th Comput.: Australasian Theory Sympos. (CATS)*, volume 65 of *CRPIT*, pages 7–14, Ballarat, Australia, 2007. ACS.
3. O. Aichholzer, F. Aurenhammer, and B. Palop. Quickest paths, straight skeletons, and the city Voronoi diagram. In *Proc. 18th Annu. ACM Sympos. Comput. Geom.*, pages 151–159, 2002.
4. S. W. Bae and K.-Y. Chwa. The farthest city Voronoi diagram. In *Proc. of the First Meeting of AAAC*, 2008.
5. S. W. Bae, J.-H. Kim, and K.-Y. Chwa. Optimal construction of the city Voronoi diagram. In *Proc. 17th Annu. Internat. Sympos. Algo. Comput. (ISAAC)*, volume 4288 of *LNCS*, pages 183–192, 2006.
6. B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. Farthest neighbors and center points in the presence of rectangular obstacles. In *Proc. 17th Annu. ACM Sympos. Comput. Geom.*, pages 164–171, 2001.
7. J. Cardinal, S. Collette, F. Hurtado, S. Langerman, and B. Palop. Moving walkways, escalators, and elevators. *CoRR*, abs/0705.0635, 2007.
8. B. Chazelle. An algorithm for segment dragging and its implementation. *Algorithmica*, 3:205–221, 1988.
9. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 4:387–421, 1989.
10. J. S. B. Mitchell.  $L_1$  shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.
11. J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.*, 6(3):309–331, 1996.
12. Y. Ostrovsky-Berman. The transportation metric and related problems. *Inform. Process. Lett.*, 95:461–465, 2005.