유클리드 씨티 보로노이 다이어그램 계산을 위한 향상된 알고리즘

배 상 원[1]*, 김 재 훈[2], 좌 경 룡[1]
[1]한국과학기술원 전자전산학과 전산학전공
{swbae*,kychwa}@jupiter.kaist.ac.kr
[2]부산외국어대학교 컴퓨터공학과
jhoon@pufs.ac.kr

Improved Algorithm for Constructing Euclidean City Voronoi Diagrams

Sang Won Bae[1]*, Jae-Hoon Kim[2], Kyung-Yong Chwa[1]
[1]Div. of Computer Science, Dept. of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology
[2]Division of Computer Engineering, Pusan University of Foreign Studies

요 약

This paper presents an improved algorithm for constructing the city Voronoi diagram under the Euclidean metric given a transportation network consisting of roads having a constant number of speeds and orientations. The algorithm applies the continuous Dijkstra paradigm and its efficiency follows from new geometric insights that are first observed in this paper.

1 Introduction

The city Voronoi diagram represents a generalized Voronoi diagram in a modern city where several transportation networks such as bus networks, railroads, subway, or taxi systems are built and thus people can use them for faster movement.

In our sense, a *transportation network* models such a real transportation network. A transportation network is defined as a plane graph and each edge is called a *road*. We assume that along each road of the network, one moves at a certain fixed speed that is faster than out of the network, and he/she can access or leave the road at any point on it. In this situation, shortest (travel time) paths using the given network are of considerable interest, and so are Voronoi diagrams when we are given a set of equally attractive facilities. We thus address these proximity problems in the presence of roads.

Aichholzer et al. [1] first proposed the term, "the city Voronoi diagram"; in their results, they considered shortest paths using given roads under the L_1 metric and the induced metric by the given roads, called the *city metric*. The Voronoi diagram under such a city metric is called the *city Voronoi diagram*. For the city Voronoi diagram under the L_1 metric, there has been constant effort to find an optimal algorithm [1, 2, 5], and at last Bae et at. achieved it [4].

In this paper, however, we particularly consider the Euclidean metric, and investigate roads, shortest paths implied by the given roads, and the city Voronoi diagram under the Euclidean metric. The Euclidean city Voronoi diagram first appears in a Spanish paper by Hurtado et al. [6], although they did not use the term "city". Recently, Bae and Chwa [3] extends their results to more general cases but the solutions are far from the optimal ones.

This paper presents an improved algorithm that builds a Euclidean city Voronoi diagram and also a shortest path map (hereafter, we may call it SPM) in the presence of a transportation network which in particular consists of roads classified into a constant number of speed-orientation pairs. Indeed, we first obtain an algorithm for building a SPM for a single source by applying the continuous Dijkstra paradigm, and then extend to that for the Voronoi diagram. The following is our main result.

Theorem 1. *A shortest path map under the city metric induced by the Euclidean metric and n roads of k speed-orientation pairs can be constructed in $O(k^{5/2}n^{3/2+\epsilon})$ time and $O(k^2n)$ space. Also, the Euclidean city Voronoi diagram for m sites can be computed in $O(k^{5/2}(n+m)^{3/2+\epsilon})$ time and $O(k^2(n+m))$ space.*

If we regard k as a constant, for every $m \in o(n^2)$, this improves the previously best known algorithms by Bae and Chwa [3], where two different algorithms are presented; $O(nm + n^2 + m\log m)$ time with $O(n(m+n))$ space and $O(nm\log m + n^2\log n)$ time with $O(m+n)$ space.

As mentioned above, the city Voronoi diagram can be computed by using the algorithm for the SPM. Thus, we shall focus on the algorithm for constructing a SPM for a given source s in the body of this paper. In Section 2, we introduce some preliminaries related to our work and, in Section 3, we briefly describe our algorithm and then give an analysis in Section 4. Finally, we conclude in Section 5.

2 Preliminaries

Transportation Networks A *transportation network* on the Euclidean plane is represented as a planar straight-line graph $G(V, E)$ such that each edge $e \in E$ has its supporting *speed* $\nu(e) > 1$ and its orientation $\beta(e)$ with $0 \leq \beta(e) < \pi$. An edge in E is often called a *road* and a vertex in V a *node*. A transportation network together with its underlying metric induces a new metric, called a *transportation distance* or a *city metric*, which is defined as the shortest travel time between two points using G [2, 3].

In this paper, we are given a transportation network G under the Euclidean metric consisting of n roads of k speed-orientation pairs. We let d_2 be the Euclidean distance on the plane and d the city metric (or the transportation distance, interchangeably) induced by G and d_2. We shall denote $d(s, p)$ just by $d(p)$, throughout this paper.

Roads on the Euclidean Plane The most fundamental observation about roads on the Euclidean plane is given as follows.

Observation 2 (Bae and Chwa [3]). *To reach a destination as quickly as possible, in accessing a road $e \in E$ from a point p, the accessing angle from p to the road e must be $\pi/2 \pm \sin^{-1} \frac{1}{\nu(e)}$ if possible, otherwise, the access should be through the closer node incident to e from p.*

We let $\alpha(e) := \sin^{-1} \frac{1}{\nu(e)}$ for each road $e \in E$. Note that, by the above observation, unless we enter a road through its incident node, the entering direction is one of $4k$ predetermined directions, namely $D := \{\beta(e) \pm \pi/2 \pm \alpha(e) \mid e \in E\}$, since we have only k speed-orientation pairs for roads.

We call a path P is called *primitive*, if P contains no nodes in V except its endpoints and passes through at most one road. Also, we call a primitive path with shortest length *a shortest primitive path*.

Lemma 3 (Bae and Chwa [3]). *Given a transportation network G on the Euclidean plane, for two points $p, q \in \mathbb{R}^2$, there exists a shortest path P connecting p and q such that P is a sequence of shortest primitive paths whose endpoints are p, q, or nodes in V.*

This lemma is very helpful in devising efficient algorithms for the problem. If we fix the source s, turning points of shortest paths are well restricted. We define $F(v)$ for every $v \in V \cup \{s\}$ as follows: $p \in F(v)$ if and only if p is a point on a road that is first hit at p by a ray starting at v and following a direction in D. We let $V' := V \cup \{s\} \cup \bigcup_{v \in V \cup \{s\}} F(v)$ be the set of *vertices*. Then, the following lemma is implied based on the above observation and lemma.

Lemma 4. *For any point $t \in \mathbb{R}^2$, there exists a shortest path $\pi = (s = v_0, \cdots, v_k = t)$ connecting s and t using the transportation network G such that $v_i \in V'$ for $i = 0, 1, \cdots, k-1$.*

Also, we can easily compute V' by a standard plane sweep algorithm and point location queries in $O(kn \log n)$ time and $O(kn)$ space. [3, Section 6]

Needles A *needle* is a generalized Voronoi site proposed by Bae and Chwa [3] for easy explanation of geometry induced by transportation networks. A needle can be viewed as a generalized type of a line segment with additive weight; a needle, indeed, is a line segment with a weight function that is linear over all points along the segment.

More specifically, a needle p can be represented by a 4-tuple $(p_1(p), p_2(p), t_1(p), t_2(p))$ with $t_2(p) \geq t_1(p) \geq 0$, where $p_1(p), p_2(p)$ are two endpoints and $t_1(p), t_2(p)$ are additive weights of the two endpoints, respectively. For convenient references to a needle, we define some terms associated with a needle: We let $s(p)$ be the segment $\overline{p_1(p)p_2(p)}$ and $v(p)$ the *speed* of p defined as $d_2(p_2(p), p_1(p))/(t_2(p) - t_1(p))$. The L_1 distance from any point x to a needle p is measured as $d_2(x, p) = \min_{y \in s(p)}\{d_2(x, y) + w_p(y)\}$, where $w_p(y)$ is the weight assigned to y on $s(p)$, given as $w_p(y) = t_1(p) + d_2(y, p_1(p))/v(p)$, for all $y \in s(p)$.

It was shown that a SPM in our setting may have at most $O(kn)$ size, and further can be represented as a Voronoi diagram of $O(kn)$ needles under the Euclidean metric, which can be computed in optimal time and space [3].

Applying the Continuous Dijkstra Method We apply the continuous Dijkstra paradigm to obtain a shortest path tree (SPT, in short) rooted at the given source s. The framework of our algorithm is not so different from that of Mitchell [7] and that of Bae et al. [4] but most of specific details are quite distinguishable. The continuous Dijkstra method simulates the wavefront propagation by tracking effects of the *pseudo-wavefront* that is easy to maintain but sufficient to express the *true* wavefront.

In our case, the pseudo-wavefront is represented by a set of straight line segments and circular arcs, called *wavelets*. A wavelet ω is an arc of a "circle" centered at *root* $r(\omega)$.[1] Note that the root

[1] Here, a "circle" means the set of all the equidistant points from a given center, generally being a set of points or a needle.

$r(\omega)$ of a wavelet ω will be a needle along a certain road in our situation. Each wavelet ω has a left and a right *track*, denoted by $lt(\omega)$ and $rt(\omega)$, respectively, and expands with its endpoints sliding along its tracks. Each track is either a portion of a straight line or a portion of a bisecting curve between two certain roots. A bisector $B(r, r')$ between two roots r and r' under the Euclidean metric is a piecewise quadratic - it may contain parabolic and hyperbolic curves - and can be computed explicitly in constant time, even when the two roots are needles in general [3].

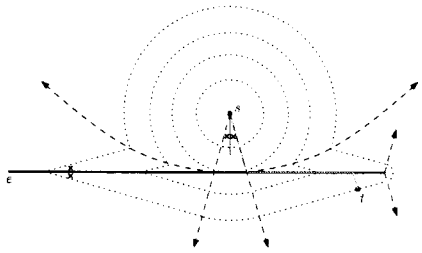

Figure 1: A road e and the wavefront propagating from s; dotted lines represent the wavefront of each radius and dashed arrows are tracks of each wavelet. The thick gray path shows the shortest path from s to t and the angles marked by $\times$ are exactly $\alpha(e)$.

On the Euclidean plane, the wavelets are basically circular arcs. However, a road e makes wavelets which are straight segments inclined at angles $\beta(e) \pm \alpha(e)$. Since we have k speed-orientation pairs for the roads, we have only $2k$ angles for the inclinations of straight wavelets. Tracks of straight wavelets are initially also straight and their directions are $\beta(e) \pm \pi/2 \pm \alpha(e) \in D$.

After running the continuous Dijkstra method, we obtain a SPT, a vertex-labeled tree rooted at the given source s such that every path to s through its vertices in the tree leads us to a shortest path with respect to the city metric d. We take V' as the set of vertices of our SPT. We observe that V' contains $O(kn)$ vertices and guarantees that we detect and handle combinatorial and geometric changes of wavelets by Lemma 4. Furthermore, we are able to determine a SPT uniquely with the vertices V'.

3 The Algorithm for the SPM

Our algorithm works with two steps: we compute a SPT rooted at s by applying the continuous Dijkstra method and then construct a shortest path map from the SPT. The framework of our algorithm is the same as in [4] which solves the city Voronoi diagram problem under the L_1 metric. Thus, we shall focus on how different our setting is from the L_1 case and how to handle such differences, rather than presenting a full description of the algorithm.

Each vertex $v \in V'$ has a label $\ell(v)$ and initially $\ell(v) = \infty$. After completing the continuous Dijkstra method, v will have a finite label $\ell(v) < \infty$ and its predecessor $pre(v)$ in the resulting SPT; indeed, at the end of the first step, $\ell(v)$ will become equal to $d(v)$, which is the length of the corresponding shortest path from s to v.

For that purpose, we maintain the set of wavelets and keep track of their effects by tracking *events* at every step. An event is associated with each wavelet and has its corresponding *event point* and *event distance*. We predict and handle two sorts of events during the algorithm: *Closure events* occur when two tracks of a wavelet meet at a point, namely the *closure point*, and the associated wavelet degenerates to the closure point. *Vertex events* occur when a wavelet, either its interior or its endpoint along a track, hits a vertex in V'.

We initially set the current event distance δ to zero and increase δ. As the current event distance δ increases, we keep track of *active* wavelets that represent the circle with radius δ centered at s. This can be done by maintaining following data structures:

- *Event Queue* $\mathcal{Q}$ is a priority queue containing active wavelets indexed by event distance. A wavelet ω is stored in $\mathcal{Q}$ with its left/right tracks $lt(\omega)$ and $rt(\omega)$, its root $r(\omega)$, its left/right neighbors $L(\omega)$ and $R(\omega)$, and its corresponding event.
- $SPM(\delta)$*-subdivision* is a partial subdivision of the plane. Each cell in $SPM(\delta)$-subdivision represents the locus of a wavelet until event distance δ from when it has been created or modified, and is either a pseudo-triangle or a pseudo-quadrilateral whose boundary may consist of quadratic curve segments such as parabolic and hyperbolic arcs. In particular, we allow $SPM(\delta)$ cells to overlap since two wavelets can collide with each other. We, however, fix up such unpleasant situations in a conservative way so that we will have no overlap among $SPM(\delta)$ cells at the end of the algorithm. A cell is called *open* if it consists of an active wavelet in its boundary or, otherwise, *closed*. Every open cell contains exactly one active wavelet in the pseudo-wavefront at distance δ.

We instantiate, modify, or terminate a wavelet when a certain event occurs. Performing such operations for an active wavelet is accompanied with updates for data structures such as the event queue $\mathcal{Q}$ and $SPM(\delta)$-subdivision. Every time we instantiate or modify a wavelet, we also do the following procedure; (1) we determine the corresponding event point and event distance, (2) insert the wavelet into $\mathcal{Q}$ or modify it in $\mathcal{Q}$, and (3) create a new corresponding cell in $SPM(\delta)$-subdivision when instantiating a new wavelet, or make the corresponding $SPM(\delta)$ cell closed and create a new one when modifying an existing wavelet (so that we maintain all the $SPM(\delta)$ cells to be pseudo-triangles or quadrilat-

erals). When we terminate a wavelet, we remove it from $\mathcal{Q}$ and set the corresponding cell in $SPM(\delta)$-subdivision to be closed by making up its boundary appropriately.

Determining the event point and the event distance for a wavelet is performed by computing the distance to its closure point (for a closure event), if any, and the distance to the first vertex $v \in V'$ that is encountered as the wavelet propagates (for a vertex event). The second one can be computed via a *wavelet dragging query*, which costs $O((kn)^{1/2+\epsilon})$ query time with $O(kn \log kn)$ preprocessing time and $O(kn)$ space, where $\epsilon > 0$ [7]. In fact, we need to modify the original structure for wavelet dragging queries but this adoption is not very difficult.

While such events completely check when a wavelet disappears or when a wavelet collides with a vertex, what remains difficult is detecting collisions among wavelets. We detect such collisions also by vertex events. When a vertex event occurs at $v \in V'$ due to wavelet ω with root $r = r(\omega)$, the label $\ell(v)$ is set to the current event distance δ if $\ell(v) = \infty$ yet. Otherwise, if $\ell(v) < \infty$ (in fact, $\ell(v) < \delta$), this means that another wavelet ω' has already hit the vertex v and hence two wavelets have been colliding with each other. This collision is also represented by an overlap, which has been swept over twice, in $SPM(\delta)$-subdivision. In order to fix up this kind of errors, we shall use subroutines *Clip-and-Merge* and *Trace-Bisector* when such collisions among wavelets (overlaps in $SPM(\delta)$-subdivision, equivalently) are detected by a vertex event. (Thus, we handle collisions among wavelets in a conservative way.) These subroutines are summarized as follows: The two wavelets ω and ω' with roots r and r', respectively, can be identified, and also the two corresponding $SPM(\delta)$ cells containing v. Consider walking along the path π from v to $p_1(r)$ that is implied by ω, and during this walk, we keep track of which cell we are in from the cell corresponding to root r'. We search a point q on π such that $d_2(q, r'') = d_2(q, r)$ where r'' is the root corresponding to a cell that is encountered during the walk and contains q. Note that such q is guaranteed to exist on π by the continuity of the function $f(x) := d_2(x, r)$ over all $x \in \pi$. In other words, q is a point on the bisector between two sets of roots corresponding to overlapped cells. *Clip-and-Merge* works with two phases: Find the point q as described above, and then run *Trace-Bisector* (as described below) with q as input to trace out the merge curve γ. *Trace-Bisector* runs with a point q which is assumed to be a point on the merge curve γ, and traces γ with q as a starting point as is done in merging two Voronoi diagrams. We first walk along $B(r, r'')$, which is a portion of γ, in both directions out of q, and shift one of the two roots at the boundary of a corresponding $SPM(\delta)$ cell to correctly follow portions of bisectors. During this walk, we terminate wavelets that we have walked across, and we stop walking at point z that is a crossing point among two wavelets and the merge curve γ, replacing the appropriate left/right tracks of the two wavelets by the bisector between their roots.

A brief description of our algorithm is stated as follows: First, we initialize structures we need, and instantiate four zero-length circular wavelets along 4 vertical or horizontal tracks from s that have root s (or, equivalently, needle $(s, s, 0, 0)$). While $\mathcal{Q}$ is not empty, we extract the upcoming event from the front of $\mathcal{Q}$ and do a proper procedure according to its type. This process needs geometric observations about roads under the Euclidean metric but such observations are very natural and easy to see. Especially, if the present event is a vertex event on $v \in V'$ and its label $\ell(v)$ has been already labeled before, then we run *Clip-and-Merge* to fix up the collision between the corresponding wavelets. Once this main loop has ended, we obtain the SPT rooted at s which consists of labeled vertices and directed links among them. From this information,we gather the set of needles (or roots), $\mathcal{N} = \bigcup_{v \in V'} R_v$. Note that the Voronoi diagram $\mathcal{V}(\mathcal{N})$ under the Euclidean metric coincides with a SPM for the source s. This has been already argued in earlier results [3]. Consequently, we can build a SPM from the resulting SPT in $O(|\mathcal{N}| \log |\mathcal{N}|)$ time and $O(|\mathcal{N}|)$ space. For more details, we refer to Bae et al. [4].

4 Analysis

Now, we show the correctness of the algorithm described in the previous section, and then bound its runtime.

Theorem 5. *The algorithm correctly constructs the SPT. In other words, if the event distance is $\delta > 0$ during the algorithm, for all vertices v with $d(v) < \delta$, v has been correctly labeled with $\ell(v) = d(v) = d(s, v)$.*

Proof. The proof is by induction on the number of vertices that have been labeled. If the labeled vertex is only one, s, the theorem holds. Suppose that k vertices have been correctly labeled, and let v be the next vertex which will be labeled (with $\ell(v) = \delta$) by collision of a wavelet ω rooted at $r = r(\omega)$. Then, we have a path to v from s following the current (partial) SPT so that the length of the path is $\delta = d_2(v, r) = d_2(v, p_1(r)) + \ell(p_1(r))$ by the construction of V'. Further, the inductive hypothesis gives $\delta = d_2(v, p_1(r)) + d(p_1(r)) \leq d(v)$.

Assume now that strictly $d(v) < \delta$. We claim that any point $p \in \mathbb{R}^2$ with $d(p) < \delta$ has been swept over at least once when the pseudo-wavefront reaches distance δ. This statement immediately follows from Lemma 4 and the fact that process for collisions among wavelets is done conservatively. Thus, v should have been swept over at least once by another wavelet, and thus a vertex event on v should have been processed before ω hit v. This contradicts to our choice of v, and hence implies that v will be correctly labeled

with $\ell(v) = d(v) = \delta$.

We now discuss the complexity of the algorithm. In doing so, we investigate nearest neighbor graphs for vertices under the city metric d. The *nearest neighbor graph* of a set P of points under a metric is built as follows: For each point $p \in P$, there is a directed edge to point $q \in P$ such that q is the nearest neighbor of p with respect to the given metric among the points P.

Here, we give an upper bound on the maximum in-degree, say Δ, of the nearest neighbor graph for $V' \cup \{p\}$ for any point p in the plane. By the construction of V', it is revealed in Lemma 7 that Δ is surprisingly bounded by a constant. This observation will be very helpful in showing the efficiency of the algorithm through the following sequence of lemmas and corollaries. This process starts with the following simple fact.

Fact 6. *Let P be a set of points in the plane. The nearest neighbor graph for P under the Euclidean metric has maximum in-degree of at most 6.*

Proof. Pick a point $p \in P$. Let N_p be the set of points in P whose nearest neighbor is p. Then, we can find a disk D_q centered at each $q \in N_p$ such that D contains p but any other points in N_p. We just fix D_q as the disk centered at q whose radius is $d_2(q,p)$ so that D_q touches p and contains no other points $q' \in N_p$ in its interior.

We now subdivide the plane into six unbounded regions by cutting the plane along three lines which pass through p and have slopes of 0 and $\pm\sqrt{3}$, respectively. For more rigorous discussion, let us say that, each region includes one ray as its boundary but not two; in fact, the boundary of each region is two rays starting at p. Assume that two points $q, r \in N_p$ lie in such a region. Since $q, r \in N_p$, we assert that $d_2(q,p) \leq d_2(q,r)$ and $d_2(r,p) \leq d_2(q,r)$. However, we easily observe that D_r contains q in its interior if $d_2(r,p) \geq d_2(q,p)$, since the angle $\angle qpr$ is strictly smaller than 60°. Thus, no two points in N_p lie in the same region and hence $|N_p| \leq 6$.

Lemma 7. *For any point p in the plane, the in-degree of p in the nearest neighbor graph for $V' \cup \{p\}$ is at most a constant.*

Proof. We first observe a good property of nearest neighbors graph among $V' \cup \{p\}$: Consider the shortest path π from any vertex $v \in V'$ to p. By the construction of V', if π goes through a road that is not incident to v, then it must pass through another vertex $v' \in V'$, which implies that p cannot be the nearest neighbor of v since v' is closer to v along π than p is. Thus, any vertex $v \in N_p$ approaches p either by the direct path or by a path using only one road incident to v if any. Also, D_v is either just a Euclidean disk or a union of at most $2k$ needle shapes in the sense of Bae and Chwa [3].

Next, we consider a subset D'_v of D_v defined as follows: D'_v is the Euclidean disk centered at v with radius $d_2(v,p)$ if the path from v to p is not using any road. Otherwise, if the path is using one road e incident to v, D'_v is the disk centered at v as if e is the only road incident to v. Note that D'_v still contains p, and that since D_v does not contain any other vertices $v' \in N_p$ in its interior, neither does D'_v.

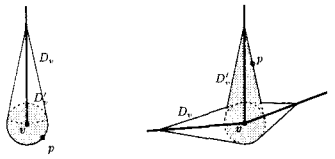

Figure 2: D_v (regions bounded by solid thin segments) and D'_v (gray regions) for $v \in N_p$.

Now, the problem is switched to how many such D'_v's can be there with each containing no other vertices $v' \in N_p$ than v. By Fact 6, we know that there are at most 6 D'_v's whose shapes are Euclidean disks. For needle shapes, we fix one of k speed-orientation pairs and let $e \in E$ be a road falling into the case. Then, the boundaries of needle shapes produced by roads of the same type have straight segments whose inclinations are $\beta(e) \pm \alpha(e)$. Thus, we subdivide the plane into four unbounded regions by two lines which pass through p and have slopes $\beta(e) \pm \alpha(e)$. Then, we can easily see that no two vertices in N_p that make such needle shapes cannot lie in the same region, and hence there are at most 4 D'_v whose shapes are needle shapes of a certain speed-orientation type. Since we have k speed-orientation pairs for roads, we conclude that the in-degree of p is at most a $4k + 6$. □

Almost the same proof shows a bit more extended fact, which helps to show Lemma 9.

Corollary 8. *Let P be a set of points and $N_p \subseteq P$ be the set of points q such that the nearest neighbor of q is p over all points in P. Then, $|N_p| \leq 4k + 6$ if the following condition holds for every $q \in N_p$; the shortest path from q to p is either (1) the direct path or (2) using a single road incident to q.*

Lemma 9. *Any point can be swept over at most $4k + 6$ times by the pseudo-wavefront during the algorithm.*

Proof. Suppose that a point p has been swept over K times with $K > 4k + 6$ and R is the set of K roots whose wavelets have swept over p. Let $R' := \{p_1(r) | r \in R\}$. Note that $d_2(p,r) = d_2(p,p_r) + d_2(p_r, p_1(r))/\nu(e) + d(p_1(r))$ for any root $r \in R$ on road e, where p_r denotes the point on the segment of r that leads p to the shortest path to r. The corresponding path from p to r (or to $p_1(r)$) is either just the direct path or using a single road incident

to $p_1(r)$. Thus, R' and p fulfill the condition of Corollary 8, which implies that if all the points in R' have p as their nearest neighbor, obviously $|R'| \leq 4k + 6$.

However, since $K > 4k + 6$, there should exist two points $q := p_1(r), q' := p_1(r') \in R'$ such that $d_2(p, r) > d_2(q', r)$. Thus, before a wavelet ω rooted at r hits p, another one ω'' rooted at r must hit q'. We have two possibilities; either ω'' is the first wavelet that hits q' or not. If this is the former case, $SPM(\delta)$ cells associated with wavelets rooted at r or r' cannot overlap by handling a vertex event at q'. If this is the latter case, $d(q) + d(q, q') > d(q')$ and q' should have been labeled before q' is swept over by ω''. Thus, ω'' causes a clipping at q' before ω reaches p. Consequently, in either case, not both wavelets from q and q' can sweep over p. This contradicts the assumption that $K > 4k + 6$ and hence the lemma is shown.

Lemma 10. *The algorithm processes at most $O(k^2n)$ events.*

Proof. By Lemma 9, each vertex $v \in V'$ can be swept over at most $O(k)$ times. Since every collision between a wavelet and a vertex causes a vertex event, we can bound the number of vertex events by $O(k^2n)$. Moreover, since wavelets get instantiated only at a vertex event except for the beginning of the algorithm and only a constant number of new wavelets get instantiated at each vertex event, only $O(k^2n)$ number of wavelets get instantiated. Since only wavelets having been instantiated can be terminated, the number of closure events does not exceed the number of vertex events.

Lemma 11. *The total number of $SPM(\delta)$ cells created during the algorithm is at most $O(k^2n)$.*

Proof. We create an $SPM(\delta)$ cell only when we create a new wavelet or modify an existing one. Only at each event, we create or modify a constant number of wavelets. Thus, by Lemma 10, the lemma is shown.

Lemma 12. *The overall time spent by the subroutines* Clip-and-Merge *and* Trace-Bisector *during the algorithm is $O(k^3n)$.*

Proof. Lemma 9 implies that any $SPM(\delta)$ cell can be clipped in the subroutine *Trace-Bisector* and crossed while walking the path from v to $p_1(r)$ in the first phase of *Clip-and-Merge* at most $O(k)$ times in total. By Lemma 11, we just can bound the overall time consumed in the subroutines *Clip-and-Merge* and *Trace-Bisector* only by $O(k^3n)$

Each event, before occurring, may involve either a wavelet dragging query, which costs $O((kn)^{1/2+\epsilon})$ time for each. Also, $|\mathcal{N}| = O(k^2n)$ since each vertex in V' has $O(k)$ incident roads. Thus, the total time complexity is $O(kn \log kn + (k^2n)(kn)^{1/2+\epsilon} + k^3n + k^2n \log kn)$, simplified to $O(k^{5/2}n^{3/2+\epsilon})$ since $k \leq n$. Moreover, the city Voronoi diagram for m sites can be obtained just by initializing the event queue to consist of the initial wavelets for each input site and regarding each site as a vertex. Finally, we conclude the main theorem stated at the beginning.

5 Concluding Remarks

The resulting map or diagram has information of shortest paths to the given source or to the nearest site, and of their lengths. With this information, we are able to get the length of the shortest path from a query point in logarithmic time, or the path itself in additional time proportional to the complexity of the path, with the aid of a proper point location structure.

Our algorithm can be easily extended to more general situations. An interesting extension is to composite (geodesic) metric spaces by roads together with obstacles; only by a little modification on our algorithm for processing obstacle vertices, as described in [7]. Subsequently, we can construct the SPM and the Voronoi diagram in the same time and space bounds, letting n be the number of all the endpoints introduced by the roads and the obstacles.

References

[1] 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.

[2] S. W. Bae and K.-Y. Chwa. Shortest paths and Voronoi diagrams with transportation networks under general distances. In *Proc. 16th Annu. Internat. Sympos. Algorithms Comput.*, volume 3827 of *LNCS*, pages 1007–1018, 2005.

[3] S. W. Bae and K.-Y. Chwa. Voronoi diagrams for a transportation network on the euclidean plane. *Internat. J. Comp. Geom. Appl.*, 16(2–3):117–144, 2006.

[4] S. W. Bae, J.-H. Kim, and K.-Y. Chwa. Optimal construction of the city Voronoi diagram. In *Proc. 17th Annu. Internat. Sympos. Algorithms Comput.*, to appear.

[5] R. Görke and A. Wolff. Computing the city Voronoi diagram faster. In *Proc. 21st Euro. Workshop on Comput. Geom.*, pages 155–158, 2005.

[6] F. Hurtado, B. Palop, and V. Sacristán. Diagramas de voronoi con funciones temporales. *VIII Encuetos en Geometria Computacional*, 1999.

[7] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *Internat. J. Comput. Geom. Appl.*, 6(3):309–331, 1996.