Constructing the City Voronoi Diagram Faster

Robert Görke*
Universität Karlsruhe
http://i11www.ira.uka.de/algo/group

Alexander Wolff†
Universität Karlsruhe
http://i11www.ira.uka.de/algo/group

Abstract

Given a set S of n point sites in the plane, the City Voronoi diagram partitions the plane into the Voronoi regions of the sites, with respect to the City metric. This metric is induced by quickest paths according to the Manhattan metric and an accelerating transportation network that consists of c non-intersecting axis-parallel line segments. We describe an algorithm that constructs the City Voronoi diagram (including quickest path information) in $O((c+n)\text{polylog}(c+n))$ time using a wavefront expansion. For $c \in \Omega(\sqrt{n}\log^3 n)$ our algorithm is faster than an algorithm by Aichholzer et al., which takes $O(n \log n + c^2 \log c)$ time.

Key words: wavefront expansion, City Voronoi diagram, straight skeleton, closest pair, minimization query, transportation network

1. Introduction

Imagine Manhattan in 2050—void of car traffic. Only a network of conveyors accelerates the movement of countless busy visitors in this huge pedestrian zone. As is known streets are arranged isothetically in Manhattan, so given a general direction, pedestrians can intuitively find a footpath to one of the many post offices. But time is precious, and thus a technique is required, telling an arbitrary pedestrian the quickest path to the post office that can be reached most quickly. Detours utilizing the transportation network should be accepted if they help to save time. A courier service with several staging posts faces a similar problem. For any incoming job it has to be determined how and starting from which post the pickup point can be reached most quickly.

We concretize the situation as follows. We are given a transportation network $C = \{e_1, \ldots, e_c\}$ which consists of c isothetic line segments that are only allowed to touch and n point sites $S = \{\omega_1, \ldots, \omega_n\}$ in the plane. Movement off the network takes place with unit speed with respect to the Manhattan metric, while a segment e_i can be used to move with some speed $g_i > 1$ into either direction. We require that the number of different speeds is constant. A segment can be accessed and left at any point. The Manhattan metric, or L_1-metric, is induced by the distance $d_{\text{Manhattan}}$ of two points $a = (x_a, y_a)$ and $b = (x_b, y_b)$ in the plane being defined as $d_{\text{Manhattan}}(a,b) = |a_x - b_x| + |a_y - b_y|$. In this paper we use the following notion of distance based on the Manhattan metric. If we measure the length of a path Π as

$$d(\Pi) = d_{\text{Manhattan}}(\Pi \setminus C) + \sum_{e_i \in C} \left(\frac{d_{\text{Manhattan}}(e_i \cap \Pi)}{g_i} \right),$$

we can define the distance d between two points a and b in the plane to be the temporal length of the quickest path Π between them:

$$d(a,b) = \min_{\Pi \text{ isothetic } a\text{-}b \text{ path}} d(\Pi).$$

The definition of quickest paths induces a metric in the plane that we call *City metric*. As usual we define the *Voronoi region* $\text{reg}(\omega_i)$ of a site ω_i as the set of all points that are not closer to any other site ω_j. If we associate borders between regions in an arbitrary manner with one of the involved sites, we get a partition of the plane called the *City Voronoi diagram* $V_C(S)$. Given a query point $q \in \mathbb{R}^2$, the site in S closest to q can be determined by point location in time logarithmic in the complexity of $V_C(S)$. By virtue of our construction method we obtain a *refinement* $\mathcal{V}_C(S)$ [3] of the City Voronoi

*Supported by the European Commission within FET Open Projects DELIS.

†Supported by grant WO 758/4-2 of the German Science Foundation (DFG).

diagram that can then also report the quickest path to the closest site in additional time $O(L)$, with L being the path complexity.

The technique we present for the construction of the City Voronoi diagram can be viewed as an example of a more general approach to solving geometric problems. In a setting that involves objects with a high complexity we often have to realize the following three concurrent requirements on our data. First, the objects need to be simplified in a way such that they allow fast handling and processing. This is best accomplished by guaranteeing constant complexity of data objects. Second, this data simplification must not result in a substantial increase in the number of objects. And third, the simplified objects must help to solve the problem on the original data efficiently. Our refinement of the City Voronoi diagram meets these three requirements, as we shall see.

The *trapezoidal decomposition* is a well-known method of answering point-location queries in the plane that also follows the stated paradigm. A given planar subdivision is augmented by drawing vertical extensions through all vertices. The extensions stop when they meet another edge of the subdivision. Again, this yields a refined planar subdivision with simplified objects. The search structure for this subdivision, a tree, is built by a randomized incremental algorithm. Due to the simple shapes of the trapezoidal regions, the search tree can be built efficiently. The refinement does not increase the complexity of the subdivision asymptotically. And finally, a query point can be located efficiently in the original subdivision via the trapezoidal regions.

This paper is structured as follows. In Section 2 we go through the previous work on City Voronoi diagrams. In Section 3 we analyze the mechanics of the wavefront expansion. Here we also determine the complexity of the diagram and present a lifting into 3-space, where the additional dimension represents the elapsed time. This helps us to apply orthogonal range queries for predicting the next change in the shape of the wavefront. In Section 4 we describe our main contribution, an algorithm that efficiently maintains the shape of the wavefront during the expansion. In Section 5 we put things together. This yields the overall result, the construction of the City Voronoi diagram in time $O((c+n)\text{polylog}(c+n))$. We conclude the paper with a short discussion and an outlook in Section 6. To give an impression of the City Voronoi diagram of a moderately complicated transportation network and several sites, we provide a large example with Figure 16, at the end of this paper.

2. Previous work

The City metric was introduced by Abellanes et al. [1] who derived basic properties of quickest-path metrics. Moreover they gave an $O(n \log n)$-time construction algorithm for the City Voronoi diagram $V_C(S)$ for the special case that the transportation network is a single straight line. Hurtado et al. [8] discuss some results for single-line transportation networks under the Euclidean metric. Based on the concept of weighted regions, introduced by Mitchell and Papadimitriou [10], Gewali et al. [7] studied a special case that connects to the City Voronoi diagram. The segments of a given transportation network C can be viewed as one-dimensional instances of weighted regions. The authors construct the quickest path between two points in time $O(c^2)$. Another variant of our setting is the airlift Voronoi diagram, which restricts access to the network to a set of stations. Recently Ostrovsky-Berman [11] presented the first time-optimal algorithm for airlift Voronoi diagrams, running in time $O((n + s)\log(n + s) + c)$ with s being the number of stations. Aichholzer et al. [3] presented an algorithm that constructs the City Voronoi diagram of n sites and c segments given a uniform network speed in $O(n \log n + c^2 \log c)$ time using $O(c + n)$ space. The resulting data structure, the refined City Voronoi diagram $\mathcal{V}_C(S)$, answers quickest-path queries in $O(L + \log(c + n))$ time. In their algorithm the authors first prepare a set of time-stamped nodes in the grid induced by the segments using the continuous Dijkstra method [9]. Then carefully adapted straight skeleton figures scheduled at these nodes are computed by employing techniques for the construction of abstract Voronoi diagrams. Recently Bae and Chwa [4] presented an algorithm that establishes a City Voronoi diagram in the Euclidean plane, admitting arbitrary orientation and speed of network segments. Their technique, being similar to the approach of Aichholzer et al. [3], requires time $O(nc^2 \log n + c^3 log c)$ and space $O(c(c + n))$.

The two fundamental techniques used in this

paper, namely the expansion of a wavefront and the maintenance of closest pairs in dynamic sets, have been employed before, e.g. by Mitchell et al. [9] for solving the discrete geodesic problem and by Agarwal et al. [2] for collision detection in kinetic data structures.

3. The wavefront expansion

Our algorithm constructs the City Voronoi diagram $V_C(S)$ by simulating the expansion of a wavefront starting at time $t_0 = 0$ at the set S of sites. At time $t \geq 0$ the wavefront is the set of all points whose distance from S is t in the City metric. The key observation is that during the course of the expansion each point of the plane is reached by the quickest possible path starting from S. In order to tell the quickest path from p to S we therefore need to store information about how the wavefront reached p and invert the path taken by the wavefront. This is done as follows. By storing where the wavefronts of different sites merge and by tracing vertices resulting from such mergings, we immediately obtain a partition of the plane. The borders of this partition consist of all points that can be reached equally quickly from at least two sites, thus the partition obtained is the City Voronoi diagram $V_C(S)$, see Figure 1. We can trace the path of other wavefront vertices in order to obtain a refinement of $V_C(S)$. Since the wavefront consists exclusively of vertices and straight line segments (due to the properties of the City metric), this refined City Voronoi diagram $\mathcal{V}_C(S)$ partitions $V_C(S)$ into regions of uniform wavefront expansion. Thus, if we store for each such region the direction in which the wavefront swept over the region, we can tell for all points of that region how to reach the *oldest* object of this region. This oldest object can either be a vertex or a line segment, being the part of the region that was reached first by the wavefront. By doing this repeatedly, we ultimately reach S, tracing back the expansion of the wavefront. See Figure 2 for an example. We discretize the continuous expansion of the wavefront at the points in time when a collision between the wavefront and the network or between two parts of the wavefront happens. We call each of these points in time *events*. At an event the combinatorial shape of the wavefront changes. An event is a pair of a timestamp and a locus in the plane, which is either a point or a line segment.

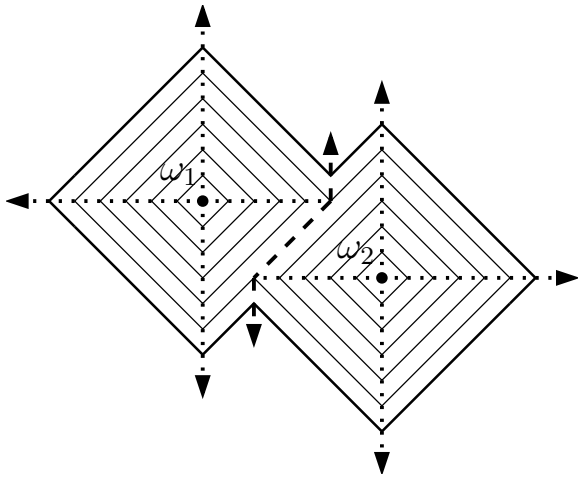

Figure 1. The wavefronts of two sites merge, tracing out a border (dashed).

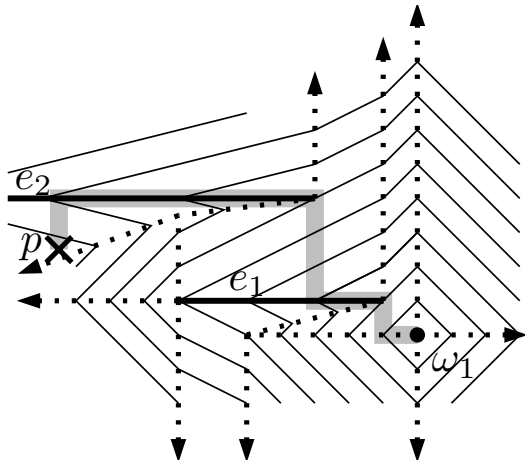

Figure 2. The expansion of the wavefront guides the way from p back to ω_1.

3.1. Events

We distinguish four types of events, depending on the situation. A vertex of the wavefront hitting a segment generates a type-A event, while an edge of the wavefront sliding into a network node triggers a type-B event. A type-C event occurs when a wavefront edge shrinks to zero length and finally, a type-D event is a collision of two parts of the wavefront. See Figures 3 to 6 for examples. It is not hard to see the following:

Observation 1. *For any type of event the number of changes in the wavefront is constant.*

As a consequence of this observation, we only need to focus on the detection of events. An up-

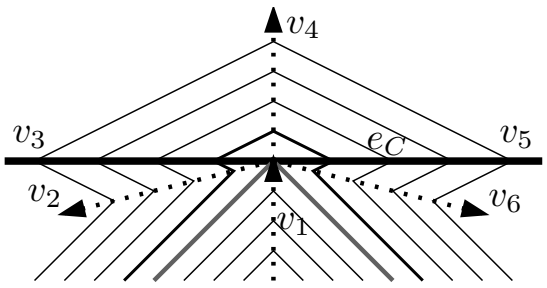

Figure 3. v_1 hits e in a type-A event.

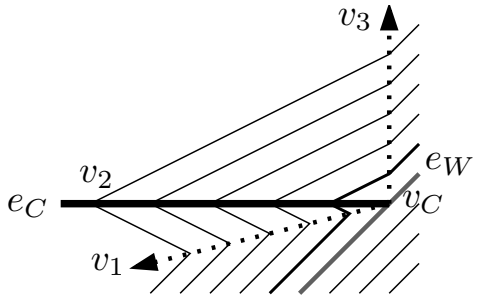

Figure 4. e_W slides into v_C in a type-B event.

coming event can be detected by comparing for all edges and vertices of the wavefront the timestamp of their next collision. This comparison leads us to the notion of *virtual* events. A virtual event is an event that seems likely to occur during the wavefront expansion, but is then prevented by some other event that happens earlier, see Figures 7 and 8 for an example. We can even detect events that do happen, but still do not contribute to the complexity of $V_C(S)$. We call such events *redundant*, see Figure 9 for an example. Events that are neither redundant nor virtual are *relevant* and take part in shaping $V_C(S)$. Next we discuss an important result about the total number of relevant events.

3.2. The linear complexity

Adapting a result of Aichholzer et al. [3] to a constant number of network speeds we get the following result:

Theorem 1. *The number of relevant events and the complexity of the refined City Voronoi diagram $\mathcal{V}_C(S)$ is $O(n+c)$.*

Opposed to that, the number of redundant events can amount to $\Theta(c(c+n))$ (see Figure 9), and the number of virtual events can even add up to $\Theta((c+n)^2)$. While these events are easy to identify, we cannot treat them explicitly. Thus

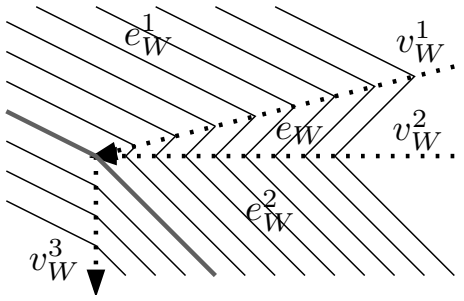

Figure 5. e_W causes a type-C event.

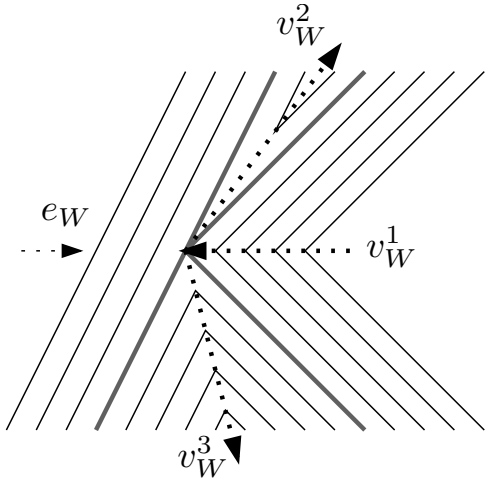

Figure 6. v_W^1 and e_W cause a type-D event.

we are left with the task of efficiently detecting the next event while implicitly ignoring irrelevant ones. In the next subsection we consider a unifying approach for detecting all four types of events.

3.3. The wavefront in space

We now add a third dimension (z-axis) to our view of the wavefront expansion, such that a positive z-component represents the time that passed since the start of the wavefront in the x-y-plane. Consequently wavefront vertices and edges trace out rays and polygons, respectively. Note that once the wavefront has reached the last event, the orthogonal projection of these polygons onto the plane yields the regions of the refined City Voronoi diagram. In a similar fashion we extend all network segments to vertically unbounded rectangles and all network nodes to vertical half-lines. If the z-component of the wavefront expansion has unit speed we can easily tell the timestamp of an event by its z-coordinate in space. Figure 10 shows how the z-axis is added. Analyzing each type of event

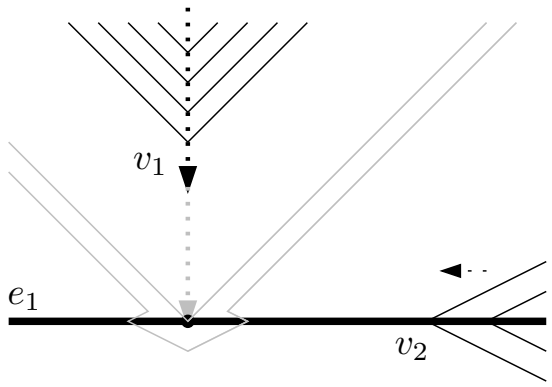

Figure 7. A type-A event is pending.

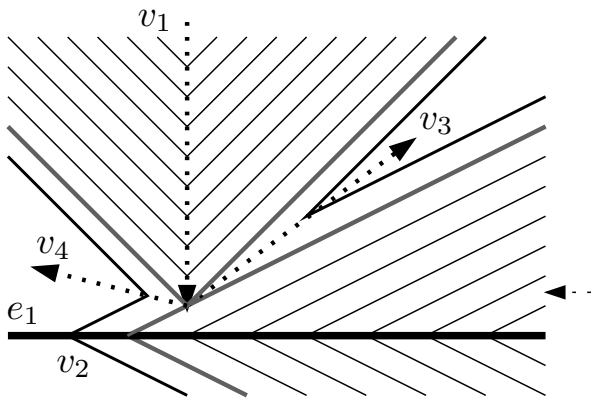

Figure 8. The event has been prevented.

in space, we observe the following:

Observation 2. *In space any event can be described as a collision between a ray and a polygon.*

Such collisions can be computed using *ray-shooting* techniques, but general methods for ray shooting have unsatisfactory time bounds. Furthermore we still need to take care of redundant and virtual events. The next section describes how we can efficiently detect upcoming events.

4. Maintaining the next event

Since each event is a collision of a ray and a polygon we can always determine the next event by maintaining the closest pair between these two dynamic sets of objects (polygons and rays). If we can do this quickly and repeatedly, we can efficiently simulate the expansion of the wavefront. In the following we will present a hierarchy of event-prediction mechanisms, culminating in the prediction of the next event of the expansion.

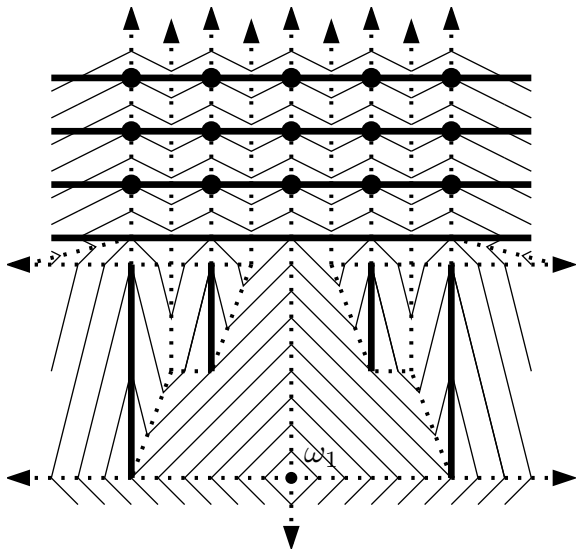

Figure 9. Cascade of redundant type-A events (marked by large disks).

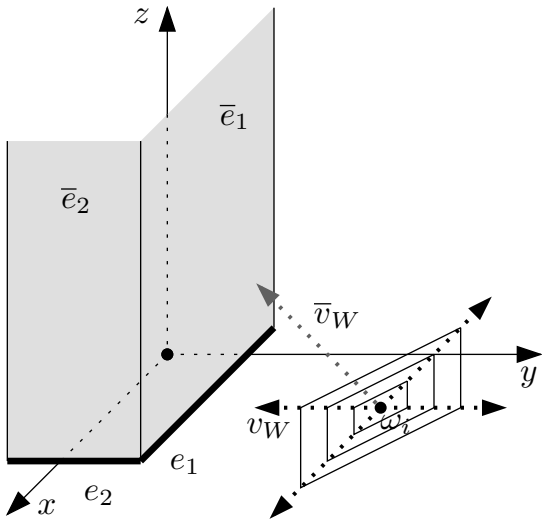

Figure 10. A type-A event in space involving $\overline{v}_W$ and $\overline{e}_1$ is imminent.

4.1. The global prediction

Eppstein and Erickson [6] proposed a method of maintaining the closest pair among two changing sets R and B of objects according to a given distance measure d that can be computed in constant time. As a prerequisite the sets R and B need to support *minimization queries*, i.e. for any object $b \in B$ an object $r \in R$ minimizing $d(r, b)$ can be determined and vice versa. In our application R and B will be partially unbounded polygons and foot points of rays, respectively. We use the following result:

Theorem 2 ([6]). *Suppose that after $P(n)$ pre-*

processing time, we can maintain a data structure of size $S(n)$ that supports insertions, deletions, and minimization queries, each in amortized time $T(n)$. Then after $O(P(n) + nT(n))$ preprocessing time, we can maintain the closest pair between R and B in $O(S(n))$ space, $O(T(n)\log(n))$ amortized insertion time, and $O(T(n)\log^2(n))$ amortized deletion time.

Employing this theorem we are left with the lesser problem of efficiently performing minimization queries. If we confer the requirements of the theorem to our situation we need to determine for any given ray (i.e. vertex) the region it hits next and the inverse, for any given region the ray that hits next. These two well-known types of queries are ray-shooting queries and *lowest-intersection* queries, respectively. Let us call the results of such queries *local* predictions and the result of the above theorem the *global* prediction. We now face the challenge of simplifying our data as to speed up minimization queries while taking implicit care of irrelevant events.

4.2. Simplification of wavefront data

The data we deal with for the purpose of local predictions comprises rather complicated, potentially unbounded polygons in space. Recall that the orthogonal projection of these polygons onto the plane yields the regions of the refined City Voronoi diagram. If we split these regions along the current wavefront as depicted in Figure 11, namely each time the region hits the locus of an event, we obtain *slabs* that are possibly unbounded triangles or quadrilaterals, both in two and three dimensions. We distinguish four types of slabs depending on the number of bounded edges and the presence of parallel edges. As long as a slab has not yet been involved in an event (except for the one that created the slab) we call it *active*. Analogously we define active rays. If an active slab is involved in a second event it becomes inactive and changes its shape. The new slab is a subset of the previous and is still contained in the corresponding region of the refined City Voronoi diagram. Inactive slabs cannot take part in a relevant event. They have already been swept over by the wavefront and thus we exclude them from further event detection. By Theorem 1 the number of events that occur during the wavefront expansion is linear in $c+n$. Since by Observation 1 each

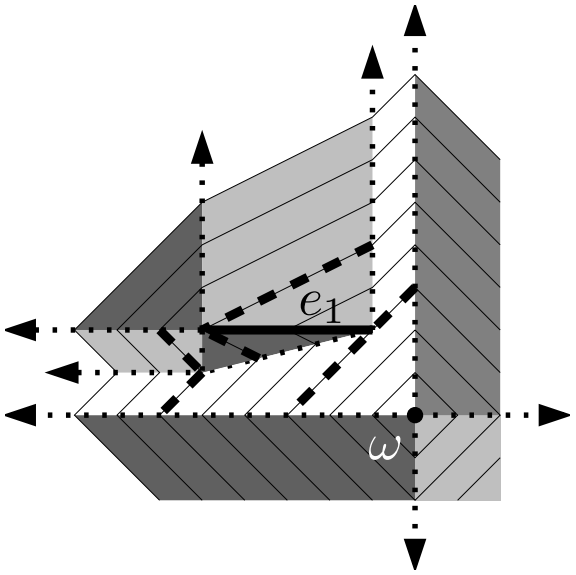

Figure 11. Division (dashed) of regions (shaded) into slabs.

event causes a constant number of changes in the wavefront, we obtain the following corollary:

Corollary 1. *Subdividing the refined City Voronoi diagram $\mathcal{V}_C(S)$ into slabs yields a partition of complexity $O(c + n)$.*

We are now left with answering minimization queries for a linear number of triangles and unbounded quadrilaterals. Note that active slabs cover areas beyond the current wavefront. The key observation is that we do not need to know exactly how far a slab has actually been traced out by the wavefront. The slabs have been designed to cover only those points of the plane that they would cover in the finished diagram, if they are not made inactive prematurely by some event involving them. The same holds for rays.

4.3. Orthogonalized sublocal queries

We now define slabs to be *similar* if their sides pairwise have the same angular position. For an example see Figure 12. Rays are similar if they merely point in the same direction and move at the same speed. These definitions at hand, the following holds for classes of similarity of slabs and of rays:

Lemma 1. *The number of classes of similarity of slabs and of rays is constant.*

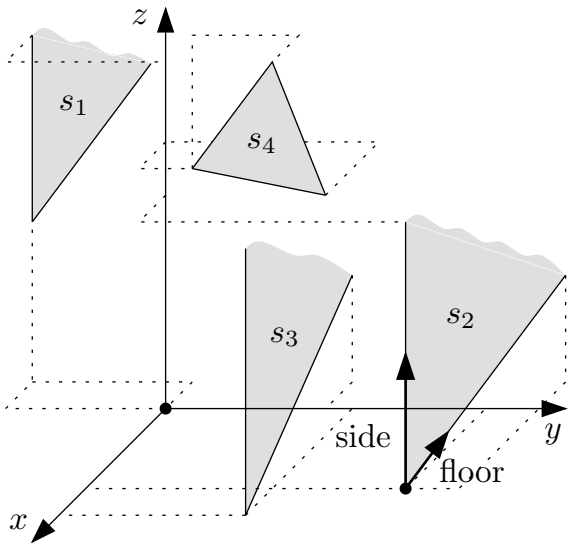

Figure 12. Only slabs s_1 and s_2 are similar.

The proof of Lemma 1 builds upon the fact that the expansion of a wavefront edge can only be accelerated, i.e. altered from the simple Manhattan metric expansion, by a single network segment. Since the number of different speeds and angular positions of segments is constant, the wavefront can expand only with a constant number of orientations and speeds. Since vertices of the wavefront are the intersection points of two edges, the same holds for wavefront vertices.

Let us now consider an arbitrary combination of one class of slabs with one class of rays. We call the result of a minimization query involving all objects of exactly these two classes a *sublocal* prediction. Since by Lemma 1 the number of ray and slab classes is constant, the number of pairs of ray and slab classes is constant, too. Therefore, any local prediction can easily be computed from sublocal predictions in constant time. Within a sublocal data structure considerable simplifications are possible. For each such data structure we can define a coordinate transformation f consisting of at most one rotation and four concatenated shearings. First the rotation aligns the rays with the z-axis and one side of the slabs with the x-axis. Then step by step each side of the slabs is orthogonalized to two of the three axes. As shown in Figure 13, we end up with simple orthogonal range queries instead of ray-shooting or lowest-intersection queries. Note that in order to handle type-3 slabs (bounded triangles) we need to introduce the additional Ψ-axis (see Figure 14), adding one more level to the range-searching data structure.

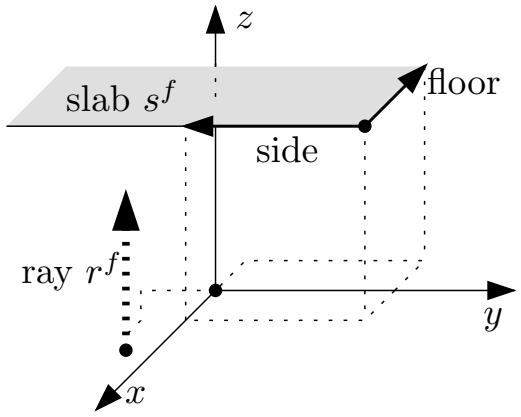

Figure 13. A transformed slab-ray pair.

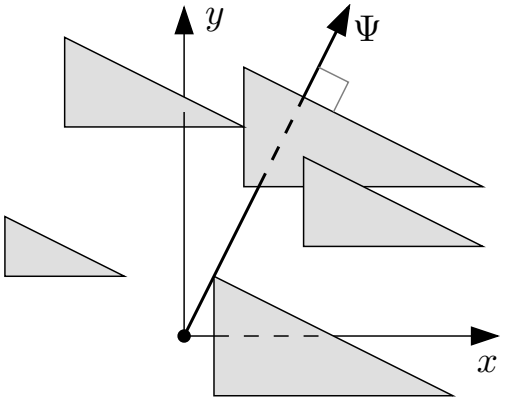

Figure 14. The additional Ψ-axis.

4.4. Feeding the global prediction

As stated earlier the global prediction relies on local predictions. Local predictions in turn are based on a constant number of sublocal queries, each being answered with a multi-dimensional orthogonal range query. The general picture of our algorithm is given in Figure 15. Making use of Theorem 1 and of well-known results about multi-level range trees and fractional cascading (see e.g. [5]) we can state the following:

Observation 3. *After preprocessing our sublocal data structures in time* $O((n+c)\log^3(n+c))$*, each can each handle insertions, deletions and queries in* $O(\log^3(c+n)\log\log(c+n))$ *time using* $O((c+n)\log^3(c+n))$ *space. The same holds for local data structures.*

Comparing slabs with rays we observe that while slabs are static, rays are not and thus a ray cannot simply be represented by its static foot point

p_{foot}. In order to do justice to the dynamic nature of rays we should in fact use the (moving) tip of the rays. However, instead of repeatedly advancing the tips of all rays we can simply apply a time-correction when inserting the foot points into our lowest-intersection data structures and querying our ray-shooting data structures: We set $p_{\text{foot}}^{\text{new}} := p_{\text{foot}} - (0,0,t)|\vec{v}|$, with t being the elapsed time since the start of the wavefront at $t_0 = 0$ and $\vec{v}$ being the direction of the ray. They key observation is that inside each individual sublocal data structure these modified foot points represent at all times the relative position of the tips of their rays, once the transformation f has been applied.

4.5. Ignoring redundant and virtual events

While the statement of Observation 3 is crucial to relevant events, we can show that due to the careful design of our slabs we do not need to invest any time ignoring redundant and virtual events. As indicated in Figure 9, a redundant event is due to a wavefront vertex hitting a segment with equal or lower speed than segments hit by the same wavefront vertex earlier. If we simply refrain from forwarding local queries to sublocal data structures designed for segments with equal or lower speed, we implicitly ignore all redundant events.

Virtual events will be omnipresent in sublocal and local data structures as a narrowed view of the wavefront expansion does not allow us to foresee whether some impending event will be anticipated. However, Theorem 2 takes care of this, thus we simply have to show that no virtual event will ever be predicted globally. According to our definition of virtual events (see Section 3.1), a virtual event will never be the next event of the wavefront expansion to happen, since some other event will prevent this. Consequently the global prediction yields the earlier event. Note that this is only true since slabs are carefully designed to be subsets of their corresponding regions. If for the sake of low slab complexity this property is violated (e.g. if we always use rectangular slabs), virtual events outside the corresponding region could indeed be globally predicted. Such events would then have to be taken care of explicitly. This yields the following lemma:

Lemma 2. *The total number of globally predicted events is $O(c+n)$.*

5. Main result

The structure of our algorithm is shown in Figure 15, and in the previous sections we have described all its vital data structures and procedures. We now state our main result:

Theorem 3 (Construction of $\mathcal{V}_C(S)$). *Given an isothetic transportation network C with c disjoint isothetic segments, a constant number of different speeds on these segments and a set S of n sites, the refined City Voronoi diagram can be computed in $O((c+n)\log^5(c+n)\log\log(c+n))$ time using $O((c+n)\log^5(c+n))$ storage. The refined City Voronoi diagram answers queries asking for the quickest path to S in $O(L+\log(c+n))$ time, where L is the complexity of the path.*

Now we can put things together to prove Theorem 3. According to Lemma 2, $O(c+n)$ events are treated. Using Observation 3, Theorem 2 allows us to predict each event in $O(\log^5(c+n)\log\log(c+n))$ time. Hence, the total time used for the global prediction is $O((c+n)\log^5(c+n)\log\log(c+n))$, which dominates the time needed to handle events (see Observation 1 and Lemma 2). Observation 3 and thus Theorem 2 require $O((c+n)\log^3(c+n))$ space. The preprocessing time of $O((c+n)\log^3(c+n)\log\log(c+n))$ for the global prediction dominates the preprocessing times of the other data structures (see Observation 3). The total preprocessing time in turn is dominated by the running time of the global prediction.

6. Conclusion and open problems

In this paper we have presented an algorithm for the construction of the refined City Voronoi diagram. By carefully simplifying the data objects involved, we could utilize fast techniques for the simulation of a wavefront expansion. On the other hand, the space consumption of our simplified data and the data structures is only slightly superlinear. While our algorithm runs in subquadratic time it relies heavily on certain constraints in the setup. In particular these are the constant number of different network speeds and the isothetic network layout. Our results can be generalized to arbitrarily oriented network segments and to weighted sites as long as the number of different orientations and weights is constant. However, it is a challenge to find a general solution that

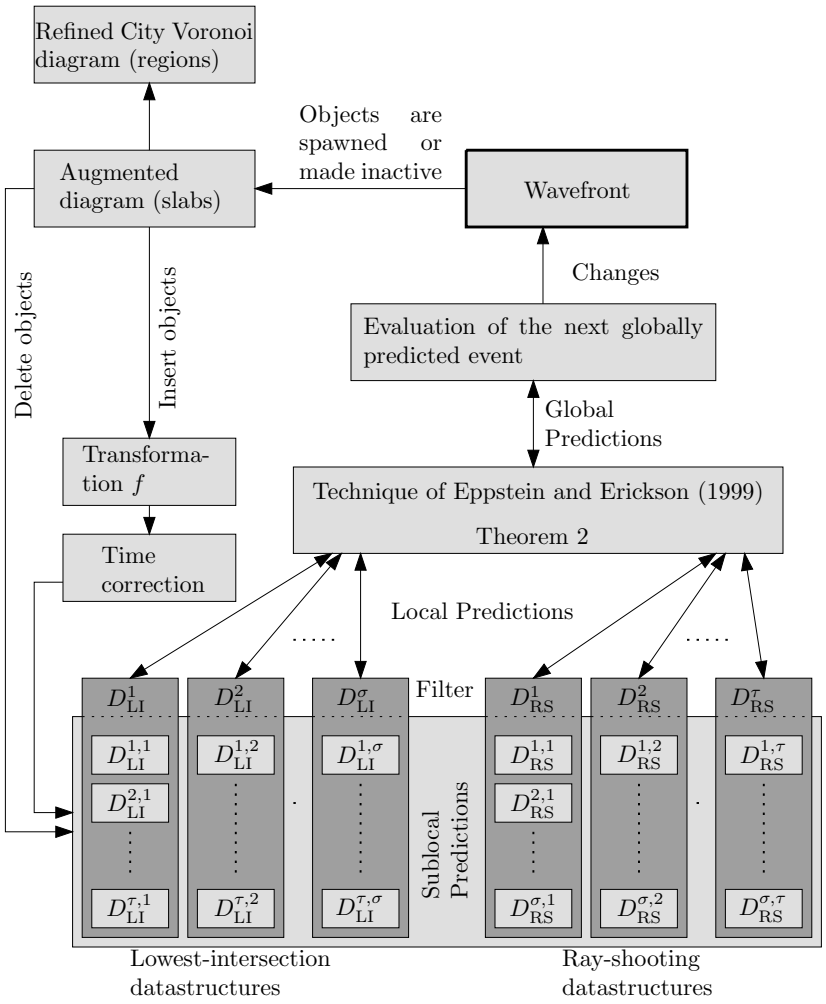

Figure 15. Sketch of the construction algorithm.

runs in subquadratic time and can handle arbitrary network speeds, site weights, and segment orientations.

Acknowledgments

We thank Jae-Hoon Kim and especially Chan-Su Shin for discussions about the City Voronoi diagram.

References

[1] M. Abellanas, F. Hurtado, C. Icking, R. Klein, E. Langetepe, L. Ma, B. Palop del Río, and V. Sácristan. Proximity problems for time metrics induced by the l_1 metric and isothetic networks. In *Actas de los IX Encuentros de Geometría Computacional*, pages 175–182, Universitat de Girona, 2001.

[2] P. K. Agarwal, J. Basch, L. J. Guibas, J. E. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection. In *Proc. 4th Int. Workshop on Algorithmic Foundations of Robotics (WAFR'00)*, 2000.

[3] O. Aichholzer, F. Aurenhammer, and B. Palop del Río. Quickest paths, straight skeletons, and the City Voronoi diagram. *Discrete & Computational Geometry*, 31(1):17–35, 2004.

[4] S. W. Bae and K.-Y. Chwa. Voronoi diagrams with a transportation network on the Euclidean plane. In R. Fleischer and G. Trippen, editors, *Proc. 15th International Symposium on Algorithms and Computation (ISAAC'04)*, volume 3341 of *Lecture Notes in Computer Science*, pages 101–112. Springer-Verlag, 2004.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry*. Springer Verlag, 2nd edition, 2001.

[6] D. Eppstein and J. G. Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete Computational Geometry*, 22(4):569–592, June 1999.

[7] L. P. Gewali, A. C. Meng, J. S. Mitchell, and S. Ntafos. Path planning in 0/1/infinity weighted regions with applications. *INFORMS Journal on Computing*, 2(3):253–272, 1990.

[8] F. Hurtado, B. Palop del Río, and V. Sacristán. Diagramas de Voronoi con fonciones temporales. In *Actas de los VIII Encuentros de Geometría Computacional*, pages 279–287. Universitat Jaume I, 1999.

[9] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–667, Aug. 1987.

[10] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[11] Y. Ostrovsky-Berman. Computing transportation Voronoi diagrams in optimal time. In *Proc. 21st European Workshop on Computational Geometry (EWCG'05)*, pages 159–162, Eindhoven, 9–11 Mar. 2005.

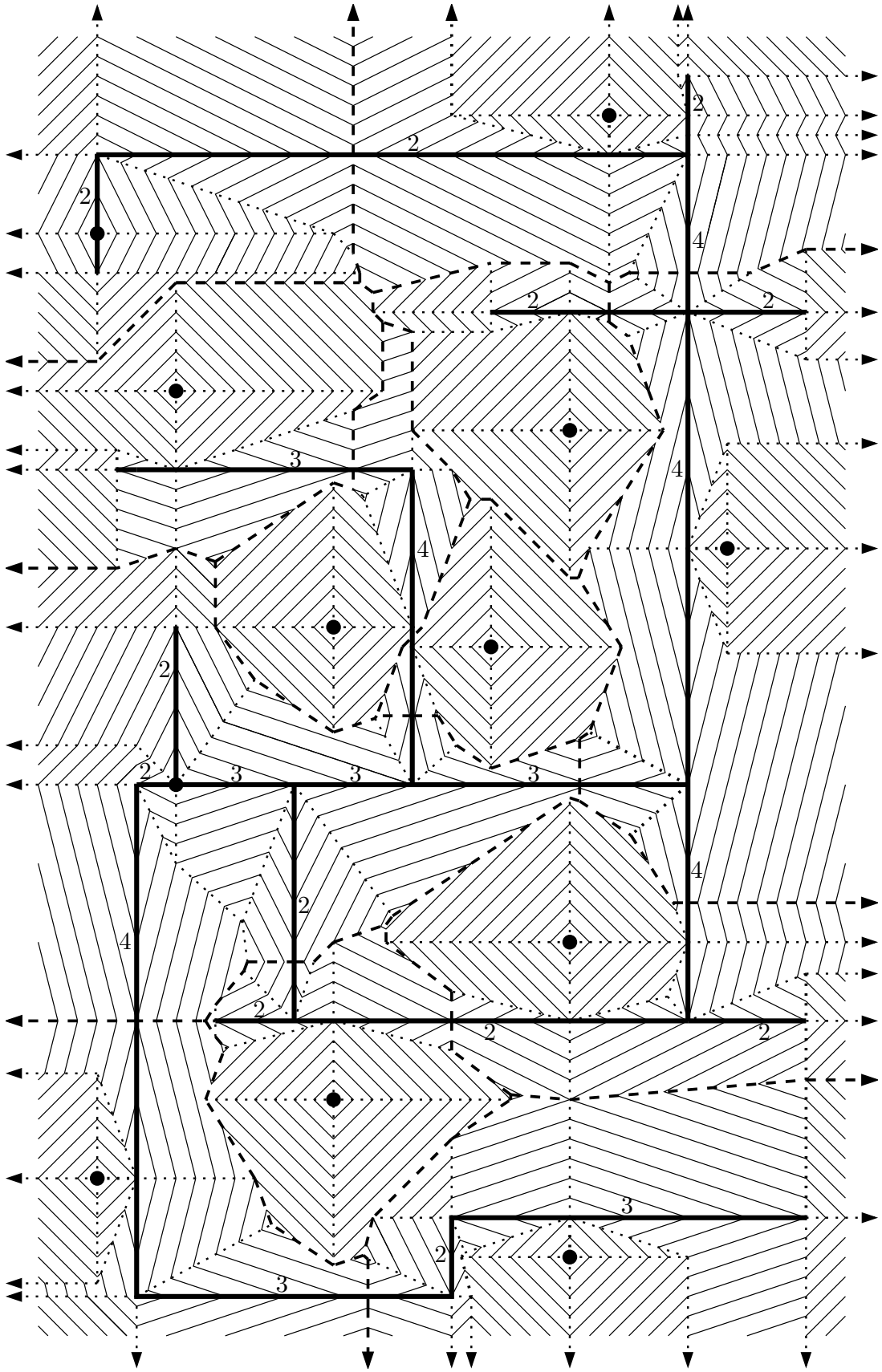

Figure 16. The City Voronoi diagram (dashed) of a complex transportation network (bold) and twelve sites (disks). The individual speeds of the network segments are given next to the segments. Snaphots of the expanding wavefront are given by thin solid lines, while the additional edges of the refined diagram, being the traces of wavefront vertices, are indicated by dotted lines.