Constructing the City Voronoi Diagram Faster

Robert Görke* Alexander Wolff*

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. [2], which takes $O(n \log n + c^2 \log c)$ time.

1 Introduction

Let's assume Manhattan to be void of car traffic by the year 2050. 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.

2 Concretization

We concretize the situation as follows. The transportation network $C = \{e_1, \ldots, e_c\}$ consists of c isothetic line segments, that are only allowed to touch at endpoints. Each segment e_i is assigned a speed $g_i > 1$. We require that the number of different speeds is constant. A segment can be accessed and left at any point. The n sites $S = \{\omega_1, \ldots, \omega_n\}$ are scattered arbitrarily 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 (bidirectionally) to move with speed g_i. We 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: $d(a,b) = d_{\text{Manhattan}}(\Pi \setminus C) + \sum_{e_i \in C}(\frac{d_{\text{Manhattan}}(e_i \cap \Pi)}{g_i})$.

*Fakultät für Informatik, Universität Karlsruhe, http://i11www.ira.uka.de/people. R. G. supported by the European Commission within FET Open Projects DELIS, A. W. supported by grant WO 758/4-1 of the German Science Foundation (DFG)

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* $\mathcal{V}_C(S)$ [2] of the City Voronoi diagram that can then also report the quickest path to the closest site in additional time $O(L)$, with L being the path complexity. We now state our main result. The proof will be given at the end of this paper.

Theorem 1 (Construction of $\mathcal{V}_C(S)$) *Given an isothetic transportation network C with c edges, a constant number of different speeds on these edges 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.*

3 Previous work

The City metric was first introduced by Abellanes et al. [1] who derived basic results for a single straight line as transportation network. Aichholzer et al. [2] 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 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 plane using the continuous Dijkstra method [4]. Then carefully adapted straight skeleton figures scheduled at these nodes are computed by employing well known techniques for the construction of abstract Voronoi diagrams.

4 The wavefront expansion

Our algorithm constructs the City Voronoi diagram $V_C(S)$ by simulating the expansion of a wavefront

starting at the set S of sites. At time t 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 note 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 the partition $V_C(S)$, see Figure 1. We can trace the path of 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 the wavefront moved in, we can tell for all points of that region how to reach the youngest object of this region. By doing this repeatedly, we 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 an interaction, collision or an interference between the wavefront and the network or even another part of the wavefront happens. We call each of these points in time *events*. At an event the combinatorial shape of the wavefront changes.

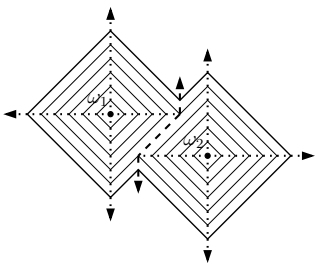

Figure 1: The wavefronts of two sites merge, tracing out a border.

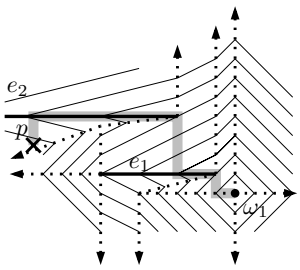

Figure 2: The path of the wavefront guides the way from p back to ω_1.

4.1 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. It is not hard to see the following:

Observation 1 *For any type of event the extent of changes on the wavefront is constant.*

As a consequence of this observation, we need to focus on the detection of events. An upcoming 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 with a lower timestamp, see Figures 3 and 4 for an example. We can even detect events that do happen, but still don't contribute to the complexity of $V_C(S)$. We call such events *redundant*, see Figure 5 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.

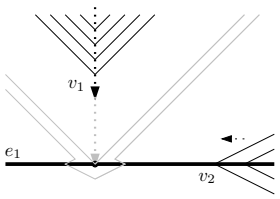

Figure 3: A type-A-event is pending.

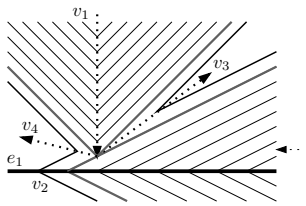

Figure 4: The event has been prevented.

4.2 The linear complexity

Adapting a result of Aichholzer et al. [2] to a constant number of network speeds we get the following result:

Theorem 2 *The number of edges, vertices and faces of the refined City Voronoi diagram $\mathcal{V}_C(S)$ is linear in the number c of segments and the number n of sites.*

This gives rise to the fact that the number of relevant events occurring during the wavefront expansion is also linear in $(c+n)$. Opposed to that, the number of redundant events can amount to $\Theta(c(c+n))$ (see Figure 5), 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 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 detection of all four types of events.

4.3 The wavefront in space

We now add a third dimension (z-axis) to our situation, such that a positive z-component is added to the wavefront expansion which starts in the x-y-plane. Consequently wavefront vertices and edges trace out rays and polygons, respectively. Accordingly all network segments are extended to vertical unbounded rectangles and network nodes are extended to half-lines, both being unbounded in positive z-direction. If

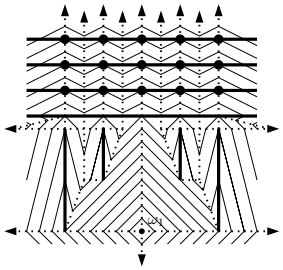

Figure 5: Cascade of redundant type-A-events (marked by disks).

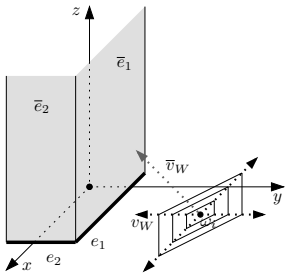

Figure 6: A type-A-event in space involving $\overline{v}_W$ and $\overline{e}_1$ is imminent.

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 6 shows how the z-axis is added. Now, reconceiving the nature of each event in space, we can observe the following:

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

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

5 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 changing sets of objects (polygons and rays).

5.1 The global prediction

Eppstein and Erickson [3] 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 the foot points of rays and partially unbounded polygons, respectively. We use the following result:

Theorem 3 ([3]) *Suppose that after $P(n)$ preprocessing 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.*

Note that we can neglect the linear preprocessing time of the starting wavefront in our situation. Employing this theorem we are left with the lesser problem of efficiently performing both ray-shooting queries and their inverse, called *lowest-intersection queries*. 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.

5.2 Simplification of wavefront data

The data we deal with for the purpose of local predictions comprises arbitrarily shaped, partially unbounded polygons in space. If we split these surfaces at each event along the current wavefront as depicted in Figure 7, we obtain *slabs* that are either triangles or quadrilaterals. 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. Inactive slabs can't take part in a relevant event. By Theorem 2 this augmentation of $\mathcal{V}_C(S)$ retains the linear complexity. 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 any slab has actually been traced out by the wavefront at any given time. 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.

5.3 Orthogonalized sublocal queries

We now define slabs to be *similar* if their sides pairwise have the same angular position. For an example see Figure 8. Rays are similar if they merely point in the same direction and move at the same speed.

Lemma 4 *The number of classes of similarity of slabs and of rays is constant.*

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 4 the number of ray and slab classes is constant, the number of pairs of ray and slab classes is constant, too. Clearly, any local prediction can easily be computed out of 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 con-

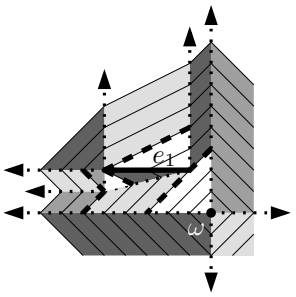

Figure 7: Division into slabs (dashed lines).

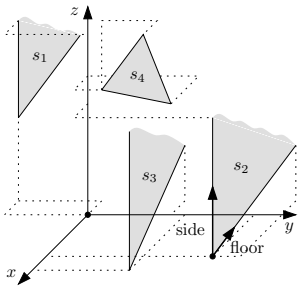

Figure 8: Only slabs s_1 and s_2 are similar.

sisting 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 9, 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 10), adding one more level to the data structure.

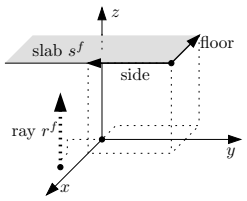

Figure 9: A transformed slab-ray pair.

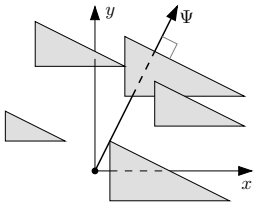

Figure 10: The additional Ψ-axis.

5.4 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 multidimensional orthogonal range query. Making use of Theorem 2 and of well-known results about multi level range trees and fractional cascading we can state the following:

Observation 3 *Sublocal data structures 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 they cannot simply be represented by their static foot points p_{foot}. In order to do justice to the dynamic nature of rays we should in fact regard the (moving) tip of the rays. But instead of repeatedly advancing the tips of all rays we can simply apply a time corrected insertion: $p^{new}_{foot} := p_{foot} - (0,0,t)|\vec{v}|$, with t being the elapsed time and $\vec{v}$ being the direction of the ray. They key observation is that these modified foot points represent at all times the relative position of the tips of their rays, after the transformation f has been applied.

5.5 Ignoring irrelevant 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 don't need to add up any time for irrelevant events. As indicated in Figure 5, 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. Due to the fact that slabs comprise only points they would actually reach if unhindered by events it is not hard to see that no virtual event will ever be globally predicted. Thus by Theorem 2 we get:

Lemma 5 *The total number of globally predicted events is* $O(c+n)$.

6 Proof of main result

Now we can put things together to prove Theorem 1. According to Lemma 5, $O(c+n)$ events are treated. Using Observation 3, Theorem 3 lets us 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 5). Observation 3 and thus Theorem 3 require $O((c+n)\log^3(c+n))$ space.

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, Universidad de Girona, 2001.

[2] O. Aichholzer, F. Aurenhammer, and B. Palop del Río. Quickest paths, straight skeletons, and the City Voronoi diagram. In *Proc. 18th Symp. Computational Geometry*, pages 151–159. ACM Press, June 2002.

[3] 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.

[4] 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.