I N F O R M A T I K

A simple way to recognize a correct Voronoi diagram of line segments

Christoph Burnikel, Kurt Mehlhorn, Michael Seel

MPI–I–1999–1–007 October 1999

FORSCHUNGSBERICHT RESEARCH REPORT

M A X - P L A N C K - I N S T I T U T
F Ü R
I N F O R M A T I K

Im Stadtwald 66123 Saarbrücken Germany

Author's Address

Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
email: {burnikel,mehlhorn,seel}@mpi-sb.mpg.de

Acknowledgements

This work was supported in part by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141(ALCOM II) and the BMFT(Förderungskennzeichen ITS 9103).

Abstract

Writing a program for computing the Voronoi diagram of line segments is a complex task. Not only there is an abundance of geometric cases that have to be considered, but the problem is also numerically difficult. Therefore it is very easy to make subtle programming errors. In this paper we present a procedure that for a given set of sites S and a candidate graph G rigorously checks that G is the correct Voronoi diagram of line segments for S. Our procedure is particularly efficient and simple to implement.

Keywords

Program checking, Verification, Voronoi diagram of line segments, Planarity, Consistency, Winding number

1 Introduction

Algorithms for computing the Euclidean Voronoi diagram of line segments are hard to implement. One reason is the numerical complexity of the problem[1]. Another reason is the geometric complexity of this particular Voronoi diagram that is due to an abundance of geometric cases that an implementation has to take into account. In particular, the core of our LEDA implementation of the randomized-incremental algorithm [3] has more than ten thousand lines of C++ code. Although it uses exact arithmetic to decide numerically difficult cases, how can we be sure that the result of our program is correct? Is there a *simple* and efficient procedure that rigorously proves or disproves the correctness of the returned graph for an arbitrary set of input sites? We answer this question positively, presenting a new program checker in the spirit of [4]. Our procedure works also for other types of Voronoi diagrams, including Voronoi diagrams for closed convex sites under convex distance functions.

A program checker with the desired properties is known for the Voronoi diagram of points in the plane. It is based on the following fact: A triangulation that is locally Delaunay is also globally Delaunay. See [7], Lemma 3 or [2]. Translated into a statement for the Voronoi diagram, this means that a planar embedded graph that is locally Voronoi is automatically globally Voronoi. Unfortunately, the proof of this result does not seem to extend to other types of Voronoi diagrams. Even if a similar statement is proven for the Voronoi of line segments, it will not necessarily provide an easy method of program checking. This is because testing the planarity of a graph embedded by arcs of Voronoi bisectors is nontrivial for the case of line segment sites. More specifically, how do we check for an edge cycle given by a closed path of bisector segments that it forms a simple curve, without checking all pairs of bisector segments for intersection? We will see that such an explicit planarity test can be avoided in our checker. Beside checking all single Voronoi edges *locally*, we need only one *global* test to verify that the sites that the program puts at the convex hull actually form a convex set. This implies as well a simplification of the program checker for the case of point sites.

Our paper has the following outline. In Section 2 we specify the problem in more detail and give basic definitions. In Section 3 we develop our program checker in several steps and at the same prove its correctness. In Section 4 we sketch how the checker can be actually implemented. Finally, in Section 5 generalize the checker to other types of Voronoi diagrams.

2 Basics

Clearly we first have to tell *what exactly* we are testing before we can seriously think about a program checker. We suppose that the input set S consists of planar segments that may intersect only at their endpoints. Segments may degenerate to single points. For the definition of the Voronoi diagram $VD(S)$ we refer to [6]. The program that we want to test shall compute a *combinatorial embedding* of the Voronoi diagram $VD(S)$. This is a bidirected[1] graph $G(S)$ realizing the Voronoi vertices and edges such that each edge is contained in exactly one *face* associated with a site. In this context the face associated with a site is the counterclockwise cycle of edges that bound the Voronoi region of that site. We assume that $VD(S)$ is turned into a planar graph by restricting the diagram to a circle Γ that is big enough to contain all

[1] In a bidirected graph, for every egde (v, w) there is a reverse edge (w, v). It is only a matter of presentation whether we use a bidirected graph or an undirected graph.

vertices and bounded edges of $VD(S)$. An example is given by Figure 1. This graph can

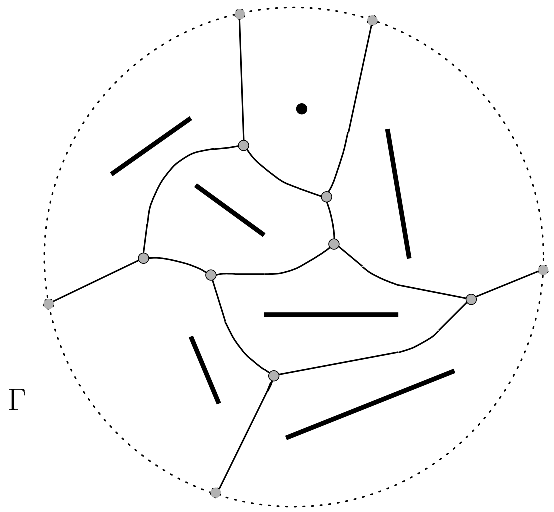

Figure 1: The Γ-embedding of $G(S)$

be viewed as a representation of the Voronoi diagram of the set S, extended by an artificial site ∞; see e.g. [3]. Note that the restriction of $VD(S)$ to Γ introduces new vertices and edges on Γ. These edges are called Γ-edges and the resulting edge cycle is called the Γ-face, representing the Voronoi region of ∞. Unlike all other faces the Γ-face is clockwise oriented.

In our definition, Voronoi regions having area zero do not generate a face of $G(S)$; we consider these regions as empty. An example is a point site p that is the common endpoint of two collinear input segments. In this case the Voronoi region of p following the definition given in [6] is contained in a line. Likewise, Voronoi edges of length zero do not appear in $G(S)$.

Our Voronoi program returns the graph $G(S)$ only if it has computed $VD(S)$ correctly. In general it returns some graph G and the task of our program checker is to verify that $G = G(S)$. The graph G is the input of our program checker and is called *the candidate graph.* In our checker we can merely assume that G is some bidirected graph such that each edge knows its reverse edge, its successor and predecessor on the face to which it belongs and also the site to which the face is associated. Note that the actual data structure that our program uses to represent the Voronoi diagram might contain more information, such as numerical approximations of the Voronoi vertices. This information is however *not* subject to our program checker.

3 The Voronoi checker

In this section we develop a method to verify that the candidate graph G returned by our Voronoi program is the correct graph $G = G(S)$. There are four steps in our method that we outline as follows.

1. Check that G is a bidirected graph whose edges are consistently labelled by faces associated with input sites.

2. For each edge of G, identify the four sites that define the corresponding edge in $VD(S)$ and check that this edge is *locally Voronoi*.

3. Check that there is exactly one face of ∞ in G whose Γ-embedding is simple.

4. Check that every input site whose Voronoi region is nonempty has a face in G.

The first three checks ensure that the candidate graph G correctly represents the Voronoi diagram $VD(T)$ for the subset T of sites present in G, but possibly with some sites of S missing. Hence we need a fourth, final check that the computed Voronoi diagram is complete. There is no need for a global planarity test; only the Γ face is tested for simplicity. This test basically amounts to testing whether the sites whose faces border on the ∞ face form a convex set. In the next subsections we describe the steps of our checker in more detail.

3.1 First step: Combinatorial consistency

In the first step we check that the candidate graph G has the combinatorial structure of a planar graph with faces. More precisely, we consider the labels `succ(e)` of the face successor and `pred(e)` of the face predecessor of each edge e and verify that these labels consistently define face cycles. Here we test

(a) For every edge e, `succ(e)` and `pred(e)` are associated with the same site as e.

(b) For every edge e we have
`pred(succ(e)) = succ(pred(e)) = e`.

If the first step is passed our graph G is partitioned into faces associated with sites, but we do not yet know whether there is a one-to-one correspondence between the sites and the faces. This will turn out later.

3.2 Second step: Local Voronoi Test

The tests in the second step have the goal to let us define an appropriate embedding of G that agrees with the Voronoi diagram $VD(S)$, if G is correct. This embedding is never actually computed but we will refer to it in the rest of our checker. What we test is for every pair of reverse directed edges $e = (e^+, e^-)$ of G whether it represents a locally correct Voronoi edge in the way indicated by the labels of G. The following sites are relevant here:

- the sites p, q associated with e^+ and e^-,
- the sites r_p and s_p associated with the reverse edges of the predecessor and the successor of e^+ in the face of p,
- the sites r_q and s_q associated with the reverse edges of the successor and the predecessor of e in the face of q.

See Figure 2 for an illustration. Here we denote the unique point v having the same distance from p, q, r such that the three sites appear in this clockwise cyclic order around v by $v = v_{prq}$.

First we assume that none of p, q and r_p, r_q, s_p, s_q is ∞. Clearly the endpoints v_{pr_pq} of e^+ and v_{pr_qq} of e^- have to be the same point if e^+ and e^- are reverse edges. Likewise, the

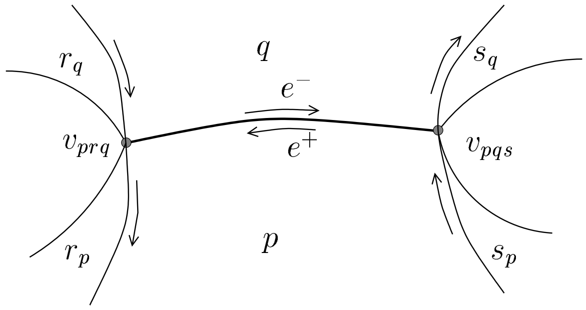

Figure 2: Checking edge e locally

endpoints v_{pqs_p} and v_{pqs_q} must be the same. Note that r_p and r_q are identical except for the degenerate case where r_p, r_q, p, q touch a common circle. The analogous statement is true for the sites s_p, s_q. Let now r be one of the sites $\{r_p, r_q\}$ and s one of the sites $\{s_p, s_q\}$. If e represents a Voronoi edge, the order of the vertices v_{prq} and v_{pqs} on the pq-bisector must be correct: v_{pqs} must be before v_{prq} if the pq bisector is oriented such that p is left. Hence we do

(a) if r_p and r_q are different, test whether p, q, r_p, r_q are cocircular,

(b) if s_p and s_q are different, test whether p, q, s_p, s_q are cocircular,

(c) test whether the sites $\{p, r, q\}$ and $\{p, q, s\}$ actually define points v_{prq} and v_{pqs},

(d) test whether v_{pqs} is strictly before v_{prq} on the pq-bisector, oriented such that p is left.

If the tests are passed we can embed e by the segment of the pq bisector between v_{prq} and v_{pqs}.

The same tests apply if p, q are finite, but one of the other sites is ∞. In this case e represents an unbounded Voronoi edge. If one of the sites r_p, r_q is ∞, they must be both equal to ∞; the same is true for s_p, s_q. Note that test (d) above makes still sense, because $v_{pq\infty}$ and $v_{p\infty q}$ can be thought of as the two ends of the pq bisector lying at infinity. In particular, $v_{pq\infty}$ and $v_{p\infty q}$ are always well-defined. For the embedding of e we fix a circle Γ that includes all the finite edges that are embedded as described before. Now we embed e by the part of the bisector between p_{prq} and v_{pqs} that is contained in Γ.

Finally, we consider the case that one of p, q is ∞, say, $p = \infty$. Then we must have $q \neq \infty$ and $r_p = r_q$, $s_p = s_q$. Note that test (d) now does not make sense anymore; we omit it. The two vertices of e are embedded by intersecting the appropriate bisectors with Γ. e itself is embedded by joining its two vertices on Γ.

3.3 Third step: Planarity

We suppose that the candidate graph passes the tests of steps 1 and 2. In the third step of our checker we want to ensure that the embedding defined in the previous section is planar. What we already know is this: G is partitioned into faces associated with the sites in $S \cup \{\infty\}$. Each face f of a site $s \neq \infty$ is embedded by a closed path γ_f that consists of s-bisector segments and possibly one or two Γ-edges; the face of ∞ is embedded by a closed path γ_Γ on

Γ. Furthermore we know that every edge used in the embedding occurs in two face cycles, once in every direction. We want to show now that every one of these paths is a simple curve and the regions bounded by the paths have no common interior points. To this end we consider the winding number of each curve γ around a point $x \in \mathbb{R}^2$ not on γ. It can be formally defined by the complex curve integral

$$W_\gamma(x) := \frac{1}{2\pi i} \int_\gamma \frac{1}{z-x} dz$$

where we identify $\mathbb{R}^2$ with $\mathbb{C}$. The following facts are well-known. Let G_γ denote the planar embedded graph induced by γ. This graph has exactly one unbounded face. Clearly, if γ is simple then G_γ has only two faces by Jordan's curve theorem. We have

1. W_γ is constant within every face of G_γ and is zero in the unbounded face.

2. Let x and y be two points in the interior of two neighboring faces f_x and f_y of G_γ. Furthermore let $\gamma^\perp$ be a curve from x to y that lies completely in $f_x \cup f_y$ and crosses γ in the single, simple point $\gamma \cap \gamma^\perp$. If γ crosses $\gamma^\perp$ from left to right, we have

$$W_\gamma(f_y) = W_\gamma(f_x) + 1$$

and if γ crosses $\gamma^\perp$ from right to left, we have

$$W_\gamma(f_y) = W_\gamma(f_x) - 1.$$

Using these facts we can successively compute the winding number for every face of G_γ, starting at the unbounded face where the winding number is zero. As an example we consider a face cycle γ of correctly oriented Voronoi edges around the point s where the winding number around s is 3. Such a face cycle is shown in Figure 3. Note however that the edges that make up the cycle γ have only the topology of Voronoi edges. The above procedure for computing

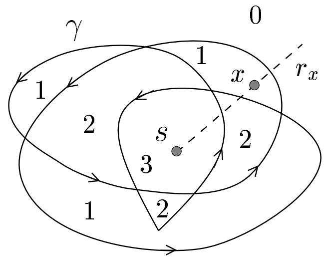

Figure 3: The winding numbers for every component of $\mathbb{R}^2 \setminus \gamma$

winding numbers also works in principle if γ contains infinitely many multiple points (say, multiple edges). We only have to split γ into finitely many simple (= not self-intersecting) pieces $\gamma_1, \gamma_2, \ldots$. Taking a curve $\gamma^\perp$ as above that does not contain any endpoint of a γ_i, we count the number LR of pieces γ_i that cross $\gamma^\perp$ in $\gamma \cap \gamma^\perp$ from left to right and also the number RL of pieces that cross $\gamma^\perp$ from right to left. In the above notation we get then $W_\gamma(f_y) = W_\gamma(f_x) + LR - RL$.

Coming back to the example of Figure 3, we remark that the winding number is strictly positive in the bounded faces of G_γ. We call such paths *positively oriented*. A path γ is called *negatively oriented* if $-\gamma$ is positively oriented. The next lemma shows that all face cycles of our embedding except γ_Γ are positively oriented.

Lemma 1 *Let γ be the embedded cycle of a face in G, labeled with the finite site s. Then γ is positively oriented.*

Proof: Here we use the particular geometry of the Voronoi diagram. Recall that every dominance region D of s is *star-shaped with respect to* s, i.e., for every point $x \in D$ and a point s_x on s that is nearest to x the segment $\overline{s_x x}$ is contained in D. This property is used to compute the winding number of a bounded face f of the graph G_γ, as follows. We choose an arbitrary point x in the interior of f and consider the ray r_x from s_x through x, as in Figure 3. We can assume that r_x intersects γ in only finitely many points, none of which is a vertex of f; otherwise we perturb x slightly. Now let e be a st-edge in the face cycle of s. Since $D(s,t)$ is star-shaped with respect to s, the ray r_x can cross e at most once. If e crosses r_x, it is from right to left because e is forward oriented on the st-bisector directed such that s is left. Also, the one possible Γ edge of γ is oriented counterclockwise and so this edge can cross r_x from right to left only. Hence the winding numbers of the points on the ray are strictly decreased whenever the ray intersects γ. Moving our reference point on r_x from x to infinity we eventually reach the unbounded face of G_γ where the winding number is zero. We conclude that the winding number is strictly positive inside f. ■

Now we consider the sum of the winding numbers of all cycles γ. We argue that this sum is zero everywhere.

Lemma 2 *Let in the following sum γ vary over all cycles used in our embedding of the candidate graph G. Then we have*

$$\sum_{\gamma} W_{\gamma}(x) = 0$$

for all x not contained in any cycle γ.

Proof: Let $\mathcal{E}$ be the set of pairs $e = (e^+, e^-)$ of reverse edges in G. Then the paths $\{\gamma \mid \gamma \text{ embedded face cycle in } G\}$ are exactly composed of all the edges $\{e^+, e^- \mid e \in \mathcal{E}\}$. By definition of the winding number we get for every point x in the plane that is not on any cycle γ

$$\begin{aligned}
\sum_{\gamma} W_{\gamma}(x) &= \frac{1}{2\pi i} \sum_{\gamma} \int_{\gamma} \frac{1}{z-x} dz \\
&= \frac{1}{2\pi i} \sum_{e \in \mathcal{E}} \left(\int_{e^+} \frac{1}{z-x} dz + \int_{e^-} \frac{1}{z-x} dz \right) \\
&= \frac{1}{2\pi i} \sum_{e \in \mathcal{E}} 0 \\
&= 0.
\end{aligned}$$

Here the sum is zero just because we are integrating over each edge in both directions. There is also a more intuitive way to see this, as follows. Crossing any edge e^+ of a face cycle γ by a path $\gamma^{\perp}$, we find the reverse edge e^- in another face cycle γ'. Hence the change of $W_{\gamma}(x)$ in the crossing of e^+ is compensated by the inverse change of $W'_{\gamma}(x)$ caused by the crossing of e^-. This means that the sum of all winding numbers is a constant. Since all winding numbers are zero outside Γ, the sum is zero. ■

Recall that we want to ensure that all cycles of our embedding are simple and that all the regions bounded by the cycles are disjoint. The next lemma shows that here it is enough to verify that the face γ_{Γ} of ∞ is simple.

Lemma 3 *Let γ_Γ be a negatively oriented simple cycle. Then our embedding of the candidate graph G is planar and connected.*

Proof: By our assumption, the winding number of γ_Γ is -1 inside Γ and hence, by Lemma 2, we have

$$\sum_{\gamma \neq \gamma_\Gamma} W_\gamma(x) = 1 \tag{1}$$

for all $x \in D(\Gamma)$. Since every term in the sum is nonnegative and integral, we conclude that there is always exactly one cycle γ such $W_\gamma(x) = 1$ and $W_{\gamma'}(x) = 0$ for all $\gamma' \neq \gamma$. In particular, all winding numbers are either zero or one. Hence for an arbitrary cycle γ every bounded face of G_γ has winding number one. This is only possible if γ is a simple curve. Then we can define $V(\gamma)$ as the region bounded by γ. By Equation 1, all other regions $V(\gamma')$ are disjoint from $V(\gamma)$ which means that our embedding is planar.

Suppose now that G is not connected. Then we can drop all face cycles in the sum of Equation 1 that are not in the same component as the Γ-face, without changing the result. Hence the winding number of every dropped γ is identically zero, which is a contradiction. This completes the proof of our lemma. ■

Hence for the planarity test we do the following.

(a) Test whether there is exactly one face of ∞, the Γ-face.

(b) Test whether the γ_Γ cycle is simple.

3.4 Conclusion: G is Voronoi

It is time to summarize the knowledge that we have if the candidate graph G passes the tests from the first three steps. We know that G is a connected planar embedded graph where every face cycle associated with a finite site s is a simple curve that consists of counterclockwise oriented s-bisector segments or circular Γ arcs. Let for every site s that has a face f in the candidate graph $V(f)$ denote the open region bounded by the embedded face cycle of f. We have to ensure that $V(f)$ equals the true Voronoi region $VR(s,S)$ of s in the Γ-embedding of $G(S)$. One problem is that there could be sites with more than one face in G, another is that there may be sites in S that are not present in G but have a Voronoi region in $VD(S)$. We postpone the last check and first prove that $V(f)$ is always equal to the region $VR(s,T)$ in the Γ-embedding of $G(T)$ where T is the set T of sites that are associated with a face in G.

Lemma 4 *If for a face f associated with the finite site s we have $\overline{V(f)} \cap \overline{VR(s,T)} \neq \emptyset$, then $VR(s,T)$ is contained in $V(f)$.*

Proof: Let $x \in \overline{V(f)} \cap \overline{VR(s,T)}$. Then for any $y \in VR(s,T)$ there is a path $\varphi : [0,1] \mapsto \overline{VR(s,T)}$ from x to y. If y is outside $\overline{V(f)}$, then φ leaves the boundary of $V(f)$ at some point $z = \varphi(u), u \in (0,1)$. Let z be contained in a st-edge of the s-face. Then φ crosses the bisector $\mathcal{B}(s,t)$ and hence clearly $\varphi(v)$ is nearer to t than to s for all $v > u$ in a small neighborhood of u, which is a contradiction to our assumption $y \in VR(s,T)$. Hence y is in $\overline{V(f)}$. But y cannot lie on the boundary of $V(f)$ since it would then be on a s-bisector, which proves the lemma. ■

In our Voronoi diagram, any point on a site s is in the closure of the Voronoi region of s and is also in the closure of any region $V(f)$ where f is associated with s. This shows that the condition of the preceding lemma is always satisfied.

Lemma 5 *We have $G = G(T)$ where T is the set of sites associated with a face of G.*

Proof: We denote the disk defined by the Γ circle by $D(\Gamma)$. By the previous lemma, every Voronoi region $VR(t,T)$ is contained in the region $V(f)$ of a face associated with t. Since $\{VR(s,T)|s \in T\}$ is a partition of $D(\Gamma)$ just as $\{V(g)|g$ is a face in $G\}$, we conclude that $VR(t,T)$ is equal to $V(f)$ whenever $VR(t,T)$ is nonempty. Of course, there can be only one such f. Hence the faces of G are exactly the nonempty Voronoi regions of $VD(T)$. ■

3.5 Fourth step: Completeness

It remains to test whether every site that has a Voronoi region in $VD(S)$ has also a face in the candidate graph. The problem is that (the interior of) the Voronoi region of a segment endpoint p may be empty. This is the case if and only if the set of segments in S with endpoint p is not contained in any open halfplane having p in its boundary. For example, this condition holds if two collinear segments have endpoint p. For that reason we check

(a) test whether every segment site has a face in G,

(b) test whether every point site with nonempty Voronoi region has a face in G.

By Lemma 5 our theorem follows.

Theorem 1 *A candidate graph G that passes all of the tests from steps 1 to 4 represents the correct Voronoi diagram $VD(S)$.*

3.6 An example

Although the presented program checker is relatively simple, the question remains: Could we have done better? At first sight, testing the convexity of the sites on the Γ face might seem obsolete. However, the following example shows that this is not so. It is best presented for the dual Delaunay graph. First consider the correct Delaunay triangulation of a sample set of 5 points in Figure 4. Now we change the coordinates of our sample points, but preserving

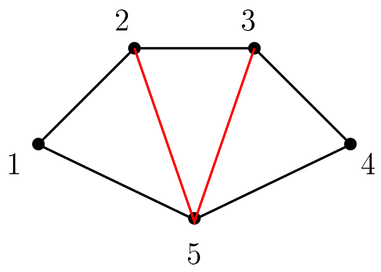

Figure 4: A Delaunay triangulation

the graph structure; see Figure 5. This graph is obviously not a triangulation, but it is nevertheless locally Delaunay:

- The cycle of points $1, 2, 3, 4, 5$ on the outer face consists of right-turns.
- The points on the triangles $\triangle(1,2,5)$, $\triangle(2,3,5)$ and $\triangle(3,4,5)$ are all in correct right-turn orientation.

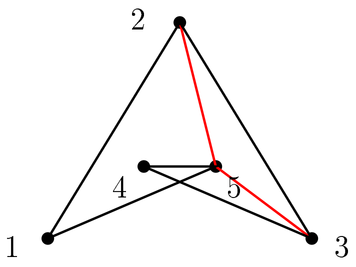

Figure 5: A graph that is only locally Delaunay

- Points $1, 4$ are outside $\bigcirc(2, 3, 5)$, point 2 is outside $\bigcirc(3, 4, 5)$ and point 3 is outside $\bigcirc(1, 2, 5)$.

Our checker would refuse this graph because the point 'cycle' $1, 2, 3, 4, 5$ does not form a convex set; there is a small triangular region where its winding number is 2. This shows that step 3 of our checker is really necessary.

4 Implementation of the checker

In this section we sketch the implementation of our Voronoi checker.

The check for combinatorial consistency of G in step 1 is straightforward to implement.

For the completeness check of step 4 we first sort the sites in S and the sites present in G separately in any total order. Then it is straightforward to identify those points in S that have an empty Voronoi region. Now we simply compare the list of sites in G with the list of relevant sites in S.

We consider the check of simplicity for the Γ cycle of G in step 3. If it is correct, the sites on the convex hull must appear in cyclic order on the Γ cycle. In particular, the sites $s_1, \ldots, s_k$ that border on the Γ face must form a convex polygon. This is easy to test as described in [4].

Finally we consider step 2 where we check that the edges of G are locally Voronoi. The two nontrivial computations which have to be done are

- Computation of Voronoi vertices
- Sorting of Voronoi vertices on a bisector

These tasks do not impose any particular problem on the geometric side. However the correctness of the implementation depends on the use of exact arithmetic as described in [1].

We remark that the first three steps of our checker can be done in linear time. The fourth check, however, takes time $O(n \log n)$ because we have to sort the input sites.

5 Other Voronoi diagrams

Our verification procedure is not restricted to the Voronoi diagram of line segments. In fact, very little of our argumentation in proving the correctness of the checker depends on the diagram type. In Lemma 1 we need that the Voronoi region of a site s is star-shaped with respect to s. Furthermore, there must be a simple way to decide whether a site has nonempty Voronoi region. This is true for e.g.

- Voronoi diagrams of convex sites under convex distance functions, see Lemma 4.30 of [5],
- most types of Euclidean Voronoi diagrams.

There are more types of Voronoi diagrams where Voronoi regions are star-shaped, like the *power diagram* for point sites and the Voronoi diagrams of *weighted* point sites. The only potential problem here is that even isolated sites may have empty Voronoi regions. For sites that do not have a face in the candidate graph often there is no simple way of proving that their Voronoi region is really empty. In these cases we suggest to let the Voronoi program return for every site s without a face *a proof* that the Voronoi region is empty. One way of doing this is by providing a nonempty Voronoi region V of s in a subdiagram $VD(\{s\} \cup T), T \subset S$ together with another site t whose insertion into the Voronoi diagram destroys V completely.

References

[1] C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections.* PhD thesis, Universität des Saarlandes, 1996.

[2] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Verlag, 1986.

[3] R. Klein, K. Mehlhorn, and S. Meiser. On the construction of abstract Voronoi Diagrams. *LNCS*, 450:138–154, 1990.

[4] K. Mehlhorn, S. Näher, T. Schilz, S. Schirra, R. Seidel, M. Seel, and Uhrig. C. Checking geometric programs or verification of geometric structures. *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, pages 159–165, 1996.

[5] S. Meiser. *Zur Konstruktion abstrakter Voronoi-Diagramme.* PhD thesis, Univ. des Saarlandes, 1993.

[6] Atsuyuki Okabe, Barry Boots, and Kokichi Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Chichester, UK, 1992.

[7] R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21:243–245, 1978.

I N F O R M A T I K

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Im Stadtwald
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-1999-3-005	T.A. Henzinger, J. Raskin, P. Schobbens	Axioms for Real-Time Logics
MPI-I-1999-3-004	J. Raskin, P. Schobbens	Proving a conjecture of Andreka on temporal logic
MPI-I-1999-3-003	T.A. Henzinger, J. Raskin, P. Schobbens	Fully Decidable Logics, Automata and Classical Theories for Defining Regular Real-Time Languages
MPI-I-1999-3-002	J. Raskin, P. Schobbens	The Logic of Event Clocks
MPI-I-1999-3-001	S. Vorobyov	New Lower Bounds for the Expressiveness and the Higher-Order Matching Problem in the Simply Typed Lambda Calculus
MPI-I-1999-2-005	J. Wu	Symmetries in Logic Programs
MPI-I-1999-2-004	V. Cortier, H. Ganzinger, F. Jacquemard, M. Veanes	Decidable fragments of simultaneous rigid reachability
MPI-I-1999-2-003	U. Waldmann	Cancellative Superposition Decides the Theory of Divisible Torsion-Free Abelian Groups
MPI-I-1999-2-001	W. Charatonik	Automata on DAG Representations of Finite Trees
MPI-I-1999-1-006	M. Nissen	Integration of Graph Iterators into LEDA
MPI-I-1999-1-005	J.F. Sibeyn	Ultimate Parallel List Ranking ?
MPI-I-1999-1-004	M. Nissen, K. Weihe	How generic language extensions enable "open-world" desing in Java
MPI-I-1999-1-003	P. Sanders, S. Egner, J. Korst	Fast Concurrent Access to Parallel Disks
MPI-I-1999-1-002	N.P. Boghossian, O. Kohlbacher, H.-. Lenhof	BALL: Biochemical Algorithms Library
MPI-I-1999-1-001	A. Crauser, P. Ferragina	A Theoretical and Experimental Study on the Construction of Suffix Arrays in External Memory
MPI-I-98-2-018	F. Eisenbrand	A Note on the Membership Problem for the First Elementary Closure of a Polyhedron
MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-014	Y. Gurevich, M. Veanes	Partisan Corroboration, and Shifted Pairing
MPI-I-98-2-013	H. Ganzinger, F. Jacquemard, M. Veanes	Rigid Reachability
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard

MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid E-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-2-002	F. Jacquemard, C. Meyer, C. Weidenbach	Unification in Extensions of Shallow Equational Theories
MPI-I-98-1-031	G.W. Klau, P. Mutzel	Optimal Compaction of Orthogonal Grid Drawings
MPI-I-98-1-030	H. Brönniman, L. Kettner, S. Schirra, R. Veltkamp	Applications of the Generic Programming Paradigm in the Design of CGAL
MPI-I-98-1-029	P. Mutzel, R. Weiskircher	Optimizing Over All Combinatorial Embeddings of a Planar Graph
MPI-I-98-1-028	A. Crauser, K. Mehlhorn, E. Althaus, K. Brengel, T. Buchheit, J. Keller, H. Krone, O. Lambert, R. Schulte, S. Thiel, M. Westphal, R. Wirth	On the performance of LEDA-SM
MPI-I-98-1-027	C. Burnikel	Delaunay Graphs by Divide and Conquer
MPI-I-98-1-026	K. Jansen, L. Porkolab	Improved Approximation Schemes for Scheduling Unrelated Parallel Machines
MPI-I-98-1-025	K. Jansen, L. Porkolab	Linear-time Approximation Schemes for Scheduling Malleable Parallel Tasks
MPI-I-98-1-024	S. Burkhardt, A. Crauser, P. Ferragina, H. Lenhof, E. Rivals, M. Vingron	q-gram Based Database Searching Using a Suffix Array (QUASAR)
MPI-I-98-1-023	C. Burnikel	Rational Points on Circles
MPI-I-98-1-022	C. Burnikel, J. Ziegler	Fast Recursive Division
MPI-I-98-1-021	S. Albers, G. Schmidt	Scheduling with Unexpected Machine Breakdowns
MPI-I-98-1-020	C. Rüb	On Wallace's Method for the Generation of Normal Variates
MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence
MPI-I-98-1-017	A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, E. Ramos	Randomized External-Memory Algorithms for Some Geometric Problems
MPI-I-98-1-016	P. Krysta, K. Loryś	New Approximation Algorithms for the Achromatic Number
MPI-I-98-1-015	M.R. Henzinger, S. Leonardi	Scheduling Multicasts on Unit-Capacity Trees and Meshes
MPI-I-98-1-014	U. Meyer, J.F. Sibeyn	Time-Independent Gossiping on Full-Port Tori
MPI-I-98-1-013	G.W. Klau, P. Mutzel	Quasi-Orthogonal Drawing of Planar Graphs
MPI-I-98-1-012	S. Mahajan, E.A. Ramos, K.V. Subrahmanyam	Solving some discrepancy problems in NC*
MPI-I-98-1-011	G.N. Frederickson, R. Solis-Oba	Robustness analysis in combinatorial optimization
MPI-I-98-1-010	R. Solis-Oba	2-Approximation algorithm for finding a spanning tree with maximum number of leaves
MPI-I-98-1-009	D. Frigioni, A. Marchetti-Spaccamela, U. Nanni	Fully dynamic shortest paths and negative cycle detection on diagraphs with Arbitrary Arc Weights
MPI-I-98-1-008	M. Jünger, S. Leipert, P. Mutzel	A Note on Computing a Maximal Planar Subgraph using PQ-Trees
MPI-I-98-1-007	A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Schönherr	On the Design of CGAL, the Computational Geometry Algorithms Library