Computing Delaunay Triangulations in Manhattan and Maximum Metric

Michael Jünger Volker Kaibel Stefan Thienel

Institut für Informatik
Universität zu Köln

January 9, 1995

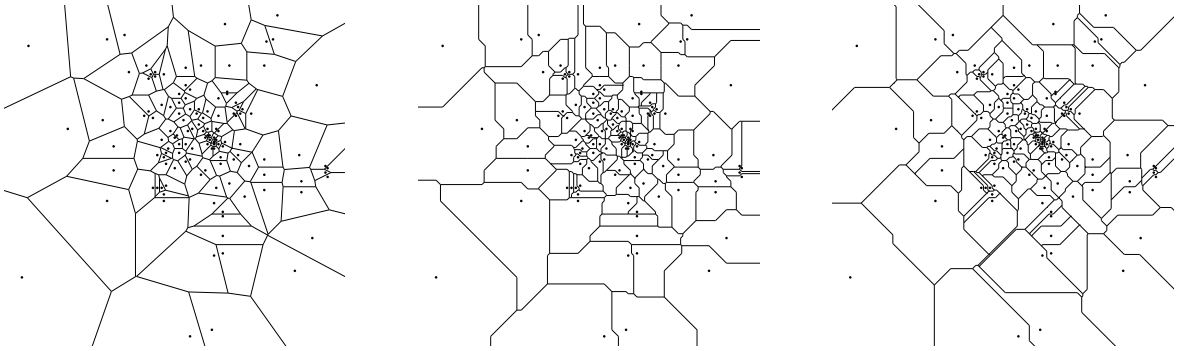
Abstract

We modify the incremental algorithm for computing Voronoi diagrams in the Euclidean metric proposed by Ohya, Iri and Murota [6] in order to obtain an algorithm for computing Voronoi diagrams (resp. Delaunay triangulations) in Manhattan and Maximum metric, that is rather simply to implement. We generalize the notions of "Voronoi diagram" and "Delaunay triangulation" in such a way that these structures still can be computed by an algorithm very similar to the one of Ohya, Iri and Murota, and that they contain – as special cases – analogons to the Euclidean Voronoi diagram (Delaunay triangulation) in the Manhattan and Maximum metric. In this paper, we give a detailed description of the algorithm, that makes it (rather) easy to write a computer program that computes Delaunay triangulations for Manhattan or Maximum metric.

Keywords: Delaunay triangulation, Voronoi diagram, Manhattan metric, Maximum metric

1 Introduction

Voronoi diagrams are structures like those shown in Figure 1. The dots are called "generators", and the Voronoi diagram consists of a set of regions, each one containing exactly one generator, having the property that for all the points inside such a region the generator in that region is a nearest neighbour among all the generators. If we measure distances by the Euclidean metric, the Voronoi diagram looks like Figure 1(a), if we choose Manhattan metric or Maximum metric, the Voronoi diagrams have shapes like shown in Figures 1(b) resp. 1(c). If we consider these diagrams as (embedded) planar graphs we can construct the dual graphs, and these ones are – roughly spoken – the Delaunay triangulations of the generator set (according to the respective metric). Such Delaunay triangulations contain

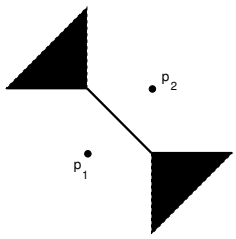


(a) Euclidean metric (b) Manhattan metric (c) Maximum metric

Figure 1: Voronoi diagrams of `bier127` out of the TSPLIB for three metrics

much information about the neighbourhood structure of the generators, and therefore, they are useful tools in many algorithms (e.g., computing minimum spanning trees, k-nearest-neighbour graphs, etc.).

In case of the Euclidean metric, one may define those regions (called "Voronoi regions") simply as the set of all points being closer to the respective generator than to any other one. In case of the Manhattan metric (and also for the Maximum metric) this is not practicable anymore. The occuring problems are due to the fact that the set of all points having equal distance to two distinct generators p_1 and p_2 in the Euclidean metric is simply the perpendicular bisector of the line segment joining p_1 and p_2, while it has the following shape in case of the Manhattan metric (and a similar one in the Maximum metric), and therefore, is no longer adequate to be used as "separator" of the Voronoi regions.



Hence, in case of the Manhattan metric, one cannot longer base all notions and algorithmic decisions just on the metric.

In this paper, we propose an abstract algorithmic frame for computing Delaunay triangulations (from which one can also compute the corresponding Voronoi diagrams) that uses an incremental method. We describe how to fill this frame in order to obtain a "Voronoi diagram" with respect to the Manhattan metric. It will turn out that computing Delaunay triangulations in the Maximum metric can easily be reduced to the computation in the Manhattan metric. How to use the frame in order to derive an (numerical stable) algorithm for the Euclidean metric is described in JÜNGER, KAIBEL AND THIENEL [2].

The aim of this paper is to present a complete detailed description of the algorithm such that it is possible to write a computer program for computing Delaunay triangulations in Manhattan metric (and Maximum metric) without many difficulties. We decided to focus on this algorithmical aspect rather than on the theoretical one. Therefore, we do not give proofs for most of our claims, since otherwise, without shortening of the algorithmical part, the extent of the paper would have grown over the acceptable limit. For a complete proof of the correctness of the algorithm we refer to KAIBEL [3].

2 The Abstract Algorithmic Frame

In order to derive an abstract algorithmic frame for the computation of Voronoi diagrams (resp. Delaunay triangulations) in various metrics, we first develop an abstract definition of these objects inspired by two goals. First, we want the abstract objects to be the well known Voronoi diagrams (resp. Delaunay triangulations) in the special case of the Euclidean metric, and second, we want to stay able to compute these structures by a refined version of the incremental algorithm proposed by OHYA, IRI AND MUROTA [6]. The basic concept of our abstract formulation is the notion of a *bisector* – defined independently from the notion of *metric*. This approach is similar to that of KLEIN [4] and KLEIN, MEHLHORN AND MEISER [5]. The difference of their and our ways of defining the *abstract Voronoi diagram* is due to the reason that we intend to use the "simple" incremental algorithm for its computation.

Let us stipulate some notations. Let $\lambda \subset \{c : \mathbb{R} \longrightarrow \mathbb{R} \mid c \text{ Jordan curve}\}$ the set of all Jordan curves satisfying $\lim_{t\to\infty} \|c(t)\| = \lim_{t\to-\infty} \|c(t)\| = \infty$. We denote the set of supports of such curves by $\Lambda := \{c(\mathbb{R}) \mid c \in \lambda\}$. Hence, each $C \in \Lambda$ divides the affine plane $\mathbb{R}^2$ into two regions G_1 and G_2, such that $\mathbb{R}^2 = G_1 \cup C \cup G_2$ and $G_1 \cap G_2 = G_1 \cap C = G_2 \cap C = \emptyset$ hold. By $\Delta := \{(t,t) \mid t \in \mathbb{R}\}$ we denote the diagonal of $\mathbb{R}^2$. For a subset $A \subset \mathbb{R}^2$ we denote by ∂A its boundary and by $\overline{A}$ its closure, both with respect to the canonical (norm-induced) topology on $\mathbb{R}^2$.

We come to the central definition.

Definition 2.1 *A map $b : (\mathbb{R}^2 \times \mathbb{R}^2) \setminus \Delta \longrightarrow \Lambda$ is called a "bisector function" if for any three pairwise distinct points $p_1, p_2, p_3 \in \mathbb{R}^2$ the following five conditions are valid:*

(i) $b(p_1, p_2) = b(p_2, p_1)$
(Symmetry)

(ii) $b(p_1, p_2)$ *divides* $\mathbb{R}^2$ *into two regions* $G(p_1, p_2)$ *and* $G(p_2, p_1)$ *satisfying* $p_1 \in G(p_1, p_2)$ *and* $p_2 \in G(p_2, p_1)$.
(In particular, we have $p_1, p_2 \notin b(p_1, p_2)$.*)*

(iii) $b(p_1, p_2) \cup b(p_1, p_3)$ *divides the plane* $\mathbb{R}^2$ *into not bounded regions.*
($b(p_1, p_2)$ *and* $b(p_1, p_3)$ *shall have a connected intersection.)*

(iv) $|b(p_1,p_2) \cap b(p_2,p_3) \cap b(p_3,p_1)| \leq 1$
(This property will enable us to give an adequate definition of a "Voronoi point".)

(v) $p \in G(p_1,p_2), G(p_2,p_3) \Rightarrow p \in G(p_1,p_3)$
(This transitivity condition is a generalization of the relation "is closer than".)

If b is any bisector function, $b(p_1,p_2)$ is called the "bisector of p_1 and p_2".

One readily sees that especially the map defined via the *perpendicular bisectors* of two points (yielding to the usual Euclidean Voronoi diagram) satisfies the requirements demanded in the above definition.

Now, we define a number of notions known in connection with Euclidean Voronoi diagrams.

Definition 2.2 *Let $p_1,p_2,p_3 \in \mathbb{R}^2$ be three pairwise distinct points in $\mathbb{R}^2$, and let b be a bisector function. If $b(p_1,p_2) \cap b(p_2,p_3) \cap b(p_3,p_1) \neq \emptyset$ holds, the (by part (iv) of the preceding definition uniquely determined) point $vp(p_1,p_2,p_3) \in \mathbb{R}^2$ with $\{vp(p_1,p_2,p_3)\} = b(p_1,p_2) \cap b(p_2,p_3) \cap b(p_3,p_1)$ is called the "Voronoi point" of p_1,p_2 and p_3.*

Definition 2.3 *Let $\Omega = \{g_1,\ldots,g_n\} \subset \mathbb{R}^2$ be a finite subset of $\mathbb{R}^2$.*

(i) For all $g \in \Omega$ we call $vr(g) := vr_\Omega(g) := \bigcap_{h \in \Omega \setminus \{g\}} G(g,h)$ the "Voronoi region" of g.

(ii) We call $vd(\Omega) := \bigcup_{g \in \Omega} \partial vr(g)$ the "Voronoi diagram" of Ω.

(iii) For all $g,h \in \Omega$ $(g \neq h)$ $ve(g,h) := ve_\Omega(g,h) := \overline{vr(g)} \cap \overline{vr(h)}$ is called the "Voronoi edge" of g and h.

For the rest of the paper, let $\Omega = \{g_1,\ldots,g_n\} \subset \mathbb{R}^2$ be a set of "generators" and b any bisector function.

By exploiting the features of a bisector function, one can show the following five properties:

(i) $vr(g)$ is a simply connected region for any $g \in \mathbb{R}^2$.

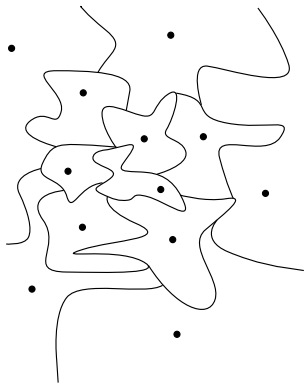
(ii) $\partial vr(g)$ is a Jordan curve for any $g \in \mathbb{R}^2$.

(iii) $ve(g,h)$ is connected for all $g,h \in \Omega$ $(g \neq h)$.

(iv) The closed hulls of all Voronoi regions cover the whole plane, i.e, $\mathbb{R}^2 = \bigcup_{g \in \Omega} \overline{vr(g)}$.

(v) The cover in *(iv)* is "nearly a partition" of the plane, more precisely we have $vr(g) \cap vr(h) = \emptyset$ for all $g,h \in \Omega$.

These five facts indicate that $vd(\Omega)$ has a shape as shown in Figure 2, where each of the regions is the Voronoi region of the (unique) generator it contains.



The incremental algorithm we will describe later makes essential use of the (intuitively distinguished) "branching points" in the Voronoi diagram. The following proposition can be proved by showing that $ve(g,h) \subset b(g,h)$ holds for any pair $g,h \in \Omega$ $(g \neq h)$.

Proposition 2.4 *Let $g,h,f \in \Omega$ be pairwise distinct, and let $\overline{vr(g)} \cap \overline{vr(h)} \cap \overline{vr(f)} = \{s\}$. Then $s = vp(g,h,f)$ holds.*

Suppose, we know that three Voronoi regions touch each other in one of those branching points. Then the proposition provides the possibility to determine the Voronoi point of the three involved generators just by examining the bisectors generated by these three generators.

Next, we want to derive from our notion of a Voronoi diagram the notion of a "Delaunay triangulation".

Definition 2.5 *The "Pre-Delaunay graph" $pd := pd(\Omega) := (\Omega, E)$ of Ω is the graph defined by $(g,h) \in E \Leftrightarrow |ve(g,h)| > 1$.*

Note that (due to the fact that the Voronoi regions are regions, and therefore path connected) $vd(\Omega)$ induces an embedding of the planar graph $pd(\Omega)$. According to this embedding, the branching points in $vd(\Omega)$ correspond to the inner faces of $pd(\Omega)$. Now suppose that a branching point in $vd(\Omega)$ has "degree" greater than three. Then the corresponding face in $pd(\Omega)$ is no triangle. This is the reason, why people often assume the generators to be in *general position*, i.e., with respect to the Euclidean metric, no four of them lying on a common circle; an assumption that is unlikely to hold for practical instances. A difficulty arising from the occurrence of such faces (being no triangles) in a Delaunay triangulation is that it makes the incremental algorithm more complicated.

Definition 2.6 *A "Delaunay triangulation" $dt = dt(\Omega)$ of Ω is a plane extension of $pd(\Omega)$, where the outer face was not changed, and all inner faces are triangles.*

Any Delaunay triangulation determines in a unique way a Voronoi diagram but the converse is not true. When the bisectors are the perpendicular bisectors (as for the Euclidean metric), it follows from the convexity of the Voronoi regions that any Delaunay triangulation may be embedded straight line, i.e. using straight line segments for the edges. Moreover, it is possible to show this also for the bisectors that we will define for the Manhattan metric in the following section, although they do not give rise to convex Voronoi regions.

For the sequel, let dt be a Delaunay triangulation of Ω and vd the corresponding Voronoi diagram. Proposition 2.4 leads to the observation that every triangle in dt represents the Voronoi point of the three generators defining that triangle.

Definition 2.7 *We say, such a Voronoi point belonging to a triangle of the Delaunay triangulation dt is "active" in dt.*

Now we are able to describe the algorithmic frame. In our exposition, we will use both the notions Voronoi diagram and Delaunay triangulation, always keeping in mind their relationship established above. The algorithm is based on a combination of the methods proposed by OHYA, IRI AND MUROTA [6] and SUGIHARA [8].

The incremental step is much easier if the newly arising Voronoi region is known to be bounded. To guarantee this for all the incremental steps, we initially determine a set Ω_0 of dummy generators having the property that for all $g \in \Omega$ the Voronoi region $vr_{\Omega_0 \cup \{g\}}(g)$ in the Voronoi diagram of $\Omega_0 \cup \{g\}$ is bounded. Then, we start the computation by determining a Delaunay triangulation of Ω_0 (this shall be easy if Ω_0 is chosen appropriately), and afterwards we add successively all the generators of Ω by the incremental step described below. After all the generators are inserted, we remove the dummies, and adjust the Delaunay triangulation. Choosing and removing of the generators are dependent on the bisector function, of course.

We stipulate the following notations. Let $\Omega = \{g_1, \ldots, g_n\}$ be a set of generators. For any $j \in \{1, \ldots, n\}$ we set $\Omega_j := \{g_1, \ldots, g_j\}$, we denote the Voronoi diagram of Ω_j by vd_j, and we write $vr_j(g)$ for the Voronoi region of any $g \in \Omega_j$ in vd_j.

The Figure 2 illustrates the insertion step of the incremental algorithm.

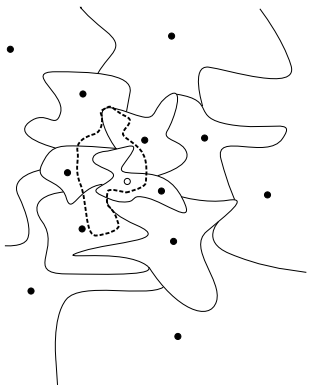


Figure 2: The insertion step

Besides introducing and removing the dummies, the algorithm has as its only bisector specific components the following two functions.

The first is defined for any pairwise distinct generators $z, a, b \in \Omega \cup \Omega_0$.

`Closer`(z, a, b)

TRUE, if $z \in G(a, b)$
FALSE, otherwise

The second is defined for any four pairwise distinct generators $z, a, b, c \in \Omega \cup \Omega_0$, where dt' is a Delaunay triangulation of $\Omega \setminus \{z\}$ and $vp(a, b, c)$ is active in dt'.

`CheckVP`(z, a, b, c)

IN, if there exists $\omega \in \{a, b, c\}$ such that $vp(a, b, c) \in G(z, \omega)$
OUT, if there exists $\omega \in \{a, b, c\}$ such that $vp(a, b, c) \in G(\omega, z)$
ON, otherwise

Before giving the detailed algorithm, we briefly describe how the augmenting (incremental) step is performed. Therefore suppose, g_i ($i \in \{1, \ldots, n\}$) is the generator to insert.

We first determine a generator $nn \in \Omega_{i-1}$, such that $g_i \in \overline{vr_{i-1}(g_j)}$. This is done by the subroutine NEARESTNEIGHBOUR that starts its search at a candidate $cand = ini\_guess(i) \in \Omega_{i-1}$. The array ini_guess is determined in a procedure PREPROCESSING that also may reorder the generators in Ω (which may lead to significant running time improvements, cf. OHYA, IRI AND MUROTA [6]). NEARESTNEIGHBOUR searches (using the function CLOSER) all the neighbours of $cand$ in dt' for anyone being "closer" to g_i than $cand$ itself. If one of the neighbours does so, it becomes the new $cand$, and we proceed in the same way. Otherwise, $cand$ is the generator we searched for. The correctness of this procedure can be proved by exploiting the bisector properties.

Now, starting with nn, we modify all the Voronoi regions "touched" by the new one and "connect" the corresponding generators with g_i in the new Delaunay triangulation (cf. Figure 2). This update is done by searching those of the in dt' active Voronoi points on the boundary of the currently modified Voronoi region that will not be active anymore in the new Delaunay triangulation. These decisions are done by the function CHECKVP. Having found all these Voronoi points, we also know the Voronoi region to modify next. The modification of the Voronoi region of nn is done by the procedure INITIALMODIFY. When modifying the further Voronoi regions, we can exploit the additional information by which Voronoi edge we "entered" the region to speed up computations. Hence, for all the other concerned generators we use the procedure MODIFY.

We conclude this section by a detailed description of the whole algorithmic frame.

Algorithm INCREMENTALDELAUNAY

INPUT : $\Omega = \{g_1, \ldots, g_n\} \subset \mathbb{R}^2$
OUTPUT : A Delaunay triangulation of Ω

(1) PREPROCESSING
– Order the generators
– Determine the initial guesses $ini\_guess(g_i)$
(2) $\Omega_0 :=$ INTRODUCEDUMMIES(Ω)
(3) Determine Delaunay graph D_0 of Ω_0
(4) for $i := 1$ to n do
(5) $\quad D_i :=$ AUGMENTDELAUNAY($D_{i-1}, g_i, ini\_guess(g_i)$)
(6) $D :=$ REMOVEDUMMIES(D_n)
(7) return D

Subroutine AUGMENTDELAUNAY($D_{old}, new\_gen, initial\_guess$)

(1) $D_{new} := D_{old}$
(2) $nn :=$ NEARESTNEIGHBOUR($D_{old}, new\_gen, initial\_guess$)
(3) $(next\_vr, depart\_edge) :=$ INITIALMODIFY($D_{new}, D_{old}, nn, new\_gen$)
(4) while $next\_vr \neq nn$ do
$\quad$ {
(5) $\quad vr := next\_vr$
(6) $\quad arrive\_edge := depart\_edge$
(7) $\quad (next\_vr, depart\_edge) :=$ MODIFY($D_{new}, D_{old}, vr, arrive\_edge, new\_gen$)
$\quad$ }

Subroutine NEARESTNEIGHBOUR($D, new\_gen, initial\_guess$)

(1) $act\_gen := initial\_guess$
(2) for all $cand \in Inz_D(act\_gen)$ do
$\quad$ {
(3) $\quad$ if CLOSER($new\_gen, cand, act\_gen$) then
$\quad\quad$ {

```
(4)                 act_gen := cand
(5)                 goto (2)
                    }
                }
(6)     return act_gen
```

The following macro is defined in order to ease reading.

STEP
$i := j$
Let (vr, j) be the successor of (vr, i) in $Inz_{D_{old}}(vr)$

Subroutine INITIALMODIFY$(D_{old}, D_{new}, vr, new_gen)$

```
(1)     Determine i and j such that (vr, j) is the successor of (vr, i) in Inz_{D_old}(vr) and
        CheckVP(new_gen, vr, i, j) = OUT.
(2)     found := FALSE
(3)     while not found do
                {
(4)             pos := OUT
(5)             while pos = OUT do
                        {
(6)                     STEP
(7)                     pos =CheckVP(new_gen, vr, i, j)
                        }
(8)             if pos = IN then found := TRUE
(9)             arrive_edge := (vr, i)
(10)            while pos ≠ OUT do
                        {
(11)                    STEP
(12)                    pos =CheckVP(new_gen, vr, i, j)
(13)                    if pos = IN then found := TRUE
                        }
(14)            depart_edge := (vr, i)
(15)            if found then goto (16)
                }
(16)    Remove from D_new all edges, that coincide with vr between
        arrive_edge and depart_edge
(17)    Insert (vr, new_gen) into D_new
(18)    return (next_vr, depart_edge)
```

Subroutine MODIFY($D_{old}, D_{new}, vr, arrive\_edge, new\_gen$)

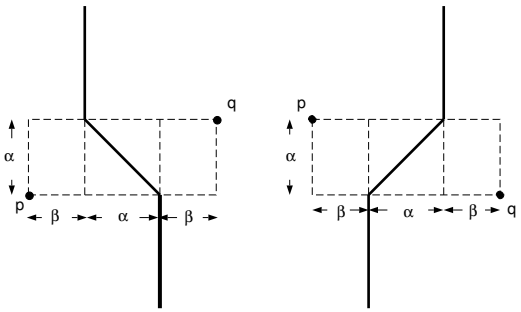
(1) Let $(vr, i) := arrive\_edge$
(2) Let (vr, j) be the successor of (vr, i) in $Inz_{D_{old}}(vr)$
(3) while `CheckVP`$(new\_gen, vr, i, j) \neq$ OUT do **STEP**
(4) $depart\_edge := (vr, i)$
(5) Remove from D_{new} all edges, that coincide with vr between $arrive\_edge$ and $depart\_edge$
(6) Insert (vr, new_gen) into D_{new}
(7) return $(i, depart\_edge)$

3 Implementation for the Manhattan Metric

If $d : \mathbb{R}^2 \times \mathbb{R}^2 \longrightarrow \mathbb{R}$ is any metric, we call for any pair $p, q \in \mathbb{R}^2$ $(p \neq q)$ the set $\{r \in \mathbb{R}^2 \mid d(r,p) = d(r,q)\}$ the "metric bisector" of p and q (according to the metric d).

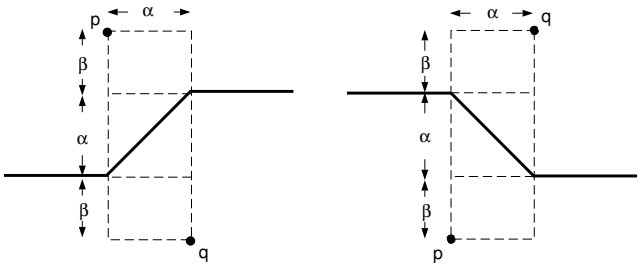
If d_1 is the Manhattan metric (i.e., $d_1(p,q) = |p_x - q_x| + |p_y - q_y|$ for all $p, q \in \mathbb{R}^2$), then for any two distinct points $p, q \in \mathbb{R}^2$ $(p \neq q)$ the metric bisector of p and q (w.l.o.g. $p_x \leq q_x$) looks as shown in the following figures, where in the third case the grey regions belong to the metric bisector:

1. $|p_x - q_x| > |p_y - q_y|$



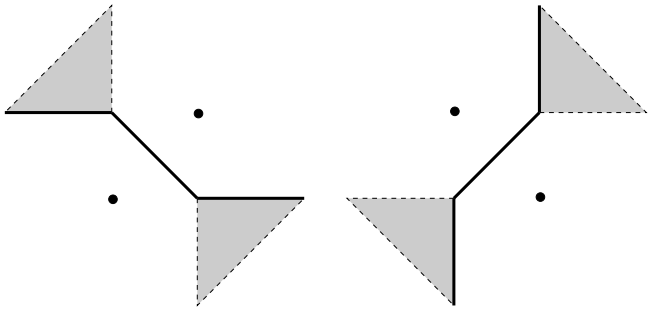
($\alpha := |p_y - q_y|$ and $\beta := \frac{|p_x - q_x| - |p_y - q_y|}{2}$)

2. $|p_x - q_x| < |p_y - q_y|$



($\alpha := |p_x - q_x|$ and $\beta := \frac{|p_y - q_y| - |p_x - q_x|}{2}$)

3. $|p_x - q_x| = |p_y - q_y|$



We define a function $b_1 : (\mathbb{R}^2 \times \mathbb{R}^2) \setminus \Delta \longrightarrow \Lambda$ by assigning to any pair $(p, q) \in (\mathbb{R}^2 \times \mathbb{R}^2) \setminus \Delta$ the fat line in the appropriate figure above.

By making many case distinctions, one can prove the following proposition.

Proposition 3.1 *The function b_1 is a bisector function.*

For the rest of this section, all notions corresponding to a bisector function correspond to b_1. Obviously, for any triple $p, q, r \in \mathbb{R}^2$ $(p \neq q)$ the relation $d_1(r, p) < d_1(r, q)$ implies $r \in G(p, q)$.

Observation 3.2 *If we have for $p \in \mathbb{R}^2$ and $g \in \Omega$ as well as $d_1(p, g) < d_1(p, h)$ for all $h \in \Omega \setminus \{g\}$ then p is contained in $vr_\Omega(g)$.*

3.1 Inserting the dummies

Suppose, all the generators out of Ω are located in the grey part Π of the following figure, where $\alpha = \frac{y_{max} - y_{min}}{2}$ and $\beta = \frac{x_{max} - x_{min}}{2}$.

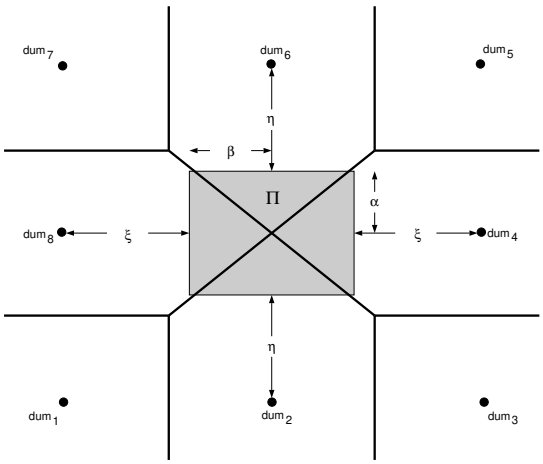


Figure 3: Inserting the dummies

Proposition 3.3 *If $\Omega_0 = \{dum_1, \ldots, dum_8\}$ is defined according to Figure 3.1, and if furthermore $\xi > \alpha$ and $\eta > \beta$, then $b_1(g, dum_i) \cap vr_{\Omega_0}(dum_i)$ is bounded for all $g \in \Omega$ and for all $i \in \{1, \ldots, 8\}$.*

This proposition leads immediately to the following procedure for inserting the dummy generators, where we choose $\xi := \eta := 2(\alpha + \beta) + 1$ due to reasons, that become clear in the next subsection.

Subroutine INTRODUCEDUMMIES

INPUT : $\Omega = \{g_1, \ldots, g_n\} \subset \mathbb{R}^2$
OUTPUT : $\{dum_1, \ldots, dum_8\} \subset \mathbb{R}^2$, such that in the Voronoi diagram of $\Omega_0 \cup \{g\}$ the Voronoi region of g is bounded for all $g \in \Omega$.

(1) x_{max} := maximum of all x-coordinates of Ω
x_{min} := minimum of all x-coordinates of Ω
y_{max} := maximum of all y-coordinates of Ω
y_{min} := minimum of all y-coordinates of Ω
(2) $\alpha := \frac{y_{max} - y_{min}}{2}$
$\beta := \frac{x_{max} - x_{min}}{2}$
(3) $\xi := \eta := 2(\alpha + \beta) + 1$
(4) $dum_1 := (x_{min} - \xi, y_{min} - \eta)$
$dum_2 := (x_{min} + \beta, y_{min} - \eta)$
$dum_3 := (x_{max} + \xi, y_{min} - \eta)$
$dum_4 := (x_{max} + \xi, y_{min} + \alpha)$
$dum_5 := (x_{max} + \xi, y_{max} + \eta)$
$dum_6 := (x_{min} + \beta, y_{max} + \eta)$
$dum_7 := (x_{min} - \xi, y_{max} + \eta)$
$dum_8 := (x_{min} - \xi, y_{min} + \alpha)$
(5) return $\{dum_1, \ldots, dum_8\}$

3.2 Removing the dummies

The following observation follows immediately from the definition of the bisector function b_1.

Observation 3.4 *For any triple $a, b, c \in \Omega$ $vp(a, b, c)$ is not outside the rectangle Π in Figure 3.1.*

Using our special choice of ξ and η, one can show that any point in Π is closer (with respect to d_1) to every generator out of Ω than to any of the dummy generators. This together with Observation 3.2 implies the following proposition.

Proposition 3.5 *For any $g \in \Omega$ and any $i \in \{1, \ldots, 8\}$ the Voronoi region $vr(dum_i)$ in V_{dum} satisfies $vr(dum_i) \cap \Pi = \emptyset$.*

Now we can conclude that in order to turn a Voronoi diagram of $\Omega \cup \Omega_0$ into a Voronoi diagram of Ω, we just have to remove all Voronoi edges that are belonging to dummy generators, and afterwards extend all those Voronoi edges to infinity that are ending with no continuation after this removal. The correctness of this procedure is due to the facts that these "elongations" start outside Π (by Proposition 3.5) and so they do not give rise to new Voronoi points (by Observation 3.4).

Subroutine REMOVEDUMMIES

INPUT : Delaunay triangulation D_{dum} of $\Omega \cup \Omega_0$
(where $\Omega_0 =$ INTRODUCEDUMMIES(Ω))
OUTPUT : Delaunay triangulation of Ω

(1) $D := D_{dum}$
(2) Remove from D all dummies and the edges incident to them.
(3) return D

3.3 Numerical operations

To complete the algorithm for the Manhattan metric it remains to define the two functions CLOSER and CHECKVP. Therefore, we first define two auxiliary functions. The first one decides if the line passing through two given (distinct) points has slope $+1$, -1 or a slope in $\mathbb{R} \setminus \{-1, +1\}$.

We call any line having slope $+1$ or -1 a "critical line". For any point $r \in \mathbb{R}^2$, we denote by $g_+(r)$ resp. $g_-(r)$ the line passing through r and having slope $+1$ resp. -1.

Function `OnLine`

INPUT : $a, b \in \mathbb{R}^2$
OUTPUT : $+(-)$, if a, b are on a line with slope $+1(-1)$,
otherwise FALSE

(1) if $a_x - a_y = b_x - b_y$ then return $+$
(2) if $a_x + a_y = b_x + b_y$ then return $-$
(3) return FALSE

The second auxiliary function permits us to use formulations like "if $z \in G(a,b) \ldots$".

Function ASSIGN

INPUT : $z, a, b \in \mathbb{R}^2$
OUTPUT : FIRST, if $z \in G(a,b)$
SECOND, if $z \in G(b,a)$
ON, if $z \in b_1(a,b)$

```
(1)   d_a := d_1(z,a)
      d_b := d_1(z,b)
(2)   if d_a < d_b then return FIRST
(3)   if d_b < d_a then return SECOND
(4)   exception := OnLine(a,b)
(5)   if exception = FALSE then return ON
(6)   if a_x > b_x then
                  {
(7)               Swap a,b
(8)               swapped := TRUE
                  }
(9)   else
(10)              swapped := FALSE
(11)  if exception = + then
                  {
(12)              if z_y < a_y then
(13)                    ret := FIRST
(14)              else if z_y > b_y then
(15)                    ret := SECOND
(16)              else
(17)                    return ON
                  }
(18)  else
                  {
(19)              if z_x < a_x then
(20)                    ret := FIRST
(21)              else if z_x > b_x then
(22)                    ret := SECOND
(23)              else
(24)                    return ON
                  }
(25)  if swapped then
                  {
```

```
(26)          if ret = FIRST then
(27)                  return SECOND
(28)          else
(29)                  return FIRST
              }
(30)  else
(31)          return ret
```

The definition of the function CLOSER is now trivial.

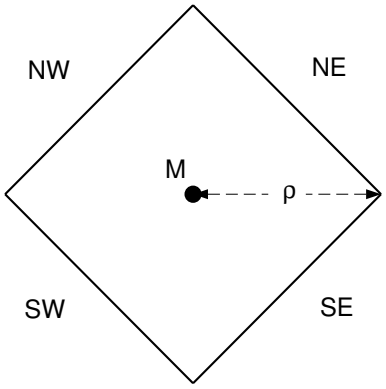
Function CLOSER

INPUT : $z, a, b \in \mathbb{R}^2$
OUTPUT : TRUE, if $z \in G(a, b)$, otherwise FALSE

```
(1)   if z ∈ G(a, b) then
(2)           return TRUE
(3)   else
(4)           return FALSE
```

We come to the definition of the function CHECKVP. For the sequel, let $z, a, b, c \in \mathbb{R}^2$ be pairwise distinct, such that the Voronoi point $vp := vp(a, b, c)$ exists. The definition of the bisector function b_1 implies that $d_1(vp, a) = d_1(vp, b) = d_1(vp, c)$. One possibility to compute the function CHECKVP is to calculate vp first, and afterwards perform the necessary checks for each generator out of $\{a, b, c\}$. Computing vp explicitly can be done in the case that no two points out of a, b, c lie on a common critical line by solving a 2×2 equation system that is not a linear one, since it contains expressions like $|x - y|$. However, by examining the relative position of a, b, c to each other, one can get rid of these "absolute values". If two generators out of a, b, c are located on a common critical line, one has to perform lots of case distinctions to figure out vp. We also implemented this version of CHECKVP, but we found out that our other method (computing CHECKVP without explicitly calculating vp) is more efficient. Therefore, we just describe this second method here.

We know from the above equation that vp is the center of a certain sphere (with respect to the Manhattan metric) having a, b and c on its boundary. In the Manhattan metric, a sphere $S_{M,\varrho} := \{p \in \mathbb{R}^2 \mid d_1(p, M) = \varrho\}$ having center $M \in \mathbb{R}^2$ and radius $\varrho \in \mathbb{R}^2$ looks as follows:



The *diameter* of the sphere is $D := 2\varrho$. From now on, let S be the sphere having vp as center and a, b, and c on its boundary. Obviously, we have *opposite sides* of such a sphere (i.e., sides being parallel but not identical), and hence we can speak about a side being located *between two opposite sides*.

Our algorithm to compute the function CHECKVP is based on the following two lemmata that describe how to determine the value of the function CHECKVP just from knowledge about the position of z relative to S. The rest of the algorithm just consists of determining that position.

The first lemma describes the case that z does not lie on the boundary of S. One proves it by simply comparing some distances.

Lemma 3.6 *We have:*

(i) If z lies inside S then there exists an $\omega \in \{a, b, c\}$ such that $vp(a, b, c) \in G(z, \omega)$ holds.

(ii) If z lies outside S then there exists an $\omega \in \{a, b, c\}$ such that $vp(a, b, c) \in G(\omega, z)$ holds.

The second one deals with the more complicated case that z is located on the boundary of S. Its proof proceeds by several case distinctions concerning the relative position of z and a, b, c.

Lemma 3.7 *Let z be on the sphere S, and let be $\delta \in \{a, b, c\}$. Then the following implications hold:*

(i) If z is not on the same side of S as δ then $vp \in b_1(z, \delta)$.

(ii) If z is on the same side Σ of S as δ then:

(a) $\Sigma = NW$:

$$vp \in \left\{ \begin{array}{ll} G(z,\delta), & \text{if } z_x < \delta_x \text{ and } z \neq W \\ G(\delta,z), & \text{if } \delta_x < z_x \text{ and } \delta \neq W \\ b(z,\delta), & \text{otherwise} \end{array} \right\}$$

(b) $\Sigma = NE$*:*

$$vp \in \left\{ \begin{array}{ll} G(z,\delta), & \text{if } z_x < \delta_x \text{ and } z \neq N \\ G(\delta,z), & \text{if } \delta_x < z_x \text{ and } \delta \neq N \\ b_1(z,\delta), & \text{otherwise} \end{array} \right\}$$

(c) $\Sigma = SE$*:*

$$vp \in \left\{ \begin{array}{ll} G(z,\delta), & \text{if } z_x > \delta_x \text{ and } z \neq E \\ G(\delta,z), & \text{if } \delta_x > z_x \text{ and } \delta \neq E \\ b_1(z,\delta), & \text{otherwise} \end{array} \right\}$$

(d) $\Sigma = SW$*:*

$$vp \in \left\{ \begin{array}{ll} G(z,\delta), & \text{if } z_x > \delta_x \text{ and } z \neq S \\ G(\delta,z), & \text{if } \delta_x > z_x \text{ and } \delta \neq S \\ b_1(z,\delta), & \text{otherwise} \end{array} \right\}$$

To derive a complete algorithm for the computation of the function CHECKVP we have to do two things:

(i) We have to develop a criterion that enables us to decide whether z lies inside, outside or on the boundary of S.

(ii) We need a test that decides for a given corner Γ of S and a point e on the boundary of S, if $e = \Gamma$.

The second task will be solved by Lemma 3.12, the first one will be performed by Proposition 3.10. To apply that proposition, it will be nessecary to rename a, b, c into p, q, r such that p and q lie on opposite sides Σ_p resp. Σ_q of S, and r lies on a side between Σ_p and Σ_q. That a, b, c are always in such a constellation, and how to determine it, is told by the following two lemmata. Again, the first one describes a simple case, namely the one in which no two points of a, b, c are located on a critical line. Then, the three points a, b and c lie on three different sides of S, and one easily proves the following lemma.

Lemma 3.8 *Let $a, b, c \in \mathbb{R}^2$ have the property that no two of the three points are located on a critical line. Choose $p, q \in \{a, b, c\}$ such that $d_1(p, q)$ is the maximal distance between any two of the three points a, b, c.*
Then we have $D = d_1(p, q)$, and p and q are located on opposite sides of S. The third point r lies between p and q.

Now, we come to the more complicated case that there are (at least) two points among a, b, c lying on a critical line. The key to handle this case is to "perturb" a, b, c slightly in order to return to the easy case (actually, we just perturb b).

Due to the fact that $vp(a,b,c)$ is supposed to exist it follows from the definition of the bisector function b_1 that it is impossible for all three points a,b,c to be on one common critical line. Let us assume that a and b are located on a critical line. If c neither lies on a critical line with a nor with b then we say that one "critical pair" exists. Otherwise, we have (at least) two critical pairs, and we assume that b and c are located on a common critical line. Note that these two critical lines (the one of a,b and the one of b,c) cannot have the same slope, since otherwise a,b,c would lie on a common critical line. Hence, we assume (in the case of two critical pairs) that a and b are located on a critical line having slope $+1$, and b and c on one with slope -1.

The following two tables (for the two cases of one or two critical pairs) define the "ε-perturbation" of b, where in the first one the "$+$" and "$-$" in the first column mean that a and b lie on a critical line having slope $+1$ resp. -1.

	$a_x < b_x$	$a_x > b_x$
$+$	$b'_x := b_x \ , \ b'_y := b_y + \varepsilon$	$b'_x := b_x \ , \ b'_y := b_y - \varepsilon$
$-$	$b'_x := b_x + \varepsilon \ , \ b'_y := b_y$	$b'_x := b_x - \varepsilon \ , \ b'_y := b_y$

Table 1: One critical pair

	$a_x < b_x$	$a_x > b_x$
$b_x < c_x$	$b'_x := b_x - \varepsilon \ , \ b'_y := b_y$	$b'_x := b_x \ , \ b'_y := b_y - \varepsilon$
$b_x > c_x$	$b'_x := b_x \ , \ b'_y := b_y + \varepsilon$	$b'_x := b_x + \varepsilon \ , \ b'_y := b_y$

Table 2: Two critical pairs

The following lemma (that one can prove by examination of the relative positions of a,b,c in some case distinctions) gives the method how to rename a,b,c to p,q,r in the demanded way in the case of existence of critical pairs.

Lemma 3.9 *Let $a,b,c \in \mathbb{R}^2$ be in such positions, that a and b lie on a common critical line, and that in the case of the existence of another critical pair b,c is such a pair. If more than one critical pairs exist then let a,b lie on a critical line having slope $+1$ and b,c*

on one with slope -1. *Let* $\varepsilon := \frac{1}{2}$ *and let* b' *be the* ε*-perturbation of* b. *Let* $p', q' \in \{a, b', c\}$ *be chosen such that* $d_1(p', q')$ *is the maximal distance between any two of the three points* a, b' *and* c.
Let

$$p := \left\{ \begin{array}{ll} b, & \text{if } p' = b' \\ p', & \text{otherwise} \end{array} \right\} \quad \text{and} \quad q := \left\{ \begin{array}{ll} b, & \text{if } q' = b' \\ q', & \text{otherwise} \end{array} \right\}.$$

Then $D = d_1(p, q)$ *holds, and* p *and* q *are located on opposite sides of* S. *The third point* r *lies between* p *and* q.

The following proposition states how to compute (with the knowledge of the two preceding lemmata) the position of z relative to S.

Proposition 3.10 *Let* p *and* q *be located on opposite sides* Σ_p *and* Σ_q *of* S, *and let* r *lie on a side between* Σ_p *and* Σ_q *of* S *having slope* $\tau \in \{+, -\}$. *Let for* $i \in \{p, q, r\}$

$$t_i := \left\{ \begin{array}{lll} IN, & \text{if} & d_1(z, i) < D \\ ON, & \text{if} & d_1(z, i) = D \\ OUT, & \text{if} & d_1(z, i) > D \end{array} \right\}.$$

Let furthermore

$$\delta := \left\{ \begin{array}{llll} \uparrow, & \text{if} & p \text{ or } q & \text{are above } g_\tau(r) \\ \downarrow, & \text{if} & p \text{ or } q & \text{are below } g_\tau(r) \end{array} \right\}$$

(p *and* q *can neither lie on different sides of* $g_\tau(r)$, *nor both on* $g_\tau(r)$*) and*

$$\gamma := \left\{ \begin{array}{llll} \uparrow, & \text{if} & z & \text{is above } g_\tau(r) \\ \circ, & \text{if} & z & \text{is on } g_\tau(r) \\ \downarrow, & \text{if} & z & \text{is below } g_\tau(r) \end{array} \right\}.$$

Finally, let

$$t'_r := \left\{ \begin{array}{lll} IN, & \text{if} & \gamma = \delta \\ ON, & \text{if} & \gamma = \circ \\ OUT, & \text{otherwise} & \end{array} \right\}.$$

Then z *is located*

in the interior of S, *if all* t_p, t_q, t_r, t'_r *are* IN
in the exterior of S, *if any of* t_p, t_q, t_r, t'_r *is* OUT,
on S, *otherwise.*

The value τ in the preceding proposition can be computed as described in the following remark.

Remark 3.11 *If p and q lie on opposite sides Σ_p and Σ_q then we have (note that p, q, r do not lie on a common side):*

$$\begin{array}{lll} \{\Sigma_p, \Sigma_q\} = \{SW, NE\}: & r \in NW, & \text{if } p \text{ od } q \text{ are below } g_+(r) \\ & r \in SE, & \text{if } p \text{ od } q \text{ are above } g_+(r) \\ \{\Sigma_p, \Sigma_q\} = \{NW, SE\}: & r \in NE, & \text{if } p \text{ or } q \text{ are below } g_-(r) \\ & r \in SW, & \text{if } p \text{ or } q \text{ are above } g_-(r) \end{array}$$

The question if a point s lies above or below a line $g_\varepsilon(t)$ ($\varepsilon \in \{+, -\}$) may be decided as follows:

$$\begin{array}{lll} \varepsilon = +: & s \text{ lies above } g_+(t), & \text{if } s_y - s_x > t_y - t_x \\ & s \text{ lies below } g_+(t), & \text{if } s_y - s_x < t_y - t_x \\ \varepsilon = -: & s \text{ lies above } g_-(t), & \text{if } s_y + s_x > t_y + t_x \\ & s \text{ lies below } g_-(t), & \text{if } s_y + s_x < t_y + t_x \end{array}$$

The final lemma in this section provides the needed test if a given point e on the boundary of S is a certain corner Γ of S.

Lemma 3.12 *Let $\Gamma \in \{W, N, E, S\}$ and $z \in S$, as well as*

$$z' := \left\{ \begin{array}{llll} z & + & (D, 0), & \text{if } \Gamma = W \\ z & + & (0, -D), & \text{if } \Gamma = N \\ z & + & (-D, 0), & \text{if } \Gamma = E \\ z & + & (0, D), & \text{if } \Gamma = S \end{array} \right\}.$$

Let furthermore t_p, t_q, t_r and t'_r be as in Proposition 3.10 with z' instead of z. Then we have:

$$z = \Gamma \text{ if and only if } t \neq OUT \text{ holds for all } t \in \{t_p, t_q, t_r, t'_r\}.$$

Now, we have introduced all the needed tools to propose the desired algorithm for computing the function CHECKVP.

Funktion CHECKVP

INPUT : $z, a, b, c \in \mathbb{R}^2$
OUTPUT : IN, if $\omega \in \{a, b, c\}$ exists such that $vp(a, b, c) \in G(z, \omega)$
OUT, if $\omega \in \{a, b, c\}$ exists such that $vp(a, b, c) \in G(\omega, z)$
ON, otherwise

(1) Check, if there are critical pairs under a, b, c;
if yes, rename a, b, c as decribed above.
(2) Determine p, q, r and D according to Lemma 3.8 resp. 3.9.
(3) Determine t_p, t_q, t_r and t'_r as in Proposition 3.10.
(4) if $t =$ IN holds for all $t \in \{t_p, t_q, t_r, t'_r\}$ then return TRUE
(5) if $t \in \{t_p, t_q, t_r, t'_r\}$ exists such that $t =$ OUT holds then return FALSE
(6) for each $\omega \in \{a, b, c\}$ do
{
(7) By using Lemma 3.7 determine if
$vp \in G(z, \omega)$, $vp \in G(\omega, z)$ or $vp \in b_1(z, \omega)$.
(8) if $vp \in G(z, \omega)$
(9) return IN
(10) if $vp \in G(\omega, z)$
(11) return OUT
}
(12) return ON

4 A Gift: Implementation for the Maximum Metric

Let $d_\infty : \mathbb{R}^2 \times \mathbb{R}^2 \longrightarrow \mathbb{R}$ be the Maximum metric, i.e., $d_\infty(p, q) = \max\{|q_x - p_x|, |q_y - p_y|\}$ for all $p, q \in \mathbb{R}^2$.

Again, let $\Omega \subset \mathbb{R}^2$ be a finite set of generators.

Definition 4.1 *The mapping $\psi : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ is defined by $\psi(p) := \frac{1}{2}(p_x + p_y, p_y - p_x)$ for all $p \in \mathbb{R}^2$.*

ψ is a bijection that is continuous in both directions (according to the canonical topology). The following proposition allows us to reduce the problem in the Maximum metric to a problem in the Manhattan metric, as already mentioned by JÜNGER, REINELT AND ZEPF [1]. It holds due to the fact that ψ maps the "1-sphere" in the Maximum metric onto the "1-sphere" in the Manhattan metric (and vice versa).

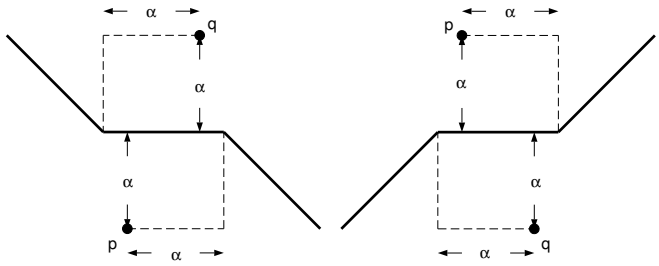
Proposition 4.2 *$d_\infty(p, q) = d_1(\psi(p), \psi(q))$ holds for all $p, q \in \mathbb{R}^2$.*

For $p, q \in \mathbb{R}^2$ $(p \neq q)$ we define $b_\infty(p, q) := \psi^{-1}(b_1(\psi(p), \psi(q)))$.
Since ψ is a continuous bijection, the property of being a bisector function carries over from b_1 to b_∞.

Proposition 4.3 *The function b_∞ is a bisector function.*

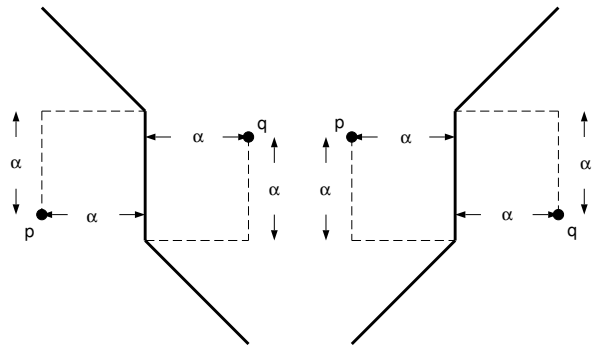
Let be $p, q \in \mathbb{R}^2$ $(p \neq q)$. Then $b_\infty(p, q)$ looks as follows (w.l.o.g. let $p_x \leq q_x$):

1. $|p_x - q_x| < |p_y - q_y|$



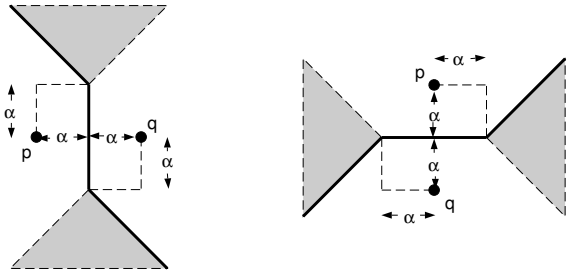
$(\alpha := \frac{|p_y - q_y|}{2})$

2. $|p_x - q_x| > |p_y - q_y|$



$(\alpha := \frac{|p_x - q_x|}{2})$

3. $|p_x - q_x| = |p_y - q_y|$



As a bijection, ψ commutes with intersection of sets. Hence, by applying the transformation ψ, the Voronoi regions in the L_1-Voronoi diagram of Ω transform to the Voronoi regions of the L_∞-Voronoi diagram of Ω. Due to the continuity of ψ, also the closures of these regions transform to each other. Hence, we get the final proposition.

Proposition 4.4 *If dt is a Delaunay triangulation of $\psi(\Omega)$ (according to the bisector function b_1), then dt is (via identification of Ω with $\psi(\Omega)$ using the bijection ψ) a Delaunay triangulation of Ω according to the bisector function b_∞.*

5 Computational Results

We implemented the proposed algorithm for computing Delaunay triangulations in the Manhattan metric and in the Maximum metric, as well as in the Euclidean metric (cf. JÜNGER, KAIBEL AND THIENEL [2]. There we describe how we build a numerically robust algorithm basing on the abstract algorithmic frame we proposed in section 2) on a SUN SPARCstation 10 model 41 using the programming language C. We chose SUN's compiler `acc (1.0)` with the compiler optimization option `-fast`. By sending an e-mail request to `kaibel@informatik.uni-koeln.de` you can obtain the library `delvor` (together with a short documentation) that provides functions to compute Delaunay triangulations in Manhattan metric, Maximum metric or Euclidean metric, as well as functions to calculate the corresponding Voronoi diagrams.

We give a brief overview in two tables of our computational experiences with this code. They both show running times of our program for Manhattan metric and Maximum metric, where we give both times, for calculating the function CHECKVP either by explicit computation of the Voronoi point (columns labeled "$+vp$") or without its explicit computation (columns labeled "$-vp$").

Table 3 deals with instances arising from the TSPLIB [7]. The first column contains the name of the instance, where the number in that name is always the number of generators.

It is a little striking that the difference between the running times for computation with and without explicit determination of the Voronoi points is smaller in case of the Maximum metric. In our opinion, this is due to the reason that many instances in the TSPLIB have lots of generators lying on lines being parallel to the coordinate-axes. Applying the transformation ψ of Section 4 yields many critical pairs. In case of no two out of a, b, c lying on a common critical line the method for computing CHECKVP we described in Section 3 finishes nearly immediately after detection of that "nice case", while the method with explicit computation of the Voronoi point still has to perform some comparisons in order to get rid of those "$|\ldots|$" in the equation system. If we are not in the "nice case", both methods have to make a similar number of comparisons, and therefore, if the "bad cases" outweigh, the difference of efficiency of the two methods gets smaller.

Table 4 shows the same running time data for generator sets being uniformly pseudo-randomly distributed in $[0,1]^2$. The first column contains the number of generators. Here, we do not observe a great difference between the times for Manhattan metric and Maximum metric, which supports our interpretation of the results for the TSPLIB instances.

Problem	L_1 (+vp)	L_1 (-vp)	L_{inf} (+vp)	L_{inf} (-vp)
rl1323	0.53	0.33	0.50	0.45
nrw1379	0.52	0.35	0.52	0.35
fl1400	0.55	0.37	0.57	0.43
u1432	0.57	0.43	0.55	0.55
fl1577	0.63	0.40	0.60	0.57
d1655	0.72	0.43	0.63	0.60
vm1748	0.67	0.48	0.68	0.60
u1817	0.80	0.50	0.70	0.70
rl1889	0.72	0.48	0.70	0.65
d2103	0.97	0.60	0.82	0.83
u2152	0.95	0.60	0.82	0.78
u2319	1.05	0.77	0.95	0.88
pr2392	0.92	0.62	0.92	0.75
pcb3038	1.17	0.75	1.17	0.83
fl3795	1.75	1.12	1.62	1.48
fnl4461	1.68	1.12	1.73	1.13
rl5915	2.35	1.55	2.23	2.22
rl5934	2.33	1.55	2.25	2.15
d6960	2.83	1.90	2.80	2.28
pla7397	3.28	2.25	2.75	2.75
rl11849	5.02	3.33	4.80	4.42
brd14051	5.58	3.85	6.58	4.07
d18512	7.28	4.90	7.65	5.07
pla33810	15.50	10.35	13.72	12.48
pla85900	39.85	27.02	34.82	32.23

Table 3: Instances of the TSPLIB

#gen	L_1 (+vp)	L_1 (-vp)	L_{inf} (+vp)	L_{inf} (-vp)
4	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.02
16	0.00	0.02	0.02	0.02
32	0.02	0.00	0.02	0.00
64	0.02	0.00	0.03	0.02
128	0.02	0.03	0.05	0.02
256	0.08	0.05	0.10	0.07
512	0.18	0.12	0.22	0.12
1024	0.38	0.23	0.42	0.25
2048	0.83	0.52	0.88	0.53
4096	1.65	1.00	1.72	1.05
8192	3.43	2.08	3.58	2.15
16384	7.08	4.35	7.45	4.57
32768	14.32	9.10	15.53	9.43
65536	28.85	18.15	29.95	21.03
131072	62.23	40.28	65.10	40.10

Table 4: Instances arising from pseudo-randomly distributed generators

6 Conclusions

We have built the abstract algorithmic frame (basing on our definition of an abstract Voronoi diagram) in order to derive fast and well implementable algorithms for computing Delaunay triangulations in Euclidean metric, Manhattan metric, and Maximum metric. Of course, the abstract definition admits many other bisector definitions. Once one has defined such a bisector function, it "still remains" to implement procedures for inserting and removing the dummies, as well as for calculating the two functions CLOSER and CHECKVP. It may be interesting to examine, if it is possible to get a computer code for computing Delaunay triangulations and Voronoi diagrams in any L_p-metric ($p = 1, 2, 3, \ldots$) using our framework.

References

[1] M. Jünger, G. Reinelt and D. Zepf: Computing Correct Delaunay Triangulations, *Computing* 47 (1991), 43-49.

[2] M. Jünger, V. Kaibel and S. Thienel: A Practical Method for Computing Correct Delaunay Triangulations in the Euclidean Metric, Report No. 94.158, Angewandte Mathematik und Informatik, Universität zu Köln (1994).

[3] V. Kaibel: Delaunay-Triangulation in verschiedenen Metriken, Diploma thesis, Institut für Informatik, Universität zu Köln (1993).

[4] R. Klein : Concrete and Abstract Voronoi Diagrams, *Springer LNCS 400* (1989).

[5] R. Klein, K. Mehlhorn und S. Meiser : Randomized Incremental Construction of Abstract Voronoi Diagrams, *Informatik: Festschrift zum 60. Geburtstag von Günter Hotz, hrsg. von J. Buchmann, H. Ganzinger und W.J. Paul*, Stuttgart (1992), 283-308.

[6] T. Ohya, M. Iri und K. Murota: Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparisons of Various Algorithms, *Journal of the Operations Research Society of Japan* 27 (1984), 306-337.

[7] G. Reinelt: TSPLIB – A Traveling Salesman Problem Library, *ORSA Journal on Computing 3* (1991), 376-384.

[8] K. Sugihara: A Simple Method for Avoiding Numerical Errors and Degeneracy in Voronoi Diagram Construction, *Research Memorandum RMI 88-14, Faculty of Engineering, University of Tokyo* (1988).

[9] K. Sugihara und M. Iri: Geometric Algorithms in Finite-Precision Arithmetic, *Research Memorandum RMI 88-10, Faculty of Engineering, University of Tokyo* (1988).

Michael Jünger
Institut für Informatik
Universität zu Köln
Pohligstr. 1
D-50969 Köln
Germany
Telephone: 49 221 4705313
e-mail: mjuenger@informatik.uni-koeln.de

Volker Kaibel
Institut für Informatik
Universität zu Köln
Pohligstr. 1
D-50969 Köln
Germany
Telephone: 49 221 4705314
e-mail: kaibel@informatik.uni-koeln.de

Stefan Thienel
Institut für Informatik
Universität zu Köln
Pohligstr. 1
D-50969 Köln
Germany
Telephone: 49 221 4705307
e-mail: thienel@informatik.uni-koeln.de