Proximity structures in the fixed orientation metrics

Christian Wulff-Nilsen[*]

Abstract

We present algorithms computing two types of proximity structures in the plane with a fixed orientation metric. Proximity structures have proven useful for Steiner tree heuristics in the Euclidean plane and may play a similar role for the fixed orientation metrics where Steiner trees are important in the area of VLSI design. We show how to find an all nearest neighbour graph $NNG(Z)$ of a set Z of n points in $O(an \log n)$ time using $O(n)$ space where a is the number of fixed orientations. The algorithm does not use the Voronoi diagram of Z. We present an algorithm that computes the Gabriel graph $GG(Z)$ of Z in $O(an \log n)$ time using $O(an)$ space under the assumption that no three points of Z are on a line parallel to one of the boundary edges of a σ-circle where σ is the set of fixed orientations. We show that if this assumption is not satisfied then $GG(Z)$ may contain $\Omega(n^2)$ edges. Both algorithms are optimal for constant a.

1 Introduction

Let σ be a set of a angles $\alpha_1, \ldots, \alpha_a$ sorted counter-clockwise. These angles are called *(fixed) σ-orientations* and a line, halfline, or line segment with a σ-orientation is said to be *σ-oriented*. The *σ-distance* d_σ between two points is the length of a shortest path of σ-oriented line segments between the points and d_σ is called the *σ-metric*. We refer to the σ-metric as a *fixed orientation metric*. We will sometimes write $|pq|_\sigma$ instead of $d_\sigma(p, q)$.

In recent years, fixed orientation metrics have played an important role in the area of VLSI design. This is due to the fact that the orientations of wires on a print are typically restricted to a finite set of fixed orientations. In VLSI routing, an important objective is to minimize the total length of an interconnection. This involves computing a Steiner tree. Since the Steiner tree problem is NP-hard, heuristics can be applied. Proximity structures have proven efficient for Steiner tree heuristics in the Euclidean plane [3] and could possibly play a similar role for Steiner tree heuristics in fixed orientation metrics.

We consider two types of proximity structures in the σ-metric. The first is an *all nearest neighbour graph $NNG(Z)$* of Z where Z is a finite set of n points or *terminals* in the plane. This graph has an edge (z_1, z_2) if and only if z_2 is a *nearest neighbour* of z_1, i.e. $d_\sigma(z_1, z_2) = \min_{z \in Z \setminus \{z_1\}} d_\sigma(z_1, z)$, or if z_1 is a nearest neighbour of z_2. If a terminal has more than one nearest neighbour, only one of them is picked as a nearest neighbour.

For $c \in \mathbb{R}^2$ and $r > 0$, the *σ-circle* $C_\sigma(c, r)$ with center c and radius r is the set of points having σ-distance at most r to c, i.e. $C_\sigma(c, r) = \{p \in \mathbb{R}^2 | d_\sigma(c, p) \leq r\}$. The second proximity structure we consider is the *Gabriel graph $GG(Z)$ of Z*. This graph has an edge (z_1, z_2) if and only if the σ-circle with center $c = \frac{1}{2}(z_1 + z_2)$ and radius $|cz_1|_\sigma = |cz_2|_\sigma$ contains no terminals in its interior.

We present algorithms computing $NNG(Z)$ and $GG(Z)$ in the σ-metric.

The organization of the paper is as follows. In section 2, we make various definitions and present some basic properties related to the σ-metric. In section 3, we show how to find $NNG(Z)$ in $O(an \log n)$ time using $O(n)$ space. We also show how to compute a similar all nearest neighbour graph and, using this, we present an $O(an \log n)$ time and $O(an)$ space algorithm computing $GG(Z)$ in section 4. To obtain these bounds, we make a simplifyng assumption about Z since otherwise, $GG(Z)$ may contain $\Omega(n^2)$ edges. Our algorithms are optimal for constant a as we show in section 6. Finally, we make some concluding remarks in section 7.

2 Definitions and basic properties

Associate with each $z \in Z$ the region of points not farther from z than from any other terminal in the σ-metric. This region is a (possibly unbounded) polygon [2] and is referred to as the *Voronoi polygon of z*. Since Voronoi polygons are not always disjoint, they need not define a partition of the plane.

We define a *Voronoi diagram of Z*, *$Vor(Z)$*, to be a partition of the plane into regions such that each region contains some $z \in Z$ together with (some of the) points not farther from z than from any other terminal in Z. The region containing z is a (possibly unbounded) polygon [2] called the *Voronoi region (of z)* and the line segments defining the boundary of a Voronoi region are called *Voronoi edges*. A Voronoi diagram $Vor(Z)$ of Z can be found in $O(an \log n)$ time using $O(an)$ space [2].

Given a point p, a *σ-cone of p* is the region bounded

[*]Department of Computer Science, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark, koolooz@diku.dk

by halflines emanating from p having orientations α_i and α_{i+1} respectively for some i (if $i = a$ then the orientations are α_a and α_1 respectively). We order the σ-cones of p counter-clockwise starting with the σ-cone having orientations a_1 resp. a_2.

Let q be another point. The union of σ-oriented lines through p resp. q partitions the plane into regions called *fields*. The *bisector of p and q* is the set of points r such that $d_\sigma(p, r) = d_\sigma(q, r)$. When the bisector of p and q is restricted to a field, it is either empty, a line segment, or the whole field [2]. In the latter case, we will refer to the field as a *bisector field of p and q*.

We will need another type of all nearest neighbour graph, denoted $NNG(M, M')$, where each point in M is incident to a nearest neighbour among points in M' for finite point sets M and M'.

As we shall see later, $GG(Z)$ may contain $\Omega(n^2)$ edges unless we make some simplifying assumption about Z. We will show that one such assumption is the following: no three terminals are on the same line parallel to a boundary edge of a σ-circle. If this assumption is satisfied, we say that the terminals are in *general position*.

3 All nearest neighbour graph

In this section, we show how to find $NNG(Z)$ in $O(an \log n)$ time using $O(n)$ space. This is done without computing $Vor(Z)$, as opposed to the algorithm suggested in [2]. The idea is to linearly transform Z by mapping σ-cones to north-east (NE) quadrants in such a way that finding a nearest neighbour in a σ-cone is equivalent to finding a nearest NE-neighbour in the transformed terminal set in the L_1-metric. Nearest NE-neighbours are then found using the algorithm suggested in [1].

We will briefly describe the algorithm in [1] since we need to modify it later. It finds nearest north-east neighbours in the L_1-metric by partioning the given point set into a left and a right half, recursively finding nearest NE neighbours in the two halves, and then finding nearest NE neighbours of points in the left half among points in the right half. To do the latter, the algorithm keeps track of three pointers, *left*, *right*, and *min*. During the course of the algorithm, *left* advances down the list of points in the left half, *right* advances down the list of candidate nearest NE neighbours of *left* in the right half, and *min* keeps track of the nearest NE neighbour of *left* in the right half found so far. Pointer *left* makes only one pass through the points in the left half. Similarly, *right* makes only one pass through the points in the right half.

Let K be the ith σ-cone of a point $p = (p_x, p_y)$ and let $q = (q_x, q_y)$ be a point in K. By rotating if necessary, we may assume that the right leg of K is aligned with the x-axis. Now, letting $s = \cot\theta$, where $\theta = \alpha_{i+1} - \alpha_i$, we have (see Figure 1)

$$d_\sigma(p, q) = (q_x - p_x) + (q_y - p_y)\left(\sqrt{s^2+1} - s\right).$$

Define $T_i : \mathbb{R}^2 \to \mathbb{R}^2$ by

$$T_i(x, y) = \left(\frac{x - sy}{\sqrt{s^2+1}}, y\right).$$

The following lemma shows that T_i is the linear transformation we are looking for.

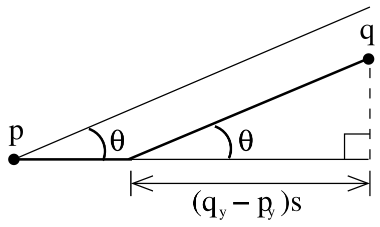

Figure 1: The σ-distance from p to q is the sum of lengths of the two σ-oriented line segments marked in bold.

Lemma 1 *With the above definitions, T_i is linear and maps K to the first quadrant of $T_i p$. If $T_i q$ is in the first quadrant of $T_i p$ then $q \in K$. If $q_1, q_2 \in K$ then $d_\sigma(p, q_1) \le d_\sigma(p, q_2)$ if and only if $L_1(T_i p, T_i q_1) \le L_1(T_i p, T_i q_2)$.*

The algorithm computing $NNG(Z)$ is as follows. For $i = 1, \ldots, a$, we make a call to the algorithm in [1] on $T_i(Z)$. By Lemma 1, this gives us nearest neighbours in the ith σ-cones of terminals in Z. To ensure $O(n)$ space requirement, we maintain only the nearest neighbour of each terminal found so far during the iterations.

The time spent in each of the a iterations is bounded by the time spent in the algorithm in [1] which is $O(n \log n)$. Thus, the total running time of our algorithm is $O(an \log n)$. The space used is $O(n)$, the amount required by the algorithm in [1].

In order to construct $NNG(M, M')$, where $|M| = m$ and $|M'| = m'$, we need to modify the update of pointers *left* and *right* in the algorithm in [1]. We do this as follows. The input point set is $M \cup M'$. Since we need to find nearest neighbours of points in M among points in M' pointer *left* needs to skip points in M' and pointer *right* needs to skip points in M. With this modification, we can construct $NNG(M, M')$ in $O(a(m+m') \log(m+m'))$ time using $O(m+m')$ space.

4 Gabriel graph

Next, we consider the Gabriel graph $GG(Z)$ of Z. We will show how to find this graph in $O(an \log n)$ time using $O(an)$ space. We assume that a supergraph

$S(Z)$ of $GG(Z)$ containing $O(n)$ edges is given. In section 5, we present an algorithm that finds such a supergraph in $O(an \log n)$ time and $O(an)$ space. However, we assume that terminals are in general position since otherwise, $GG(Z)$ may contain $\Omega(n^2)$ edges, see Figure 2.

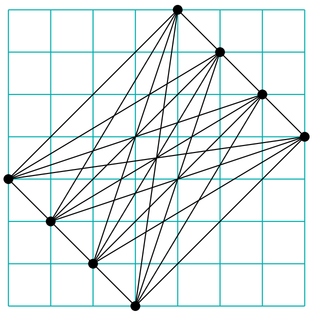

Figure 2: $GG(Z)$ with a quadratic number of edges, shown in the rectilinear metric.

Let M be the set of midpoints of edges of the supergraph $S(Z)$. An edge $e = (z_1, z_2) \in S(Z)$ belongs to $GG(Z)$ if and only if a nearest neighbour terminal of midpoint m of e is no closer to m than z_1 and z_2. From this it follows that, given $NNG(M, Z)$, which can be constructed in $O(an \log n)$ time using $O(n)$ space, determining whether e belongs to $GG(Z)$ takes $O(1)$ time. Hence, edges of $S(Z)$ not belonging to $GG(Z)$ can be discarded in $O(n)$ time. This shows that $GG(Z)$ can be found in $O(an \log n)$ time using $O(an)$ space.

5 A supergraph of $GG(Z)$

In the Euclidean metric, the *Delaunay graph of Z* is the straight-line dual of the Voronoi diagram of Z. The Delaunay graph of Z contains the Gabriel graph of Z. Unfortunately, this does not always hold in the σ-metric since a Voronoi region need not equal the corresponding Voronoi polygon, see Figure 3. The algorithm in [2] constructing $Vor(Z)$ does not always pick an entire bisector field but merely one of the halflines bounding this field when constructing the boundary of a Voronoi region P. Hence, the bisector parts bounding P will not always be included in P and the straight-line dual of $Vor(Z)$ will therefore not always contain $GG(Z)$.

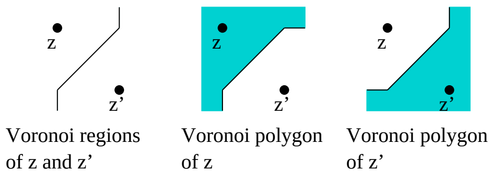

Figure 3: Voronoi regions and Voronoi polygons of terminals z and z', shown in the rectilinear metric.

To remedy this, we expand each Voronoi region of $Vor(Z)$ to its corresponding Voronoi polygon by including those parts of the bisector fields that belong to the Voronoi polygon. In the following, assume that $Vor(Z)$ is found using the algorithm in [2]. In the σ-metric, we define the *Delaunay graph $DG(Z)$ of Z* to be the graph having an edge between each pair of terminals whose Voronoi polygons overlap (possibly only on the boundaries). We will later see that $DG(Z)$ can be used as a supergraph $S(Z)$.

In order to efficiently construct $DG(Z)$, we will need a few results. Suppose Voronoi edges of each Voronoi region are directed clockwise. This involves replacing each edge of $Vor(Z)$ by two oppositely directed edges. Let $e = (p, q)$ be a Voronoi edge of the Voronoi region P of a terminal z_1 and let (q, p) belong to the Voronoi region P' of a terminal z_2. The following lemma gives a necessary and sufficient condition for expanding P at e.

Lemma 2 *With the above definitions, e bounds a bisector field F of z_1 and z_2 such that F does not intersect the interior of P if and only if there are two lines l and l′ with orientations α_i and α_{i+1} respectively for some i such that $e, z_2, p \in l$ and $z_1, p \in l'$ (Figure 4).*

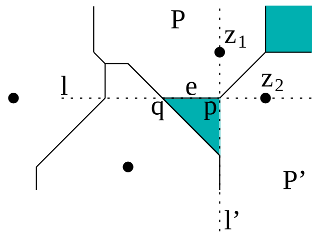

Figure 4: The situation in Lemma 2.

Next, we show that we only need to consider Voronoi regions adjacent to P when expanding.

Lemma 3 *Let z_1, z_2, and z_3 be distinct points. If F_1 is a bisector field of z_1 and z_2 and F_2 is a bisector field of z_1 and z_3 then $F_1 \cap F_2 = \emptyset$.*

We can quickly find those Voronoi edges of the neighbouring Voronoi region P' that belong to the expanded P as the next lemma shows.

Lemma 4 *With the above definitions, if e bounds a bisector field F of z_1 and z_2 such that F does not intersect the interior of P then the Voronoi edges of P′ intersected by F occur sequentially when walking around the Voronoi edges of P′ starting in p.*

Theorem 5 *Given $Vor(Z)$, $DG(Z)$ can be constructed in $O(an)$ time using $O(an)$ space and $DG(Z)$*

consists of $O(n)$ edges. Furthermore, $GG(Z) \subseteq DG(Z)$.

Proof. We construct $DG(Z)$ from $Vor(Z)$ by initializing the edge set of $DG(Z)$ to the empty set and doing the following for each Voronoi region P of $Vor(Z)$. Let z_1 be the terminal of P. For each edge $e = (p, q)$ of P, let z_2 be the terminal of the Voronoi region P' sharing edge e with P. We check if e bounds a bisector field F of z_1 and z_2 such that F intersects the interior of P (Lemma 2). If so, we add edge (z_1, z_2) to $DG(Z)$. Otherwise, we need to expand P at e. We do not do this explicitly since we only need to construct $DG(Z)$. Instead, we make a counter-clockwise detour into P', starting in q. Each edge in this detour is adjacent to the Voronoi region of a terminal $z_3 \neq z_2$. We add (z_1, z_3) to $DG(Z)$ if (z_1, z_3) is not already in $DG(Z)$. The detour stops when we encounter an edge not intersecting F or if there are no more edges in the tour (this may happen when P' is unbounded).

The correctness of the above algorithm follows from Lemma 3 and Lemma 4. The running time is bounded from above by the number of edges we traverse. An (undirected) edge e of $Vor(Z)$ is traversed at most four times, namely once in each of the two traversals of Voronoi regions adjacent to e and, by Lemma 3, at most twice in detours. Thus, the running time is $O(an)$. The space requirement is clearly $O(an)$.

Let $Vor'(Z)$ be the graph obtained from $Vor(Z)$ by merging edges of $Vor(Z)$ meeting in degree two vertices into one edge. Since no vertex of $Vor'(Z)$ has degree less than three, the number of edges in this graph is $O(n)$ by Euler's formula (the unbounded faces are easily handled). In the above algorithm, an edge traversal of a Voronoi region or a detour in $Vor(Z)$ induces an edge traversal in $Vor'(Z)$. An argument similar to the one above shows that the number of times we traverse any edge of $Vor'(Z)$ is bounded by a constant. Thus, $DG(Z)$ contains $O(n)$ edges.

Let $e = (z_1, z_2)$ be an edge of $GG(Z)$ and let $m = (z_1 + z_2)/2$ be the midpoint of e. Since no terminals are strictly closer to m than z_1 and z_2 it follows that the Voronoi polygons (the expanded Voronoi regions) of z_1 and z_2 overlap in m, hence $e \in DG(Z)$. □

6 Optimality of algorithms

Finally, we show that for fixed a, the two algorithms presented are optimal.

Space requirement is clearly optimal. That our GG-algorithm has optimal running time follows from reduction of sorting. Let $c_1, \ldots, c_n$ be n distinct real numbers. Consider the corresponding set Z of n terminals $z_{c_1}, \ldots, z_{c_n}$, where $z_{c_i} = (c_i, 0)$ for $i = 1, \ldots, n$. Let τ be the permutation of $c_1, \ldots, c_n$ such that $\tau(c_1) < \tau(c_2) < \cdots < \tau(c_n)$ and let $G = (Z, E)$ where E is the set of edges $(z_{\tau(c_i)}, z_{\tau(c_{i+1})})$ for $i = 1, \ldots, n-1$. Then $GG(Z) = G$, showing the optimality of the GG-algorithm.

Now, with Z as above, construct $NNG(Z)$. This graph is a collection of connected components, each of which corresponds to a sublist of the sorted list of numbers. Letting $Z' \subseteq Z$ be the set of left endpoint of each sublist, the fact that each terminal has a nearest neighbour implies $|Z'| \leq \frac{1}{2}|Z|$, see figure 5. We repeat the procedure on $NNG(Z')$. Eventually we

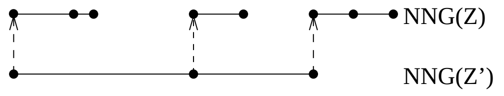

Figure 5: We can use $NNG(Z)$ to sort numbers.

end up with a single component. Using the generated graphs, we find the sorted list of n numbers in linear time. If we could construct $NNG(Z)$ in $o(n \log n)$ time, we could construct the above sequence of graphs in $o(n \lg n)$ time, a contradiction.

7 Conclusion

We presented algorithms computing an all nearest neighbour graph $NNG(Z)$ and the Gabriel graph $GG(Z)$ for a set Z of n points in the plane with a fixed orientation metric. For a constant number of fixed orientations, both algorithms have $O(n \log n)$ optimal running time and $O(n)$ optimal space requirement. We assumed that no three points of Z are on the same line parallel to a boundary edge of a σ-circle when constructing $GG(Z)$ and showed that $GG(Z)$ may contain $\Omega(n^2)$ edges in general.

It should be possible to extend both algorithms to the *weighted fixed orientation metrics* in which each fixed orientation has an associated weight.

Finally, it should be possible to relax the general position assumption without affecting the asymptotic time and space bounds of our algorithms by only requiring the following: at most a constant number of terminals are on the same line parallel to a boundary edge of a σ-circle.

References

[1] Leo J. Guibas and Jorge Stolfi. On computing all north-east nearest neighbors in the L_1 metric. Information Processing Letters 17 (1983) 219-223, North-Holland.

[2] P. Widmayer, Y. F. Wu and C. K. Wong. On some distance problems in fixed orientations. Siam J. Comput. Vol. 16, No. 4, August 1987.

[3] M. Zachariasen and P. Winter. Concatenation-Based Greedy Heuristics for the Euclidean Steiner Tree Problem. Algorithmica (1999) 25: 418-437.