Theoretical Computer Science 590 (2015) 73–85

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

New sequential and parallel algorithms for computing the β-spectrum ☆

Mirosław Kowaluk, Gabriela Majewska *

Institute of Informatics, University of Warsaw, Warsaw, Poland

ARTICLE INFO

Article history:
Received 13 December 2013
Received in revised form 15 March 2015
Accepted 18 March 2015
Available online 25 March 2015

Keywords:
Neighborhood graph
β-Skeleton
β-Spectrum
Delaunay triangulation
Parallel algorithms

ABSTRACT

β-skeletons, prominent members of the neighborhood graph family, have interesting geometric properties and various applications ranging from geographic networks to archeology. This paper focuses on computing the β-spectrum, a labeling of the edges of the Delaunay triangulation, $DT(V)$, which makes it possible to quickly find the lune-based β-skeleton of V for any query value $\beta \in [1, 2]$. We consider planar n-point sets V with L_p metric, $1 < p < \infty$. We present an $O(n \log^2 n)$ time sequential, and an $O(\log^4 n)$ time parallel, β-spectrum labeling. We also show a parallel algorithm, which for a given $\beta \in [1, 2]$ finds the lune-based β-skeleton in $O(\log^2 n)$ time. The parallel algorithms use $O(n)$ processors in the CREW-PRAM model.

1. Introduction

The β-skeletons [15] in R^2 belong to the family of proximity graphs, geometric graphs in which two vertices (points) are connected with an edge if and only if they satisfy particular geometric requirements. In the case of β-skeletons those requirements are dependent on a given parameter $\beta \geq 0$.

The β-skeletons are both important and popular because of many practical applications. The applications span a wide spectrum of areas: from geographic information systems to wireless ad hoc networks and machine learning. They also facilitate reconstructing shapes of two-dimensional objects from sample points, and are also useful in finding the minimum weight triangulation of point sets.

Gabriel graphs (1-skeletons), defined by Gabriel and Sokal [9], are an example of β-skeletons. Matula and Sokal [18] showed that Gabriel graphs can be computed from the Delaunay triangulation in a linear time.

The *relative neighborhood graph* (*RNG*) is another example of the β-skeleton graph family, for $\beta = 2$. The *RNG* was introduced by Toussaint [21] in the context of their applications in pattern recognition. Supowit [20] showed how to construct the *RNG* of a set of n points in $O(n \log n)$ time. Later, Jaromczyk and Kowaluk [12] described how to construct the *RNG* from the Delaunay triangulation *DT* for the L_p metric $(1 < p < \infty)$ in $O(n\alpha(n))$ time, where α is a functional inverse of Ackermann's function. This result was further improved to $O(n)$ time [13,16] for β-skeletons where $1 \leq \beta \leq 2$.

Hurtado, Liotta and Meijer [11] presented an $O(n^2)$ algorithm for the β-skeleton when $\beta < 1$. This algorithm is optimal, since for any $\beta < 1$ there exists a set of points in general position whose β-skeleton is a complete graph.

Two different forms of β-neighborhoods have been studied for $\beta > 1$ (see for example [2,8]) leading to two different families of β-skeletons: lune-based β-skeletons and circle-based β-skeletons. Regions defining circle-based and lune-based

☆ This research is supported by the ESF EUROCORES program EUROGIGA, CRP VORONOI.
* Corresponding author.
E-mail addresses: kowaluk@mimuw.edu.pl (M. Kowaluk), gm248309@students.mimuw.edu.pl (G. Majewska).

http://dx.doi.org/10.1016/j.tcs.2015.03.033
0304-3975/

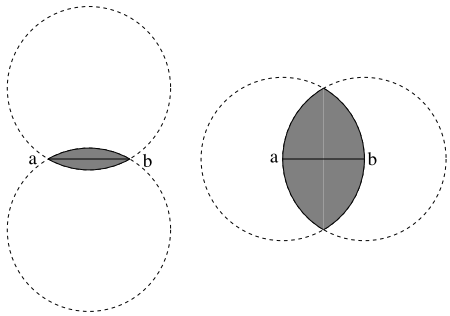

Fig. 1. The regions $N_2(a, b, \beta)$ for some β: $0 < \beta < 1$ (left) and $1 \leq \beta < \infty$ (right).

β-skeletons are different (the regions are unions or intersections of discs, respectively), so the algorithms for both cases are different.

For $\beta > 1$, the circle-based β-skeletons can be constructed in $O(n \log n)$ time from the Delaunay triangulation DT with a simple test to filter edges of the DT. On the other hand, so far the fastest algorithm for computing the lune-based β-skeletons for $\beta > 2$ runs in $O(n^{\frac{3}{2}} \log^{\frac{1}{2}} n)$ time [14].

With each pair of vertices u, v we can associate the largest value β (called the β-value of uv) for which the edge uv belongs to the β-skeleton. The set of all edges spanned by a set of points V (a complete graph) and labeled with their β-values is called the β-spectrum of V.

Hurtado, Liotta and Meijer [11] presented an algorithm for computing the lune-based and the circle-based β-spectrum for a set of n points in $O(n^2)$ time.

In this paper we show that an improvement of this time complexity is possible for selected intervals of the parameter β. For $\beta > 1$, the circle-based β-spectrum can be easily found in $O(n \log n)$ time from the Delaunay triangulation DT. We present an algorithm computing the lune-based β-spectrum for $1 \leq \beta \leq 2$ in $O(n \log^2 n)$ time. Our algorithm also uses the Delaunay triangulation DT but the analysis of $DT(V)$ edges is much more complicated than that for the circle-based β-skeletons. Its time complexity exceeds the complexity of the algorithm computing the lune-based β-skeleton for $1 \leq \beta \leq 2$.

On the other hand, there are very few parallel algorithms for proximity graphs [1,3,6,7,10]. Those algorithms are focused on the most known graphs, e.g., the Delaunay triangulation, the relative neighborhood graph and the minimum spanning tree. The algorithms are defined for different machine architectures, e.g., mesh [19] and CRCW PRAM [17]. Their time complexity usually is logarithmic or polylogarithmic. However, parallel algorithms for β-skeletons have not been studied and this paper makes an initial effort to fill this gap. We present two parallel algorithms using $O(n)$ processors in the CREW PRAM model. We focus on the lune-based β-skeletons for $1 \leq \beta \leq 2$. The first algorithm computes the β-spectrum in $O(\log^4 n)$ time and the second one constructs the β-skeleton in $O(\log^2 n)$ time.

The paper is organized as follows. The definition and basic properties of β-skeletons are introduced in Section 2. In Section 3 we present two algorithms, a sequential one and parallel one, for computing the β-spectrum for $1 \leq \beta \leq 2$. In Section 4 we describe a parallel algorithm which computes the β-skeleton. In Section 5 we discuss a way of using the ideas presented in this paper in distributed computations.

2. Basic facts and definitions

We consider a two-dimensional plane R^2 with the L_p metric (with distance function d_p), where $1 < p < \infty$.

Definition 1. For a given set of points $V = \{v_1, v_2, \ldots, v_n\}$ in R^2 and parameters $\beta \geq 0$ and p we define graph $G_\beta(V)$—called a lune-based β-skeleton—as follows: two points v_1, v_2 are connected with an edge if and only if no point from $V \setminus \{v_1, v_2\}$ belongs to the set $N_p(v_1, v_2, \beta)$ where (see Fig. 1):

1. for $\beta = 0$, $N_p(v_1, v_2, \beta)$ is equal to the segment v_1v_2;
2. for $0 < \beta < 1$, $N_p(v_1, v_2, \beta)$ is the intersection of two discs in L_p, each of them has radius $\frac{d_p(v_1v_2)}{2\beta}$ and their boundaries contain both v_1 and v_2;
3. for $1 \leq \beta < \infty$, $N_p(v_1, v_2, \beta)$ is the intersection of two L_p discs, each with radius $\frac{\beta d_p(v_1v_2)}{2}$, whose centers are in points $(\frac{\beta}{2})v_1 + (1 - \frac{\beta}{2})v_2$ and in $(1 - \frac{\beta}{2})v_1 + (\frac{\beta}{2})v_2$, respectively;
4. for $\beta = \infty$, $N_p(v_1, v_2, \beta)$ is the unbounded strip between lines perpendicular to the segment v_1v_2 and containing v_1 and v_2.

Furthermore, we can consider open or closed $N_p(v_1, v_2, \beta)$ regions that lead to *open* or *closed* β*-skeletons*. For example, the *Gabriel graph* is the closed 1-skeleton and the *relative neighborhood graph* is the open 2-skeleton.

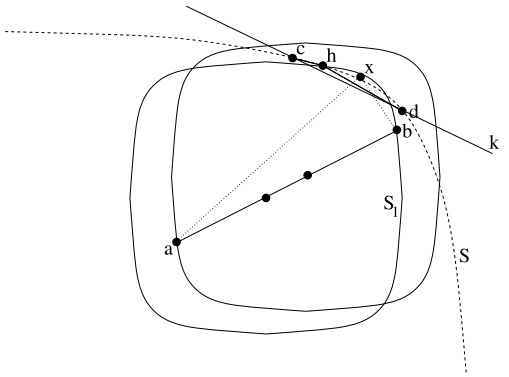

Fig. 2. If the edge cd twice intersects only one arc of the lune and intersects $\triangle axb$ then $cd \notin DT(V)$.

The following definition is important for our further considerations.

Definition 2. For an edge v_1v_2, the largest real number $\bar{\beta}$ such that no point in $V \setminus \{v_1, v_2\}$ belongs to $N_p(v_1, v_2, \bar{\beta})$ is called *the β-value* for v_1v_2. The set of all edges spanned by V, each labeled by its β-value is called *the β-spectrum* of V.

Fact 1. *For $1 \leq \beta \leq \beta' \leq 2$ the following inclusions hold true: $MST(V) \subseteq RNG(V) \subseteq G_{\beta'}(V) \subseteq G_{\beta}(V) \subseteq GG(V) \subseteq DT(V)$.*

Additionally, *the β-spectrum of V for $\beta \in [x, y]$* is the maximum subset of β-spectrum of V such that β-values for all edges in this subset satisfy $x \leq \beta$-value $\leq y$.

Let us assume that points in V are in general position. The following fact connecting β-skeletons with the minimum spanning tree $MST(V)$ and Delaunay triangulation $DT(V)$ of V was proved by Kirkpatrick and Radke [15]:

Definition 3. The intersection points of the two circles defining $N_p(a, b, \beta)$ are called *vertices of the lune*. The arcs between vertices of the lune are called *arcs of the lune*. We say that $v \in V$ *eliminates* edge $ab \in DT(V)$ if and only if v belongs to $N_p(a, b, \beta)$. The edge ab divides the lune $N_p(a, b, \beta)$ into two halves.

Lemma 1. *If in a given half of the lune $N_p(a, b, \beta)$ there exists a point x eliminating ab then there is no edge of $DT(V)$ which intersects both $\triangle axb$ and exactly one arc of the lune.*

Proof. Let us suppose that there exists an edge cd in $DT(V)$ which intersects $\triangle axb$ and cd intersects exactly one arc of the lune $N_p(a, b, \beta)$ twice (the one defined by a circle S_1). From the definition of the lune, the edge ab lies inside the circle S_1. Let S be the circle circumscribed on $\triangle cdh$ where $\triangle cdh \in DT(V)$ and h and x lie on the same side of the line k containing an edge cd; because the edge cd is not the edge of the convex hull of V the vertex h always exists (see Fig. 2). Let D_1 and D be the interiors of S_1 and S respectively. Note that circle S does not have x in its interior. Therefore, S intersects S_1 in points on the same side of the line k as point x. Hence, the part of D_1 on the opposite side of k from x belongs to D and consequently $a, b \in D$. This is a contradiction with respect to properties of $DT(V)$. □

$\widetilde{DT}(V)$ is the graph with vertices corresponding to the edges of $DT(V)$ and arcs connecting two vertices if and only if their corresponding edges in $DT(V)$ belong to the same triangle. For each vertex $v \in V$, we construct a directed path $path(v, e_0) = (e_0, \ldots, e_s)$ in $\widetilde{DT}(V)$ as follows:

- e_0 is an edge opposite to v in some triangle in $DT(V)$,
- v eliminates edges e_i for $0 \leq i \leq s$ and
- edges e_i and e_{i+1} are incident in $\widetilde{DT}(V)$ for $0 \leq i \leq s-1$.

Thanks to [12,13] we have the following definition and results that will be important in the further discussion:

Definition 4. $path(v, e_0) = (e_0, \ldots, e_s)$ is called *the elimination path* for v.

Lemma 2.

1. *Once two elimination paths meet, they never split. That is, if two paths share a common edge e_j, all of the edges that follow e_j in these two elimination paths are identical.*
2. *Every eliminated edge belongs to at least one elimination path.*

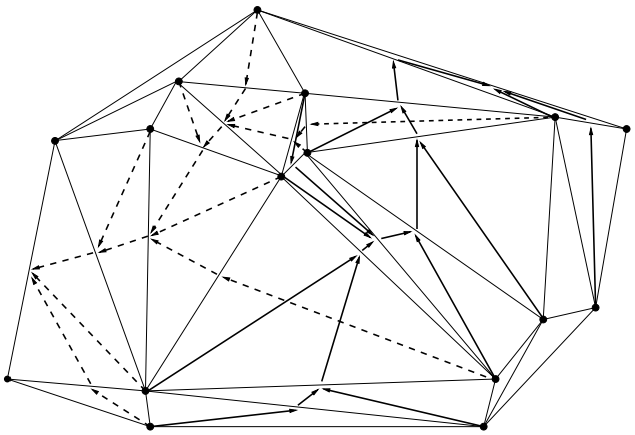

Fig. 3. Generalized elimination paths.

Theorem 1. *The following equality holds*

$$G_\beta(V) = DT(V) \setminus \bigcup_{\triangle uvw \in DT(V)} path(v, uw),$$

where we view both sides of the equality as undirected graphs.

Elimination paths are critical in designing our β-skeleton algorithms. Our main strategy is to identify edges belonging to elimination paths and to find β values for which these edges are eliminated from $DT(V)$.

3. β-Spectrum for $\beta \in [1, 2]$

In this section we describe two algorithms for computing the β-spectrum for $1 \leq \beta \leq 2$. For this purpose we generalize elimination paths and we construct polygons containing $DT(V)$ triangles traversed by generalized paths which join in one path. Such polygons can have very complicated shapes. Therefore, in each union of generalized paths we extract only one path which divides the polygon into smaller parts. Each such part has at most a half of vertices of the previous polygon. We continue this process recursively. For vertices of each such subpolygon we create a hierarchical structure of Voronoi diagrams. It allows us to find for each edge of $DT(V)$ a set of vertices that are near this edge. In this set we find the vertex defining the smallest value of β for which the edge is eliminated.

Triangles $\triangle t_1$ and $\triangle t_2$ in $DT(V)$ are called neighbors if they have a common edge in $DT(V)$.

For each vertex v and edge e ($v \notin e$) which determine the triangle $t_0 \in DT(V)$, we construct a sequence of edges in $DT(V)$ that starts with $e_0 = e$. Each pair of consecutive edges in this sequence belongs to a triangle in $DT(V)$. Inductively, let us assume that we have already constructed a sequence $e_0, \ldots, e_i$ and for $i > 0$ let t_i in $DT(V)$ be defined by edges e_{i-1} and e_i. Furthermore, for $i \geq 0$ let t_{i+1} and t_i be neighbors that share edge e_i in $DT(V)$. As e_{i+1}, we select the longer one of the two edges of the triangle t_{i+1} other than e_i, if there exists such a longer edge. When we reach a triangle and an edge, such that its neighbor triangle does not exist, or we reach a base of an isosceles triangle, or the next part of constructed sequence becomes cyclic then the construction of the sequence ends (see Fig. 3). Edges of the elimination path for vertex v are contained in the sequence constructed for v and some edge that determines with v a triangle in $DT(V)$. Note also that if two sequences $e_0, \ldots, e_i$ and $e'_0, \ldots, e'_m$ contain the same edge $e = e_k = e'_l$ that corresponds to the same triangle $t = t_k = t'_l$ in $DT(V)$ then from this point one of these sequences will be contained in the other one.

Definition 5. A sequence of edges $e = e_0, \ldots, e_i$, constructed in the way described above, is called a *generalized elimination path* (*GEP*) for the vertex v and the edge e. The last edge in this sequence is called the *root* of the generalized elimination path.

By merging generalized paths in $DT(V)$, we form *generalized elimination trees* ($GET(V)$) and a set of all such trees is called *a generalized elimination forest* ($GEF(V)$). The root edge of *GEP* that is not the last edge of any other *GEP* in the same $GET(V)$ is called the root of the generalized elimination tree containing those paths.

Fact 2. *Each edge from $DT(V)$ is contained in at most two generalized elimination trees.*

Proof. Let $e = t_1 \cap t_2$, where t_1, t_2 are neighbors in $DT(V)$. Then all elimination paths passing e from t_1 to t_2 belong to the same elimination tree. It also holds true for paths passing e in the opposite direction. □

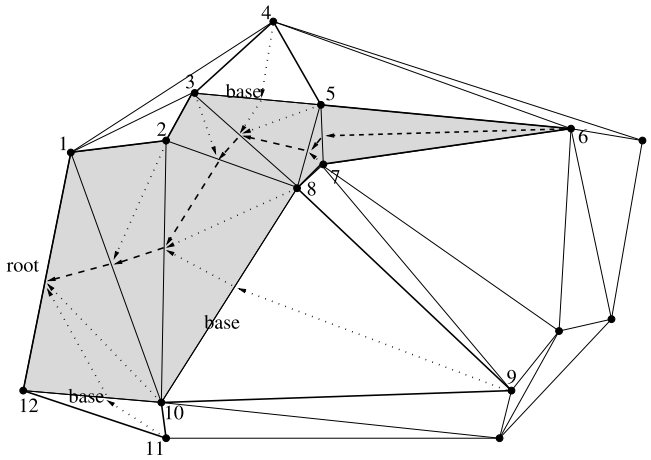

Fig. 4. Central polygon.

Fact 3. *The forest of generalized elimination trees can be constructed in linear time.*

Let edges of $DT(V)$ correspond to vertices of a graph and generalized elimination paths define sequence of directed edges of this graph. We start from a vertex with the outdegree equal 0 we construct a generalized elimination tree using the DFS algorithm. Since $DT(V)$ has $O(n)$ edges and each edge is visited at most twice, we can construct the forest of generalized elimination trees in linear time.

3.1. Sequential algorithm for computing β-spectrum for $\beta \in [1, 2]$

As it was mentioned before *GET* is a tree of the generalized elimination paths. *The polygon of GET* (*PGET*) is defined as the union of all the $DT(V)$ triangles that are traversed by paths belonging to the *GET*.

Definition 6. The vertices of the central polygon are ordered clockwise starting with 1 assigned to the left end of the root edge and going around the polygon to the right end of the root (see Fig. 4).

The *central polygon* for *GET* is a part of the *PGET* corresponding to the $DT(V)$ triangles intersected by the generalized elimination path in *GET* which starts from the middle vertex on the boundary of *PGET*.

We extract the central polygon from corresponding *PGET*. Then *PGET* splits into smaller polygons (unions of closed $DT(V)$ triangles). Recursively, we construct central polygons in each of those smaller polygons. The common edge of a pair of central polygons created in two consecutive steps of *PGET* partition are called *base* of the polygon generated in the later step (see Fig. 4).

Lemma 3. *All of the central polygons can be constructed in $O(n)$ time.*

Proof. Building all the generalized elimination trees takes $O(n)$ time. To traverse the tree and number all its vertices we also need $O(n)$ time. Dividing the polygons into central polygons can be done in linear time, since every triangle belonging to the analyzed polygon is tested only once. □

As the next step, for each central polygon, we construct its *logarithmic structure of Voronoi diagrams*, as follows. The vertices of the central polygon are numbered starting from the leftmost vertex of the root or the base edge in the *CPT*. We start by constructing Voronoi diagrams for individual vertices on the boundary of this central polygon. Then, Voronoi diagrams are constructed for sets of vertices labeled with numbers from the interval $[s2^k + 1, (s + 1)2^k]$ where $0 \leq s \leq \lfloor \frac{n}{2^k} \rfloor - 1$, $0 \leq k \leq \lfloor \log n \rfloor$. This structure will allow us to find vertices eliminating an analyzed edge in a faster and easier way.

Lemma 4. *Constructing the logarithmic structure of Voronoi Diagrams for a given central polygon takes $O(n \log n)$ time.*

Proof. The construction relies on the *divide and conquer* method and requires linear time in each of the $O(\log n)$ steps. □

Now we are ready to present the algorithm. At first, let us consider the Euclidean metric. The bisector of edge *ab* contains vertices of the lune $N_2(a, b, \beta)$ for all $1 < \beta \leq 2$ (for $\beta = 1$ the bisector is not defined by vertices of the lune and therefore

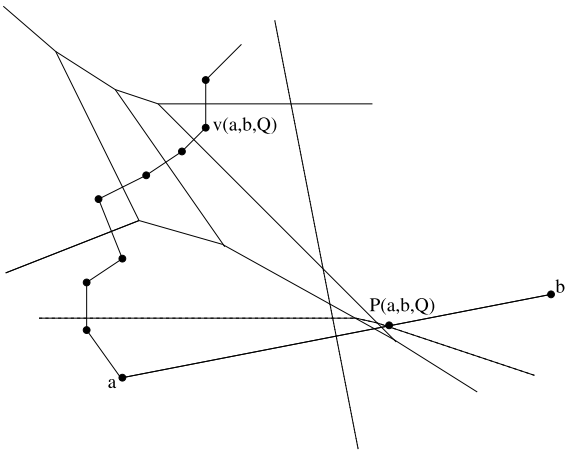

Fig. 5. Points $v(a, b, Q)$ and $P(a, b, Q)$. The dotted line denotes a border of the region which center is C.

we have to compute 1-skeleton separately in $O(n \log n)$ time and compare it with the results computed for $1 < \beta \leq 2$). The bisector and the line containing the segment ab divide the plane into four parts. We consider logarithmic structure of Voronoi diagrams for each closed quarter plane. Let Q be a given closed quarter plane and let c be the endpoint of edge ab such that $c \in Q$. We search for the closest point $P(a, b, Q)$ to c (see Fig. 5), which is an intersection point of the edge ab and the Voronoi diagram of $Q \cap V$ (if it exists).

Lemma 5. *If $P(a, b, Q) \in Q$ then the minimum β (β_{min}), such that the edge ab is eliminated, is smaller than 1. If the point $P(a, b, Q)$ does not exist then $\beta_{min} > 2$. If the point $P(a, b, Q)$ exists and does not belong to Q then it is the center of the circle defining the lune of $N_2(a, b, \beta_{min})$.*

Proof. Point $P(a, b, Q)$ is equidistant from c (an endpoint of ab) and some point $v \in V \cap Q \setminus \{c\}$ since it is on an edge of Voronoi diagram for $V \cap Q$. If $P(a, b, Q)$ and v belong to Q, then the circle with diameter ab and a center in the middle of ab contains v inside.

In other case, the lune, defined by circles with a radius of the length equal to $d_2(v, P(a, b, Q))$, has v and c on its border and no points from $V \cap Q$ inside.

If the point $P(a, b, Q)$ does not exist, then $N_2(a, b, 2) \cap (V \cap Q \setminus \{c\}) = \emptyset$, so $\beta_{min} > 2$. □

We analyze the boundary of the Voronoi region for c and elements of the logarithmic structure of the Voronoi diagrams for the central polygon's vertices belonging to the quarter plane Q (we investigate diagrams which contain the maximum number of centers and where neither of the two diagrams does not contains the same center). We obtain one candidate for β-value for each of the analyzed Voronoi diagrams in the logarithmic structure (Algorithm 1).

We traverse edges of the $DT(V)$ along the generalized elimination paths starting it at the leaves of the GET. Analyzing the logarithmic structure of the Voronoi diagrams for vertices of central polygon we search the border between the Voronoi region of certain vertex of the central polygon and the Voronoi region for c. Simultaneously we need to check if the center of a given Voronoi region is in the correct quarter-plane. If not, then we decrease the size of the examined set of points from the border of central polygon and continue the analysis. If we manage to find $P(a, b, Q)$ and $v(a, b, Q)$ is in the correct quarter-plane, then we save the β value. This way we get a candidate to be β_{min} and we continue the algorithm.

Note that the border of a connected part of a central polygon might consist of two non-intersecting parts (see Fig. 6). Therefore, we have to check respective parts of the border on both sides of the line k: first, we find the parts between ends of ab and the line k, then we analyze following fragments of the border which are on the second side of the line k.

Continuing from the point where the previous analysis has terminated, we traverse the Voronoi diagrams in one direction: we analyze vertices of the central polygon from the point c towards the intersection of the boundary of the central polygon with the bisector of the edge ab.

Lemma 6. *Let $v(a, b, Q)$ be the point from V that defines $P(a, b, Q)$. Points of V which are in the angle $\angle cP(a, b, Q)v(a, b, Q)$ (where $c = a$ or $c = b$ and $c \in Q$) do not affect the value of the minimum β for edges following ab in the generalized elimination path.*

Proof. Note that edges following ab in the generalized elimination tree are separated from the points in the angle $\angle cP(a, b, Q)v(a, b, Q)$ by Voronoi regions for $v(a, b, Q)$ and the vertex c. Hence, the minimum β for the following edges can be defined only by those points and the centers of Voronoi regions which were not examined yet. □

Algorithm 1: Procedure SEARCH.

Input: central polygon CP, edge e, query point c
Output: a value of a candidate for minimal β

```
procedure SEARCH(e,CP,c) ;
i := 0;
v := c;
candidate := ∞ ;
k := bisector of e;
while i > −1 do
    find the first Voronoi diagram D, which centers precede v on the boundary of the CP and the size of D is 2^i;
    w := last so far examined center in D on the same side of k as v ;
    if there exist centers of Voronoi diagram on the opposite side of k as v then
        i := i − 1 ;
    else
        compute candidate q for the minimal β (corresponding to some center of D) ;
        if q < candidate then
            candidate := q ;
        end
        store the location of a center defining q ;
        v := the last center of D ;
        i := i + 1 ;
    end
end
return(candidate)
```

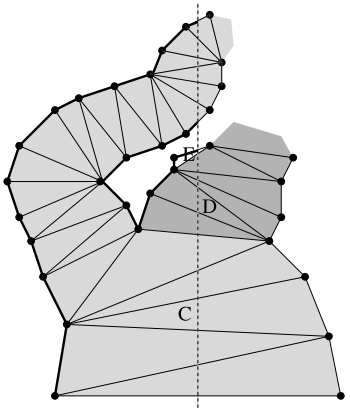

Fig. 6. Central polygons (gray C, dark gray D and white E), which need to be analyzed. The border of the left part of the gray figure consists of two pieces (thick lines).

Lemma 7. *To find the minimum β value for an edge ab in central polygon C it is enough to examine the borders of connected parts of C which are cut off by the edge ab and a line k passing through vertices of lune $N_2(a, b, \beta)$.*

If the minimum β value for ab is defined by points from a central polygon C', where $ab \notin C'$ then k cuts the base of C'.

Proof. Let $NM(a, b, \beta)$ be the maximum lune for ab such that $NM(a, b, \beta)$ has no vertices of C inside. Hence, point defining this lune belongs to the connected part of C separated from the rest of C by ab and k.

In the second case, due to Lemma 1, base of C' intersects both arcs of the lune $N_2(a, b, \beta)$, so it also has to intersect line k. □

We look for intersections of the bisector for the analyzed edge $ab \in DT(V)$ with bases of consecutive central polygons in the given *PGET*. We use the ray shooting approach [5]. Unfortunately, a generalized elimination path can interchange, i.e. it forms a cycle or traverses the same edges in different directions. In this case we analyze smaller parts of the central polygon which are simple polygons (see Fig. 7). Let us consider edges of *GEP* defining the analyzed central polygon of *PGET*. Let e be the edge of *GEP* that is the first to be traversed in both directions. We construct the first simple polygon by merging triangles of $DT(V)$ corresponding to the part of *GEP* from the root to the edge preceding the second appearance of the edge e in *GEP*. Then we consider the part of *GEP* starting from the edge next to the first appearance of the edge e in *GEP*. We continue the above mentioned procedure for this part of *GEP*.

Consecutive parts of the partition correspond to loops of the generalized elimination path. Hence, a ray shooted from a point x inside the central polygon intersects edges of this polygon that belong to the part of partition containing x or to the neighboring polygon. Moreover, the total size of polygons in the partition is at most twice greater than the size of

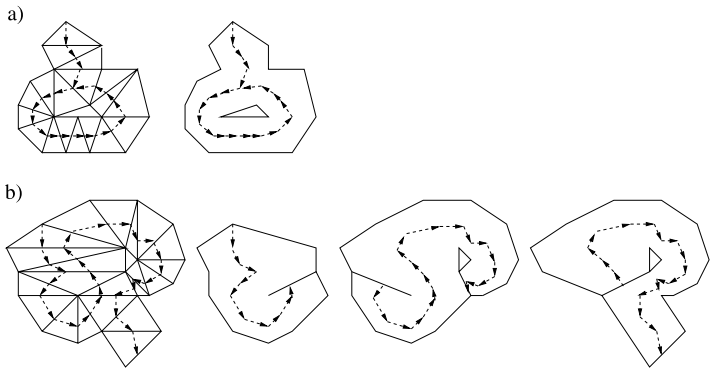

Fig. 7. Partition of the central polygon into smaller simple polygons.

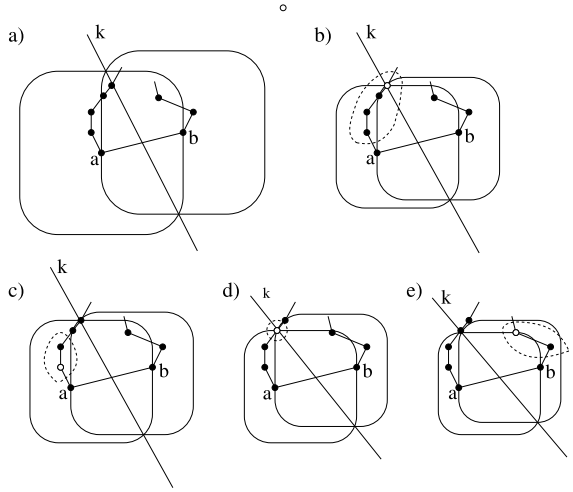

Fig. 8. The computing of candidates for minimum β-value of the edge ab: (a) the starting position for $\beta = 2$, (b)–(d) the first step of the algorithm, (e) the second step of the algorithm. White points define candidates for corresponding logarithmic structures of Voronoi diagrams.

the analyzed central polygon. Then the ray shooting problem can be solved in $O(\log n)$ time with a data structure of size $O(n)$ [5].

We also can find the intersection of edges of the central polygon with the given line k without partition of the central polygon. We test an intersection of the line k with convex hulls created for sets of points defining elements of the logarithmic structure of Voronoi diagrams. However, in this case we need $O(\log^2 n)$ time to find the intersection point.

In L_p metrics for $1 < p < \infty$, $p \neq 2$ vertices of regions $N_p(a, b, \beta)$ for a given edge ab and $1 \leq \beta < \infty$ form the Voronoi diagram for centers a and b. Line passing through the vertices of $N_p(a, b, \beta)$ for fixed $\beta \neq 1$ is not the bisector of ab and the direction of this line depends on the parameter β.

However, that makes no difference for the algorithm. We analyze the location of centers of Voronoi regions online, so we are not dependent on the direction of line k. The analysis of vertices of the central polygon is done in two steps as follows.

First, we consider vertices of the central polygon in the direction of the increasing parameter β. We use the logarithmic structure of Voronoi diagrams in the same way as for L_2 metric. We start with $\beta = 2$. If we find a vertex that belongs to $N_p(a, b, \beta)$ we decrease the value of β. Then we test if the analyzed vertex is on the proper side of the line k generated for the new value of β. The positive result allows us to analyze the next vertex in the same Voronoi diagram structure. In the other case we test vertices defining the Voronoi diagram for a smaller set of centers (see Fig. 8).

Next, we consider vertices of the central polygon in the direction of the decreasing parameter β. We start with the minimum β computed in the previous step. If we find a vertex that belongs to $N_p(a, b, \beta)$ we decrease the value of β (the line k walks away from the analyzed point).

Lemmas 6 and 7 do not depend on metrics. The minimum β value defines the direction of a line passing through vertices of the lune and allows finding the correct sequence of central polygons that should be tested.

Corollary 1. *β-Spectrum in the L_p metrics, where $1 \leq \beta \leq 2$, $1 < p < \infty$ and $p \neq 2$ can be computed in the same time as in the Euclidean metric (Algorithm 2).*

Algorithm 2: Sequential algorithm computing β-spectrum.

Input: $DT(V)$
Output: β-spectrum for V

construct the generalized elimination forest ;
construct polygons for generalized elimination trees and divide them into central polygons, store them in a central polygons tree *CPT* structure;
for *every central polygon* C **do**
  build a logarithmic Voronoi diagram structure for C ;
end
while T *is not empty* **do**
  $P :=$ a leaf of T;
  $T := T - \{P\}$;
  $Q :=$ queue of edges from P ;
  while Q *is not empty* **do**
    $ab :=$ POP(Q) ;
    END := false ;
    while *not END* **do**
      $x := a$;
      $y := b$;
      SEARCH(ab, P, x) ;
      c:=first vertex which precede x on the boundary of P, on the opposite side of a line passing through vertices of $N_p(a, b, \beta)$ than a;
      SEARCH(ab, P, y) ;
      d:=first vertex which precede y on the boundary of P, on the opposite side of a line passing through vertices of $N_p(a, b, \beta)$ than b;
      SEARCH(ab, P, c) ;
      SEARCH(ab, P, d) ;
      if *line passing through vertices of* $N_p(a, b, \beta)$ *intersects base* fg *of central polygon* P' **then**
        $P := P'$;
        $x := f$;
        $y := g$;
      else
        END:=true ;
      end
    end
    among computed candidates find minimal β ;
  end
end

Theorem 2. *β-Spectrum for a set V of n points in L_p metrics, where $1 \leq \beta \leq 2$ and $1 < p < \infty$ can be computed in $O(n \log^2 n)$ time.*

Proof. The generalized elimination forest and the central polygons forest can be found in $O(n)$ time, while it takes $O(n \log n)$ time to compute the logarithmic Voronoi diagrams structure.

In all of the steps of the algorithm *SEARCH* procedure examines no more than $O(\log^2 n)$ Voronoi diagrams per edge. However, for consecutive edges of a given central polygon the search is done in only one direction. By virtue of Lemma 6, this search needs $O(n \log n)$ time.

To construct the sequence of central polygons we can use the ray shooting approach [5]. The corresponding data structures for the central polygons can be constructed in $O(n)$ time. Since we need $O(\log n)$ time to find the next central polygon in a sequence, creating elements in all of the sequences takes $O(n \log^2 n)$ time. Hence, in total the algorithm needs $O(n \log^2 n)$ time. □

3.2. Parallel algorithm for computing β-spectrum for $\beta \in [1, 2]$

Due to results presented in the previous section β-skeletons can be constructed from β-spectrum in constant time, i.e. in total $O(\log^4 n)$ time, However, one can solve this problem faster.

In this section we show an $O(\log^2 n)$ time algorithm which computes β-spectrum of a set V of n points in the plane for $\beta \in [1, 2]$ using $O(n)$ processors in a CREW-PRAM model. We adapt the idea of the algorithm presented in the previous section. The big number of processors allows parallel performing of operations on the logarithmic Voronoi diagrams structures. For fast search of points $P(a, b, Q)$ and $v(a, b, Q)$ we have to construct an auxiliary data structure.

We start from constructing of the Delaunay triangulation for V; this takes $O(\log^2 n)$ time [3]. Next, by using pointer jumping, we create the generalized elimination trees and divide polygons corresponding to these trees into central polygons in $O(\log^2 n)$ time. Then we build the logarithmic structure of the Voronoi diagrams for all of these central polygons. It takes $O(\log^3 n)$ time in total, since each of $O(\log n)$ levels of the structure needs $O(\log^2 n)$ time.

Unfortunately, we cannot compute here candidates for β_{min} in the same way as in the sequential algorithm. Instead, we have to maintain an additional data structure for quick searches in all of the Voronoi diagrams.

We are interested in finding Voronoi regions that intersect a respective diagonal of the central polygon. We compute the convex hull *CH* of the set of centers of the Voronoi diagram [4] and the intersection of diagram edges with the border of *CH*.

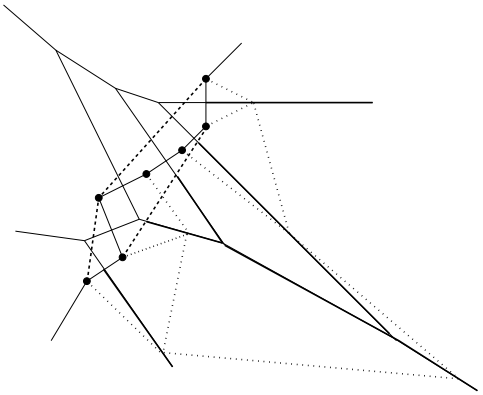

Fig. 9. Voronoi regions intersecting the border of the convex hull (dashed line) and the interior of the central polygon; and the scheme of the structure for $v(a, b, Q)$ search (dotted line).

Algorithm 3: Parallel algorithm for computing the β-spectrum.

Input: set of vertices V
Output: β-spectrum for V

compute $DT(V)$, construct generalized elimination forest, polygons connected to generalized elimination trees, central polygons, logarithmic structure of Voronoi diagrams and search structure;
for *every edge of* $DT(V)$ **do in parallel**
  while *a central polygon exists with vertices which can influence the value of the minimum* β **do**
    find correct Voronoi diagrams and candidates for minimum β defined by those diagrams;
    compute the current minimum β;
    construct the line passing through the vertices of the current lune and check if it defines another central polygon which should be checked;
  end
end

Each edge corresponds to a processor that performs the binary search; this takes $O(\log n)$ time. This way we construct a sequence of the centers belonging to regions intersecting the border of the convex hull and the interior of the central polygon. The auxiliary data structure has a form of a tree (see Fig. 9). We divide the set of centers into halves. Edges that separate the corresponding two sets of the Voronoi regions are ordered by the pointer-jumping procedure and stored in the node of the tree (this step takes $O(\log n)$ time). We repeat this procedure until the regions are separated.

Lemma 8. *Let Q be the part of the plane between ab and the line containing vertices of $N_p(a, b, \beta)$. The auxiliary data structure can be built in $O(\log^2 n)$ time. The points $P(a, b, Q)$ and $v(a, b, Q)$ can be found in $O(\log^2 n)$ time using this structure.*

Proof. The set of regions is partitioned in $O(\log n)$ steps. In each stage, the construction of the data structure takes $O(\log n)$ time. Hence, the data structure can be built in $O(\log^2 n)$ time.

We locate $P(a, b, Q)$ in $O(\log n)$ time using the binary search for testing its position with respect to the sequence of edges stored in the analyzed node of the structure. We find the Voronoi region (respectively $v(a, b, Q)$) whose intersection with the line passing through the edge ab contains $P(a, b, Q)$ in $O(\log n)$ steps. Hence, the total time is $O(\log^2 n)$. □

Corollary 2. *Building the auxiliary data structures for all diagrams in the logarithmic structure of Voronoi diagrams takes $O(\log^3 n)$ time.*

If for a given diagonal ab, $v(a, b, Q) \notin Q$ then we need to reduce the size of the examined Voronoi diagram. For each edge we check $O(\log n)$ diagrams in one central polygon. The next central polygon which can influence the β_{min} can be found in $O(\log n)$ time (Algorithm 3).

Theorem 3. *For a set V of n points we can compute the β-spectrum for $1 \leq \beta \leq 2$ in $O(\log^4 n)$ time using $O(n)$ processors in PRAM-CREW model.*

Proof. Building all of the structures takes $O(\log^3 n)$ time. The number of central polygons examined for a given diagonal is $O(\log n)$. Each processor assigned to this task has to check $O(\log n)$ Voronoi diagrams and each search takes $O(\log^2 n)$ time. Hence, the whole algorithm can be executed in $O(\log^4 n)$ time. □

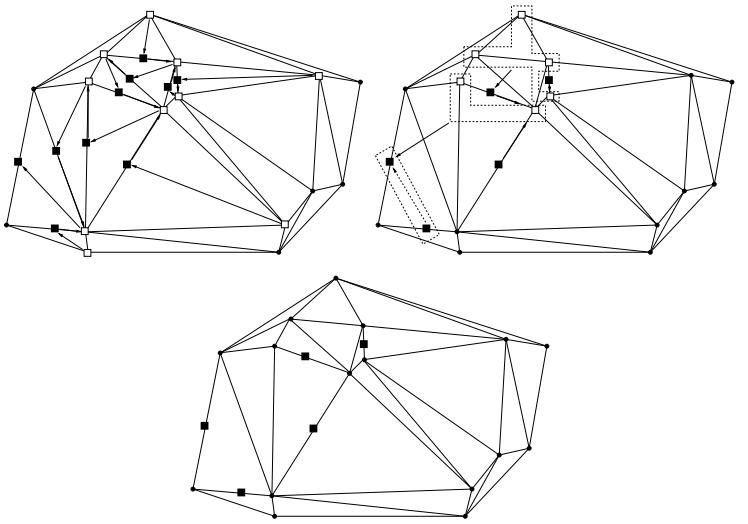

Fig. 10. Orders of white and black processors.

4. Parallel algorithm for computing the β-skeleton for $\beta \in [1, 2]$

In this section we describe a parallel algorithm for computing the β-skeleton for $\beta \in [1, 2]$ in a PRAM-CREW model with $O(n)$ processors, where n is the number of points in a given set V.

At first, we create a partial order of alternating groups of black and white processors using a generalized elimination path. The vertices which are leaves of the generalized elimination tree correspond to white processors. The edges which belong to the *GET* correspond to black processors. Moreover, the endpoints of these edges which compose a triangle in $DT(V)$ with the succeeding edges on the respective generalized elimination paths correspond to white processors. The black processor is either a successor of the white processor corresponding to vertex of the preceding edge on the respective *GEP*, or (if such edge doesn't exist) a successor of the white processor corresponding to the respective leaf. The white processor is successor of the black processor corresponding to the edge which precedes on the *GEP* the edge corresponding to successor of this white processor. A white processor can have two black predecessors and only one black successor (see Fig. 10).

We consider each of the linear orders of the processors (i.e. generalized elimination path from a leaf to the root) separately.

The main idea of this algorithm is to eliminate in each generalized elimination tree groups of white or black processors in each step. At the beginning these groups consist of single white and black processors. Then, we check if any vertex corresponding to a processor in a given set of white processors eliminates all of the edges from the next set of black processors. It is only necessary to check if the farthest edge is eliminated. If the test result is positive we cross out the corresponding set of black processors and join the set of white processors with the next existing set of white processors. Otherwise, we need to check which edges corresponding to processors from the black set are eliminated by vertices corresponding to processors from the white set. They can be found by the binary search on a generalized elimination path. Note that edges assigned to the black processors from different linear orders may form a tree in the partial order.

Lemma 9. *The intersection of the sets of edges assigned to the black processors eliminated by vertices assigned to two separate sets of white processors, is empty.*

Proof. Let us assume that there exists an edge e eliminated by vertices from two different sets. Hence, this edge belongs to two different generalized elimination paths. Let edge ab, where $a, b \in V$, be the first common edge of these two paths, i.e the edge ab is also eliminated by analyzed vertices. Note that there exists a vertex v which lies opposite to the edge ab and it belongs to the both elimination paths. Hence, the white processor assigned to v has already been crossed out (otherwise, the edge ab would be eliminated by v first). Consequently, v does not eliminate ab, so one of the edges of $\triangle abv$ intersects at most one arc of the lune $N_p(a, b, \beta)$. By virtue of Lemma 1 the vertices preceding this edge on the generalized elimination path cannot eliminate ab. Therefore, they cannot eliminate e either. □

Corollary 3. *There is no collision between the white processors eliminating edges corresponding to the black processors.*

The algorithm uses two functions. The function *Destroy* destroys a whole set of processors. For a given set of white processors W, function *Eliminate* selects edges corresponding to the next set of black processors, which are eliminated by vertices corresponding to processors in W. Only these black processors are removed (Algorithm 4).

Algorithm 4: Parallel algorithm for computing β-skeleton.

Input: set of white processors W and set of black processors B for $DT(V)$, parameter $\beta \in [1, 2]$
Output: set of edges of the β-skeleton for V

construct sequences corresponding to generalized elimination paths ;
while *sets of white processors exist* **do**
  for *each set of white processors w* **do in parallel**
    if *if w eliminates all processors in* `Next(w)` **then**
      find the next not eliminated set z of processors after w ;
      `Destroy(`*Next(w)*`)` ; `Next(w) := z`
    else
      find the next not eliminated set of processors after *Next*(w) ;
      `Eliminate(w, Next(w))` ;
      `Destroy(w)` ;
    end
    join neighboring black sets ;
    join neighboring white sets ;
  end
end

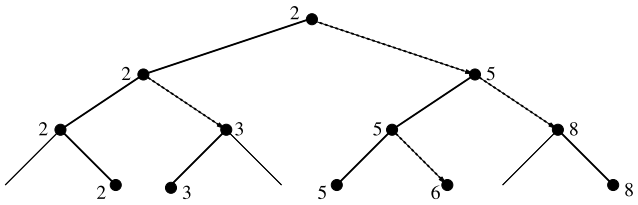

Fig. 11. Broadcasting of information between processors. Only labeled processors are interested in receiving data.

Theorem 4. *The presented parallel algorithm computes β-skeleton for a set V of n points and $\beta \in [1, 2]$ in $O(\log^2 n)$ time with $O(n)$ processors.*

Proof. The algorithm uses $O(n)$ white and $O(n)$ black processors due to linear size of $DT(V)$. Note that in each repetition of **while** loop at least half of the existing sets of processors in every linear orders are crossed out. Black processors are never removed totally, because $MST(V)$ is a subgraph of β-skeleton for $\beta \in [1, 2]$. Therefore, the algorithm stops after $O(\log n)$ repetitions of the **while** loop. Functions *Eliminate* and *Destroy* take $O(\log n)$ time. Hence, the algorithm computes the β-skeleton in $O(\log^2 n)$ time. □

5. Distributed algorithms

Both of the presented parallel algorithms can be easily transformed into a form which uses $O(n)$ processors in the PRAM-EREW model. For example, the processors can avoid conflict in reading data by communicating with each other according to the scheme of complete binary tree (see Fig. 11—the processors distribute information to the others (dotted lines) by traversing the tree along the selected paths). This change increases the complexity of the algorithms by $O(\log n)$.

Let us assume that each processor is a separate computer with its own memory. Unlike the PRAM model, this distributed model lacks the global shared memory we have used so far to communicate between processors. Instead, the appropriate parts of the memory are located in the individual computers, which we label. We want to divide the list of computers into subsets according to their destinations. In the i-th step we create lists of computers communicating with computers whose labels belong to the interval $[k\lceil\frac{n}{2^i}\rceil + 1, (k+1)\lceil\frac{n}{2^i}\rceil]$, where $0 \le k \le 2^i - 1$ (see Fig. 12). One step requires $O(\log n)$ time. We continue this procedure until we allocate to each computer a sequence of computers which want to connect with it. We use pointer-jumping method to broadcast the information in these sequences. The same procedures can be used by storing results of computations.

Corollary 4. *Using the method of communication between computers presented above, we can construct distributed algorithms for $O(n)$ computers with sufficient amount of memory that solve problems of computing the β-spectrum and finding the β-skeleton for $\beta \in [1, 2]$ in polylogarithmic time.*

When we have $o(n)$ computers (each storing more information), the complexity of distributed algorithm increases by the factor $O(\frac{n}{k})$, where n is the size of the input and k is the number of computers.

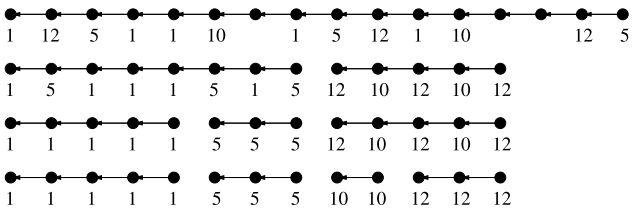

Fig. 12. Separation of computers (black points) with different destinations (numbers below the points).

6. Conclusions

In this work, we have presented three new algorithms to construct the lune-based β-skeletons for $\beta \in [1, 2]$. We showed a sequential algorithm which computes the β-spectrum in $O(n \log^2 n)$ time, and a parallel one which works in $O(\log^4 n)$ time. We also showed a parallel algorithm which constructs the β-skeleton in $O(\log^2 n)$ time. Both of those parallel algorithms use $O(n)$ processors in the CREW-PRAM model. It would be interesting to know if the presented algorithms are optimal. The circle-based β-spectrum can be found in a $O(n \log n)$ time sequentially and one could expect a similar result for the lune-based β-spectrum. Similarly, the exponents in time complexity functions for the analyzed parallel algorithms are high and possibly they could be decreased. The construction of parallel algorithms for finding β-skeletons and β-spectrum in higher dimensions is another open problem.

Acknowledgements

The authors would like to thank Jerzy W. Jaromczyk for important discussions and comments.

References

[1] S.G. Akl, K.A. Lyons, Parallel Computational Geometry, Prentice Hall, 1993.
[2] N. Amenta, M.W. Bern, D. Eppstein, The crust and the β-skeleton: combinatorial curve reconstruction, Graph. Models Image Proc. 60/2 (2) (1998) 125–135.
[3] A. Aggarwal, C. O'Dunlaing, C. Yap, Parallel Computational Geometry, Courant Institute of Mathematical Sciences, New York University, New York, 1987.
[4] M.J. Atallah, M.T. Goodrich, Efficient parallel solutions to some geometric problems, J. Parallel Distrib. Comput. 3 (1986) 293–327.
[5] B. Chazelle, H. Edelsbrunner, M. Grigni, L.J. Guibas, J. Hershberger, M. Sharir, J. Snoeyink, Ray shooting in polygons using geodesic triangulations, Algorithmica 12 (1994) 54–68.
[6] R. Cole, M.T. Goodrich, Optimal parallel algorithms for polygon and point-set problems, Algorithmica 7 (1992) 3–23.
[7] M. Connor, P. Kumar, Fast construction of k-nearest neighbor graphs for point clouds, IEEE Trans. Vis. Comput. Graph. 4 (2010) 599–608.
[8] D. Eppstein, β-Skeletons have unbounded dilation, Comput. Geom. 23 (2002) 43–52.
[9] K.R. Gabriel, R.R. Sokal, A new statistical approach to geographic variation analysis, Syst. Zool. 18 (1969) 259–278.
[10] N.F. Huang, A divide-and-conquer algorithm for constructing relative neighborhood graph, BIT 30 (1990) 196–206.
[11] F. Hurtado, G. Liotta, H. Meijer, Optimal and suboptimal robust algorithms for proximity graphs, in: Computational Geometry, North-Holland, Amsterdam, 1985, pp. 217–248.
[12] J.W. Jaromczyk, M. Kowaluk, A note on relative neighborhood graphs, in: Proceedings 3rd Annual Symposium on Computational Geometry, ACM Press, Canada, Waterloo, 1987, pp. 233–241.
[13] J.W. Jaromczyk, M. Kowaluk, F. Yao, An optimal algorithm for constructing β-skeletons in L_p metric, manuscript, 1989.
[14] M. Kowaluk, Planar β-skeleton via point location in monotone subdivision of subset of lunes, in: EuroCG 2012, Italy, Assisi, 2012, pp. 225–227.
[15] D.G. Kirkpatrick, J.D. Radke, A framework for computational morphology, in: Computational Geometry, North Holland, Amsterdam, 1985, pp. 217–248.
[16] A. Lingas, A linear-time construction of the relative neighborhood graph from the Delaunay triangulation, Comput. Geom. 4 (1994) 199–208.
[17] P.D. MacKenzie, Q.F. Stout, Ultra-fast expected time parallel algorithms, J. Algorithms 26 (1998) 1–33.
[18] D.W. Matula, R.R. Sokal, Properties of Gabriel graphs relevant to geographical variation research and the clustering of points in plane, Geogr. Anal. 12 (1984) 205–222.
[19] S. Olariu, I. Stojmenović, A.Y. Zomaya, Time-optimal proximity graph computations on enhanced meshes, J. Parallel Distrib. Comput. 49 (1998) 204–217.
[20] K.J. Supowit, The relative neighborhood graph, with an application to minimum spanning trees, J. ACM 30 (3) (1983) 428–448.
[21] G.T. Toussaint, The relative neighborhood graph of a finite planar set, Pattern Recogn. 12 (1980) 261–268.