All Approximating Segments for a Sequence of Points

Ghobad Emadi* Alireza Zarei†

Abstract

In this paper, we consider the problem of approximating a sequence of n points by a line segment in such a way that the distance of each point from this segment is not greater than a given constant. Furthermore, the distance between the first(last) input point and the start(end)-point of the approximating segment must not be greater than the given constant. This is a sub-problem in solving unrestricted line simplification and minimum-link path problems. We propose an $O(n \log n)$ algorithm for computing a representation of these segments and we prove that the lower time complexity of finding all such segments (in a specific representation) is $\Omega(n \log n)$ on the algebraic computation tree model which means that our algorithm is optimal.

Keywords: line simplification, line stabbing, algebraic computation tree model, minimum-link path.

1 Introduction

A basic technique in data reduction is to approximate a collection of data by another collection of smaller size. Then, the resulted data are easier to be processed or maintained, in cost of accuracy in further processes. An example of such large-scale data is the ordered sequence of points describing a path or a region boundary.

In such applications, we have an arbitrary sequence of points and the goal is to approximate the trace of these points by a chain of smaller number of segments so that it describes the initial sequence well. This problem is known as the line simplification [1, 2, 3] or ordered stabbing [5, 6] in the literatures. In this problem, each line segment of the answer is used as an approximation for a sub-sequence of the input points with an *error* which is the maximum *distance* of the points of this sub-sequence to the approximating segment. Then, the *error* of an approximating chain is either the maximum or sum of the *error* of its segments. The Euclidean distance is the most popular and practical distance measure in computing the *error* of a line segment.

In almost all line simplification solutions there is the sub-problem of finding an approximating segment for a sub-sequence of input points. In this paper, we consider the general version of this problem. We are given a sequence of points $p_1, p_2, \ldots, p_n$, and the goal is to find all approximating segments st in such a way that the distance of any point p_i to st as well as the length of the segments sp_1 and tp_n are not greater than a given constant ϵ. The distances and lengths are measured by L_2, the Euclidean distance metric. This problem can be formulated as having a sequence of closed disks $o_1, o_2, \ldots, o_n$, all with radius ϵ, and the goal is to find all segments that start from a point in o_1, intersect all disks and end at a point in o_n. This formulation of the problem is similar to the known ordered stabbing problem while the first definition is used in line(path) simplification literature.

This problem has many applications in online query processing on geometric data sets, CAD/CAM and computer graphics. In these applications, the set of approximating segments are computed in a preprocessing step, and at query time, for a fixed start or end point, a possible approximating segment is computed and returned efficiently. For example, by moving one end point of an approximating segment in a CAD/CAM application the proper position of this segment must be determined quickly.

We first give a proper representation model for these approximating segments in Section 2. Then, in Section 3, we propose an $O(n \log n)$ time algorithm for computing the proper representation of the approximating segments. Finally, in Section 4, we prove that obtaining such a representation has $\Omega(n \log n)$ time complexity on the algebraic computation tree model which means that our algorithm is optimal.

2 The Representation Model

Any approximating segment must start from a region $S \subseteq o_1$ and end at a region $T \subseteq o_n$. Thus, in order to find all possible approximating segments we must find the regions S and T, and a method that for each point $p \in S$ identifies the points $t \in T$ where st is an approximating segment. As we will see later, both regions S and T are convex and their boundaries are composed of line segments and some arcs of o_i's boundaries. Hence, we can maintain the regions S and T by having the ordered sequence of their boundary edges and arcs. Moreover, for each point $s \in S$ we show that there is a convex region $T_p \subset T$ where for each point $t \in T_p$, st is an approximating segment. Therefore, in a proper repre-

*Department of Mathematical Sciences, Sharif University of Technology, ghobad_emadi@alum.sharif.ir

†Department of Mathematical Sciences, Sharif University of Technology, zarei@sharif.edu

sentation of S and T we need a data structure from which the region T_p can be obtained efficiently(e.g. in logarithmic time). We call such a representation as a proper representation model for the approximating segments.

3 The Proposed Algorithm

In this section, we propose an $O(n \log n)$ time algorithm to find a proper representation for all approximating segments of a sequence of points $p_1, p_2, \ldots, p_n$. We assume that the corresponding disks o_1 and o_n of p_1 and p_n are disjoint. In this algorithm, we use some notation and definitions from [6].

For any direction $\overrightarrow{\alpha}$, $\overrightarrow{-\alpha}$ denotes its reverse direction and $L_{\overrightarrow{\alpha}}$ is a line parallel to this direction. A disk o_i is called a *support disk* for a direction $\overrightarrow{\alpha}$ if there is a line $L_{\overrightarrow{\alpha}}$ where o_i is tangent to $L_{\overrightarrow{\alpha}}$ from left(in direction of $\overrightarrow{\alpha}$) and there is no other disk strictly to the left of $L_{\overrightarrow{\alpha}}$. Then, $L_{\overrightarrow{\alpha}}$ is called the *support line* for direction $\overrightarrow{\alpha}$ and the point $p = L_{\overrightarrow{\alpha}} \cap o_i$ is called a *support point* for this direction. The gray disk in Figure 1 is a support disk for direction $\overrightarrow{\alpha}$.

Observation 1 *The set of lines(if exists any) which are parallel to $L_{\overrightarrow{\alpha}}$ and intersect all disks $o_1, o_2, \ldots, o_n$ must lie between $L_{\overrightarrow{\alpha}}$ and $L_{\overrightarrow{-\alpha}}$ (see Figure 1).*

A line L is a *limiting line* if it is the support line in both directions $\overrightarrow{\alpha}$ and $\overrightarrow{-\alpha}$. A limiting line is an internal bitangent of a pair of disks. $L_{\overrightarrow{\beta}}$ and $L_{\overrightarrow{\beta'}}$ are two limiting lines in Figure 1. Any limiting line $L_{\overrightarrow{\alpha}}$ has two support points $s_{\overrightarrow{\alpha}}$ and $e_{\overrightarrow{-\alpha}}$ which are respectively support points in directions $\overrightarrow{\alpha}$ and $\overrightarrow{-\alpha}$. A limiting line $L_{\overrightarrow{\alpha}}$ is called clockwise(resp. counter-clockwise) if the direction of the vector $\overrightarrow{s_{\overrightarrow{\alpha}} e_{\overrightarrow{-\alpha}}}$ is the same as the direction of $\overrightarrow{-\alpha}(\overrightarrow{\alpha})$ and is denoted by *cw*(resp. *ccw*). Figure 1 shows *cw* and *ccw* limiting lines.

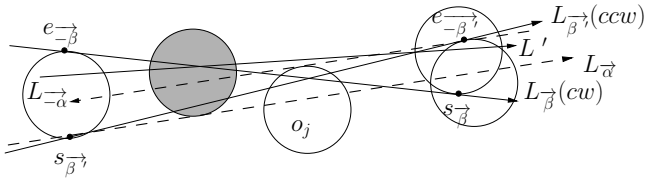

Figure 1: The gray disk is a support disk for the line $L_{\overrightarrow{\alpha}}$, and $L_{\overrightarrow{\beta}}$ and $L_{\overrightarrow{\beta'}}$ are two limiting lines. $L_{\overrightarrow{\beta}}$ is clockwise limiting line and $L_{\overrightarrow{\beta'}}$ is counter-clockwise.

It is simple to verify that a *cw* or *ccw* limiting line intersects all disks and if we rotate a *cw*(resp. *ccw*) clockwise(resp. counter-clockwise) it does not intersect all disks any more. Moreover, there are at most two limiting lines(a *cw* and a *ccw*) which delimit slopes of the lines intersecting all disks. Assume that *cw* is $L_{\overrightarrow{\alpha}}$ and *ccw* is $L_{\overrightarrow{\beta}}$. Then, the direction of all approximating lines must be between $\overrightarrow{\alpha}$ and $\overrightarrow{\beta}$.

Other than *cw* and *ccw* there is another limitation for the set of approximating segments. When we rotate the *cw* limiting line, before reaching the *ccw*, it may leave a disk which means that it is not an approximating line any more. For example, the line L' in Figure 1 does not intersect disk o_j while it lies inside the slope defined by *cw* and *ccw*. This restriction for the approximating lines can also be defined by the notion of support lines. Assume that A is the set of support disks for directions $[\alpha, \beta]$ and B is the set of support disks for directions $[-\beta, -\alpha]$. The sets A and B are respectively called the above and below *support hulls*. It is simple to see that any one of the disks defining *cw* and *ccw* belongs to at least one of the sets A and B.

Lemma 1 *If a line L intersects all disks in A and B, it intersects all disks $o_1, o_2, \ldots, o_n$.*

Proof. Assume that $L_{\overrightarrow{\gamma}}$ intersects all disks in A and B, but it does not intersect a disk o_i where $o_i \notin A \cup B$. Without loss of generality (W.l.o.g), assume that o_i lies to the left of $L_{\overrightarrow{\gamma}}$. Consider the line $L'_{\overrightarrow{\gamma}}$ which is parallel to $L_{\overrightarrow{\gamma}}$ and is tangent to o_i, and o_i lies to the left of $L'_{\overrightarrow{\gamma}}$.

Trivially, $L'_{\overrightarrow{\gamma}}$ is a support line in direction $\overrightarrow{\gamma}$. On the other hand, $L'_{\overrightarrow{\gamma}}$ intersects support disk of *cw* and *ccw* which means that $\gamma \in [\alpha, \beta]$ or $\gamma \in [-\beta, -\alpha]$. Therefore, o_i belongs to A or B which is a contradiction. □

The final restriction on the approximating segments in our problem is that such a segment must start from o_1 and end at a point in o_n.

Summarizing these three limitations, we propose an algorithm which begins by o_1 and o_n and incrementally adds the other disks o_2 to o_{n-1}. Next, it updates the limiting lines *cw* and *ccw*, support hulls A and B, and start and end regions S and T. As initialization, *cw* and *ccw* are the proper internal bitangents of o_1 and o_n(o_1 and o_n does not intersect each other), both of A and B are the set $\{o_1, o_n\}$, $S = o_1$ and $T = o_n$. Then, the disks are considered one by one and these elements are updated accordingly. The pseudo code of the algorithm and its sub-functions are listed in Appendix A.

In this algorithm, A and B are maintained in two balanced binary search trees as follows. Assume that $A =< o_{i_1}, o_{i_2}, \ldots, o_{i_k} >$ is the sequence of disks that their order is defined so that the lower external bitangent of any consecutive pair of these disks is a support line (see Figure 2). Each line L that intersects all the disks must lie above the lower envelope of these bitangents (drawn in bold in Figure 2). Similarly, there is such an ordering for disks in B. Moreover, the lower envelope of these bitangents for A is convex-downward and the upper envelope of the bitangents for B is convex-upward.

According to the direction of cw and ccw, the first disk in A(resp. B) is called $head(A)$(resp. $head(B)$) and the last one is called $tail(A)$(resp. $tail(B)$).

The order of disks in A and B (implies the first and the last disks) is derived from their position in their corresponding binary search tree. For the initiative, the order of o_1 is less than o_n. As we see later, a new disk will be inserted into A(resp. B) when it lies above(resp. below) the lower(resp. upper) envelope of the bitangent of disks in A(resp. B). The order of such a disk is defined by the projection of its center on the corresponding envelope. It is simple (by an inductive argument) to see that cw and ccw are respectively internal bitangents of ($head(B)$,$tail(A)$) and ($head(A)$,$tail(B)$).

The algorithm considers disks o_2 to o_{n-1} one by one and for a disk o_i does as follows: When o_i lies between two consecutive disks in A(resp. B) and completely lies below(resp. above) the lower(resp. upper) envelope then it is impossible to find a line intersecting all disks $o_1, o_2, \ldots, o_i$ and o_n. Moreover, if o_i lies beyond $tail(B)$(resp. $tail(A)$) and to the left(resp. right) of ccw(resp. cw) or o_i lies before $head(B)$(resp. $head(A)$) and to the left(resp. right) of cw(resp. ccw), there is no approximating segment for disks $o_1, o_2, \ldots, o_i$ and o_n. The correctness of these claims are simple and are derived from the definition of A, B, cw, and ccw.

The disk o_i may change the limiting lines cw or ccw. This happens when o_i is not intersected by cw or ccw. Based on the position of o_i, (with respect to cw or ccw) these limiting lines must be rotated accordingly until it will be tangent to o_i. This is done by calling procedures $UpdateCW(A, B, o_i, cw)$ and $UpdateCCW(A, B, o_i, ccw)$.

While the set of approximating segments must have a direction between cw and ccw, some parts of A and B are not valid any more and in the $UpdateCW$ and $UpdateCCW$ procedures these invalid areas are removed. If o_i has not been inserted as $head(A)$ and $Tail(A)$ in calling $UpdateCW$ and $UpdateCCW$ procedures and if it lies between two consecutive disks in A and above the lower envelope of A, it must be inserted in A in its correct position. Then, this may cause the lower envelope to be non-convex. If so, the incorrect dicks are removed from A. These operations are done by calling the $UpdateA$ procedure (see Appendix A). Similarly, it may be required to update B using $UpdateB(B, o_i)$ procedure.

Finally, the algorithm must update the regions S and T according to the new values of cw, ccw and the position of o_i. S is *at most* the intersection of the previous S, the left half plan of ccw and the right half plan of cw. We say, *at most* because the new disk o_i may lies before (comparing to the direction cw and ccw) the disk o_1. Then, any segment which starts from o_1 must go backward to intersect o_i which means that it can not end at a point in o_n. Therefore, when cw or ccw leaves o_i before exiting S we must remove from S the region that lies in front of o_i(according to the direction of cw or ccw). Symmetrically, the region T must be updated accordingly. S and T are always the intersection of convex regions. Thus, they will remain convex and their boundaries are composed of line segments from cw and ccw and circle arcs from o_i disks.

After handling the disk o_i, if either of the regions S or T is empty it means that there is no approximating segment which is reported by the algorithm.

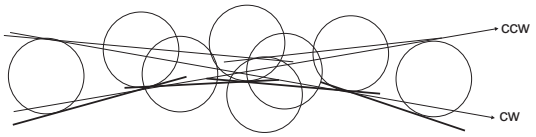

Figure 2: Support hulls and the lower envelope of disks.

Theorem 2 *The time complexity of the above algorithm for a sequence of n disks is $O(n \log n)$ and requires $O(n)$ space.*

Proof. Having the proper search structures for A and B, the position of a disk o_i is obtained in $O(\log(|A|))$ and $O(\log(|B|))$ and the size of A and B are at most n. The running time of $UpdateCW$, $UpdateCCW$, $UpdateA$ and $UpdateB$ procedures depend on the number of changes these procedures apply on A and B. Each of these changes can be done in $O(\log(|A|))$ and $O(\log(|B|))$. Each disk is inserted into A(or B) and removed from it at most once which means that the total number of insert/delete operations on A and B is $O(n)$. Therefore , the amortized cost of any one of the procedures $UpdateCW$, $UpdateCCW$, $UpdateA$ and $UpdateB$ is $O(\log n)$.

While the regions S and T are convex, finding the intersection of these regions and a half-plane or a convex region of complexity $O(1)$ can be handled in logarithmic time in terms of the complexity of their boundaries. Any call to $UpdateS$ or $UpdateT$ increases the complexity of the region by a constant number which means that the complexity of the boundary of S and T is $O(n)$. Therefore, the time complexity of $UpdateS$ and $UpdateT$ is $O(\log n)$. Hence, the total time complexity of the algorithm is $O(n \log n)$ and requires $O(n)$ space. □

Having the regions S and T and the binary search tree data structures for A and B, for any query point $s \in S$ we can report the region $T' \subseteq T$ where for any $t \in T'$, st is an approximating segment. This can be done by finding tangent lines to A and B from s and obtaining the intersection of T and the region contained between these two tangents. Therefore, the following theorem can be concluded.

Theorem 3 *Having the regions S and T and the data structures A and B, the set of all approximating segments starting from a given query point $s \in S$ can be obtained in $O(\log n)$ time.*

4 The Reduction Method

In this section, we propose a method for sorting a set of n numbers using just the proper representation of the region S of a set of n disks to show that finding the proper representation of all approximating segments requires $\Omega(n \log n)$ time on algebraic computation tree model. In this model, add, multiply, division, and square-root operations are done in constant time [4] and the lower bound of sorting n distinct numbers on this model is $\Omega(n \log n)$. Let $a_1, a_2, \ldots, a_n$ be n positive real, non-zero and distinct numbers and $\Phi : \mathcal{R} \rightarrow D((x, y), 2\epsilon)$ be a mapping function which maps a number a_i to a disk $D(C_i, 2\epsilon)$ whose radius is 2ϵ and its center is $C_i = (x_i, y_i) = (\frac{\epsilon}{2} - \frac{\epsilon a_i}{\sum_{j=1}^{n} a_j}, \sqrt{\epsilon^2 - (\frac{\epsilon}{2} - \frac{\epsilon a_i}{\sum_{j=1}^{n} a_j})^2})$ in cartesian coordinate. In this mapping, $-\frac{\epsilon}{2} < x_i < \frac{\epsilon}{2}$ and y_i is y-coordinate of the intersection point of line $x = x_i$ and circle $D((0, 0), \epsilon)$ (see Figure 3). According to this mapping, each number a_i corresponds to a disk $D_i = \Phi(a_i)$ whose center lies on ϵ-radius circle $D((0, 0), \epsilon)$. We add two other disks D_0 and D_{n+1} such that the correct ordering of the numbers $a_1, a_2, \ldots, a_n$ can be obtained from the proper representation of the starting region S of the approximating segments for the sequence $D_0, D_1, D_2, \ldots, D_n, D_{n+1}$ of disks. D_0 is defined as $D((0, -\epsilon), 2\epsilon)$ and D_{n+1} will be defined later.

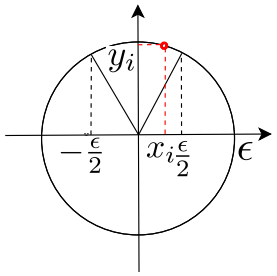

Figure 3: Mapping a number a_i to a disk with center (x_i, y_i).

This mapping can be done with $O(n)$ number of add, multiply, division, and square-root operations and these operations are allowed on algebraic computation tree model. So, this reduction can be done in $O(n)$ time. Having the proper representation of the solution of this derived problem, we prove that we can find the correct ordering of the input numbers. This implies that the lower bound of the approximating segments problem is $\Omega(n \log n)$ as well.

Assume that C_i is the center of the disk D_i and its coordinates are denoted by (x_i, y_i).

Observation 2 *For each disk D_i $(1 \leq i \leq n)$ we have $-\frac{\epsilon}{2} < x_i < \frac{\epsilon}{2}$ and for each pair of disks D_i and D_j $(1 \leq i \leq n$ and $1 \leq j \leq n)$ we have $x_i < x_j$ if and only if $a_i > a_j$.*

This observation is a direct result of the definition of C_i and the assumption that the numbers are positive, non-zero, and distinct.

Observation 3 *Each pair of distinct disks D_i and D_j $(1 \leq i \leq n$ and $1 \leq j \leq n)$ intersect in exactly two points on their boundaries.*

This observation is derived from the facts that these disks have the same radius, and the distance between their centers is smaller than their radius.

In the rest of this paper, intersection points of two disks are the intersections between their boundaries.

Lemma 4 *Exactly one intersection point of each pair of distinct disks D_i and D_j $(1 \leq i \leq n$ and $1 \leq j \leq n)$ lies inside the disk D_0 and this point lies under x-axis.*

Proof. By the construction, the centers of D_i and D_j (C_i and C_j) lie on the boundary of $D((0, 0), \epsilon)$ and above x-axis, and their distance is smaller than 2ϵ. This implies that one of the intersection points of D_i and D_j lies above x-axis and outside D_0 (point p_{out} in Figure 4). On the other hand, the distances of C_i and C_j from x-axis are smaller than 2ϵ which imply that the other intersection point must lie under this line (point p_{in} in Figure 4). We prove that this intersection point lies inside D_0. W.l.o.g, assume that $p_{in} = (x_{in}, y_{in})$ is this intersection point and $0 \leq x_{in} < \frac{\epsilon}{2}$. Then, at least one of x_i and x_j must be in the range $(-\frac{\epsilon}{2}, 0]$ (otherwise, x_{in} cannot be in the range $0 \leq x_{in} < \frac{\epsilon}{2}$). This means that at least one of the centers C_i and C_j lies on the second quadrant on the boundary of disk $D((0, 0), \epsilon)$. Then, the shortest distance from this center to the outside of D_0 in forth quadrant happens when it lies on point $p = (0, \epsilon)$ as shown in Figure 4. Trivially, this distance is greater than 2ϵ. Therefore, the intersection point cannot lie outside D_0. □

We denote by S the region $D_0 \cap D_1 \cap \ldots \cap D_n$ which is the intersection of these disks. Obviously, S can be represented as an ordered sequence of arcs, each of which belongs to a disk $D_i \in \{D_0, D_1, \ldots, D_n\}$. Any pair of consecutive arcs have a common point. These points are called the vertices of S.

Lemma 5 *If a_i and a_j are adjacent in the sorted order of numbers $a_1, a_2, \ldots, a_n$, the intersection point of their corresponding disks D_i and D_j $(1 \leq i \leq n$ and $1 \leq j \leq n)$ that lies inside D_0 is a vertex of $S = D_0 \cap D_1 \cap \ldots \cap D_n$.*

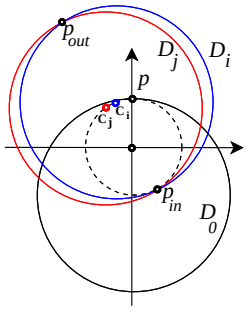

Figure 4: Positions of the intersection points of disks D_i and D_j.

Proof. Let $p = (x_p, y_p)$ be the intersection point for D_i and D_j (see Figure 5). We prove that if a_i and a_j are adjacent in the sorted order of $a_1, a_2, \ldots, a_n$, then $p \in S$ which proves the lemma. Consider the disk $D_p = D((x_p, y_p), 2\epsilon)$. W.l.o.g, assume that $x_j < x_i$. As shown in Figure 5, D_p contains the center of any disk D_k which $x_k < x_j$ or $x_k > x_i$. Moreover, D_p does not contain the center of a disk D_k where $x_j < x_k < x_i$. Furthermore, centers of disks D_i and D_j lie on the boundary of D_p. These three facts show that $p \in S$ if and only if there is no disk D_k whose x_i lies between x_j and x_i and, in this case p lies on the boundary of S. □

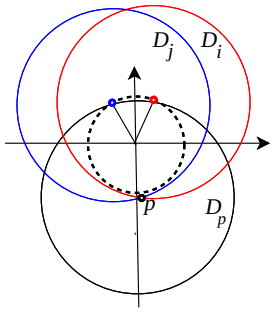

Figure 5: The vertices of the boundary of S.

We denote by max(resp. min) the index of the disk D_i $(1 \leq i \leq n)$ with maximum(resp. minimum) value of x_i. This means that D_{max}(resp. D_{min}) corresponds to the number a_{max}(resp. a_{min}) with the maximum(resp. minimum) value among the numbers $a_1, a_2, \ldots, a_n$. Let q and q' be the vertices of S that are respectively the intersection points of D_0 and D_{min}(lies on the left side of y-axis), and D_0 and D_{max}(lies on the right side of y-axis) (see Figure 6).

Observation 4 *The region S is convex and has $n+1$ vertices. $n-1$ vertices are the intersection points of D_i and $D_j (1 \leq i \leq n$ and $1 \leq j \leq n)$ pairs where their corresponding numbers a_i and a_j are adjacent in the sorted order of the numbers $a_1, a_2, \ldots, a_n$. The other two vertices are q and q'.*

This observation is a simple result of Lemma 5, the fact that the intersection of convex regions is convex, and the configuration of D_0 and D_i $(1 \leq i \leq n)$ disks.

Observation 5 *If we traverse the vertices of S from q to q', we obtain the sorted order of x_i's which corresponds to the sorted order of the numbers $a_1, a_2, \ldots, a_n$.*

Therefore, we can sort a set of positive, non-zero, and distinct numbers by converting them to a set of disks D_i and traversing the boundary of their intersection. Both of these operations can be done in linear time.

Now, we select the disk D_{n+1} so that a segment st be an approximating segment for the sequence of disks $D_0, D_1, D_2, \ldots, D_n, D_{n+1}$ if and only if $s \in S = D_0 \cap D_1 \cap \ldots \cap D_n$ and $t \in D_{n+1}$. Assume that t and t' are respectively the tangent lines to D_{min} at point q and D_{max} at point q' (see Figure 6). The disk D_{n+1} is defined to be the 2ϵ-radius disk that is tangent to both t and t' and lies on the opposite side of the intersection point of t and t' with respect to S (see Figure 6). Lemma 6 shows that D_{n+1} is well-defined. Each disk $D_i \in \{D_1, D_2, \ldots, D_n\}$ intersects D_0 in two points which lie on right and left side of y-axis. These points respectively are called the *first* and the *last* intersection points.

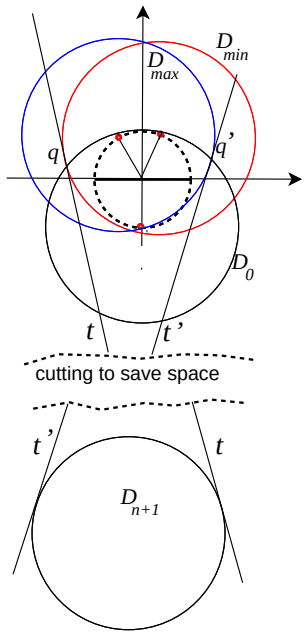

Figure 6: Position of D_{n+1}.

Lemma 6 *For each pair of disks D_i and D_j from $\{D_1, D_2, \ldots, D_n\}$, the tangent line of D_i at its first*

intersection point intersects the line that is tangent to D_j at D_j's last intersection point. Moreover, the intersection point of these tangent lines is closer to that side of S that does not belong to the boundary of D_0 and this intersection point lies under x-axis.

Proof. Each disk $D_i((x_i, y_i), 2\epsilon)$ intersects D_0 in two points. These points lie below that diameter of D_i which is parallel to x-axis. The reason is that the distance of C_i to any point on that part of the boundary of D_0 which lies above this diameter is less than 2ϵ. On the other hand, one of the intersection points of D_0 and D_i lies on the right of the vertical diameter of D_i and the other one lies on the left of that diameter. The slope of tangent lines to D_i at the left intersection point is negative and this for the right intersection point is positive. Therefore, any right(resp. left) tangent line of D_i intersect any left(resp. right) tangent line of D_j. □

The following Lemma identifies the set of all approximating segments for the sequence $D_0, D_1, D_2, \ldots, D_n, D_{n+1}$ of disks.

Lemma 7 *A segment st is an approximating segment for the sequence $D_0, D_1, D_2, \ldots, D_n, D_{n+1}$ of disks if and only if $s \in S$ and $t \in D_{n+1}$.*

Proof. Trivially, any segment that starts from a point of S and ends at a point of D_{n+1} is an approximating segment for $D_0, D_1, D_2, \ldots, D_n, D_{n+1}$. We prove the reverse of the lemma by showing that any segment st where $s \in D_0 - S$ and $t \in D_{n+1}$ does not intersect the region S. As shown in Figure 7, the region $D_0 - S$ can be partitioned into 3 parts A, B and C. It is simple to check that if s belongs to any one of these regions, it cannot intersect S. This means that st does not intersect any one of the disks $D_1, D_2, \ldots, D_n$ and, therefore, cannot be an approximating segment. □

Now, we can conclude the main result:

Theorem 8 *The lower bound on the time complexity of algorithms that solve the approximating segments problem of a sequence of n disks on the algebraic computation tree model is $\Omega(n \log n)$.*

5 Conclusion

In this paper, we considered the problem of approximating a set of points by a line segment. This is a subproblem in line(path) simplification and ordered stabbing problems which used in cartography and geometric data reduction applications. We showed that the time complexity of any algorithm for finding all approximating segments in a specific representaion is $\Omega(n \log n)$ and proposed an optimal algorithm for finding these segments by which the set of all approximating segments

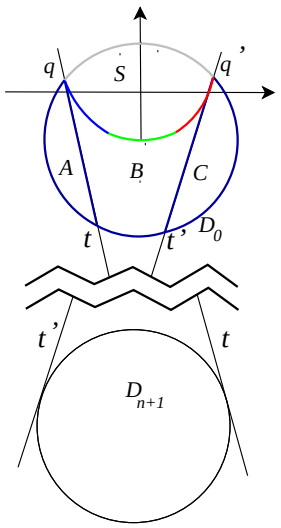

Figure 7: Locus of the approximating segments.

from a given query point can be obtained in $O(\log n)$ time. A practical and immediate related open problem is to find an optimal algorithm for finding only one approximating segment or solving its decision problem.

References

[1] Abam, M.A., de Berg, M., Hachenberger, P., Zarei, A. Streaming algorithms for line simplilcation. *Discrete & Computational Geometry*, vol. 43(3), pp. 497-515, 2010.

[2] P.K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete & Compututationl Geometry*, 23(2):273-291, 2000.

[3] P.K. Agarwal, S. Har-Peled, N.H. Mustafa and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42:203-219, 2005.

[4] M. Ben-Or. Lower bounds for algebraic computation trees. *Proc. 15th Annual Symposium on Theory of Computing.*, pp. 80-86, 1983.

[5] P. Egyed and R. Wenger. Ordered stabbing of pairwise disjoint convex sets in linear time. *Discrete Applied Mathematics*, 31:133-140, 1991.

[6] L. J. Guibas, J. E. Hershberger, J. S.B. Mitchell, and J.S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Internat. J. Comput. Geom. Appl.*, 3(4):383-415, December 1993.

Appendix

In the following procedures we use $head(tail)$ for A and B which corresponds to the first(last) disk in A or B according to direction of cw and ccw. Also, $next$ and $prev$ items follow this ordering. Here, we only listed the procedures $UpdateCCW$, $UpdateA$ and $UpdateS$. The other procedures($UpdateCW$, $UpdateB$ and $UpdateT$) are analogously the same.

Algorithm 1 main procedure

Initialize A=$< o_1, o_n >$, B=$< o_1, o_n >$, cw=$o_1 \searrow o_n$, ccw=$o_1 \nearrow o_n$, S=o_1, T=o_n.
for $i \leftarrow 2$ to $n-1$ **do**
  if o_i lies between two consecutive disks of $A(B)$ and below(above) their lower(upper) external bitangent **then**
    There is no approximating segment.
  end if
  if o_i lies after $Tail(B)(Tail(A))$ and to the left(right) of $ccw(cw)$ or before $Head(B)(Head(A))$ and to the left(right) of $cw(ccw)$ **then**
    There is no approximating segment.
  end if
  if o_i does not intersect ccw **then**
    $UpdateCCW$.
  end if
  if o_i does not intersect cw **then**
    $UpdateCW$.
  end if
  if o_i lies between two consecutive disks of $A(B)$ and above(below) their lower(upper) external bitangent and it is not inserted as $Tail(A)(Tail(B))$ or $Head(A)(Head(B))$ **then**
    $UpdateA(A, o_i)(UpdateB(B, o_i))$.
  end if
  $UpdateS$.
  $UpdateT$.
  if S or T is empty **then**
    There is no approximating segment.
  end if
end for

Algorithm 2 $UpdateS$ procedure

$S \leftarrow S \cap$ (the left halfplan of ccw) $\cap$ (the right halfplan of cw)
if cw or ccw leaves o_i before leaving S **then**
  remove from S the region lies in front of o_i
end if

Algorithm 3 $UpdateCCW$ procedure

if o_i lies on the right of ccw **then**
  $ccw \leftarrow$ counter-clockwise tangent from $Head(A)$ to o_i.
  while ccw enters $Tail(B)$ after entering o_i **do**
    remove $Tail(B)$ from B.
  end while
  while $Tail(B)$ lies above the upper external bitangent of o_i and $Tail(B) \rightarrow prev$ **do**
    remove $Tail(B)$ from B.
  end while
  Add o_i as the $tail$ of B.
  while $Head(A) \rightarrow next$ does not lie below ccw **do**
    remove $Head(A)$ from A.
    $ccw \leftarrow$ counter-clockwise tangent from $Head(A)$ to o_i.
  end while
end if
if o_i lies on the left of ccw **then**
  $ccw \leftarrow$ counter-clockwise tangent from o_i to $Tail(B)$.
  while ccw exit $Head(A)$ before exiting o_i **do**
    remove $Head(A)$ from A.
  end while
  while $Head(A)$ lies below the lower external bitangent of o_i and $Head(A) \rightarrow next$ **do**
    remove $Head(A)$ from A.
  end while
  Add o_i as the $head$ of A.
  while $Tail(B) \rightarrow prev$ does not lie above ccw **do**
    remove $Tail(B)$ from B.
    $ccw \leftarrow$ counter-clockwise tangent from o_i to $Tail(B)$.
  end while
end if

Algorithm 4 $UpdateA$ procedure

/* Suppose o_i lies betweeen two members of A such as A_{prev} and A_{next} */
while A_{next} lies below the lower external bitangent of o_i and $A_{next} \rightarrow next$ **do**
  $Tmp \leftarrow A_{next}$.
  $A_{next} \leftarrow A_{next} \rightarrow next$.
  remove Tmp from A.
end while
while A_{prev} lies below the lower external bitangent of o_i and $A_{prev} \rightarrow prev$ **do**
  $Tmp \leftarrow A_{prev}$.
  $A_{prev} \leftarrow A_{prev} \rightarrow prev$.
  remove Tmp from A.
end while
Insert o_i into A.