Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computational Geometry 30 (2005) 41–58
Computational Geometry
Theory and Applications
www.elsevier.com/locate/comgeo

Polygonal chain approximation: a query based approach

Ovidiu Daescu *, Ningfang Mi

Department of Computer Science, University of Texas at Dallas, MS-EC31, Richardson, TX 75080, USA

Received 12 September 2003; received in revised form 28 June 2004; accepted 26 July 2004

Available online 11 September 2004

Communicated by P. Agarwal

Abstract

In this paper we present a new, query based approach for approximating polygonal chains in the plane. We give a few results based on this approach, some of more general interest, and propose a greedy heuristic to speed up the computation. Our algorithms are simple, based on standard geometric operations, and thus suitable for efficient implementation. We also show that the query based approach can be used to obtain a subquadratic time exact algorithm with infinite beam criterion and Euclidean distance metric if some condition on the input path holds. Although in a special case, this is the first subquadratic result for path approximation with Euclidean distance metric.

Keywords: Polygonal chain; Approximation; Query method; Subquadratic time

1. Introduction

A polygonal path or chain P in the plane, with n vertices, is defined as an ordered set of vertices $(p_1, p_2, \ldots, p_n)$, such that any two consecutive vertices p_i, p_{i+1} are connected by the line segment $\overline{p_i p_{i+1}}$, for $1 \leqslant i < n$. The polygonal path approximation problem is to approximate a general polygonal path P by another polygonal path P' whose vertices are constrained to form an ordered subset of the vertices of P. The problem appears as a subproblem in many applications in geographic information systems (GIS), cartography, computer graphics, medical imaging and data compression. For example,

* Corresponding author.
E-mail address: daescu@utdallas.edu (O. Daescu).

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2004.07.001

with recent imaging technologies, medical and satellite images have become very rich in details. Analyzing such images or using them in interactive remote WWW-based applications may require a significant amount of time. Thus, path approximation algorithms can be used to simplify/compress those images, for the purpose of efficient storage or for progressive transmission to a remote site. Similar observations apply to map representation in geographic information systems, where in many situations (animations, etc.) a high level of detail is unnecessary or even unwanted.

1.1. Problem definition

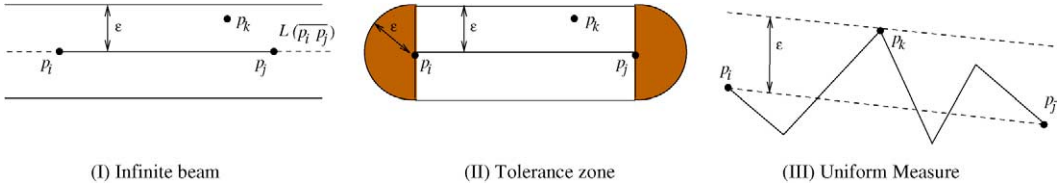
Given a planar polygonal path $P = (p_1, p_2, \ldots, p_n)$, the polygonal path approximation problem is to find a path $P' = (p_{i_1} = p_1, p_{i_2}, \ldots, p_{i_m} = p_n)$ such that, for each $j \in \{1, 2, \ldots, m-1\}$: (i) $i_j < i_{j+1}$, where $i_j, i_{j+1} \in \{1, 2, \ldots, n\}$ and (ii) the subpath $(p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P is contained in some *error tolerance* region of the line segment $\overline{p_{i_j} p_{i_{j+1}}}$. We will also consider the case when the vertices of the approximation are not restricted to a subset of the original path.

The *error tolerance* region of a line segment $\overline{p_i p_j}$ is defined by an error measure criterion. A few error criteria have been used in solving various polygonal path approximation problems, not necessarily of the form above [1–3,5,9–11,13,16,17,21–26,28]. If not otherwise specified, we consider the error criterion used in [10,16,23,28], called the *infinite beam* or *parallel-strip* criterion, with the Euclidean, L_2 measure of distance. With this criterion, the ε-tolerance region of a line segment $\overline{p_i p_j}$ is the set of points that are within distance ε from the line $L(\overline{p_i p_j})$ supporting the line segment $\overline{p_i p_j}$ (see Fig. 1(I)). If a subpath $P_{i_j,i_{j+1}} = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P is contained in the ε-tolerance region of the line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P', then $\overline{p_{i_j} p_{i_{j+1}}}$ is an *ε-approximating* segment for $P_{i_j,i_{j+1}}$. A path P' is an *ε-approximation* of P if each line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P', for $j = 1, 2, \ldots, m-1$, is an *ε-approximating* segment.

Other commonly used error criteria are the tolerance zone and uniform measure. With the *tolerance zone* criterion [21,23,24], the ε-tolerance region of a line segment $\overline{p_i p_j}$ is the region of space that is the union of all radius-ε disks centered at points along the segment $\overline{p_i p_j}$ (see Fig. 1(II)). With the uniform error measure [2], the error of a line segment $\overline{p_i p_j}$ is defined as $\max\{d(p_k, \overline{p_i p_j}) \mid i \leqslant k \leqslant j\}$, where $d(p_k, \overline{p_i p_j})$ denotes the vertical distance between p_k and $\overline{p_i p_j}$ (see Fig. 1(III)). This measure applies to x-monotone paths.

In this paper, we consider the optimization version of the polygonal path approximation problem, called the **min-# problem**: Given a polygonal path P and a positive approximation error ε, find an ε-approximating path P' of P with the smallest number of vertices.

While the tolerance zone criterion would produce a compressed version that better captures the features of the original path, the motivation for studying the problem under infinite beam criterion is two fold. The first reason is that the best known solutions under tolerance zone criterion compute approxi-



(I) Infinite beam (II) Tolerance zone (III) Uniform Measure

Fig. 1. The ε-tolerance zones of a single segment $\overline{p_i p_j}$ with infinite beam criterion, tolerance zone criterion and uniform measure.

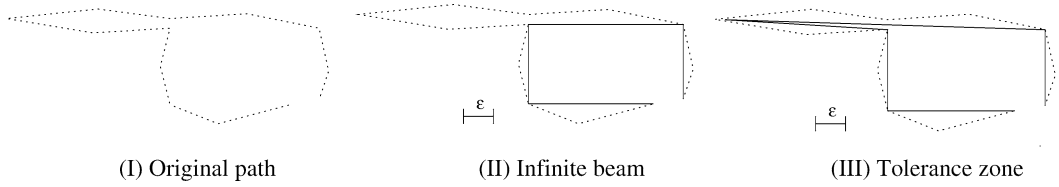


Fig. 2. The ε-approximating paths with infinite beam and tolerance zone criteria.

mating lines or semilines in order to obtain approximating segments. Thus, some of our results may be useful for this error criterion. The second reason is that infinite beam criterion gives a better degree of compression (see Fig. 2).

Lemma 1. *Let P' be an ε-approximating path of P with tolerance zone criterion. Then P' is an ε-approximating path of P with infinite beam criterion.*

Proof. If a point of P is within distance ε from a line segment $\overline{p_{i_j} p_{i_{j+1}}}$ of P', then it is within distance ε from the line $L(\overline{p_{i_j} p_{i_{j+1}}})$ supporting $\overline{p_{i_j} p_{i_{j+1}}}$. □

1.2. Previous work

Early results for the min-# problem, under various error criteria, were presented by Imai and Iri [21–23], Melkman and O'Rourke [24] and Toussaint [28]. In particular, Imai and Iri [23] have formulated the problem in terms of graph theory: construct a path approximation, unweighted directed acyclic graph G and then use breadth first search to compute a shortest path in this graph. Their formalism has been later exploited by most of the algorithms devoted to the problem [2,5,9,10,16]. However, with the exception of Agarwal and Varadarajan algorithms [2], all the other algorithms have quadratic or superquadratic time complexity. The algorithms in [2] combine the previous iterative graph based approach with divide and conquer and, using graph compression techniques and more complicated data structures, achieve $O(n^{4/3+\delta})$ time and space complexities, where $\delta > 0$ is an arbitrarily small constant. However, those algorithms work for the L_1 distance metric and it has been left as an open problem in [2] to extend them to the more general, L_2 distance metric. Very recently, algorithms with running times that depend on the size of the output have been proposed in [12], by observing that only the edges of G that are needed by the shortest path computation in G need be computed. The most popular heuristic method that is used in path approximation, the recursive simplification heuristic of Douglas and Peucker [14], can be implemented in $O(n \log^* n)$ time [19], but does not guarantee to find an optimal solution. If the vertices of the approximating path are not required to be a subset of the vertices of the input, then faster algorithms are possible [17,18,22]. Exploiting this fact, given a monotone polygonal path P and a parameter $\varepsilon > 0$, linear time algorithms are proposed in [1] for computing an approximating path with vertices *among those* of P. Specifically, they compute in linear time an optimal set of $\frac{\varepsilon}{2}$-approximating segments for P, called a *segment cover*, such that, if the segment e_j approximates the subpath $P_{i_j,i_{j+1}} = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ of P then the endpoints of e_j are within the $\frac{\varepsilon}{2}$-radius disks centered at p_{i_j} and $p_{i_{j+1}}$, respectively. Then, they prove that the corresponding line segment $\overline{p_{i_j} p_{i_{j+1}}}$ is an ε-approximating segment for $P_{i_j,i_{j+1}}$.

Solutions to polygonal subdivision simplification problem are also based on polygonal path simplification. In [13], polygonal path simplification has been used to approximate a subdivision S with N vertices

and M extra points in $O(N(N+M)\log N)$ time. Finding a minimum size simplification is NP-hard [17] and, unless P = NP, one cannot obtain in polynomial time an approximation within a factor of $n^{1/5-\delta}$ of an optimal solution, for any $\delta > 0$ [15].

1.3. Our results

Quadratic or near-quadratic time algorithms for the planar min-# problem with L_2 distance metric have been developed almost two decades ago. However, no subquadratic time algorithms are known with the L_2 metric and no nontrivial lower bound could be proved. In this paper we present a new, query based approach for solving the min-# problem with infinite beam criterion, which carries out the computation in a dual space. We give a few results based on this approach, some of more general interest, and propose a greedy heuristic to speed up the computation. In particular, our results imply an $O(n\log n)$ time breadth first search procedure for some quadratic size graphs and an $O(n\log n)$ preprocessing, $O(\log n)$ query time solution for the following problem: Given n equal radius disks $D_1, D_2, \ldots, D_n$, construct a data structure that, for a query triplet (L, i, j), where L is a line and i and j are integers such that $1 \leqslant i < j \leqslant n$, answers whether L intersects all disks $D_i, D_{i+1}, \ldots, D_j$. Under the relaxation that the vertices of the approximating path are within the ε-tolerance regions of the vertices of P (instead of being among the vertices of P), we further give an $O(n\log n)$ time, factor 2 approximation algorithm with the infinite beam criterion. Finally, we show that the dual space approach can be used to obtain a subquadratic time exact algorithm with infinite beam criterion and L_2 distance metric if the following condition on the input path holds: $d(p_i, p_j) \notin [\varepsilon, \varepsilon\sqrt{2}]$, for $1 \leqslant i < j \leqslant n$, where $d(p_i, p_j)$ is the Euclidean distance between p_i and p_j (in fact, it is enough to require that for each vertex $p_i \in P$, only a "small" (constant) number of vertices p_j are such that $d(p_i, p_j) \in [\varepsilon, \varepsilon\sqrt{2}]$). Although in a special case, this is the first subquadratic result with the L_2 metric. For the general case our algorithm matches the best known, $O(n^2\log n)$ worst case time bound under the infinite beam criterion [10]. All our algorithms require $O(n\log n)$ storage.

The algorithms we propose are simple, based on standard geometric operations. We have implemented a version of our solution and present experimental results and comparisons with previous path approximation algorithms.

2. Preliminaries

In this section, we briefly present the general structure of the previous algorithms for solving the min-# problem. We exemplify the algorithms for the infinite beam criterion. To solve the min-# problem, those algorithms first build a path approximation, directed acyclic graph G. The vertices of G are associated with the vertices of P and edges correspond to ε-approximating line segments for P. An optimal approximating path P' is obtained by a shortest path computation in G. In general, G has $O(n^2)$ complexity ($O(n^{4/3+\delta})$ in [2]). As observed in [10], in many cases one need not explicitly construct G: an optimal path can be computed while computing the ε-approximating segments. Let $D(p_i, \varepsilon)$ denote the disk with center p_i and radius ε. The algorithms for computing the ε-approximating segments use an iterative, incremental approach similar to the following.

1: **for** $i = 1$ to $n-1$ **do**
2: $\quad j = i+1$.

3: **while** $D(p_{i+1}, \varepsilon), D(p_{i+2}, \varepsilon), \ldots, D(p_j, \varepsilon)$ admit line transversal through p_i **do**
4: if $\overline{p_i p_j}$ is an ε-approximation update the path length at p_j (if needed).
5: $j = j + 1$.
6: **end while**
7: **end for**

In this approach, the path length at a vertex of P could be updated multiple times, as the computation of ε-approximating edges proceeds. A more efficient approach [12] is to use a breadth first computation of the ε-approximating edges that simulates the breadth first shortest path computation in an ε-approximating graph G. The structure of the breadth first traversal (BFT) approach is presented below, with *enqueue* and *dequeue* being the standard queue operations.

1: enqueue(1)
2: **for** $j = 2$ to n **do**
3: $visit(j) = 0$.
4: **end for**
5: **repeat**
6: $i =$ dequeue(), $j = i + 1$.
7: **while** $D(p_{i+1}, \varepsilon), D(p_{i+2}, \varepsilon), \ldots, D(p_j, \varepsilon)$ admit line transversal through p_i **do**
8: If !$visit(j)$, check if $\overline{p_i p_j}$ is an ε-approximating segment; if yes then
set $visit(j) = 1$, set the path length at p_j and enqueue(j).
9: $j = j + 1$.
10: **end while**
11: **until** p_n has been reached.

Lemma 2. *With the BFT approach, the shortest p_1-to-p_i path length at a vertex p_i of P is correctly computed when the vertex is enqueued and will not be updated by future computation.*

Proof. Simple and omitted. □

Since some of our proofs make use of the algorithm in [2] we describe it in more details. For an input path P with n vertices, the algorithm constructs a compact (bipartite clique cover) representation $\mathcal{G}$ of the ε-approximation graph $G(P)$ by using a divide-and-conquer approach. Let P_1 be the subpath $(p_1, \ldots, p_{\lfloor n/2 \rfloor})$ of P and let P_2 be the subpath $(p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n)$ of P. The algorithm recursively computes the clique covers $\mathcal{G}_1$ and $\mathcal{G}_2$ of $G(P_1)$ and $G(P_2)$, respectively. In the merge step, the algorithm computes a clique cover $\mathcal{G}_{12}$ of the edges E_{12} of $G(P)$, where an edge $e \in E_{12}$ if it is an ε-approximating edge with one endpoint in P_1 and the other one in P_2. Then, $\mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{G}_{12}$ is a clique cover of $G(P)$. The size of the cover is shown to be $O(n^{4/3+\delta})$. Using this clique cover representation a shortest path from p_1 to p_n can be computed in $O(|\mathcal{G}| + |V|)$ time.

To obtain $\mathcal{G}_{12}$, they [2] define $Cone(p_i)$ for each $p_i \in P_1$ and $Cone(p_j)$ for each $p_j \in P_2$ as follows. $Cone(p_i)$ is the set of (rightward directed) rays emanating from p_i such that a ray $\rho \in Cone(p_i)$ if it intersects the vertical segment $p_k^- p_k^+$, for every $i \leqslant k \leqslant \lfloor n/2 \rfloor$, where p_k^+ is the point $(x_k, y_k + \varepsilon)$ and p_k^- is the point $(x_k, y_k - \varepsilon)$. Symmetrically, $Cone(p_j)$ is the set of (leftward directed) rays emanating from

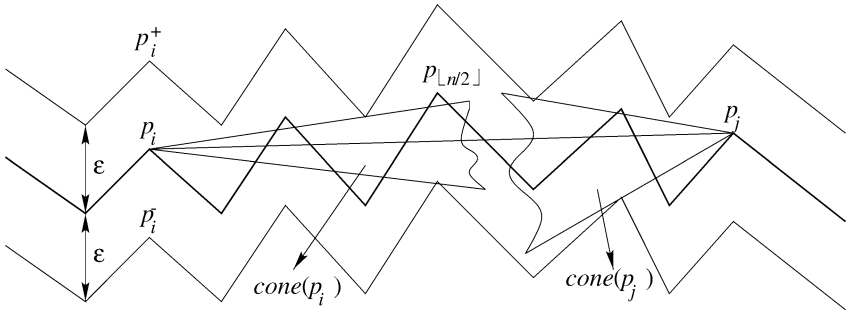


Fig. 3. Illustrating the definition of $cone(p_i)$ and $cone(p_j)$: the edge (p_i, p_j) is in $G(P)$ because $p_j \in cone(p_i)$ and $p_i \in cone(p_j)$.

p_j such that $\rho \in Cone(p_j)$ if it intersects the vertical segment $p_k^- p_k^+$, for every $\lfloor n/2 \rfloor + 1 \leqslant k \leqslant j$ (see Fig. 3).

The algorithm computes $Cone(p_i)$ for each p_i, $i = 1, 2, \ldots, n$, and then maps the lines supporting the rays in $Cone(p_i)$ to a line segment γ_i in a dual plane using a standard point-line duality transform. Let Γ_1 be the set of line segments γ_i, for $1 \leqslant i \leqslant \lfloor n/2 \rfloor$ and let Γ_2 be the set of line segments γ_i, for $\lfloor n/2 \rfloor + 1 \leqslant i \leqslant n$. Noting that $(p_i, p_j) \in E_{12}$ if and only if $p_j \in Cone(p_i)$ and $p_i \in Cone(p_j)$, the problem reduces to computing line segment intersections in the dual plane. Specifically, the problem of computing $\mathcal{G}_{12}$ reduces to computing a family $\mathcal{F} = \{(\Gamma_{11}, \Gamma_{21}), \ldots, (\Gamma_{1u}, \Gamma_{2u})\}$, where (1) $\Gamma_{1i} \subseteq \Gamma_1$ and $\Gamma_{2i} \subseteq \Gamma_2$; (2) each segment in Γ_{1i} intersects every segment in Γ_{2i} and (3) for every pair of intersecting segments $\gamma_1 \in \Gamma_1$, $\gamma_2 \in \Gamma_2$ there is a unique i such that $\gamma_1 \in \Gamma_{1i}$ and $\gamma_2 \in \Gamma_{2i}$.

The family $\mathcal{F}$ is computed by constructing a segment-intersection-searching data structure on the set Γ_2 based on a multilevel partition tree, each of whose nodes is associated with a so-called *canonical subset* of Γ_2. The total size of all canonical subsets in the tree is $O(n^{4/3+\delta})$. The queries for this structure are the segments of Γ_1. The output of a query with a segment in Γ_1 is the union of a few pairwise disjoint canonical subsets which consists of exactly those segments in Γ_2 intersecting the query segment. For each canonical subset Γ_{2i} of Γ_2, let $\Gamma_{1i} \subseteq \Gamma_1$ be the set of segments whose output contained Γ_{2i}. If $\Gamma_{1i} \neq \emptyset$, the pair $(\Gamma_{1i}, \Gamma_{2i})$ is added to the family $\mathcal{F}$.

The size of the resulting clique cover $\mathcal{G}_{12}$ of E_{12} is $\sum_i (|\Gamma_{1i}| + |\Gamma_{2i}|)$, which is $O(n^{4/3+\delta})$, and the running time for computing $\mathcal{G}_{12}$ is dominated by the time to compute the family $\mathcal{F}$, which is $O(n^{4/3+\delta})$. If $S(n)$ denotes the size of the clique cover of $G(P)$ computed by the algorithm, then the following recurrence inequality is satisfied: $S(n) \leqslant 2S(n/2) + cn^{4/3+\delta}$, where c is a positive constant. The recurrence solves for $S(n) = O(n^{4/3+\delta})$. Adding the $O(n^{4/3+\delta})$ time to compute a shortest path from p_1 to p_n once the clique cover is available gives a total of $O(n^{4/3+\delta})$ time for computing an optimal ε-approximating path with the L_1 distance metric.

3. A query based approach

In this section we present our query approach for solving the polygonal path approximation problem. We first define two key operations, $Query(\cdot, \cdot)$ and $Span(\cdot)$, and show how to implement them efficiently. Then, we show how to use them to obtain efficient algorithms with: (i) a special, "vertical" error measure,

and (ii) the L_2 distance metric, under the assumption that $d(p_i, p_j) \notin [\varepsilon, \varepsilon\sqrt{2}]$, for $1 \leqslant i < j \leqslant n$, with $d(p_i, p_j)$ the L_2 distance between p_i and p_j.

From Lemma 2 above, it follows that with the BFT approach a vertex p_j of P that has been marked as visited need not be checked again in pairs (p_i, p_j), with $i < j$. Let p_i be the vertex dequeued by the BFT algorithm. Since it may happen that many vertices immediately following p_i have been already visited, we should avoid including those vertices in future computation associated with p_i. However, incremental approaches cannot avoid this computation. Then, it may be appropriate to combine the BFT algorithm described in the previous section with a query method for computing ε-approximating edges, rather than using the incremental method of computation. This may also be interesting for practical applications mentioned earlier, such as animation, where one may want to approximate various subpaths of P at different times. To this end, we divide the vertices of P in three categories: (1) *processed*: those visited by BFT that are no longer in the queue, (2) *active*: those visited by BFT that are in the queue and (3) *inactive*: those that have not yet been reached by the BFT. With a query method, one should be able to answer fast if $\overline{p_i p_j}$ is an ε-approximating segment without first having to compute information for vertices p_k, where $i < k < j$. Specifically, the following two operations should be supported.

- *Span*(i): compute the largest index j such that there is a line stabler through p_i for the set of disks $D(p_{i+1}, \varepsilon), D(p_{i+2}, \varepsilon), \ldots, D(p_j, \varepsilon)$ ($\overline{p_i p_j}$ could be an ε-approximating segment).
- *Query*(i, j): answer if $\overline{p_i p_j}$ is an ε-approximating segment.

Let V_1, V_2 and V_3 be the sets of processed, active and inactive vertices, and consider the BFT algorithm at some stage. The queue associated with the BFT contains a set of vertices such that the shortest path lengths of any two of them differ by at most one. In other words, all vertices can be reached from p_1 with $k-1$ or k links, for some integer $2 \leqslant k \leqslant n-2$. If p_1^i denotes the shortest p_1-to-p_i path length at p_i, then $p_1^i \leqslant k$ if $p_i \in V_1 \cup V_2$ and $p_1^i \geqslant k$ if $p_i \in V_3$. We further augment the BFT algorithm with a greedy approach: vertices in V_2 are maintained in two priority queues having as keys the indices of vertices of P, such that a dequeue operation on each of the queues returns the largest index in the queue. The first priority queue corresponds to vertices V_2^1 that can be reached with $k-1$ links and the second one to those vertices V_2^2 reachable with k links.

Lemma 3. *The time to maintain the two priority queues is* $O(f(m)\log(f(m)))$, *where* $f(m) = O(n)$ *is the number of vertices of* P *that can be reached from* p_1 *with no more than* $m-1$ ε*-approximating segments and* m *is the number of vertices of a* min-# *approximating path.*

Proof. The total number of vertices visited by the BFT algorithm before p_n is reached is $O(f(m))$. Only the visited vertices are enqueued in one of the two priority queues, resulting in a total of $O(f(m)\log(f(m)))$ time. □

Observe that it is possible to have $p_i \in (V_1 \cup V_2)$ and $p_j \in V_3$ such that $i > j$. However, only pairs of the form (i, j), with $p_i \in V_2^1$, $p_j \in V_3$ and $i < j$ should be considered for *Query*(i, j) operations. Thus, we need to maintain the set of inactive vertices such that, when a vertex p_i is dequeued from the BFT queue, the inactive vertices p_j, with $j > i$, are easily available.

Lemma 4. *Assuming Span(i) is known, the inactive vertices for p_i can be found in $O(\log I + k)$ time, where I is the set of currently inactive vertices and k is the number of inactive vertices in I with indices between i and Span(i). The set I of inactive vertices can be maintained in $O(n + f(m)\log n)$ time, using $O(n)$ space.*

Proof. Use a balanced binary search tree T on the inactive vertices. Initially, p_1 is in the queue and $p_2, p_3, \ldots, p_n$ are in T. The keys in T are the indices of the vertices in T. Assume that, after a number of steps in the BFT algorithm, p_i is dequeued from the queue. We search T to find the first vertex with index larger than i. Then, the inactive vertices up to *Span*(i) can be found by a simple traversal in T. If one of the visited vertices p_j becomes active (as a result of *Query*(i, j)), we remove the vertex from T. Then, since each remove operation in T takes $O(\log n)$ time and there are $O(f(m))$ remove operations overall, we obtain the claimed time and space bounds. □

From Lemmas 3 and 4 it follows that the time complexity of the greedy BFT algorithm depends on the time complexities for performing *Span*$(\cdot)$ and *Query*$(\cdot,\cdot)$ operations and the number of failed *Query*$(\cdot,\cdot)$ operations.

Let $d(p_i, p_j)$ denote the Euclidean distance between p_i and p_j. The following lemma bounds the time complexities of *Span*(i) and *Query*(i, j) operations.

Lemma 5. *With $O(n \log n)$ preprocessing, for any pair (p_i, p_j), with $1 \leqslant i < j \leqslant n$, Query$(i, j)$ can be answered in $O(\log n)$ time. In addition, if $d(p_i, p_j) \notin [\varepsilon, \varepsilon\sqrt{2}]$, for all $1 \leqslant i < j \leqslant n$, then Span$(i)$ can be answered in $O(\log^2 n)$ time.*

To prove this lemma we first introduce some geometric structures. Let p_i and p_j, with $i < j$, be two vertices of P. For a vertex $p_k \in P$, with $i < k < j$, let σ_k denote the set of lines that are tangent to the disk $D(p_k, \varepsilon)$. Using a standard point-line duality transform, σ_k is mapped to a hyperbola H_k in the dual plane [20]. Any vertical line in the dual plane intersects each of the two branches of H_k exactly once; the upper branch H_k^a corresponds to tangents to the upper semicircle of $D(p_k, \varepsilon)$ and the lower branch H_k^b corresponds to tangents to the lower semicircle (see Fig. 4). Each branch is an x-monotone and unbounded Jordan curve and $H_k^a \cap H_k^b = \Phi$. The dual of a line transversal of $D(p_k, \varepsilon)$ corresponds to a point below H_k^a and above H_k^b. Observe that any pair (H_i^c, H_j^d) of hyperbolic branches intersects in at most one point, where $i \neq j$ and $c, d \in \{a, b\}$. (This it true only for equal-radius disks; for different radii disks each pair can intersect in at most two points.)

Let $\mathcal{L}_{ij}$ be the *lower envelope* of H_k^a and let $\mathcal{U}_{ij}$ be the *upper envelope* of H_k^b, for $k = i, i+1, \ldots, j$. Then, $L(\overline{p_i p_j})$ is a common transversal (a line stabler) of $\{D(p_{i+1}, \varepsilon), D(p_{i+2}, \varepsilon), \ldots, D(p_{j-1}, \varepsilon)\}$ (the line segment $\overline{p_i p_j}$ is a valid approximation segment) if and only if its dual point lies between $\mathcal{L}_{ij}$ and $\mathcal{U}_{ij}$. From [27], it follows that the complexities of $\mathcal{L}_{ij}$, $\mathcal{U}_{ij}$ and the region $\mathcal{I}_{ij}$ sandwiched between them are $O(j - i)$ and that $\mathcal{I}_{ij}$ has at most one connected component. (The last property is not true for different radii disks, in which case there could be $O(j - i)$ connected components.) Since for a set of n equal-radius disks, the space of line transversals that are restricted to pass through a common point p can have $O(n)$ connected components [10,12], the line dual to p_i can have $O(j - i)$ disjoint segments in $\mathcal{I}_{ij}$. This is a key property that makes the min-# problem with infinite beam criterion somehow harder than with other error criteria (such as tolerance zone and uniform measure). It gives rise to the condition in Lemma 5: requiring that $d(p_i, p_j) \notin [\varepsilon, \varepsilon\sqrt{2}]$, for $1 \leqslant i < j \leqslant n$, assures that there is only one connected

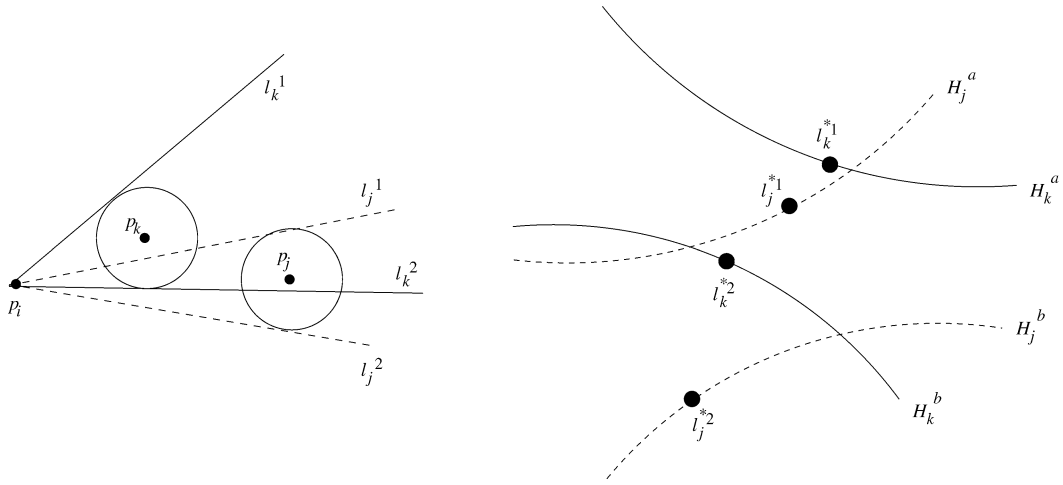


Fig. 4. Dual transforms of tangents to disks: The tangents to the disk $D(p_k, \varepsilon)$ ($D(p_j, \varepsilon)$) map to the hyperbola H_k (H_j) in dual plane and the line l_k^1 (l_k^2, l_j^1 or l_j^2) maps to the point l_k^{*1} (l_k^{*2}, l_j^{*1} or l_j^{*2}) in dual plane.

component (the details are given in [12]). Consequently, the line dual to p_i has at most one line segment in $\mathcal{I}_{ij}$.

If we use a vertical error measure, however, where the error region of a point is a vertical line segment instead of a disk, the space of line transversals that are restricted to pass through a common point can be easily seen to have at most one connected component.

Lemma 6. *The* min-# *problem with infinite beam criterion and vertical error measure can be solved in* $O(n^{4/3+\delta})$ *time and space, where* $\delta > 0$ *is an arbitrarily small constant.*

Proof. The result can be obtained by a simple modification of Agarwal and Varadarajan [2] min-# algorithm for x-monotone paths with uniform metric. For infinite beam criterion and vertical distance metric, the difference is that right (or left) oriented rays are replaced by lines (the path P is not monotone) and thus wedges are replaced by double wedges. The only places where this plays a role are in the computation of the lower and upper convex hulls (of the upper and lower endpoints of the vertical error segments) and in the computation of the double cone at p_i. The double cone at p_i can be found by computing the largest separating double cone of the two convex hulls, if such a separation is possible. This reduces to computing the tangents from p_i to the two convex hulls. Note that the convex hulls are used only to help compute the double cone at p_i, since only the double cone at p_i is important for the outcome of the algorithm. Then, to perform the incremental updating of the convex hulls and to compute the tangents in altogether $O(\log n)$ time, as in [2], it suffices to use the solution in [16], based on the on-line convex hull algorithm of Avis et al. [4]. Alternatively, we can maintain the convex hulls and answer queries efficiently by using recently developed linear space dynamic planar convex hull data structures, that allow insert and deletion of points in amortized $O(\log n)$ time per operation and support tangent queries through a given point in $O(\log n)$ time [8]. □

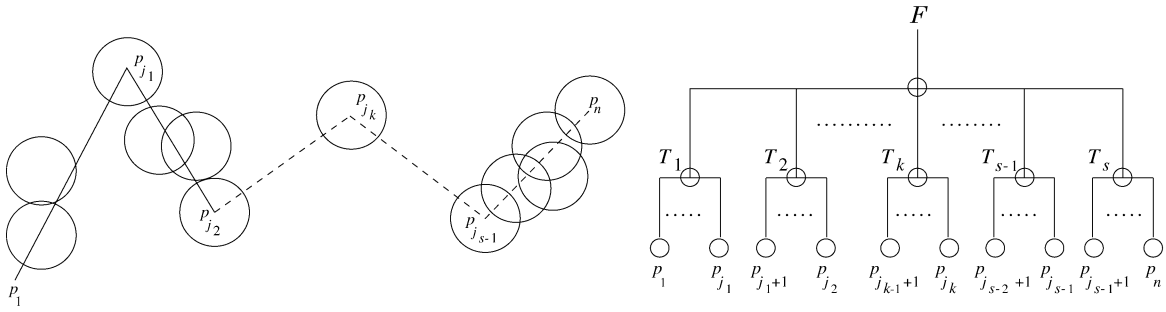


Fig. 5. Illustration of the forest $\mathcal{F}$ of trees $T_1, T_2, \ldots, T_s$.

We remark that a similar approach can be used with the L_1 metric. In that case, the space of line transversals to n L_1 disks, which are restricted to pass through a common point p, can have at most two connected components. This property is implicitly exploited by the subquadratic time algorithm in [2].

In what follows, we set up the data structures for computing *Span*(i) and *Query*(i, j). Let T be a complete binary tree such that the leaves of T are associated, in order, with the vertices of P. At each leaf l_i, we also store the hyperbolic branches H_i^a and H_i^b, the duals of the tangent lines to $D(p_i, \varepsilon)$. For an internal node v in T, let i and j be the smallest and largest indices of the leaf descendants of v. We store at v the region $I(v) = \mathcal{I}_{ij}$ sandwiched between the lower envelope $\mathcal{L}_{ij}$ of $\{H_k^a \mid i \leqslant k \leqslant j\}$ and the upper envelope $\mathcal{U}_{ij}$ of $\{H_k^b \mid i \leqslant k \leqslant j\}$. Since any two hyperbolic branches intersect each other at most once, $\mathcal{L}_{ij}$ and $\mathcal{U}_{ij}$ have complexity $O(j-i)$ [27]. We compute $I(v)$ for all vertices v of T, using a divide-and-conquer method implemented by a bottom up traversal of T. The boundary of $I(v)$ is maintained as two monotone pieces, corresponding to the contributions of the lower and upper envelopes of the two sets of hyperbolic branches stored at the leaf descendants of v. Like in [5], $I(v)$ can be computed, for all vertices v in T, in a total of $O(n \log n)$ time by simple merge-like operations. Note that in general the region $I(r)$ at the root r of T may be empty and thus T is a forest. We reorganize T, in the same time bound, in a family of trees $\mathcal{F} = \{T_1, T_2, \ldots, T_s\}$ such that T_1's leaves correspond to the longest possible prefix $D_1 = \{D(p_1, \varepsilon), D(p_2, \varepsilon), \ldots, D(p_{j_1}, \varepsilon)\}$ that admits a line transversal, T_2's leaves correspond to the longest possible prefix starting at p_{j_1+1}, and so on (see Fig. 5). Note that it is easy to construct examples where T would have $\Theta(\log n)$ more trees with nonempty intersection at the root than $\mathcal{F}$ (e.g., when all the vertices of P are on the x-axis except the last one, which is far above it). $\mathcal{F}$ is also useful in support of Lemma 7 below. We explain the process for obtaining T_1. For the remaining trees in $\mathcal{F}$ the process is similar.

Let $j_0 = 0$. For the tree T_1 of $\mathcal{F}$, we should find the largest possible index j_1 such that the region $\mathcal{I}_{j_0+1, j_1}$ is not empty. We begin to construct T_1 by forming a search path π_1 starting from the leaf p_1 of T. In general, the path π_i for some T_i has an ascending phase and a descending phase. In each step of the ascending phase of the path π_1, we go up to the parent node v_p of the current node. We stop when $I(v_p)$ is empty and we come at v_p from a left child. Clearly, the leaves with indices from $j_0 + 1$ to the largest index of a leaf node in the subtree rooted at the left child v_l of v_p appear in T_1. Let $\mathcal{I}_1$ denote the region of these leaf descendants of v_p. $\mathcal{I}_1$ is computed while traversing π_1 using the merge-sort like procedure described for T (note that for T_1 it is the region stored at v_l).

We then begin the descending phase of π_1 from the right child v_r of v_p and keep going down to the left child v_{r_l} of v_r until the intersection of $I(v_{r_l})$ and $\mathcal{I}_1$ is not empty or a leaf p_k is reached. In the later

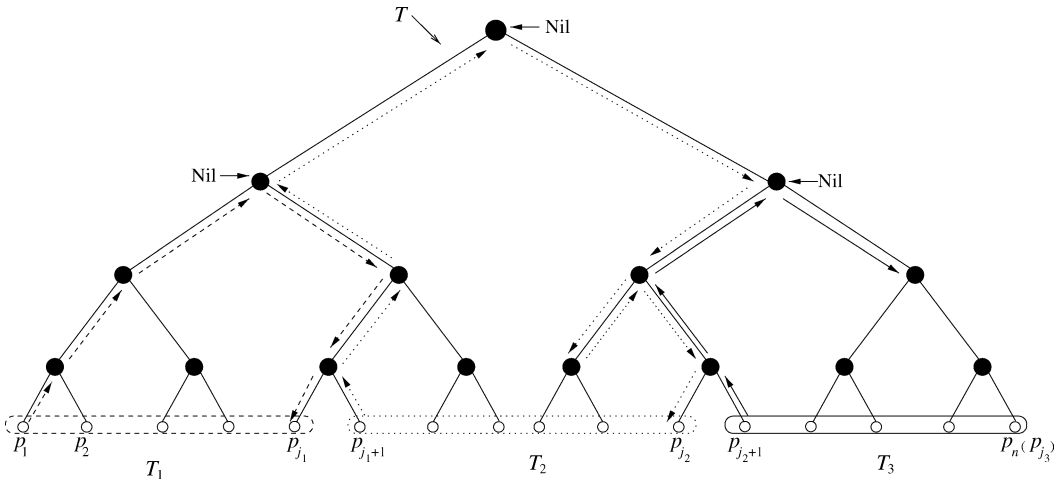


Fig. 6. The construction of the forest $\mathcal{F} = \{T_1, T_2, T_3\}$.

case, j_1 is set to $k-1$ or k, depending on whether the merged region of $I(p_k)$ and $\mathcal{I}_1$ is empty or not. In the former case, we update $\mathcal{I}_1$ as the intersection of $I(v_{r_l})$ and $\mathcal{I}_1$ and continue the descending phase from the sibling node v_{r_r} of v_{r_l} (v_r becomes v_{r_r}). The descending path stops at some leaf node and we set j_1 to the index of that node or the index of the previous node, depending on whether the last node gives a non-empty or an empty intersection with $\mathcal{I}_1$. Thus, the value of j_1 is the sought largest possible index of T_1.

With that, we begin to find the longest possible prefix for T_2 by forming another search path π_2 starting from the leaf node p_{j_1+1}. We construct T_2, T_3 and so on in the same way as we construct T_1 (see Fig. 6). Since for each tree T_i of $\mathcal{F}$, for $1 \leqslant i \leqslant s$, the time to merge the regions associated with the leaves of T_i at each node on π_i is $O(n_i)$, where n_i is the number of leaves of T_i, and the search path π_i in T has $O(\log n)$ nodes, the computation of T_i takes $O(n_i \log n)$ time. Noting that $\sum_1^s O(n_i) = n$, the total time to construct the forest $\mathcal{F}$ in $O(n \log n)$.

This implies the following result.

Lemma 7. *$\mathcal{F}$ can be used to compute an approximating path with vertices inside the error tolerance regions of the vertices of P and of size at most twice the size of an optimal ε-approximating path with vertices among the vertices of P.*

Proof. For each $T_i \in \mathcal{F}$, pick a point in the intersection stored at the root of T_i and trim the dual line to a line segment with endpoints inside the first leaf and the last leaf of T_i, resulting in $|\mathcal{F}|$ line segments. Then join the resulting segments to form a path, which adds $|\mathcal{F}| - 1$ segments. Obviously, an optimal approximating path must have at least $|\mathcal{F}|$ vertices, from which the claim follows. Once $\mathcal{F}$ is available, the construction takes $O(|\mathcal{F}|)$ time. □

Note that our 2-factor approximation solution is different from the greedy solutions in [1,17,24], which consider wedges instead of double wedges and/or may depend on some predefined stabbing order. Also, it can be easily seen that the same results hold for other distance metrics, such as L_1 metric.

We now give the proof for Lemma 5.

Proof of Lemma 5. Let p_{ij} be the dual of the line $L(\overline{p_i p_j})$. To perform a *Query*(i, j) operation, we form a search path π_{ij} in $\mathcal{F}$, from the leaf l_i to the leaf l_j. Observe that l_i and l_j may be in the same tree or in different, adjacent trees of $\mathcal{F}$. We say that a node v of T is on the *right fringe* (*left fringe*) of π_{ij} if v is not on π_{ij} but it is a right (left) child of a node on π_{ij}. Noting that the query is a decomposable search problem, we can answer *Query*(i, j) by determining if p_{ij} is inside $I(v)$ for each v that is a right (left) fringe node of π_{ij} (answering "true" if and only if it is inside them all). Using the approach in [5], based on fractional cascading technique of Chazelle and Guibas [6,7], the answer for *Query*(i, j) can be obtained in $O(\log n)$ time. *Span*(i) computation proceeds in a similar way. We form a search path π_i in $\mathcal{F}$ as follows. We start at l_i and go up the corresponding tree T_{l_i} storing l_i until we find a node with nonempty intersection on the right fringe or we reach the root of T_{l_i}. If the root has been reached, we go on a downward path on T_{l_i+1} (if it exists) until we find a node with nonempty intersection on the left fringe. Let v be the root of that subtree (right or left fringe) and let $l(p_i)$ be the dual line of p_i. We then compute the intersection of $l(p_i)$ with $I(v)$. The condition in the lemma assures that $l(p_i) \cap I(v)$ has at most one connected component. If $v \in T_{l_i}$ and $l(p_i) \cap I(v) \neq \Phi$, set $l(p_i) = l(p_i) \cap I(v)$. Then go up T_{l_i} until a node w on the right fringe of π_i with $l(p_i) \cap I(v) = \Phi$ is found or the root of T_{l_i} has been reached, while updating $l(p_i)$ along the way. Then follow a downward path (possibly in T_{l_i+1}) until a node with nonempty left fringe is found. We then proceed down the tree as follows. Let v be the current node and let v_l and v_r be the left and right children of v. If $l(p_i) \cap I(v_l) \neq \Phi$ proceed on right fringe of v with $l(p_i) = l(p_i) \cap I(v_l)$, else proceed on the left fringe of v with $l(p_i)$. The computation stops at some leaf l_j. Clearly, *Span*$(i) = j - 1$. The computation for *Span*(i) at a node in the tree is similar to that for *Query*(i, j) without fractional cascading. At each node v on the search path we compute the two endpoints of the intersection of $l(p_i)$ with the boundary of $I(v)$ using binary search on the boundary of $I(v)$, which takes $O(\log n)$ time. There are $O(\log n)$ levels on the search path and thus *Span*(i) can be answered in $O(\log^2 n)$ time. We note here that if one is willing to increase the preprocessing time to $O(n^2 \log n)$, by precomputing $l(p_i) \cap I(v)$ for all nodes $v \in \mathcal{F}$ and $i = 1, 2, \ldots, n$, then *Span*(i) can be answered in $O(\log n)$ time. □

Corollary 1. *Given a set of n equal radius disks $D_1, D_2, \ldots, D_n$, in $O(n \log n)$ time one can construct a data structure of size $O(n \log n)$ such that, for a query triplet (L, i, j), where L is a line and i and j are integers, $1 \leqslant i < j \leqslant n$, it can be decided in $O(\log n)$ time whether L intersects all disks $D_i, D_{i+1}, \ldots, D_j$.*

The results in Lemmas 3, 4 and 5 can be combined to obtain an efficient algorithm for solving the min-# problem. Alternatively, we can use Lemma 5 together with the divide and conquer procedure in [2] to obtain the following result.

Lemma 8. *Under the condition in Lemma 5, that $d(p_i, p_j) \notin [\varepsilon, \varepsilon\sqrt{2}]$, for all $1 \leqslant i < j \leqslant n$, the* min-# *problem with the infinite beam criterion and L_2 distance metric can be solved in $O(n^{4/3+\delta})$ time, where $\delta > 0$ is an arbitrarily small constant.*

Proof. We can obtain the result by applying the divide and conquer procedure in [2], enhanced with $\mathcal{F}$ for constructing a clique cover $\mathcal{G}$ of the ε-approximation graph G in $O(n^{4/3+\delta})$ time, and then computing a shortest path from p_1 to p_n in G in $O(|\mathcal{G}| + |V|)$ time. Since the size of the clique cover $\mathcal{G}$ of G is $O(n^{4/3+\delta})$, the total time to solve the min-# problem is $O(n^{4/3+\delta})$.

The main difficulty in applying the technique in [2] is in computing the two sets of segments used in the combine phase of the divide-and-conquer algorithm. Since that phase is dominated by the $O(n^{4/3+\delta})$ time to compute canonical subsets for segment intersection queries, to keep the same bound we need to compute the two sets of segments in no more than $O(n^{4/3+\delta})$ time. In what follows, we show how to use $\mathcal{F}$ to compute these two sets in $O(n \log^2 n)$ time.

With the infinite beam criterion and L_2 distance metric, the right (or left) rays in [2] are replaced by lines and thus wedges are replaced by double wedges. Therefore, the double cone at p_i, for $1 \leqslant i \leqslant \lfloor n/2 \rfloor$, contains a set of lines that pass through p_i and intersect the disk $D(p_k, \varepsilon)$, for every $i \leqslant k \leqslant \lfloor n/2 \rfloor$. Symmetrically, the double cone at p_j, for $\lfloor n/2 \rfloor + 1 \leqslant j \leqslant n$, contains a set of lines that pass through p_j and intersect the disk $D(p_k, \varepsilon)$, for every $\lfloor n/2 \rfloor + 1 \leqslant k \leqslant j$. We can use $\mathcal{F}$ to compute the line segment that is the dual of the double cone at p_i, for every $1 \leqslant i \leqslant n$, in a total of $O(n \log^2 n)$ time. The computation for some p_i is similar to that of *Span*(i): form a search path π_i in $\mathcal{F}$ and obtain the line segment $l(p_i)$ by computing the intersection of $l(p_i)$ and $I(v)$ for each node v on (the right/left fringe of) π_i. Note that if $p_{\lfloor n/2 \rfloor}$ is in some $T_k \in \mathcal{F}$, then we only need consider the vertices associated with T_k and the predecessor and successor of T_k in $\mathcal{F}$ (if there exists a line $\overline{p_i p_j}$ stabbing the disks $D(p_{i+1}, \varepsilon), D(p_{i+2}, \varepsilon), \ldots, D(p_j, \varepsilon)$, where p_i is in T_{k-2} and p_j is in T_k, then we can have T'_{k-1} starting at $p_{j_{k-2}+1}$ and ending at p_j, which gives a longer prefix than T_{k-1}). The construction of $\mathcal{F}$ takes $O(n \log n)$ time and can be done in a preprocessing step. Since the computation of the double cones for the combine phase takes $O(n \log^2 n)$ time, the combine phase remains dominated by the $O(n^{4/3+\delta})$ time to construct the canonical subsets from segment intersection queries. Then, the time recurrence remains $S(n) \leqslant 2S(n/2) + cn^{4/3+\delta}$ and we obtain the claimed time bound. □

Although in a special case, this is the first subquadratic result for path approximation with L_2 distance metric. The result is useful especially when the algorithm is used to approximate borders of fat planar regions and terrains, and in medical imaging, where the condition stated in the lemma may be often satisfied. For the general case, as shown below, the query approach matches the time complexity of the best known algorithms for polygonal chain approximation with infinite beam criterion and L_2 distance metric.

Theorem 1. *With the query approach, the* min-# *problem with infinite beam criterion and L_2 metric can be solved in* $O(n^2 \log n)$ *time and* $O(n \log n)$ *space.*

Proof. We note that we can find a solution using only *Query*$(\cdot, \cdot)$ operations. For each v_i, $1 \leqslant i < n$, there are $O(n)$ *Query*(i, j) operations, where $i < j \leqslant n$. Since each *Query*(i, j) takes $O(\log n)$ time the query based algorithm requires $O(n^2 \log n)$ time. □

Note. Some of the ideas in this section can be applied to other, unrelated problems. In particular, we can extend Lemma 4 as follows. Let G be an unweighted directed graph with n vertices, such that for each vertex $v_i \in G$, there are edges $(v_i, v_{i_1}), (v_i, v_{i_1+1}), (v_i, v_{i_1+2}), \ldots, (v_i, v_{i_2}) \in G$. That is, G can be specified by its set of vertices and the index ranges (i_1, i_2) associated with each vertex $v_i \in G$. If a standard representation is used for storing G, $O(n^2)$ space is required, since G can have $O(n^2)$ edges. Computing a single source shortest path in G using standard breadth first search would take $O(n^2)$ time.

Lemma 9. *A single source shortest path tree in a graph G specified as above can be computed in* $O(n \log n)$ *time and* $O(n)$ *space.*

Proof. Similar to the proof of Lemma 4, we begin with the $O(n \log n)$-time construction of a balanced binary search tree T on the vertices of G, where the keys in T are the indices of the vertices of G. Assume v_1 is the source vertex. Initially, v_1 is in the queue and $v_2, v_3, \ldots, v_n$ are in T. Since we have the index ranges (i_1, i_2) associated with each vertex v_i of G, when v_i is dequeued from the queue it takes $O(\log n)$ time to find the first unvisited vertex with index no smaller than i_1, by searching T with the key i_1. Then, the unvisited vertices up to v_{i_2} can be found in $O(\log n + k)$ time by a simple traversal in T, where $k \leqslant |i_2 - i_1|$. Note that the number of unvisited vertices can be smaller than $|i_2 - i_1|$ since some vertices with index in the range $[i_1, i_2]$ may have been removed from T. We enqueue the visited vertices into the queue and remove them from T. Since the removal does not increase the height of T, we need not perform a standard deletion on T for a visited node. The traversal of the subtree of T for the nodes in the range $[i_1, i_2]$ can simulate an Euler Tour traversal of that subtree, in which each node is visited at most three times (from the left, from below and from the right). Deleting a node when it is last visited in the tour takes $O(1)$ time. Then, the total time required by remove operations is $O(n)$ and we obtained the claimed time and space bounds. Obviously, the same bounds hold even if we perform a standard $O(\log n)$ time delete operation for each visited node. □

4. Experimental results

In this section, we report the results of our experiments and give comparisons for three algorithms for solving the min-# problem. The first algorithm corresponds to the iterative, incremental approach presented in Section 2. As mentioned there, the path length at a vertex could be updated more than once during the computation. For the second algorithm, which is also presented in Section 2, we use a breadth first traversal (BFT) approach and put a vertex into a standard queue if the path length of this vertex is updated (the path length at a vertex may be updated at most once). For the third algorithm, we use two priority queues having as keys the indices of vertices of P, such that a dequeue operation on each of the queues returns the largest index in the queue. We have observed that the BFT and priority queue algorithms are much more efficient than the incremental algorithm and the priority queue algorithm has better performance than the BFT algorithm on average. Specifically, we define the following three parameters to compare these algorithms.

- *sum_check*: the sum of the check time of all vertices in P, where the check time of a vertex is the number of times it is visited during computation.
- *max_check*: the max value of the check time at a vertex of P.
- *sum_edges*: the number of ε-approximating edges in the path approximating graph G.

For example, for the path in Fig. 7, (*sum_check*, *max_check*, *sum_edges*) is $(90, 9, 54)$ with the incremental algorithm, $(57, 8, 10)$ with BFT, and $(19, 2, 9)$ with the priority queue algorithm. The path in Fig. 8 gives $(4329, 49, 2063)$ with the incremental algorithm, $(3709, 37, 194)$ with BFT, and $(3092, 31, 194)$ with the priority queue algorithm.

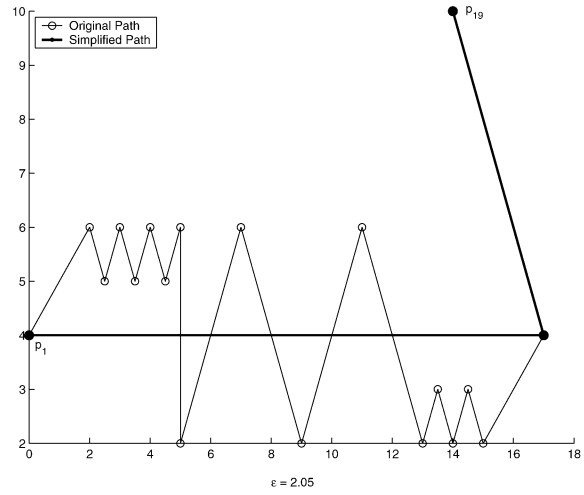


Fig. 7. A path with 19 vertices and its approximation ($\varepsilon = 2.05$).

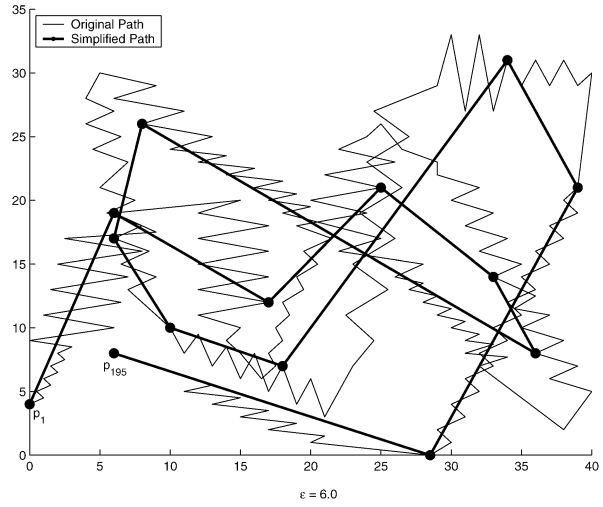


Fig. 8. A path with 195 vertices and its approximation ($\varepsilon = 6.0$).

We have generated two sets of test data with different error tolerance and varying number of vertices on the path, respectively. In each case of the first set, we randomly generate the same number of points in a $100 * 100$ area and test the inputs with different error tolerance ε. In each case of the second set, we randomly generate different number of points in a $10 * 10$ area and test the inputs with the same error tolerance. The results are illustrated in Figs. 9–14 and support our conclusion: (1) the incremental approach has the worst performance; (2) the BFT and the priority queue algorithms are much more efficient and (3) on average the priority queue is better than BFT. The difference in running times between the incremental and BFT approach increases as ε increases or n increases. This corresponds to the intuition that BFT should be much faster if coarser approximations are sought.

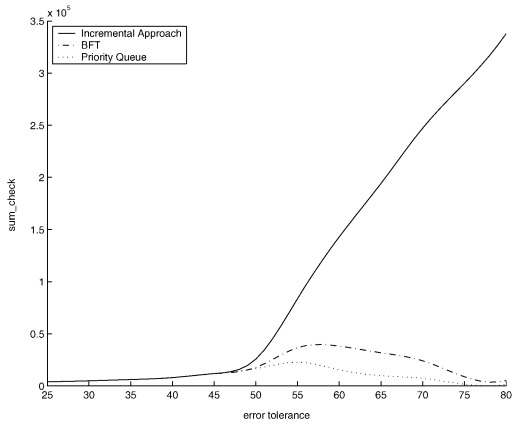


Fig. 9. Comparison of *sum_check* values with different error tolerance.

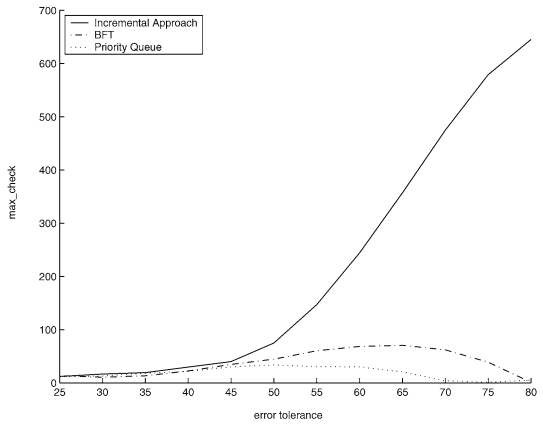


Fig. 10. Comparison of *max_check* values with different error tolerance.

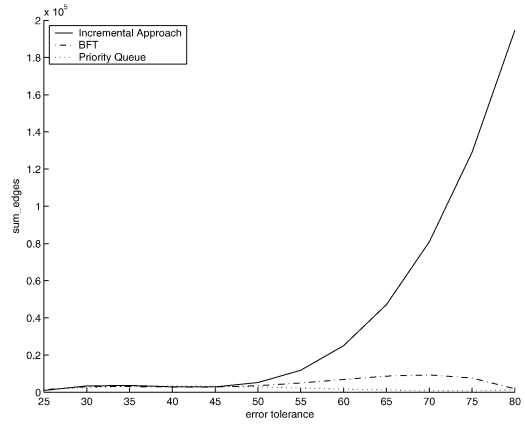


Fig. 11. Comparison of *sum_edges* values with different error tolerance.

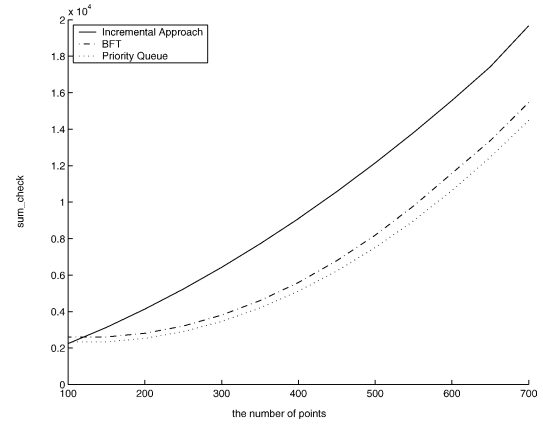


Fig. 12. Comparison of *sum_check* values with different number of points.

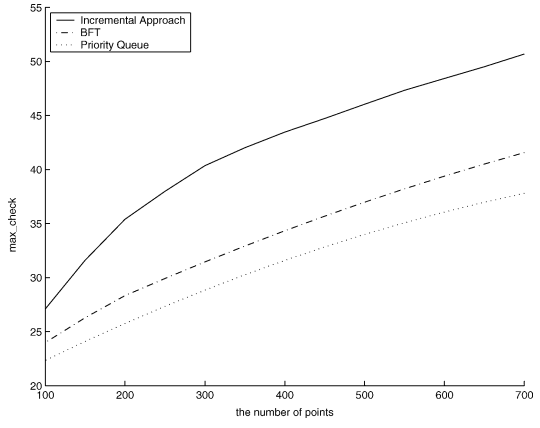


Fig. 13. Comparison of *max_check* values with different number of points.

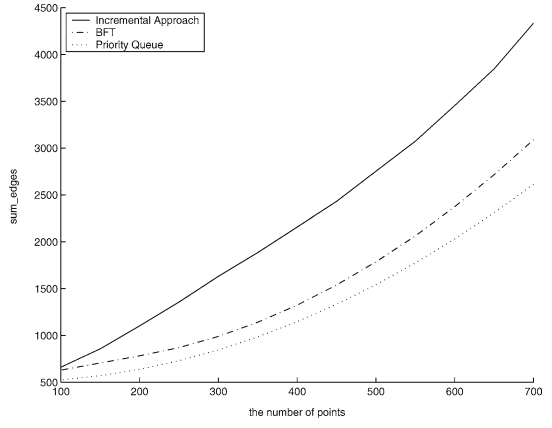


Fig. 14. Comparison of *sum_edges* values with different number of points.

5. Conclusions

We have presented a query based approach for approximating polygonal chains in the plane, with the infinite beam criterion, which matches the worst case time of the best known algorithms for this problem. We also showed that the query based approach can be used to obtain a subquadratic time exact algorithm with infinite beam criterion and Euclidean distance metric if some condition on the input path holds. Although in a special case, this is the first subquadratic result for path approximation with Euclidean distance metric. The data structures we construct can be used to obtain an $O(n \log n)$ time, factor 2 approximation algorithm for the problem.

We conjecture that our results, under the condition of Lemma 5, can be extended to the tolerance zone criterion. A technical difficulty is in deciding whether the line segment $\overline{p_i p_j}$ intersects all disks with index $i < k < j$, given that they are intersected by $L(\overline{p_i p_j})$. It remains an open problem to extend our results for general paths while maintaining subquadratic time.

Acknowledgements

The authors are thankful to Pankaj Agarwal for useful discussions.

References

[1] P.K. Agarwal, S. Har-Peled, N. Mustafa, Y. Wang, Near-linear time approximation algorithms for curve simplification, in: Proc. of 10th Annual European Sympos. Algorithms, 2002, pp. 29–41.
[2] P.K. Agarwal, K.R. Varadarajan, Efficient algorithms for approximating polygonal chains, Discrete Comput. Geom. 23 (2000) 273–291.
[3] H. Alt, J. Blomer, M. Godau, H. Wagener, Approximation of convex polygons, in: Proc. 10th Coll. on Autom. Lang. and Prog. (ICALP), 1990, pp. 703–716.
[4] D. Avis, H. ElGindy, R. Seidel, Simple on-line algorithms for convex polygons, in: G.T. Toussaint (Ed.), Computational Geometry, 1985.
[5] G. Barequet, D.Z. Chen, O. Daescu, M.T. Goodrich, J. Snoeyink, Efficiently approximating polygonal paths in three and higher dimensions, Algorithmica 33 (2) (2002) 150–167.
[6] B. Chazelle, L.J. Guibas, Fractional cascading: I. A data structuring technique, Algorithmica 1 (1986) 133–162.
[7] B. Chazelle, L.J. Guibas, Fractional cascading: II. Applications, Algorithmica 1 (1986) 163–191.
[8] G.S. Brodal, R. Jacob, Dynamic planar convex hull, in: Proc. 43rd Annual Symp. on Foundations of Computer Science, vol. 25, 2002, pp. 617–626.
[9] W.S. Chan, F. Chin, Approximation of polygonal curves with minimum number of line segments or minimum error, Internat. J. Comput. Geom. Appl. 6 (1) (1996) 59–77.
[10] D.Z. Chen, O. Daescu, Space-efficient algorithms for approximating polygonal curves in two dimensional space, Internat. J. Comput. Geom. Appl. 13 (2) (2003) 95–112.
[11] L.P. Cordella, G. Dettori, An $O(n)$ algorithm for polygonal approximation, Pattern Recognition Letters 3 (1985) 93–97.
[12] O. Daescu, New results on path approximation. Algorithmica (Special Issue on Shape Algorithmics), in press.
[13] M. de Berg, M. van Kreveld, S. Schirra, Topologically correct subdivision simplification using the bandwidth criterion, Cartography and GIS 25 (1998) 243–257.
[14] D.H. Douglas, T.K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Canadian Cartographer 10 (2) (1973) 112–122.
[15] R. Estkowski, J.S.B. Mitchell, Simplifying a polygonal subdivision while keeping it simple, in: Proc. 17th ACM Symposium on Computational Geometry, 2001, pp. 40–49.

[16] D. Eu, G.T. Toussaint, On approximation polygonal curves in two and three dimensions, CVGIP: Graphical Models and Image Processing 56 (3) (1994) 231–246.
[17] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, J.S. Snoeyink, Approximating polygons and subdivisions with minimum link paths, Internat. J. Comput. Geom. Appl. 3 (4) (1993) 383–415.
[18] S.L. Hakimi, E.F. Schmeichel, Fitting polygonal functions to a set of points in the plane, CVGIP: Graphical Models and Image Processing 53 (2) (1991) 132–136.
[19] J. Hersberger, J. Snoeyink, Cartographic line simplification and polygon csg formulae in $O(n \log^* n)$ time, in: Proc. 5th International Workshop on Algorithms and Data Structures, 1997, pp. 93–103.
[20] F. Hurtado, P. Ramos, M. Noy, C. Seara, Separating objects in the plane with wedges and strips, Discrete Appl. Math. 109 (2001) 109–138.
[21] H. Imai, M. Iri, Computational-geometric methods for polygonal approximations of a curve, Computer Vision, Graphics and Image Processing 36 (1986) 31–41.
[22] H. Imai, M. Iri, An optimal algorithm for approximating a piecewise linear function, J. Inform. Process. 9 (3) (1986) 159–162.
[23] H. Imai, M. Iri, Polygonal approximations of a curve-formulations and algorithms, in: Computational Morphology, North-Holland, Amsterdam, 1988, pp. 71–86.
[24] A. Melkman, J. O'Rourke, On polygonal chain approximation, in: Computational Morphology, North-Holland, Amsterdam, 1988, pp. 87–95.
[25] B.K. Natarajan, On comparing and compressing piecewise linear curves, Technical Report, Hewlett Packard, 1991.
[26] B.K. Natarajan, J. Ruppert, On sparse approximations of curves and functions, in: Proc. 4th Canadian Conference on Computational Geometry, 1992, pp. 250–256.
[27] M. Sharir, P.K. Agarwal, Davenport–Schinzel Sequences and Their Geometric Applications, Cambridge University Press, 1995.
[28] G.T. Toussaint, On the complexity of approximating polygonal curves in the plane, in: Proc. IASTED International Symp. on Robotics and Automation, Lugano, Switzerland, 1985.