Distance-preserving approximations of polygonal paths*

Joachim Gudmundsson[1], Giri Narasimhan[2], and Michiel Smid[3]

[1] Department of Mathematics and Computing Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands.
h.j.gudmundsson@tue.nl
[2] School of Computer Science, Florida International University, Miami, FL 33199.
giri@fiu.edu
[3] School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6.
michiel@scs.carleton.ca

Abstract. Given a polygonal path P with vertices $p_1, p_2, \ldots, p_n$ and a real number $t \geq 1$, a path $Q = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ is a t-distance-preserving approximation of P if $1 = i_1 < i_2 < \ldots < i_k = n$ and each straight-line edge $(p_{i_j}, p_{i_{j+1}})$ of Q approximates the distance between p_{i_j} and $p_{i_{j+1}}$ along the path P within a factor of t. We present exact and approximation algorithms that compute such a path Q that minimizes k (when given t) or t (when given k). We also present some experimental results.

1 Introduction

Let P be a polygonal path through the sequence of points $p_1, p_2, \ldots, p_n$. We consider the problem of approximating P by a "simpler" polygonal path Q. Imai and Iri [9, 10] introduced two different versions of this problem. In the first one, we are given an integer k and want to compute a polygonal path Q that has k vertices and approximates P in the best possible way according to some measure that compares P and Q. In the second version, we are given a tolerance $\epsilon > 0$ and want to compute a polygonal path Q that approximates P within ϵ and has the fewest vertices. Both versions have been considered for different measures that are based on variations of the notion of minimum distance between P and Q; see [1, 2, 4, 5, 7–12].

These problems have many applications in map simplification. In this paper, we consider distance-preserving approximations of polygonal paths. Distance-preserving simplifications are particularly meaningful for a path representing a meandering river or a winding road; the approximations simplify such paths without substantially distorting the length and distance information. Clearly there is a trade-off between how simple Q is made and how closely distances in Q reflect those in P. We now define our novel concept more precisely.

We denote the Euclidean distance between any two points p and q in the plane by $|pq|$. For any two vertices p_i and p_j of P, let $\delta(p_i, p_j)$ denote the Euclidean

* Gudmundsson was supported by NWO, Smid was supported by NSERC.

distance between p_i and p_j along P, i.e., $\delta(p_i, p_j) = \sum_{\ell=i}^{j-1} |p_\ell p_{\ell+1}|$. Let $t \geq 1$ be a real number. We say that a path $Q = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$ is a *t-distance-preserving approximation* of P if (i) $1 = i_1 < i_2 < \ldots < i_k = n$, and (ii) $\delta(p_{i_j}, p_{i_{j+1}}) \leq t|p_{i_j} p_{i_{j+1}}|$ for all j with $1 \leq j < k$. Hence, the straight-line edge $(p_{i_j}, p_{i_{j+1}})$ approximates the distance between p_{i_j} and $p_{i_{j+1}}$ along the path P within a factor of t. The following two problems will be considered in this paper.

Problem 1. Given a polygonal path P with n vertices and a real number $t \geq 1$, compute a t-distance-preserving approximation of P having the minimum number of vertices.

Problem 2. Given a polygonal path P with n vertices and an integer k with $2 \leq k \leq n$, compute the minimum value of t for which a t-distance-preserving approximation of P having at most k vertices exists.

We start in Section 2 by giving simple algorithms that solve Problems 1 and 2 in $O(n^2)$ and $O(n^2 \log n)$ time, respectively. In the rest of the paper, we consider approximation algorithms for both problems. In Section 3, we introduce a heuristic algorithm for Problem 1 that uses Callahan and Kosaraju's well-separated pair decomposition [3]. We use this decomposition to define a directed graph having $O(n^2)$ edges that can be represented implicitly in $O(n)$ space. This graph has the property that a simple shortest path computation (implemented using breadth-first search) gives an "approximation" to Problem 1. To be more precise, the main result in Section 3 is the following. Given real numbers $t \geq 1$ and $0 < \epsilon < 1/3$, let κ be the minimum number of vertices on any t-distance-preserving approximation of P. In $O(n \log n + (t/\epsilon)n)$ time, we can compute a $((1+\epsilon)t)$-distance-preserving approximation Q of P, having at most κ vertices. In other words, our heuristic may result in an approximation Q of P that is distance-preserving for a slightly larger value of t than desired. In Section 4, we give an approximation algorithm for Problem 2. That is, we use the result of Section 3 and the well-separated pair decomposition to design a simple binary search algorithm that computes, in $O((t^*/\epsilon)n \log n)$ time, a real number t such that $t \leq t^* \leq (1+\epsilon)t$, where t^* is the exact solution for Problem 2. In Section 5, we present some experimental results.

2 Simple exact algorithms

Consider again the polygonal path $P = (p_1, p_2, \ldots, p_n)$, and let $t \geq 1$ be a real number. For any two indices i and j with $1 \leq i < j \leq n$, we say that the ordered pair (p_i, p_j) is *t-distance-preserving* if $\delta(p_i, p_j) \leq t|p_i p_j|$.

Consider the directed graph G_t with vertex set $\{p_1, p_2, \ldots, p_n\}$ and edge set the set of all t-distance-preserving pairs of vertices. It is clear that any t-distance-preserving approximation of P having k vertices corresponds to a path in G_t from p_1 to p_n having $k-1$ edges, and vice versa. (Observe that (p_i, p_{i+1}) is an edge in G_t for each i with $1 \leq i < n$. Therefore, there exists a path in G_t from p_1 to p_n.) It follows that Problem 1 can be solved by constructing the graph G_t and computing a shortest path from p_1 to p_n. The latter can be done by performing a breadth-first search in G_t using p_1 as the source vertex. Hence, Problem 1 can be solved in a time that is proportional to the sum of the number of vertices and edges in G_t. Since the latter is $O(n^2)$, we obtain a time bound of $O(n^2)$.

Theorem 1. *Given a polygonal path P with n vertices and a real number $t \geq 1$, a t-distance-preserving approximation of P having the minimum number of vertices can be computed in $O(n^2)$ time.*

We now show that Theorem 1 can be used to solve Problem 2. Consider again the graph G_t defined above. Let κ_t be the minimum number of vertices on any t-distance-preserving approximation of P. If t and t' are real numbers with $t' > t \geq 1$, then $\kappa_{t'} \leq \kappa_t$, because G_t is a subgraph of $G_{t'}$. Problem 2 asks for the smallest real number $t \geq 1$ such that $\kappa_t \leq k$. We denote this value of t by t^*. For any two indices i and j with $1 \leq i < j \leq n$, let $t^*_{ij} := \delta(p_i, p_j)/|p_i p_j|$. The family $(G_t)_{t \geq 1}$ consists of the $\binom{n}{2}$ graphs G_t with $t \in C := \{t^*_{ij} : 1 \leq i < j \leq n\}$. Moreover, $t^* \in C$. Therefore, if we sort the elements of C and perform a binary search in the sorted set $\{\kappa_t : t \in C\}$, we obtain a solution for Problem 2. Using Theorem 1, it follows that the running time is $O(n^2 \log n)$.

Theorem 2. *Given a polygonal path P with n vertices and an integer k with $2 \leq k \leq n$, the minimum value of t for which a t-distance-preserving approximation of P having at most k vertices exists can be computed in $O(n^2 \log n)$ time.*

3 A heuristic based on well-separated pairs

In this section, we introduce a heuristic approach for solving Problem 1 that uses the *well-separated pair decomposition* of Callahan and Kosaraju [3]. We briefly recall this decomposition in Section 3.1. In Section 3.2, we describe the idea of our heuristic algorithm, analyze its output, and give a condition under which it solves Problem 1 exactly. In Section 3.3, we show how the heuristic can be implemented such that it runs in $O(n \log n)$ time.

3.1 Well-separated pairs

Definition 1. *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well-separated with respect to s, *if there are two disjoint balls C_A and C_B, having the same radius, such that C_A contains A, C_B contains B, and the distance between C_A and C_B is at least equal to s times the radius of C_A. We will refer to s as the* separation ratio.

Lemma 1. *Let A and B be two sets of points that are well-separated with respect to s, let x and x' be points of A, and let y and y' be points of B. Then $|xx'| \leq (2/s)|x'y'|$ and $|x'y'| \leq (1 + 4/s)|xy|$.*

Definition 2 ([3]). *Let S be a set of points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A* well-separated pair decomposition *(WSPD) for S (with respect to s) is a sequence $\{A_i, B_i\}$, $1 \leq i \leq m$, of pairs of non-empty subsets of S, such that (i) $A_i \cap B_i = \emptyset$ for all $i = 1, 2, \ldots, m$, (ii) for each unordered pair $\{p, q\}$ of distinct points of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that (1) $p \in A_i$ and $q \in B_i$ or (2) $p \in B_i$ and $q \in A_i$, (iii) A_i and B_i are well-separated with respect to s, for all $i = 1, 2, \ldots, m$.*

Callahan and Kosaraju show how such a WSPD can be computed. They start by constructing, in $O(n \log n)$ time, a *split tree* T having the points of S at its leaves. Given this tree, they show how a WSPD with $m = O(s^d n)$ can be computed in $O(s^d n)$ time. In this WSPD, each pair $\{A_i, B_i\}$ is represented by two nodes u_i and v_i of T. That is, A_i and B_i are the sets of all points stored at the leaves of the subtrees rooted at u_i and v_i, respectively.

Theorem 3 ([3]). *Let S be a set of n points in $\mathbb{R}^d$, and let $s > 0$ be a real number. A WSPD for S (with respect to s) with $m = O(s^d n)$ can be computed in $O(n \log n + s^d n)$ time.*

3.2 The idea of the heuristic

Consider again the polygonal path $P = (p_1, p_2, \ldots, p_n)$. We embed P into one-dimensional space by "flattening" it out: For each i with $1 \leq i \leq n$, let x_i be the real number given by $x_i = \delta(p_1, p_i)$, and let $S := \{x_1, x_2, \ldots, x_n\}$.

Let $s > 0$ be a given separation ratio, and consider the split tree T and the corresponding WSPD $\{A_i, B_i\}$, $1 \leq i \leq m$, for S. We may assume without loss of generality that any element in A_i is smaller than any element in B_i.

The following lemma shows that, for large values of s, all pairs (p, q) of vertices of P such that $\delta(p_1, p) \in A_i$ and $\delta(p_1, q) \in B_i$ are distance-preserving for approximately the same value of t.

Lemma 2. *Let p, p', q, and q' be vertices of P, and let i be an index such that $x := \delta(p_1, p) \in A_i$, $x' := \delta(p_1, p') \in A_i$, $y := \delta(p_1, q) \in B_i$, and $y' := \delta(p_1, q') \in B_i$. Let $t \geq 1$ be a real number such that $4st + 16t < s^2$. If the pair (p, q) is t-distance-preserving, then the pair (p', q') is t'-distance-preserving, where*

$$t' = \frac{(1 + 4/s)t}{1 - 4(1 + 4/s)t/s}.$$

Proof. First observe that, by the condition on t, the denominator in t' is positive. By applying Lemma 1 and the triangle-inequality, we obtain

$$\begin{aligned}
\delta(p', q') &= |x'y'| \leq (1 + 4/s)|xy| = (1 + 4/s) \cdot \delta(p, q) \\
&\leq (1 + 4/s)t|pq| \leq (1 + 4/s)t \cdot (|pp'| + |p'q'| + |q'q|) \\
&\leq (1 + 4/s)t \cdot (\delta(p, p') + |p'q'| + \delta(q', q)) \\
&= (1 + 4/s)t \cdot (|xx'| + |p'q'| + |y'y|)
\end{aligned}$$

$$
\begin{aligned}
&\leq (1+4/s)t \cdot ((2/s)|x'y'| + |p'q'| + (2/s)|x'y'|) \\
&= (1+4/s)t \cdot ((4/s)\delta(p',q') + |p'q'|) \\
&= (4(1+4/s)t/s) \cdot \delta(p',q') + (1+4/s)t|p'q'|.
\end{aligned}
$$

Rearranging terms yields $\delta(p',q') \leq t'|p'q'|$. $\square$

Let $t \geq 1$ and $0 < \epsilon < 1/3$ be real numbers, and let

$$
s = \frac{12 + 24(1+\epsilon/3)t}{\epsilon}. \tag{1}
$$

Lemma 3. *Let p, p', q, and q' be as in Lemma 2. If the pair (p,q) is t-distance-preserving, then the pair (p',q') is $((1+\epsilon/3)t)$-distance-preserving. If the pair (p,q) is $((1+\epsilon/3)t)$-distance-preserving, then the pair (p',q') is $((1+\epsilon)t)$-distance-preserving.*

For each i with $1 \leq i \leq m$, let a_i and b_i be fixed elements of A_i and B_i, respectively, and let f_i and g_i be the vertices of P such that $a_i = \delta(p_1, f_i)$ and $b_i = \delta(p_1, g_i)$. We say that the ordered pair (A_i, B_i) is (t,ϵ)-*distance-preserving* if the pair (f_i, g_i) is $((1+\epsilon/3)t)$-distance-preserving. We define a directed graph H whose vertices are the $2m$ sets A_i and B_i, $1 \leq i \leq m$. The edges of H are defined as follows. For any i with $1 \leq i \leq m$, (A_i, B_i) is an edge in H if (A_i, B_i) is (t,ϵ)-distance-preserving and $x_n \in B_i$. For any i and j with $1 \leq i \leq m$ and $1 \leq j \leq m$, (A_i, A_j) is an edge in H if (A_i, B_i) is (t,ϵ)-distance-preserving and $A_j \cap B_i \neq \emptyset$.

Let $Q = (q_1, q_2, \ldots, q_k)$ be an arbitrary t-distance-preserving approximation of the polygonal path P. We will show that Q corresponds to a path in H from some set A_i that contains x_1 to some set B_j that contains x_n. Moreover, this path in H consists of k vertices.

For each i with $1 \leq i \leq k$, let y_i be the element of S such that $y_i = \delta(p_1, q_i)$. Recall that $q_1 = p_1$ and, therefore, $y_1 = x_1$. Let i_1 be the index such that $y_1 \in A_{i_1}$ and $y_2 \in B_{i_1}$. The path in H corresponding to Q has A_{i_1} as its first vertex. Let ℓ be such that $1 \leq \ell \leq k-2$ and assume we have already converted $(q_1, q_2, \ldots, q_\ell)$ into a path $(A_{i_1}, A_{i_2}, \ldots, A_{i_\ell})$ in H, where $y_\ell \in A_{i_\ell}$ and $y_{\ell+1} \in B_{i_\ell}$. Let $i_{\ell+1}$ be the index such that $y_{\ell+1} \in A_{i_{\ell+1}}$ and $y_{\ell+2} \in B_{i_{\ell+1}}$. Since $(q_\ell, q_{\ell+1})$ is t-distance-preserving, and since $y_\ell \in A_{i_\ell}$ and $y_{\ell+1} \in B_{i_\ell}$, it follows from Lemma 3 that the pair (A_{i_ℓ}, B_{i_ℓ}) is (t,ϵ)-distance-preserving. Moreover, $A_{i_{\ell+1}} \cap B_{i_\ell} \neq \emptyset$, because $y_{\ell+1}$ is in the intersection. Therefore, $(A_{i_\ell}, A_{i_{\ell+1}})$ is an edge in H, i.e., we have converted $(q_1, q_2, \ldots, q_{\ell+1})$ into a path $(A_{i_1}, A_{i_2}, \ldots, A_{i_{\ell+1}})$ in H, where $y_{\ell+1} \in A_{i_{\ell+1}}$ and $y_{\ell+2} \in B_{i_{\ell+1}}$. We continue extending this path until we have converted $(q_1, q_2, \ldots, q_{k-1})$ into a path $(A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}})$ in H, where $y_{k-1} \in A_{i_{k-1}}$ and $y_k \in B_{i_{k-1}}$. Observe that $x_n = \delta(p_1, q_k) = y_k \in B_{i_{k-1}}$. Also, since (q_{k-1}, q_k) is t-distance-preserving, it follows from Lemma 3 that the pair $(A_{i_{k-1}}, B_{i_{k-1}})$ is (t,ϵ)-distance-preserving. Therefore, $(A_{i_{k-1}}, B_{i_{k-1}})$ is an edge in H.

Hence, any t-distance-preserving approximation $Q = (q_1, q_2, \ldots, q_k)$ of P corresponds to a path $(A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}}, B_{i_{k-1}})$ in H, where A_{i_1} contains x_1 and $B_{i_{k-1}}$ contains x_n.

What about the converse? Let $(A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}}, B_{i_{k-1}})$ be a path in H such that $x_1 \in A_{i_1}$ and $x_n \in B_{i_{k-1}}$. We will convert this path into a polygonal path Q from p_1 to p_n. Let $q_1 := p_1$, $y_1 := x_1$, and let Q be the path consisting of the single vertex q_1.

Let ℓ be an integer such that $1 \leq \ell \leq k-2$, and assume we have already converted $(A_{i_1}, A_{i_2}, \ldots, A_{i_\ell})$ into a polygonal path $Q = (q_1, q_2, \ldots, q_\ell)$ such that $y_1 \in A_{i_1}$ and $y_j := \delta(p_1, q_j) \in A_{i_j} \cap B_{i_{j-1}}$ for each j with $2 \leq j \leq \ell$. Consider the edge $(A_{i_\ell}, A_{i_{\ell+1}})$ in H. We know that $A_{i_{\ell+1}} \cap B_{i_\ell} \neq \emptyset$. Let $y_{\ell+1}$ be an arbitrary element of $A_{i_{\ell+1}} \cap B_{i_\ell}$, and let $q_{\ell+1}$ be the vertex of P for which $y_{\ell+1} = \delta(p_1, q_{\ell+1})$. We extend Q by the vertex $q_{\ell+1}$.

We continue adding vertices to Q until we have converted $(A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}})$ into a polygonal path $Q = (q_1, q_2, \ldots, q_{k-1})$ such that $y_1 \in A_{i_1}$ and $y_j := \delta(p_1, q_j) \in A_{i_j} \cap B_{i_{j-1}}$ for each j with $2 \leq j \leq k-1$. Consider the last edge $(A_{i_{k-1}}, B_{i_{k-1}})$ of the path in H. We know that $x_n \in B_{i_{k-1}}$. Let $q_k := p_n$ and $y_k := x_n$. Then $y_k = \delta(p_1, q_k) \in B_{i_{k-1}}$. We add the vertex q_k to Q, which completes the polygonal path between p_1 and p_n.

In conclusion, we have converted the path $(A_{i_1}, A_{i_2}, \ldots, A_{i_{k-1}}, B_{i_{k-1}})$ in H, where $x_1 \in A_{i_1}$ and $x_n \in B_{i_{k-1}}$, into a polygonal path $Q = (q_1, q_2, \ldots, q_k)$ such that (i) $y_1 = x_1 = \delta(p_1, q_1) \in A_{i_1}$, (ii) $y_j = \delta(p_1, q_j) \in A_{i_j} \cap B_{i_{j-1}}$ for all j with $2 \leq j \leq k-1$, and (iii) $y_k = x_n = \delta(p_1, q_k) \in B_{i_{k-1}}$.

Unfortunately, Q need not be a t-distance-preserving approximation of P. We claim, however, that Q is a $((1+\epsilon)t)$-distance-preserving approximation of P. To prove this, we let j be an arbitrary index with $1 \leq j \leq k-1$. Observe that $y_j \in A_{i_j}$ and $y_{j+1} \in B_{i_j}$. Since the pair (A_{i_j}, B_{i_j}) is (t, ϵ)-distance-preserving, it follows from Lemma 3 that the pair (q_j, q_{j+1}) is $((1+\epsilon)t)$-distance-preserving.

Theorem 4. *Let $P = (p_1, p_2, \ldots, p_n)$ be a polygonal path, let $t \geq 1$ and $0 < \epsilon < 1/3$ be real numbers, let $x_1 = \delta(p_1, p_1)$ and $x_n = \delta(p_1, p_n)$, and let H be the graph as defined above, where the separation ratio is given by (1).*

1. *Any t-distance-preserving approximation of P consisting of k vertices corresponds to a path in H from some set containing x_1 to some set containing x_n that consists of k vertices.*
2. *Any path in H from some set containing x_1 to some set containing x_n that consists of k vertices corresponds to a $((1+\epsilon)t)$-distance-preserving approximation of P consisting of k vertices.*
3. *Let κ be the minimum number of vertices on any t-distance-preserving approximation of P, and let R be a shortest path in H between any set containing x_1 and any set containing x_n. Then R corresponds to a $((1+\epsilon)t)$-distance-preserving approximation of P consisting of at most κ vertices.*
4. *Assume that for any two distinct vertices p and q of P, $\delta(p,q)/|pq| \leq t$ or $\delta(p,q)/|pq| > (1+\epsilon)t$. Then R corresponds to a t-distance-preserving approximation of P consisting of κ vertices.*

3.3 Implementing the heuristic

The results in Section 3.2 imply that we can solve Problem 1 heuristically by computing a shortest path in the graph H between any set A_i that contains

x_1 and any set B_j that contains x_n. Such a shortest path can be found by a breadth-first search computation. The problem is that the graph H can have up to $\Theta(n^2)$ edges. We will show, however, that we can run (a partial) breadth-first search without explicitly constructing the entire graph H. The main idea is to use the fact that the vertices of H correspond to nodes of the split tree T.

Let i and j be two indices such that (A_i, A_j) is an edge in the graph H. Then, by definition, $A_j \cap B_i \neq \emptyset$. Since A_j and B_i are represented by nodes of the split tree, it follows that $A_j \subseteq B_i$ (in which case the node representing A_j is in the subtree of the node representing B_i) or $B_i \subseteq A_j$ (in which case the node representing B_i is in the subtree of the node representing A_j).

We are now ready to present the algorithm. The input is the polygonal path $P = (p_1, p_2, \ldots, p_n)$ and real numbers $t \geq 1$ and $0 < \epsilon < 1/3$. The output will be a $((1+\epsilon)t)$-distance-preserving approximation of P.

Preprocessing:

Step 1: Compute the set $S = \{x_1, x_2, \ldots, x_n\}$, where $x_i = \delta(p_1, p_i)$.

Step 2: Compute the split tree T and the corresponding WSPD $\{A_i, B_i\}$, $1 \leq i \leq m$, for S with separation ratio $s = (12 + 24(1+\epsilon/3)t)/\epsilon$.

Step 3: For each i with $1 \leq i \leq m$, let a_i and b_i be arbitrary elements in A_i and B_i, respectively. Let f_i and g_i be the vertices of P such that $a_i = \delta(p_1, f_i)$ and $b_i = \delta(p_1, g_i)$. If (f_i, g_i) is $((1+\epsilon/3)t)$-distance-preserving, then keep the pair $\{A_i, B_i\}$; otherwise discard it.

For simplicity, we again denote the remaining well-separated pairs by $\{A_i, B_i\}$, $1 \leq i \leq m$. It follows from Theorem 3 that $m = O(sn) = O((t/\epsilon)n)$ and that the preprocessing stage takes time $O(n \log n + sn) = O(n \log n + (t/\epsilon)n)$. For each i with $1 \leq i \leq m$, we denote by u_i and v_i the nodes of the split tree T that represent the sets A_i and B_i, respectively. We designate the nodes u_i as *A-nodes*. We will identify each set A_i with the corresponding node u_i and each set B_i with the corresponding node v_i. Hence, the graph H in which we perform the breadth-first search will have the nodes of the split tree as its vertices. Observe that for a node w of T, there may be several indices i and several indices j such that $u_i = w$ and $v_j = w$.

The implicit breadth-first search algorithm: Our algorithm will be a modified version of the breadth-first search algorithm as described in Cormen *et al.* [6]. It computes a breadth-first forest consisting of breadth-first trees rooted at the A-nodes u_i for which $x_1 \in A_i$. The breadth-first search terminates as soon as an A-node u_i is reached such that $x_n \in B_i$. For each node w of the split tree, the algorithm maintains three variables: (i) $color(w)$, whose value is either *white*, *gray*, or *black*, (ii) $dist(w)$, whose value is the distance in H from any set A_i containing x_1 to the set corresponding to w, as computed by the algorithm, and (iii) $\pi(w)$, whose value is the predecessor of w in the breadth-first forest.

Step 1: For each node w of T, set $color(w) :=$ *white*, $dist(w) := \infty$, and $\pi(w) :=$ *nil*.

Step 2: Initialize an empty queue Q. Starting at the leaf of T storing x_1, walk up the tree to the root. For each node w encountered, set $color(w) :=$ *gray* and, if w is an A-node, set $dist(w) := 0$, and add w to the end of Q.

Step 3: Let w be the first element of Q. Delete w from Q and set $color(w) := black$. For each index i such that $u_i = w$, do the following. If $x_n \in B_i$, then set $dist(v_i) := dist(w) + 1$, $\pi(v_i) := w$, $z := v_i$, and go to Step 4. If $x_n \notin B_i$ and $color(v_i) = white$, then perform Steps 3.1 and 3.2.
Step 3.1: Starting at node v_i, walk up the split tree until the first non-white node is reached. For each white node w' encountered, set $color(w') := gray$ and, if w' is an A-node, set $dist(w') := dist(w) + 1$, $\pi(w') := w$, and add w' to the end of Q.
Step 3.2: Visit all nodes in the subtree of v_i. For each node w' in this subtree, set $color(w') := gray$ and, if w' is an A-node, set $dist(w') := dist(w)+1$, $\pi(w') := w$, and add w' to the end of Q.

After all indices i with $u_i = w$ have been processed, go to Step 3.
Step 4: Compute the path $(z, \pi(z), \pi^2(z), \ldots, \pi^{k-1}(z))$ of nodes in T, where $k = dist(z) + 1$.
Step 5: Use the algorithm of Section 3.2 to convert the path obtained in Step 4 into a polygonal path.

Observe that, if w' is the first non-white node reached in Step 3.1, all nodes on the path from w' to the root of the split tree are non-white. Also, if $color(v_i) = white$, then all nodes in the subtree of v_i (these are visited in Step 3.2) are white. Using these observations, an analysis similar to the one in Cormen *et al.* [6] shows that the path obtained in Step 4 is a shortest path in H between any set A_i containing x_1 and any set B_j containing x_n. Hence, by Theorem 4, the polygonal path obtained in Step 5 is a $((1+\epsilon)t)$-distance-preserving approximation of the input path P. To estimate the running time of the algorithm, first observe that both Steps 1 and 2 take $O(n)$ time. Steps 4 and 5 both take $O(n)$ time, because the path reported consists of at most n nodes. It remains to analyze Step 3. The total time for Step 3 is proportional to the sum of m and the total time for walking through the split tree T in Steps 3.1 and 3.2. It follows from the algorithm that each edge of T is traversed at most once. Therefore, Step 3 takes $O(m + n)$ time. We have shown that the total running time of the algorithm is $O(m + n) = O(sn) = O((t/\epsilon)n)$.

Theorem 5. *Let $P = (p_1, p_2, \ldots, p_n)$ be a polygonal path, let $t \geq 1$ and $0 < \epsilon < 1/3$ be real numbers, and let κ be the minimum number of vertices on any t-distance-preserving approximation of P.*

1. *In $O(n \log n + (t/\epsilon)n)$ time, we can compute a $((1+\epsilon)t)$-distance-preserving approximation Q of P, having at most κ vertices.*
2. *If $\delta(p,q)/|pq| \leq t$ or $\delta(p,q)/|pq| > (1+\epsilon)t$ for all distinct vertices p and q of P, then Q is a t-distance-preserving approximation of P having κ vertices.*

4 An approximation algorithm for Problem 2

Recall that, for any real number $t \geq 1$, we denote by κ_t the minimum number of vertices on any t-distance-preserving approximation of the polygonal path $P = (p_1, p_2, \ldots, p_n)$. Let k be a fixed integer with $2 \leq k \leq n$, and let $t^* :=$

$\min\{t \geq 1 : \kappa_t \leq k\}$. In this section, we present an algorithm that computes an approximation to t^*. Our algorithm will perform a binary search, which is possible because of the following lemma.

Lemma 4. *Let $t \geq 1$ and $0 < \epsilon < 1/3$ be real numbers, let Q be the output of the algorithm of Theorem 5, and let k' be the number of vertices of Q. If $k' \leq k$, then $t^* \leq (1+\epsilon)t$. If $k' > k$, then $t^* > t$.*

We first run a standard doubling algorithm to compute a real number τ that approximates t^* within a factor of two. To be more precise, starting with $t = 2$, we do the following. Run the algorithm of Theorem 5 and let k' be the number of vertices in the output Q of this algorithm. If $k' > k$, then repeat with t replaced by $2t$. If $k' \leq k$, then terminate, set $\tau := t$, and return the value of τ. It follows from Lemma 4 that

$$\tau/2 < t^* \leq (1+\epsilon)\tau. \tag{2}$$

Observe that the algorithm of Theorem 5 computes, among other things, a split tree T and a WSPD with a separation ratio s that depends on t and ϵ. Moreover, observe that T does not depend on s. Hence, it suffices to compute the split tree only once. Therefore, it follows from Theorem 5 that, when given T, the time to compute τ is $O(\sum_{i=1}^{\log \tau}(2^i/\epsilon)n) = O((\tau/\epsilon)n) = O((t^*/\epsilon)n)$.

We now show how to use binary search to compute a better approximation. Let $S = \{x_1, x_2, \ldots, x_n\}$, where $x_i = \delta(p_1, p_i)$, $1 \leq i \leq n$, and let $\{A_i, B_i\}$, $1 \leq i \leq m$, be the WSPD of S with separation ratio $s = (4 + 8(1+\epsilon)^3\tau)/\epsilon$. For each i with $1 \leq i \leq m$, let a_i and b_i be arbitrary elements of A_i and B_i, respectively, let f_i and g_i be the vertices of P such that $a_i = \delta(p_1, f_i)$ and $b_i = \delta(p_1, g_i)$, and let $t_i := \delta(f_i, g_i)/|f_i g_i|$. The following lemma states that, in order to approximate t^*, it suffices to search among the values t_i, $1 \leq i \leq m$.

Lemma 5. *There exists an index j with $1 \leq j \leq m$, such that $t_j/(1+\epsilon) \leq t^* \leq (1+\epsilon)t_j$.*

Proof. We only prove the first inequality. We have seen in Section 2 that there exist two distinct vertices p and q of P such that $t^* = \delta(p,q)/|pq|$. Let j be the index such that $\delta(p_1, p) \in A_j$ and $\delta(p_1, q) \in B_j$.

Since the pair (p, q) is t^*-distance-preserving, and since $4st^* + 16t^* < s^2$, we know from Lemma 2 that the pair (f_j, g_j) is t'-distance-preserving, where

$$t' = \frac{(1 + 4/s)t^*}{1 - 4(1 + 4/s)t^*/s}.$$

Since $s \geq 4$, we have $t' \leq (1+4/s)t^*/(1-8t^*/s)$. By our choice of s and by (2), we have $s \geq (4 + 8(1+\epsilon)t^*)/\epsilon$, which is equivalent to $(1+4/s)t^*/(1-8t^*/s) \leq (1+\epsilon)t^*$. This proves that $t_j = \delta(f_j, g_j)/|f_j g_j| \leq t' \leq (1+\epsilon)t^*$. $\square$

We proceed as follows. Define $t_0 := 1$, sort the values t_i, $0 \leq i \leq m$, remove duplicates, and discard those values that are larger than $(1+\epsilon)^2\tau$. For simplicity, we denote the remaining sorted sequence by

$$1 = t_0 < t_1 < t_2 < \ldots < t_m \leq (1+\epsilon)^2\tau. \tag{3}$$

We perform a binary search in this sorted sequence, maintaining the following invariant.

Invariant: ℓ and r are integers such that $0 \leq \ell \leq r \leq m$ and $t_\ell \leq t^* \leq (1+\epsilon)t_r$.

Initially, we set $\ell := 0$ and $r := m$. To prove that at this moment, the invariant holds, consider the index j in Lemma 5. Observe that since $t_j \leq (1+\epsilon)t^* \leq (1+\epsilon)^2\tau$, the value t_j occurs in the sorted sequence (3). Therefore, $t_\ell = 1 \leq t^* \leq (1+\epsilon)t_j \leq (1+\epsilon)t_r$.

Assume that $\ell < r - 1$. Then we use Lemma 4, with $t = t_h$ where $h = \lfloor(\ell+r)/2\rfloor$, to decide if $t^* \leq (1+\epsilon)t_h$ or $t^* > t_h$. In the first case, we set $r := h$, whereas in the second case, we set $\ell := h$. Observe that, in both cases, the invariant is correctly maintained.

We continue making these binary search steps until $\ell = r-1$. At this moment, we have $t_\ell \leq t^* \leq (1+\epsilon)t_{\ell+1}$. We now use Lemma 4, with $t = (1+\epsilon)t_\ell$, to decide if $t^* \leq (1+\epsilon)^2 t_\ell$ or $t^* > (1+\epsilon)t_\ell$. In the first case, we return the value t_ℓ, which satisfies $t_\ell \leq t^* \leq (1+\epsilon)^2 t_\ell$. Assume that $t^* > (1+\epsilon)t_\ell$. We claim that $t^* \geq t_{\ell+1}/(1+\epsilon)$. This will imply that $t_{\ell+1}/(1+\epsilon) \leq t^* \leq (1+\epsilon)t_{\ell+1}$ and, therefore, we return the value $t_{\ell+1}/(1+\epsilon)$. To prove the claim, consider again the index j in Lemma 5. We have $t_j \geq t^*/(1+\epsilon) > t_\ell$ and thus $t_j \geq t_{\ell+1}$. It follows that $t^* \geq t_j/(1+\epsilon) \geq t_{\ell+1}/(1+\epsilon)$.

We have shown that the algorithm returns a value t such that $t \leq t^* \leq (1+\epsilon)^2 t$. If we run the entire algorithm with ϵ replaced by $\epsilon/3$, then we obtain a value t such that $t \leq t^* \leq (1+\epsilon/3)^2 t \leq (1+\epsilon)t$.

Theorem 6. *Let $P = (p_1, p_2, \ldots, p_n)$ be a polygonal path, let k be an integer with $2 \leq k \leq n$, let t^* be the minimum value of t for which a t-distance-preserving approximation of P having at most k vertices exists, and let $0 < \epsilon < 1$. In $O((t^*/\epsilon) n \log n)$ time, we can compute a real number t such that $t \leq t^* \leq (1+\epsilon)t$.*

5 Experimental results

In this section, we will briefly discuss some experimental results that we obtained by implementing the algorithms presented in Sections 2 and 3. The experiments were done by running the programs on paths containing between 100 and 50,000 points. The shorter paths are from the Spanish railroad network and the longer paths (more than 3,000 points) were constructed by joining several shorter paths.

The exact algorithm: First we consider the results obtained by running the exact algorithm on the input paths with different values of t. The most striking result is that the running times and numbers of edges seem to be independent of t. The running time of the algorithm was between 94 and 98 seconds for an input path containing 20,000 points for different values of t. Even though one might expect that the algorithm would not be heavily dependent on t, it is surprising that the difference is so small. The explanation is probably that the time to perform a breadth-first search depends on the length of the optimal solution (the depth of the search tree) and the number of t-distance preserving edges (affecting the width of the tree). If t is large, the number of t-distance preserving edges is large and hence the width of the search tree is large, whereas if t is small, the optimal solution is long and hence the search tree is deep (but not very wide).

The heuristic: Just as for the exact algorithm, the running time of the heuristic is not sensitive to t, for reasons similar to the ones discussed above. On the other hand, the running time decreases when ϵ is increasing, since the number of well-separated pairs decreases. In the tests we performed, the number of pairs and the running time increased between two and four times when ϵ was decreased from 0.05 to 0.01 (for instances containing more than 2,000 points).

The well-separated pair decomposition allows us to disregard the majority of the edges in the breadth-first search, which is the reason why the heuristic is faster than the exact algorithm. However, this "pruning" step is quite costly. Comparing the construction of the well-separated pair decomposition with the breadth-first search shows that the former uses almost 98% of the total running time. Is there a way to perform the pruning more efficiently?

Comparing the algorithms: The table below shows some of the running times (in seconds) of three experiments using $t = 1.1$: The exact algorithm and the heuristic with $\epsilon = 0.01$ and $\epsilon = 0.1$. It is clear from the table that the exact algorithm quickly becomes impractical when the size of the input grows. Processing 50,000 points takes approximately 600 seconds for the exact algorithm while the same number of points can be processed in 43 and 11 seconds, respectively, by the heuristic. Plotting the running-times clearly shows a difference in their asymptotic behavior which is obviously due to the use of the well-separated pair decomposition which, as mentioned above, "prunes" the search tree. For example, when the input path consisted of 50,000 points, the exact algorithm "looked" at almost 1.2 billion edges, while the well-separated pair decomposition removed all but 36 million edges (for $\epsilon = 0.01$) and 10 million edges (for $\epsilon = 0.1$). For 20,000 points the numbers where 196 millions versus 16 millions and 4 millions. This corroborates the power of using the well-separated pair decomposition for this kind of problems.

#Points	100	500	2,000	8,000	20,000	50,000
Exact	< 1s	< 1s	1s	16s	98s	617s
Heuristic ($\epsilon = 0.01$)	< 1s	< 1s	3s	13s	20s	43s
Heuristic ($\epsilon = 0.1$)	< 1s	< 1s	1s	3s	5s	11s

References

1. H. Alt, J. Blömer, M. Godau, and H. Wagener. Approximation of convex polygons. In *Proceedings 17th International Colloquium on Automata, Languages and Programming.*, volume 443 of *Lecture Notes in Computer Science*, pages 703–716. Springer-Verlag, 1990.
2. G. Barequet, M. T. Goodrich, D. Z. Chen, O. Daescu, and J. Snoeyink. Efficiently approximating polygonal paths in three and higher dimensions. In *Proceedings 14th Annual ACM Symposium on Computational Geometry*, pages 317–326, 1998.

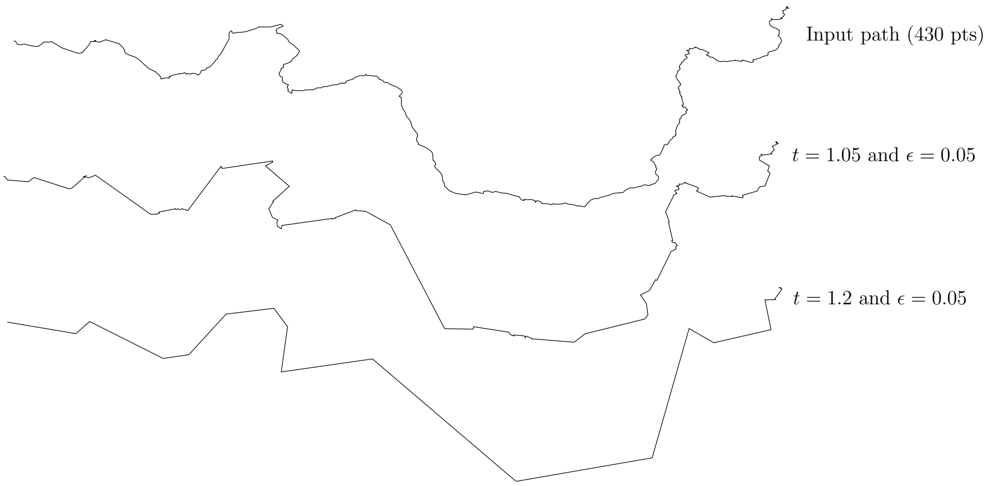

Fig. 1. The topmost figure is the original path with 430 points, in the middle and at the bottom we have two simplifications containing 126 and 22 points obtained from the heuristic using $\epsilon = 0.05$, and $t = 1.05$ and $t = 1.2$, respectively.

3. P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42:67–90, 1995.
4. W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 6:59–77, 1996.
5. D. Z. Chen and O. Daescu. Space-efficient algorithms for approximating polygonal curves in two-dimensional space. *International Journal of Computational Geometry & Applications*, 13:95–111, 2003.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.
7. M. T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete & Computational Geometry*, 14:445–462, 1995.
8. L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *International Journal of Computational Geometry & Applications*, 3:383–415, 1993.
9. H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics and Image Processing*, 36:31–41, 1986.
10. H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. North-Holland, Amsterdam, Netherlands, 1988.
11. A. Melkman and J. O'Rourke. On polygonal chain approximation. In G. T. Toussaint, editor, *Computational Morphology*, pages 87–95. North-Holland, Amsterdam, Netherlands, 1988.
12. Kasturi R. Varadarajan. Approximating monotone polygonal curves using the uniform metric. In *Proceedings 12th Annual ACM Symposium on Computational Geometry*, pages 311–318, 1996.