On the complexity of approximating Euclidean traveling salesman tours and minimum spanning trees*

Gautam Das† Sanjiv Kapoor‡ Michiel Smid§

February 20, 1996

Abstract

We consider the problems of computing r-approximate traveling salesman tours and r-approximate minimum spanning trees for a set of n points in $\mathbb{R}^d$, where $d \geq 1$ is a constant. In the algebraic computation tree model, the complexities of both these problems are shown to be $\Theta(n \log n/r)$, for all n and r such that $r < n$ and r is larger than some constant. In the more powerful model of computation that additionally uses the floor function and random access, both problems can be solved in $O(n)$ time if $r = \Theta(n^{1-1/d})$.

1 Introduction

The *Traveling Salesman Problem* (*TSP*) is one of the best known combinatorial optimization problems. In the geometric version of this problem, we are given a set S of n points in $\mathbb{R}^d$, where $d \geq 1$ is a constant. A *tour* is a closed path that visits each point of S exactly once and returns to its starting point. Each edge of such a tour has a *length* that is equal to the Euclidean distance between its endpoints. The *length* of a tour is the sum of the lengths of all its edges. The *TSP* is to compute a tour along the points of S of minimal length. Since this problem is NP-complete for dimension $d \geq 2$ (see [6]), it is natural to consider the weaker problem of designing efficient algorithms that approximate the optimal tour. We call a tour having length at most r times the length of an optimal tour an *r-approximate TSP-tour*.

It is well known that for $d = 2$, a 2-approximate *TSP*-tour can be computed in $O(n \log n)$ time. (See e.g. [7].) In fact, for any dimension $d \geq 3$ and any $\epsilon > 0$, a

*Part of this work was done while the authors were at the Max-Planck-Institut für Informatik, Saarbrücken.

†Math Sciences Dept., The University of Memphis, Memphis, TN 38152, USA. Supported in part by NSF Grant CCR-9306822. E-mail: dasg@next1.msci.memphis.edu.

‡Department of Computer Science, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India. E-mail: skapoor@cse.iitd.ernet.in.

§Department of Computer Science, King's College London, Strand, London WC2R 2LS, United Kingdom. E-mail: michiel@dcs.kcl.ac.uk.

$(2+\epsilon)$-approximate *TSP*-tour can be computed in $O(n \log n + n\,(1/\epsilon)^d \log 1/\epsilon)$ time. (This follows from results in [4, 8, 9] and Lemma 2 below.)

On the other hand, an n-approximate *TSP*-tour can be computed in $O(n)$ time. This follows from the fact that *any* tour is an n-approximate *TSP*-tour. (See Lemma 1 below.)

This leads to the question of determining, for any dimension $d \geq 1$, the complexity of computing an r-approximate *TSP*-tour for sufficiently large values of n and r. In this paper, we answer this question for algorithms that belong to the algebraic computation tree model. In particular, we prove the following result.

Theorem 1 *Let $d \geq 1$ be an integer constant. In the algebraic computation tree model, any algorithm that, given a set S of n points in $\mathbb{R}^d$ and a sufficiently large real number $r < n$, computes an r-approximate TSP-tour for S, takes $\Omega(n \log n/r)$ time in the worst case.*

Note that this lower bound even holds in dimension $d = 1$. As mentioned above, the lower bound is tight for constant values $r \geq 2$ and $d = 2$.

We prove that the lower bound is in fact tight for all values of r. That is, we give an algorithm that, given a set S of n points in $\mathbb{R}^d$ and a real number r, $8 < r < n$, computes an r-approximate *TSP*-tour for S in $O(n \log n/r)$ time. This algorithm fits in the algebraic computation tree model. (The constant 8 is somewhat arbitrary here. We concentrate on "large" values of r, because it is known already how to compute an r-approximate *TSP*-tour in $O(n \log n)$ time for values of r that are larger than two.)

We also consider the related problem of approximating the minimum spanning tree of a set of points. Again, let S be a set of n points in $\mathbb{R}^d$. Consider a graph G having the points of S as its vertices. The *weight* of G—denoted by $wt(G)$—is defined as the sum of the lengths of all edges of G. A *minimum spanning tree* (*MST*) of S is a tree of minimum weight having the points of S as its vertices. We denote an *MST* of the point set S by $MST(S)$. Its weight is equal to $wt(MST(S))$.

For $d = 2$, an *MST* can be computed in $O(n \log n)$ time, which is known to be optimal. (See [7].) For dimension $d \geq 3$, the problem becomes more difficult. For example, if $d = 3$, the fastest algorithm known today constructs an *MST* in expected time $O(n^{4/3} \log^{O(1)} n)$. (See [1, 4].)

We call a connected graph on the points of S having weight at most r times $wt(MST(S))$ an *r-approximate MST*. Note that we only require the graph to be connected; it need not be a tree. It is known that for any $\epsilon > 0$, a $(1+\epsilon)$-approximate *MST* can be computed in time $O(n \log n + n\,(1/\epsilon)^d \log 1/\epsilon)$. (See [4, 8, 9].)

We consider the problem of constructing an r-approximate *MST* for large values of r. Using the relation between an r-approximate *MST* and a $2r$-approximate *TSP*-tour (see Lemma 2 below) we have the following result.

Theorem 2 *Let $d \geq 1$ be an integer constant. In the algebraic computation tree model, any algorithm that, given a set S of n points in $\mathbb{R}^d$ and a sufficiently large real number $r < n$, computes an r-approximate MST for S, takes $\Omega(n \log n/r)$ time in the worst case.*

Again, this lower bound is tight. That is, for any set S of n points in $\mathbb{R}^d$ and any real number r, $4 < r < n$, we can in $O(n \log n/r)$ time compute a connected graph on S—in fact, a tree—having weight at most $r \cdot wt(MST(S))$. (Also here, the constant 4 is somewhat arbitrary. We concentrate on "large" values of r.)

Hence, in the algebraic computation tree model, computing an r-approximate *TSP*-tour, or an r-approximate *MST* takes $\Theta(n \log n/r)$ time. In particular, for r a (sufficiently large) constant, the complexity is $\Theta(n \log n)$. In fact, if r is a large number like $n^{1-1/d}$, the complexity is still $\Theta(n \log n)$. To give an algorithm with running time $o(n \log n)$, we need a very large approximation factor such as $r = n/\log n$.

All results mentioned so far hold for the algebraic computation tree model. In particular, they hold for algorithms that do not use the non-algebraic floor function or random access. In the final part of the paper, we consider algorithms that do have these two operations at their disposal.

Bern et al.[3] show that for any $\epsilon > 0$ and any set of n points in the plane, a $(1+\epsilon)$-approximate *MST* can be computed in $O((1/\epsilon)\, n \log\log n)$ time in this more powerful model.

We give an algorithm that, given a set S of n points in $\mathbb{R}^d$, computes a $3\sqrt{d}\, n^{1-1/d}$-approximate *MST* for S in $O(n)$ time. This yields an algorithm that computes a $6\sqrt{d}\, n^{1-1/d}$-approximate *TSP*-tour for S, also in $O(n)$ time.

The rest of this paper is organized as follows. In the next section, we recall some results that will be used in the rest of the paper. In Section 3, we prove the lower bounds. Then, in Section 4, we give the algorithm that shows that the lower bounds are tight in the algebraic computation tree model. In Section 5, we give the algorithm that operates in the more powerful model of computation. Finally, in Section 6, we give some concluding remarks.

2 Some preliminary results

We assume that the reader is familiar with the algebraic computation tree model. (See Ben-Or [2], and Preparata and Shamos [7].) Our lower bound will use the following important result.

Theorem 3 (Ben-Or [2]) *Let W be any set in $\mathbb{R}^n$ and let $\mathcal{A}$ be any algorithm that belongs to the algebraic computation tree model and that accepts W. Let $\#W$ denote the number of connected components of W. Then the worst-case running time of $\mathcal{A}$ is $\Omega(\log \#W - n)$.*

The following two lemmas are well known. We include their proofs for completeness.

Lemma 1 *Let S be a set of n points in $\mathbb{R}^d$. Any tour of S is an n-approximate TSP-tour.*

Proof: Let T be any tour of S, and let T_{opt} be an optimal *TSP*-tour. Let (p, q) be an edge of T. Consider one of the two parts of T_{opt} that connects p and q. By the triangle inequality, the distance between p and q is at most equal to the total length

of this part, which in turn is at most equal to the length of T_{opt}. Since T contains n edges, it follows that the length of T is at most equal to n times the length of T_{opt}. ∎

Lemma 2 *Let S be a set of n points in $\mathbb{R}^d$, and let G be a connected graph on S containing m edges and having weight at most r times the weight of an MST of S. Then, in $O(m)$ time, we can compute a $2r$-approximate TSP-tour for S.*

Proof: Using depth first search, compute a spanning tree G' of G. Then, double each edge of G', compute an Euler tour of the resulting graph, and, finally, by short-cutting this Euler tour, make it into a tour T. By the triangle inequality, T has length at most twice the weight of G. This, in turn, implies that T has length at most $2r \cdot wt(MST(S))$.

Let T_{opt} denote an optimal *TSP*-tour for the set S. By deleting any edge of T_{opt}, we get a spanning tree of S. Hence, the length of T_{opt} is at least equal to $wt(MST(S))$. This proves that the tour T is a $2r$-approximate *TSP*-tour for S. It is clear that T can be computed from G in $O(m)$ time. (Note that $m \geq n-1$, because G is connected.) ∎

Corollary 1 *The lower bound of Theorem 1 implies the lower bound of Theorem 2.*

Proof: Let $\mathcal{A}$ be an algorithm that, given a set S of n points in $\mathbb{R}^d$ and a sufficiently large real number $r < n$, computes a connected graph on S having weight at most $r \cdot wt(MST(S))$. If such a graph contains $\Omega(n \log n/r)$ edges, then $\mathcal{A}$ clearly takes $\Omega(n \log n/r)$ time. So assume that any such graph constructed by $\mathcal{A}$ has $o(n \log n/r)$ edges. Then, Lemma 2 and Theorem 1 imply that $\mathcal{A}$ has running time $\Omega(n \log n/r)$. ∎

3 The lower bound proof

In this section, we prove Theorem 1. By Corollary 1, this will also prove Theorem 2.

We prove Theorem 1 for algorithms that solve the r-approximate *TSP* problem for one-dimensional point sets. Clearly, this will prove the theorem for any dimension $d \geq 1$.

Throughout the rest of this section, $\mathcal{A}$ denotes any algorithm that, given a set S of n real numbers and a sufficiently large real number $r < n$, computes an r-approximate *TSP*-tour for S. We will show that the worst-case running time of $\mathcal{A}$ is $\Omega(n \log n/r)$. In fact, we prove this lower bound for even values of n. It is easy to see that this implies the lower bound for odd values of n as well.

Hence from now on, we only consider even values of n and values of r that are larger than some appropriate constant and less than n.

Here is an outline of our proof. First, we define an algorithm $\mathcal{B}$ that, when given $n+1$ real numbers $x_1, x_2, \ldots, x_n, r$ as input, runs algorithm $\mathcal{A}$ and constructs from $\mathcal{A}$'s output two lists SL and LL, the so-called source and length lists. $\mathcal{B}$ outputs the pair (SL, LL). Its running time is roughly the same as that of $\mathcal{A}$. Then, we consider the outputs of $\mathcal{B}$ on all inputs $\pi(1), \pi(2), \ldots, \pi(n), r$, where π ranges over all $n!$ permutations of $1, 2, \ldots, n$, and choose the one that occurs most frequent. Next, we define a set $W \subseteq \mathbb{R}^n$, consisting of all points $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ such that $\mathcal{B}$ computes this special output when given $x_1, x_2, \ldots, x_n, r$ as input. We show that the

logarithm of the number of connected components of W is $\Omega(n \log n/r)$. Finally, we define an algorithm $\mathcal{C}$ that accepts W and whose running time is roughly the same as that of $\mathcal{B}$ and, hence, of $\mathcal{A}$. Theorem 3 implies that algorithm $\mathcal{C}$ and, hence, also $\mathcal{A}$ have $\Omega(n \log n/r)$ running time.

Algorithm $\mathcal{B}$ does the following on an input consisting of $n+1$ real numbers $x_1, x_2, \ldots, x_n, r$.

Step B1: Run algorithm $\mathcal{A}$ on the input $x_1, x_2, \ldots, x_n, r$. Let

$$(x_{i_1}, x_{i_2}, \ldots, x_{i_n}, x_{i_1})$$

be the r-approximate *TSP*-tour that is computed by $\mathcal{A}$.

Step B2: For j, $1 \leq j \leq n/2$, let

$$e_j := \{x_{i_{2j-1}}, x_{i_{2j}}\}.$$

Give each e_j a direction, from the smaller to the larger element, breaking ties arbitrarily, and denote the resulting edge by $\vec{e_j}$. Hence,

$$\vec{e_j} = (\min(x_{i_{2j-1}}, x_{i_{2j}}), \max(x_{i_{2j-1}}, x_{i_{2j}})).$$

We call the two components of $\vec{e_j}$ its *source* and *sink*, respectively. The *weight* of the edge is defined as the difference of its sink and its source.

Step B3: Compute a *source list SL* of length n. For $1 \leq j \leq n$, the j-th element of this list is equal to x_j, if x_j is the source of some edge $\vec{e_\ell}$, and equal to a special symbol $\star$, if x_j is the sink of some edge $\vec{e_\ell}$.

Step B4: Compute a *length list LL* of length n. For $1 \leq j \leq n$, the j-th element of this list is equal to the weight of the edge $\vec{e_\ell}$ having x_j as its source, provided this edge exists. Otherwise, if x_j is the sink of some edge $\vec{e_\ell}$, the special symbol $\star$ occurs at position j.

Step B5: Output the pair of lists (SL, LL).

Note that the edges $\vec{e_j}$ form a perfect matching of $x_1, x_2, \ldots, x_n$. As an example, let $n = 4$, $x_3 < x_4 < x_1 < x_2$, and assume $\mathcal{A}$ computes the tour $(x_1, x_3, x_4, x_2, x_1)$. Then we have $\vec{e_1} = (x_3, x_1)$ and $\vec{e_2} = (x_4, x_2)$. The output of algorithm $\mathcal{B}$ consists of the lists $SL = (\star, \star, x_3, x_4)$ and $LL = (\star, \star, x_1 - x_3, x_2 - x_4)$.

Lemma 3 *Let $T_{\mathcal{A}}(n,r)$ and $T_{\mathcal{B}}(n,r)$ denote the worst-case running times of algorithms $\mathcal{A}$ and $\mathcal{B}$, respectively. Then there is a constant c independent of n and r, such that*

$$T_{\mathcal{B}}(n,r) \leq T_{\mathcal{A}}(n,r) + cn.$$

Proof: We assume that the input sequence $x_1, x_2, \ldots, x_n, r$ is stored in a linked list. Moreover, we adapt algorithm $\mathcal{A}$ such that when it computes an edge (x_i, x_j) of the r-approximate *TSP*-tour, we give the occurrences of x_i and x_j in the input list pointers to this edge. Then, by walking along the input list, we can compute the lists SL and

LL in $O(n)$ time, within the algebraic computation tree model. In particular, random access is not used. ■

We now fix an even integer n and a real number r. Let π be any permutation of $1, 2, \ldots, n$. Let (SL_π, LL_π) be the output of algorithm $\mathcal{B}$ when given as input $\pi(1), \pi(2), \ldots, \pi(n), r$. Among all these $n!$ pairs (SL_π, LL_π), let $(\mathcal{S}_{n,r}, \mathcal{L}_{n,r})$ be one that occurs most frequent.

As an example, we may have $\mathcal{S}_{n,r} = (2, \star, \star, 1)$ and $\mathcal{L}_{n,r} = (1, \star, \star, 3)$. Then the inputs $2, 3, 4, 1, r$ and $2, 4, 3, 1, r$ *may* produce these lists.

Define W as the set of all points $(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ such that algorithm $\mathcal{B}$, when given $x_1, x_2, \ldots, x_n, r$ as its input, outputs the pair $(\mathcal{S}_{n,r}, \mathcal{L}_{n,r})$.

Lemma 4 *Let $\pi_1, \pi_2, \ldots, \pi_k$ be the permutations of $1, 2, \ldots, n$ such that $(SL_{\pi_i}, LL_{\pi_i}) = (\mathcal{S}_{n,r}, \mathcal{L}_{n,r})$, $1 \leq i \leq k$. Then W has at least k connected components.*

Proof: Assume w.l.o.g. that $\mathcal{S}_{n,r}$ has the form

$$\mathcal{S}_{n,r} = (a_1, a_2, \ldots, a_{n/2}, \star, \star, \ldots, \star),$$

where $\{a_1, a_2, \ldots, a_{n/2}\}$ is a subset of $\{1, 2, \ldots, n\}$ of size $n/2$. Let $\mathcal{L}_{n,r}$ be given by

$$\mathcal{L}_{n,r} = (l_1, l_2, \ldots, l_{n/2}, \star, \star, \ldots, \star).$$

Note that all non-$\star$ elements of $\mathcal{S}_{n,r}$ and $\mathcal{L}_{n,r}$ are integers.

Let $1 \leq i < j \leq k$. We show that the permutations π_i and π_j belong to different connected components of W. (Note that both these permutations are elements of W.) This will prove the lemma.

Since $(SL_{\pi_i}, LL_{\pi_i}) = (SL_{\pi_j}, LL_{\pi_j}) = (\mathcal{S}_{n,r}, \mathcal{L}_{n,r})$, we can write the permutations π_i and π_j as

$$\pi_i = (a_1, a_2, \ldots, a_{n/2}, b_1, b_2, \ldots, b_{n/2})$$

and

$$\pi_j = (a_1, a_2, \ldots, a_{n/2}, c_1, c_2, \ldots, c_{n/2}),$$

where

$$\{b_1, b_2, \ldots, b_{n/2}\} = \{c_1, c_2, \ldots, c_{n/2}\} = \{1, 2, \ldots, n\} \setminus \{a_1, a_2, \ldots, a_{n/2}\}.$$

Let ℓ be an index such that $b_\ell \neq c_\ell$.

Consider any continuous curve in $\mathbb{R}^n$ that connects π_i and π_j. Let

$$P = (p_1, p_2, \ldots, p_{n/2}, q_1, q_2, \ldots, q_{n/2})$$

be a point on this curve such that q_ℓ is not an integer. Note that point P exists, because b_ℓ and c_ℓ are distinct integers. Let us look what happens when algorithm $\mathcal{B}$ is run on input $p_1, p_2, \ldots, p_{n/2}, q_1, q_2, \ldots, q_{n/2}, r$.

In Step B1, an r-approximate TSP-tour T is computed. In the rest of algorithm $\mathcal{B}$, a source list

$$SL = (\alpha_1, \alpha_2, \ldots, \alpha_n)$$

and a length list

$$LL = (\beta_1, \beta_2, \ldots, \beta_n)$$

is computed. We distinguish two cases.

Case 1: $(\alpha_1, \alpha_2, \ldots, \alpha_{n/2}) \neq (a_1, a_2, \ldots, a_{n/2})$.

In this case, $SL \neq \mathcal{S}_{n,r}$. Hence, point P does not belong to our set W.

Case 2: $(\alpha_1, \alpha_2, \ldots, \alpha_{n/2}) = (a_1, a_2, \ldots, a_{n/2})$.

Since SL contains exactly $n/2$ $\star$'s, we must have $SL = \mathcal{S}_{n,r}$. Consider the perfect matching of the r-approximate TSP-tour T that is computed in Step B2. Let $\vec{e}$ be the edge that contains q_ℓ. The source of this edge is contained in the source list SL, which is equal to $\mathcal{S}_{n,r}$, and which contains only integers and $\star$'s. It follows that q_ℓ must be a sink, and the weight of $\vec{e}$ is not an integer. Since this non-integer weight occurs in the length list LL, we must have $LL \neq \mathcal{L}_{n,r}$. As a result, also in this case point P does not belong to the set W.

We have shown that any curve connecting π_i and π_j passes through a point outside W. Hence, π_i and π_j are contained in different connected components of W. ■

Lemma 5 *The number of permutations π of $1, 2, \ldots, n$ such that*

$$(SL_\pi, LL_\pi) = (\mathcal{S}_{n,r}, \mathcal{L}_{n,r})$$

is at least

$$(n/2)! \Big/ \left(\binom{n}{n/2} \binom{2rn + n/2}{n/2} \right).$$

Proof: Consider again the output (SL_π, LL_π) of $\mathcal{B}$ when given $\pi(1), \pi(2), \ldots, \pi(n), r$ as input. We give an upper bound on the total number of different outputs if π ranges over all permutations of $1, 2, \ldots, n$.

A source list contains $n/2$ distinct integers from $\{1, 2, \ldots, n\}$, and $n/2$ special symbols $\star$. Hence, the total number of different source lists is at most equal to

$$\binom{n}{n/2} \cdot \frac{n!}{(n/2)!}.$$

Consider one fixed source list SL. How many different length lists LL are there such that (SL, LL) is an output of algorithm $\mathcal{B}$? Such a list LL contains $n/2$ $\star$'s, and $n/2$ non-$\star$ elements. Since we have fixed SL, the positions in LL that contain these non-$\star$'s are also fixed. Every non-$\star$ is an integer. Recall that the length list represents the edge weights of a perfect matching of the r-approximate TSP-tour computed in Step B1. Since the input is a permutation of $1, 2, \ldots, n$, we know that the optimal TSP-tour has length $2(n-1) \leq 2n$. Hence, the tour computed in Step B1 has length at most $2rn$. This, in turn, implies that the sum of the non-$\star$ symbols in the length list LL is at most equal to $2rn$. It follows that for this fixed source list SL, the total number of different corresponding length lists LL is at most equal to the number of solutions of the inequality

$$x_1 + x_2 + \cdots + x_{n/2} \leq 2rn$$

in non-negative integers x_i. It is well known (see [5, pages 103–104]) that the latter quantity is equal to

$$\binom{2rn + n/2}{n/2}.$$

We have shown that by running $\mathcal{B}$ on all $n!$ different permutations of $1, 2, \ldots, n$, we get at most

$$X := \binom{n}{n/2} \cdot \frac{n!}{(n/2)!} \cdot \binom{2rn + n/2}{n/2}$$

different outputs (SL, LL). Therefore, by the pigeon-hole principle, one of these outputs is computed on at least $n!/X$ inputs. ■

Now we can complete the proof of Theorem 1. First, we show that

$$Y := \ln \left(\frac{\left(\frac{n}{2}\right)!}{\binom{n}{n/2}\binom{2rn+n/2}{n/2}} \right) = \Omega(n \log n/r).$$

Lemma 6 *Let $0 \leq b \leq a$ be integers. Then*

$$\binom{a}{b} \leq \frac{a^a}{b^b (a-b)^{a-b}}.$$

Proof: We have

$$a^a = (b + (a-b))^a = \sum_{i=0}^{a} \binom{a}{i} b^i (a-b)^{a-i} \geq \binom{a}{b} b^b (a-b)^{a-b}. \blacksquare$$

Applying Lemma 6, and the inequalities $\binom{n}{n/2} \leq 2^n$ and $a! \geq (a/e)^a$ for $a = n/2$, we get

$$\begin{aligned} \frac{\left(\frac{n}{2}\right)!}{\binom{n}{n/2}\binom{2rn+n/2}{n/2}} &\geq \frac{\left(\frac{n}{2e}\right)^{n/2} \left(\frac{n}{2}\right)^{n/2} (2rn)^{2rn}}{2^n \, (2rn + n/2)^{2rn+n/2}} \\ &= \frac{n^n \, (2rn)^{2rn}}{2^{2n} \, e^{n/2} \, (2rn + n/2)^{2rn+n/2}}. \end{aligned}$$

Taking logarithms gives

$$\begin{aligned} Y &\geq n \ln n + 2rn \ln(2rn) - (2rn + n/2) \ln(2rn + n/2) - 2n \ln 2 - n/2 \\ &= n \ln \frac{n}{\sqrt{2rn + n/2}} + 2rn \ln \frac{2rn}{2rn + n/2} - 2n \ln 2 - n/2 \\ &= \frac{1}{2} n \ln \frac{n}{2r + 1/2} + n \ln \left(\frac{2r}{2r + 1/2} \right)^{2r} - 2n \ln 2 - n/2. \end{aligned}$$

We have

$$\left(\frac{2r}{2r + 1/2} \right)^{2r} = \left(1 - \frac{1/2}{2r + 1/2} \right)^{2r} = \left(1 - \frac{1/2}{2r + 1/2} \right)^{2r+1/2} \left(1 - \frac{1/2}{2r + 1/2} \right)^{-1/2}.$$

The quantity on the right-hand side converges to $e^{-1/2}$ if $r \to \infty$. Therefore,

$$\ln\left(\frac{2r}{2r+1/2}\right)^{2r} \geq -1$$

for all sufficiently large r. Also, since $2r + 1/2 \leq 3r$, we have $\ln(n/(2r+1/2)) \geq \ln(n/(3r)) = \ln n/r - \ln 3$. It follows that

$$Y \geq \frac{1}{2}\, n \ln \frac{n}{r} - \frac{1}{2}\, n \ln 3 - n - 2n \ln 2 - n/2 = \Omega(n \ln n/r), \tag{1}$$

which is exactly what we wanted to prove. Note that the constant factor in the Ω-bound in (1) does not depend on n and r.

Recall that we denote the number of connected components of the set W by $\#W$. Lemmas 4 and 5, and inequality (1) imply that, for our fixed values of n and r, we have

$$\log \#W \geq c_0\, n \log n/r,$$

where c_0 is a constant that does not depend on n and r.

Consider the following algorithm $\mathcal{C}$. It only accepts inputs of our fixed length n. On input $x_1, x_2, \ldots, x_n$, algorithm $\mathcal{C}$ does the following.

Step C1: Run algorithm $\mathcal{B}$ on the input $x_1, x_2, \ldots, x_n, r$. Let the output be the pair (SL, LL).

Step C2: Output YES if $SL = \mathcal{S}_{n,r}$ and $LL = \mathcal{L}_{n,r}$. Otherwise, output NO.

Since algorithm $\mathcal{C}$ only accepts inputs of our fixed length n, and since we also fixed r, we may assume that it "knows" the two lists $\mathcal{S}_{n,r}$ and $\mathcal{L}_{n,r}$. Algorithm $\mathcal{C}$ *exists*, although we have not explicitly computed these lists. The following lemma is clear.

Lemma 7 *Let $T_{\mathcal{B}}(n,r)$ and $T_{\mathcal{C}}(n,r)$ denote the worst-case running times of algorithms $\mathcal{B}$ and $\mathcal{C}$, respectively. Then there is a constant c' independent of n and r, such that*

$$T_{\mathcal{C}}(n,r) \leq T_{\mathcal{B}}(n,r) + c'n.$$

Algorithm $\mathcal{C}$ accepts exactly our set W. Hence, by Theorem 3, there is an input on which this algorithm takes time at least $c_1 n \log n/r$, for some constant c_1 that does not depend on n and r. Then, by Lemmas 3 and 7, there is an input on which algorithm $\mathcal{A}$ takes time at least $c_2 n \log n/r$, for some constant c_2. Since c_2 does not depend on n and r, this implies that the lower bound holds for all values of n and r. This completes the proof of Theorem 1.

4 Proving the matching upper bound

In this section, we show that the lower bounds of Theorems 1 and 2 are tight. By Lemma 2, it suffices to give an $O(n \log n/r)$-time algorithm that computes an r-approximate *MST* for any set of n points in $\mathbb{R}^d$, and any $4 < r < n$.

Let S be a set of n points in $\mathbb{R}^d$, and let $4 < r < n$. The algorithm does the following.

Step 1: Compute the smallest axes-parallel d-dimensional cube that contains all points of S. Let ℓ be the side length of this cube. Translate the set S such that it is contained in the cube $[0, \ell]^d$.

Step 2: Let $b := (r-2)\,\ell/(5n\sqrt{d})$. Note that

$$\frac{\ell}{b} = \frac{5n\sqrt{d}}{r-2} \leq 10\sqrt{d}\,\frac{n}{r} \leq 3n\sqrt{d}.$$

Compute the integer $\lfloor \ell/b \rfloor$ by making a scan along the integers $1, 2, \ldots, 3n\sqrt{d}$. Build a balanced binary search tree BT, storing the integers $0, 1, 2, \ldots, \lfloor \ell/b \rfloor$ in its leaves.

For each point $p = (p_1, p_2, \ldots, p_d)$ of S, and each index i, $1 \leq i \leq d$, use this tree to find the integer

$$c_i^p := \lfloor p_i/b \rfloor.$$

Hence, considering the grid over $[0, \ell]^d$ with mesh size b, point p is contained in the c_i^p-th slab along the i-th dimension. We call

$$C_p := (c_1^p, c_2^p, \ldots, c_d^p)$$

the *grid vector* of p.

Sort the n grid vectors lexicographically, by inserting them one after another into an initially empty balanced binary search tree BT'. With each leaf of this tree, store all points p with the same grid vector.

Step 3: For each grid vector C that occurs in BT', do the following. Let S_C be the set of points p of S for which $C_p = C$. Pick an arbitrary point q_C of S_C, and call it the *representative* of S_C. Then connect each point of $S_C \setminus \{q_C\}$ to this representative. This gives a tree T_C on S_C.

Step 4: Let R be the set of all representatives. Compute a 2-approximate *MST* T^R for the points of R, using any one of the (algebraic computation tree) algorithms of [4, 8, 9]. (In fact, if $d = 2$, we can even compute an exact *MST* for R, using the algorithm given in [7].)

Step 5: Let T be the tree obtained by taking the union of all trees T_C and the tree T^R. Output T.

Lemma 8 *The running time of this algorithm is bounded by* $O(n \log n/r)$.

Proof: Steps 1, 3 and 5 take $O(n)$ time. Consider Step 2. The tree BT can be built in $O(n)$ time. Given this tree, the grid vectors can be computed in time $O(n \log \ell/b) = O(n \log n/r)$. Since the tree BT' contains at most $(\ell/b)^d$ elements, its height is bounded by $O(\log(\ell/b)^d) = O(\log n/r)$. Hence, all n grid vectors can be inserted into BT' in time $O(n \log n/r)$. This proves that Step 2 takes time $O(n \log n/r)$. Step 4 takes time $O(|R| \log |R|)$. Since R is a subset of S, we have $|R| \leq n$. Also, the size of R is at most equal to the number of cells in our grid, i.e., $|R| \leq (\ell/b)^d$. Therefore, Step 4 takes time $O(n \log(\ell/b)^d) = O(n \log n/r)$. This completes the proof. ∎

It remains to show that the tree T computed by the algorithm is an r-approximate MST. Consider the minimum spanning tree $MST(S)$ for the set S. Then we have to show that $wt(T) \leq r \cdot wt(MST(S))$.

The following argument is due to Bern et al.[3]. Imagine "moving" each point of S to the representative of the cell it is contained in. Then the tree $MST(S)$ "moves" to a graph, say T', possibly containing multiple edges and loops.

Lemma 9 $wt(T') \leq 2\sqrt{d}\, bn + wt(MST(S))$.

Proof: Consider any edge (p', q') of T'. Then there is a unique edge (p, q) in $MST(S)$ that was "moved" to (p', q'). Denoting the Euclidean distance between two points x and y by $|xy|$, and using the triangle inequality, we have

$$|p'q'| \leq |p'p| + |pq| + |qq'|.$$

Since p and p' are contained in the same grid cell, we have $|p'p| \leq \sqrt{d}\, b$. Similarly, $|qq'| \leq \sqrt{d}\, b$. Hence,

$$|p'q'| \leq 2\sqrt{d}\, b + |pq|.$$

Doing this for all $n-1$ edges of T' proves the claim. ■

Lemma 10 *The tree T is an r-approximate MST for the set S.*

Proof: Let T^R_{opt} be the exact MST for the points of R. Then the graph T^R computed in Step 4 satisfies $wt(T^R) \leq 2 \cdot wt(T^R_{opt})$.

Since T' is a spanning graph of R, we have $wt(T^R_{opt}) \leq wt(T')$. This, together with Lemma 9, implies that

$$wt(T^R) \leq 4\sqrt{d}\, bn + 2 \cdot wt(MST(S)).$$

Now, consider the total weight of the edges in the trees T_C that are computed in Step 3. Clearly, each such edge has weight at most $\sqrt{d}\, b$. It follows that

$$wt(T) = wt(T^R) + \sum_C wt(T_C) \leq 5\sqrt{d}\, bn + 2 \cdot wt(MST(S)).$$

Our choice of ℓ in Step 1 of the algorithm guarantees that there are two points of S having distance at least ℓ. Hence,

$$wt(MST(S)) \geq \ell = \frac{5\sqrt{d}\, bn}{r-2}.$$

This implies that

$$wt(T) \leq (r-2)\,\ell + 2 \cdot wt(MST(S)) \leq r \cdot wt(MST(S)).$$

This completes the proof. ■

We summarize our result.

Theorem 4 *Let $d \geq 1$ be an integer constant. There is an algorithm that, given a set S of n points in $\mathbb{R}^d$ and a real number $4 < r < n$, computes an r-approximate MST for S in $O(n \log n/r)$ time. This algorithm fits in the algebraic computation tree model. Hence, the lower bound of Theorem 2 is tight.*

Corollary 2 *Let $d \geq 1$ be an integer constant. There is an algorithm that, given a set S of n points in $\mathbb{R}^d$ and a real number $8 < r < n$, computes an r-approximate TSP-tour for S in $O(n \log n/r)$ time. This algorithm fits in the algebraic computation tree model. Hence, the lower bound of Theorem 1 is tight.*

5 A non-algebraic computation tree algorithm

The results of the previous sections imply that in the algebraic computation tree model a very large approximation factor is needed in order to get a running time of $o(n \log n)$. In this section, we consider algorithms from a more powerful model of computation. More precisely, we assume that besides the operations of the algebraic computation tree model, we have the non-algebraic floor function and random access available. We will prove the following result.

Theorem 5 *Let $d \geq 1$ be an integer constant.*

1. *There is an algorithm that, given a set S of n points in $\mathbb{R}^d$, computes an r-approximate MST for S in $O(n)$ time, where $r = 3\sqrt{d}\, n^{1-1/d}$.*
2. *There is an algorithm that, given a set S of n points in $\mathbb{R}^d$, computes an r-approximate TSP-tour for S in $O(n)$ time, where $r = 6\sqrt{d}\, n^{1-1/d}$.*

Besides the operations of the algebraic computation tree model, these algorithms use the non-algebraic floor function and random access.

Let S be a set of n points in $\mathbb{R}^d$. The following algorithm computes an approximate *MST* for S.

Step 1: Let $r := 3\sqrt{d}\, n^{1-1/d}$. Compute the smallest axes-parallel d-dimensional cube that contains all points of S. Let ℓ be the side length of this cube. Translate the set S such that it is contained in the cube $[0, \ell]^d$.

Step 2: Let $b := \ell/n^{1/d}$. Initialize a d-dimensional array

$$I[0..\lfloor \ell/b \rfloor, 0..\lfloor \ell/b \rfloor, \ldots, 0..\lfloor \ell/b \rfloor].$$

Initialize an empty list with each array entry. Then store each point $p = (p_1, p_2, \ldots, p_d)$ of S in the list stored with

$$I[\lfloor p_1/b \rfloor, \lfloor p_2/b \rfloor, \ldots, \lfloor p_d/b \rfloor].$$

Step 3: For each array entry C, let S_C be the set of points of S that are stored in the list corresponding to this entry. If S_C is non-empty, pick an arbitrary point q_C in this

set and call it the *representative*. Let T_C be the tree on S_C obtained by connecting each point of $S_C \setminus \{q_C\}$ to the representative.

Step 4: Walk along the $(\ell/b)^d$ entries of the array I in lexicographical order. This defines an ordering on the representatives. Connect these representatives into a tree T^R—in fact, a path—according to this ordering.

Step 5: Let T be the tree obtained by taking the union of all trees T_C and the tree T^R. Output T.

Lemma 11 *The running time of this algorithm is bounded by $O(n)$.*

Proof: It is easy to see that the running time is bounded by $O(n + (\ell/b)^d)$. By our choice of b, this is bounded by $O(n)$. ■

Lemma 12 *Let $r := 3\sqrt{d}\, n^{1-1/d}$. Consider the minimum spanning tree $MST(S)$ of the set S. Then we have $wt(T) \leq r \cdot wt(MST(S))$, i.e., T is an r-approximate MST.*

Proof: First consider the path T^R. This path has a vertex in each grid cell that contains at least one point of S. We take an arbitrary point from each grid cell that does not contain any points of S. Let T' be the path on the representatives plus these extra points that is defined by the ordering of Step 4. Then, the representatives occur in T^R and T' in the same order. By the triangle inequality, the weight of T^R is at most equal to that of T'. It is easy to see that for each grid cell there are at most two edges of T' that intersect this cell. Hence, since there are $(\ell/b)^d$ cells, each one having diameter $\sqrt{d}\, b$, the weight of T' is bounded above by $2\,(\ell/b)^d \sqrt{d}\, b$. This implies that

$$wt(T^R) \leq 2\,(\ell/b)^d \sqrt{d}\, b = 2n\sqrt{d}\, b.$$

It is also clear that each edge of any tree T_C has weight at most $\sqrt{d}\, b$. It follows that

$$wt(T) \leq n\sqrt{d}\, b + 2n\sqrt{d}\, b = 3n\sqrt{d}\, b.$$

We know that $wt(MST(S)) \geq \ell$. Hence, it suffices to show that $3n\sqrt{d}\, b \leq r\ell$, which simplifies to $3n\sqrt{d} \leq r n^{1/d}$. But this holds with equality, by our choice of r. ■

Lemmas 11 and 12 prove the first claim of Theorem 5. The second claim follows from Lemma 2.

6 Concluding remarks

We have shown that in the algebraic computation tree model, the complexities of the r-approximate *TSP* and *MST* problems are $\Theta(n \log n/r)$, for large values of r and n such that $r < n$. We have also shown that in a model that additionally uses the floor function and random access, it is possible to solve both problems in linear time for $r = c\, n^{1-1/d}$ for some constant c.

We mention the following three open problems. First, in this more powerful model, can these problems be solved in linear time for values of r that are smaller than

$n^{1-1/d}$? Second, is it possible in this model to solve the planar version of the problems in $o(n \log\log n)$ time for a constant value of r (thereby improving the result of [3])? Finally, for dimension $d \geq 3$, can these problems be solved in $o(n \log n)$ time in this model for a constant value of r?

References

[1] P.K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. *Euclidean minimum spanning trees and bichromatic closest pairs.* Discrete & Computational Geometry **6** (1991), pp. 407-422.

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

[3] M.W. Bern, H.J. Karloff, P. Raghavan, and B. Schieber. *Fast geometric approximation techniques and geometric embedding problems.* Proceedings 5th Annual ACM Symposium on Computational Geometry, 1989, pp. 292-301.

[4] P.B. Callahan and S.R. Kosaraju. *Faster algorithms for some geometric graph problems in higher dimensions.* Proceedings 4th Annual Symposium on Discrete Algorithms, 1993, pp. 291–300.

[5] J.H. van Lint and R.M. Wilson. *A Course in Combinatorics.* Cambridge University Press, 1992.

[6] C.H. Papadimitriou. *The Euclidean traveling salesman problem is NP-complete.* Theoretical Computer Science **4** (1977), pp. 237-244.

[7] F.P. Preparata and M.I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, New York, 1985.

[8] J.S. Salowe. *Constructing multidimensional spanner graphs.* International Journal of Computational Geometry and Applications **1** (1991), pp. 99-107.

[9] P.M. Vaidya. *Minimum spanning trees in k-dimensional space.* SIAM Journal on Computing **17** (1988), pp. 572-582.