Faster exact algorithms for computing Steiner trees in higher dimensional Euclidean spaces

Fonseca, Rasmus; Brazil, Marcus; Winter, Pawel; Zachariasen, Martin

Publication date:
2014

Document version
Early version, also known as pre-print

Citation for published version (APA):
Fonseca, R., Brazil, M., Winter, P., & Zachariasen, M. (2014). *Faster exact algorithms for computing Steiner trees in higher dimensional Euclidean spaces*. Paper presented at 11th DIMACS Implementation Challenge, Providence, United States.

Faster Exact Algorithms for Computing Steiner Trees in Higher Dimensional Euclidean Spaces

Rasmus Fonseca[1], Marcus Brazil[2], Pawel Winter[1], and Martin Zachariasen[1]

[1] Department of Computer Science, University of Copenhagen,
DK-2100 Copenhagen Ø, Denmark,
rfonseca@di.ku.dk, pawel@di.ku.dk, martinz@di.ku.dk
[2] Department of Electrical and Electronic Engineering, The University of Melbourne,
Victoria 3010, Australia,
brazil@unimelb.edu.au

Abstract. The Euclidean Steiner tree problem asks for a network of minimum total length interconnecting a finite set of points in d-dimensional space. For $d \geq 3$, only one practical algorithmic approach exists for this problem — proposed by Smith in 1992. A number of refinements of Smith's algorithm have increased the range of solvable problems a little, but it is still infeasible to solve problem instances with more than around 17 terminals. In this paper we firstly propose some additional improvements to Smith's algorithm. Secondly, we propose a new algorithmic paradigm called branch enumeration. Our experiments show that branch enumeration has similar performance as an optimized version of Smith's algorithm; furthermore, we argue that branch enumeration has the potential to push the boundary of solvable problems further.

Keywords: Steiner tree problem, d-dimensional Euclidean space, exact algorithm, computational study

1 Introduction

Given a finite set of n points N in a d-dimensional space $\mathbb{R}^d$, ($d \geq 2$), the *Euclidean Steiner tree problem* (ESTP) asks for a network T^* of minimum length interconnecting N. The vertices of T^* corresponding to points of N are called *terminals* while possible additional vertices in T^* are called *Steiner points*. T^* necessarily is a tree, and is referred to as a *Euclidean minimum Steiner tree* (EMST) for N. The ESTP in the plane ($d = 2$) has a history that goes back more than two centuries [4], and is known to be NP-hard [9], even when the terminals are restricted to lie on two parallel lines [16]. The generalisation to more than two dimensions was introduced by Bopp [3] in 1879; more recent mathematical treatments can be found in [10, 13]. The problem in $\mathbb{R}^d$, $d \geq 3$, has applications to areas such as phylogenetics [6, 7, 5] and to the structure and folding of proteins [18, 20].

An EMST T^* can be viewed as a union of *full* EMSTs whose terminals have degree 1. Steiner points in T^* (in any dimension) have degree 3. The three

incident edges at a Steiner point are co-planar and each pair meets at an angle of 120^o. These degree conditions imply that full EMSTs spanning k terminals, $1 \leq k \leq n$, have $k-2$ Steiner points.

The first exact algorithms for the ESTP in the plane ($\mathbb{R}^2$) were based on the following common framework [12]. Subsets of terminals are considered one by one. For each subset, all its full EMSTs are determined and the shortest is retained. Several tests are applied to these retained full EMSTs to decide if they can belong to an EMST for N. Surviving full EMSTs are then concatenated in all possible ways to obtain trees spanning N. The shortest of them is T^*.

The main bottleneck in this approach is the generation of full EMSTs. It has been observed [24] that substantial improvements can be obtained if full RMSTs are generated simultaneously across various subsets of terminals. Very powerful geometrical pruning tests identifying non-optimal full EMSTs can then be applied not *after* but *during* their generation. As a consequence of this speed-up, the concatenation of full EMSTs became a bottleneck of the EMST computation. A remedy was based on the observation that the concatenation of full EMSTs can be formulated as a problem of finding a minimum spanning tree in a hypergraph with terminals as vertices and subsets spanned by full EMSTs as hyperedges [21]. In practice, this problem can be solved efficiently using branch-and-cut techniques. The dramatic improvements of both the generation and concatenation of full EMSTs led to the development of GeoSteiner [22, 23, 25] which can routinely solve problem instances with thousands of terminals in $\mathbb{R}^2$ in a reasonable amount of time. A similar methodology has been applied to other metrics and generalisations [11, 15, 26, 27].

Significantly less improvement has been made on exact algorithms for the problem in $\mathbb{R}^d$, $d \geq 3$. Currently, only one practical algorithmic approach exists. It was proposed by Smith [19] in 1992. A couple of recent contributions [8, 14], all based on Smith's algorithm, have pushed the boundary of solvable problems a little, but in practice it is still infeasible to solve problems for $d \geq 3$ with more than 17 terminals.

As observed by Fampa and Anstreicher [8], the bounds used in [19] do not correspond to rigorous lower bounds on the solution values of these problems, but are instead obtained from putatively near-optimal solutions. In addition, when a node fails to fathom the algorithm has no means to estimate the objective values associated with its potential children. As a result the terminal nodes are added in a fixed order, even though varying the order has the potential to substantially reduce the size of the branch-and-bound tree.

Our Contribution. In this paper we first describe and evaluate some improvements to Smith's algorithm. Then we describe a novel exact algorithm, a so-called a *branch enumeration* approach. The branch enumeration algorithm deviates significantly from Smith's algorithm. One of the advantages of the branch enumeration algorithm is that it is possible to apply stronger pruning tests early in the enumeration, e.g. based on the notion of bottleneck Steiner distances. We evaluate our new branch enumeration algorithm and our improved version of Smith's algorithm experimentally on a set of benchmark instances.

GeoSteiner [22, 23, 25] in the Euclidean plane can be viewed as a branch enumeration (to obtain a superset of full Steiner trees in the optimal solution) followed by the concatenation of these full Steiner trees (using branch and cut to solve the minimum spanning tree problem in a hypergraph). Prunning techniques in GeoSteiner are extremely powerful and make it possible to efficiently prune nonoptimal full Steiner trees. A branch enumeration algorithm has also been suggested to compute Steiner minimum trees in Hamming metric space [1]. Our approach to find Steiner minimum trees in high-dimensional Euclidean spaces is similar. However, pruning techniques in the Hamming metric space seem to be much stronger than in high-dimensional Euclidean spaces (but far from being as strong as in GeoSteiner when applied to the problems in the Euclidean plane).

Organisation of the Paper. In Section 2 we present some preliminaries on the ESTP. Smith's algorithm is introduced in Section 3. Its improvements are discussed in Section 4. Then in Section 5 we describe the branch enumeration algorithm. Computational results are presented in Section 6, and concluding remarks are given in Section 7.

2 Preliminaries

Consider a tree T in $\mathbb{R}^d$ that interconnects a set N of n points. We assume that T consists of a set of *vertices* (a superset of N) and a set of *edges* (straight line segments) connecting pairs of vertices. The *length* of T, denoted by $|T|$, is the sum of its edge lengths. The length of an edge (u, v) connecting points u and v is denoted by $|uv|$. The given points of N are called the *terminals*, and the other vertices, if any, are called *Steiner points* (see Fig. 1). The ESTP is to determine the shortest tree T^* for N. Steiner points in T^* have degree 3 while terminals have degree at most 3. Any tree satisfying these degree constraints is called a *Steiner tree* for N.

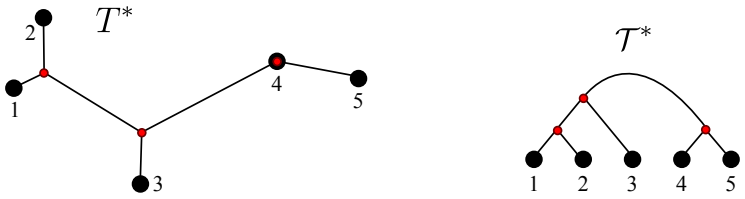


Fig. 1: The EMST of five terminals in $\mathbb{R}^2$. The terminals are shown as filled black circles, and the Steiner points are shown as smaller red circles. The topology $\mathcal{T}^*$ of T^* is shown to the right. Terminal 4 is connected to a Steiner point by a zero-length edge and therefore the Steiner point overlaps with terminal 4.

A *Steiner topology* $\mathcal{T}$ of a Steiner tree T for N is a specification of the interconnections in T, disregarding the positions of its vertices. Any Steiner topology can be transformed into a *full Steiner topology* (FST) where all terminals have

degree 1. If $\mathcal{T}$ has a terminal t adjacent to two vertices v_1 and v_2, a new Steiner point s connected to t, v_1 and v_2 is added. If T has a terminal t adjacent to three vertices v_1, v_2 and v_3, a pair of Steiner points s_1 and s_2 is added. Steiner point s_1 is connected to s_2, v_1 and t. Steiner point s_2 is connected to s_1, v_2 and v_3.

The shortest Steiner tree for a given topology, $\mathcal{T}$, is called a *relatively minimal tree* (RMT) of $\mathcal{T}$. It always exists and is unique [12]. Its Steiner points may overlap with terminals and with other Steiner points because of zero length edges. Clearly, T^* is an RMT for its FST $\mathcal{T}^*$. Unfortunately, $\mathcal{T}^*$ is not known beforehand. Unless P=NP, the only feasible way to find it seems to be the enumeration of all FSTs, the determination of their RMTs and the selection of the shortest one as T^*.

The *bottleneck edge* between two terminals t_i and t_j of N is the longest edge on the path from t_i to t_j in the minimum spanning tree of N. The length of the bottleneck edge between t_i and t_i is referred to as the *bottleneck distance* and is denoted by $\beta(t_i, t_j)$. Bottleneck distances can be determined in polynomial time in a preprocessing phase. It is well-known [12] that no edge on a path between a pair of terminals t_i and t_j in the EMST can be longer than $\beta(t_i, t_j)$.

Consider a FST $\mathcal{T}$ for N. Let s be any of its Steiner points. Let r_i, r_j and r_k denote its adjacent vertices. When s is deleted, $\mathcal{T}$ breaks into three rooted binary trees or *branches* B_i, B_j and B_k rooted at r_i, r_j and r_k, respectively, see Fig. 2. Note that roots have degree 2 unless they are terminals. Furthermore the branches are *disjoint* in the sense that they do not share any terminals. The *depth* of a terminal in a branch is the number of edges separating it from the root of the branch.

Consider the FST $\mathcal{T}_i$ obtained by *splicing away* a non-terminal root r_i of B_i and connecting its adjacent vertices with each other. For notational convenience, let RMT(B_i) denote RMT($\mathcal{T}_i$). Similar splicing away can be applied to non-terminal roots of B_j and B_k.

A pair of disjoint branches B_i and B_j rooted at respectively r_i and r_j can be combined into a branch $B = B_i \oplus B_j$ by adding a new Steiner point r as a root of B adjacent to r_i and r_j.

A triplet of disjoint branches B_i, B_j and B_k rooted at respectively r_i, r_j and r_k can be combined into a FST $\mathcal{T} = B_i \oplus B_j \oplus B_k$ by adding a new Steiner point r adjacent to r_i, r_j and r_k.

A new algorithm based on the enumeration of branches rather then the enumeration of FSTs for N will be described in Section 5. Root splicing will be used to prune branches that cannot be in the FST $\mathcal{T}^*$ of the EMST T^* of N. Disjoint triplets of the remaining feasible branches (whose union spans N) will generate feasible FSTs of N, including $\mathcal{T}^*$.

3 Smith's algorithm (Smith)

Smith's algorithm [19] enumerates FSTs of N. RMTs are then determined for each such FST. The shortest RMT encountered is an EMST T^* of N. The

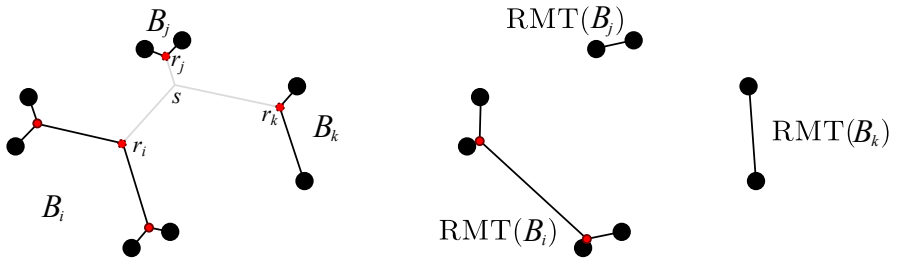


Fig. 2: Left: A FST that has been split in 3 branches B_i, B_j, and B_k by removing the Steiner point s. Right: The RMTs of each branch.

enumeration of FSTs is achieved by the following *expansion procedure*. Terminals are assumed to be in some fixed order $t_1, t_2, \ldots, t_n$. Assume that a FST $\mathcal{T}_k$ for $t_1, t_2, \ldots, t_k$, $3 \leq k < n$, is given. Note that only one FST exists for $k = 3$. Expand $\mathcal{T}_k$ into $2k-3$ FSTs, each with $k+1$ terminals, by inserting a new Steiner point s_{k-1} into every edge e of $\mathcal{T}_k$. In each expanded FST, s_{k-1} is adjacent to t_{k+1} and to the two vertices of e. This expansion process stops when $k = n$. It can be shown that every FST of N corresponds to exactly one unique sequence of such *expansions*. As described, the FSTs are generated in a breadth-first fashion. However, in order to obtain RMTs for all n terminals as quickly as possible, FSTs in [19] are generated in the best-first manner.

Given a FST $\mathcal{T}$ of N, arbitrary initial positions are assigned to its $n-2$ Steiner points. These positions are then recomputed iteratively by solving a system of $d(n-2)$ equations with $d(n-2)$ unknowns (corresponding to the locations of Steiner points). It can be shown [19] that the length of the tree reduces with each iteration and converges to RMT($\mathcal{T}$). The iterative procedure terminates if edges meet at Steiner points at angles within the interval $[2\pi/3 - \epsilon, 2\pi/3 + \epsilon]$ for an arbitrarily small constant $\epsilon > 0$. The reader is referred to [17] for the justification that a good approximation on the angles gives a good approximation of the length.

This iterative procedure can also be applied *during* the expansion process. Let $\mathcal{T}_k$ be a FST spanning k terminals, $3 \leq k < n$. Determine RMT($\mathcal{T}_k$). Suppose that it is not shorter than the shortest RMT of all n terminals encountered so far. The expansion of $\mathcal{T}_k$ can be stopped since any such expansion can only increase the lengths of RMTs of expanded FSTs. This algorithm will be referred to as Smith.

Two improvements for Smith have previously been suggested. Fampa and Anstreicher [8] used a conic formulation for finding a lower bound on a particular topology that eliminates the need to explicitly compute RMTs of children in the branch-and-bound tree. This was used to tighten lower bounds and guide the search. In the result-section we denote this method Smith-Fampa. Laarhoven and Anstreicher suggested a series of geometric criteria based on Voronoi-regions, bottleneck distances and lune-properties that could be used to discard partial topologies. Additionally they suggested exploring terminals starting with the

one furthest from the point set centroid and going toward the center. In the result-section we denote this method SMITH-LAARHOVEN.

4 Smith's modified algorithm (Smith*)

Smith's algorithm adds terminals in fixed but arbitrary order when expanding FSTs. If the low index terminals are close to each other, the corresponding FSTs will have short RMTs. As a consequence, the expansion procedure will rarely stop before $k = n$. It has been suggested to index the terminals by their non-decreasing distance to their centroid [14]. We suggest a different distance-based indexing. Terminals t_1, t_2 and t_3 have to maximize the sum of their pairwise distances. The terminal t_k, $k = 4, 5, \ldots, n$, is farthest away from $t_1, t_2, \ldots, t_{k-1}$.

To obtain a reasonable upper bound (needed to stop the expansion process), the FST $\mathcal{T}_M$ of the minimum spanning tree of N is determined. This is achieved by introducing $c-1$ Steiner points at terminals of degree c, $c \geq 1$. If $c \geq 3$, there are several ways of interconnecting these $c-1$ Steiner points with each other and with the terminal. In the current implementation, an arbitrary interconnection pattern is chosen. The length of $\mathrm{RMT}(\mathcal{T}_M)$ yields a good upper bound, denoted by UB, on the length of T^*. Given such good upper bound, the expansion of FSTs in the depth-first manner is no longer essential. Best-first expansion based on the lower bound together with the distance-based ordering of terminals results in the generation of fewer FSTs. This method is referred to as SMITH*.

The quality of lower bounds could be improved by using the conic formulations from the SMITH-FAMPA-method [8]. While smaller number of FSTs would be generated, the computation time increases significantly. Geometric criteria based on the lune-properties combined with the bottleneck distances were also used to speed up the expansion process by eliminating non-optimal FSTs [14]. However, the extra time spent on geometric computations makes the improvements minimal when compared to the distance-based sorting of terminals. None of these improvements are therefore included in the SMITH* algorithm.

5 Branch enumeration algorithm (Branch)

The approach of GeoSteiner [22] seems at first sight to be applicable to the ESTP in $\mathbb{R}^d$, $d \geq 3$. GeoSteiner has two phases. In the *generation phase*, full Steiner trees for all FSTs of all subsets of terminals are generated. Naturally, full Steiner trees that cannot be in T^* are pruned away. When FSTs are generated across different subsets, there are large groups of them that are very similar. The power of GeoSteiner rests partly in its ability to generate full Steiner trees with similar FSTs in a common pass and partly in its ability to prune away partially constructed FSTs. In the *concatenation phase*, GeoSteiner selects a subset of not pruned full Steiner trees that span all terminals and has the minimum total length. This problem can be formulated as the minimum spanning tree problem in a hypergraph. While this problem is NP-hard, a branch-and-cut algorithm

seems to be quite efficient to solve problem instances involving full Steiner trees not pruned away during the generation phase.

Unfortunately, this is not the case. First of all, the generation of full Steiner trees for *all* subsets of terminals in $\mathbb{R}^d$, $d \geq 3$, is much more difficult than in $\mathbb{R}^2$. The determination of a full Steiner tree of a given FST with more than 3 terminals requires solving high-degree polynomials [19]. As a consequence, iterative numerical approaches are the only possibility. Such numerical approaches seem to block the generation of full Steiner trees across various subsets of terminals (unlike the $\mathbb{R}^2$ case). Finally, the geometrical non-optimality tests seem to be much weaker than in $\mathbb{R}^2$.

While the generation of full Steiner trees is very challenging for $d \geq 3$, the concatenation phase could be applied without any significant modifications. However, the lack of efficient pruning tests in the generation phase permits a huge number of non-optimal RMTs to survive causing the branch-and-cut concatenation algorithm to choke. These congestion problems become more and more serious as d grows.

The *branch enumeration algorithm* that is proposed here can be seen as a compromise between the approach used by GeoSteiner (generation of full Steiner trees of *all* subsets of terminals) and the numerical approach of Smith [19] described in Section 3 (generation of RMTs for FSTs with n terminals). Rather than enumerating RMTs for all FSTs for N, branches involving subsets of terminals are generated. Consider a FST $\mathcal{T}$ of N. As already mentioned in Section 2, removing a Steiner point breaks $\mathcal{T}$ into 3 branches. As will be seen in Subsection 5.3, every FST with n terminals contains a Steiner point whose removal creates three branches with at most $\lfloor \frac{n}{2} \rfloor$ terminals. This significantly reduces the number of branches that need to be generated.

The branch enumeration algorithm consists of three phases (see Algorithm 1). The *preprocessing* phase computes EMSTs for subsets with up to 8 terminals. These EMSTs are used in subsequent phases to prune away non-optimal branches and RMSTs. The second phase *generates* branches containing up to $\lfloor \frac{n}{2} \rfloor$ terminals. EMSTs obtained in the preprocessing phase together with RMTs of root spliced branches are used to prune away branches that cannot be in T^*. The third phase generates FSTs with n terminals by *concatenating* triplets of disjoint branches. The iterative procedure described in Section 3 is then used to determine their RMTs. The shortest RMT encountered is T^*.

5.1 Preprocessing phase

The maximum size κ of subsets of terminals for which ESMTs are determined in the preprocessing phase has a large effect on the number of branches that will be pruned away during their generation described in Subsection 5.2. As branches of up to size $\lceil \frac{n}{2} \rceil$ are generated, setting $\kappa = \lfloor \frac{n}{2} \rfloor$ ensures that for any branch it is possible to find a disjoint preprocessed tree. However, choosing $\kappa > 8$ makes the preprocessing phase computationally expensive. ESMTs are therefore generated using SMITH* for all terminal subsets of size $1, 2, 3, \ldots, \kappa$, stored in $\mathcal{P}$, and finally sorted by non-increasing lengths.

Algorithm 1 Branch enumeration algorithm

Input: Set N of n terminals in $\mathbb{R}^d$.
Output: ESMT T^* of N.

1: **Preprocess:** Determine EMSTs of all subsets of terminals of size up to $\min\{8, \lceil \frac{n}{2} \rceil\}$, store them in $\mathcal{P}$, and order them by non-increasing length.
2: **Generate:** Enumerate sets $\mathcal{B}_k$ of feasible branches with k terminals, $k = 1 \dots \lfloor \frac{n}{2} \rfloor$ (pruning away non-optimal branches).
3: **Concatenate:** Combine triplets of disjoint feasible branches spanning N. Compute corresponding RMTs and let T^* be a shortest one.

5.2 Generating branches

Branches are generated by increasing number of spanned terminals. Branches of size k, $k = 1, 2, \dots \lfloor \frac{n}{2} \rfloor$, are stored in a set $\mathcal{B}_k$. It will be shown in Subsection 5.3 that only branches spanning up to $\lfloor \frac{n}{2} \rfloor$ need to be generated. A single terminal is a branch containing 1 terminal and is therefore stored in $\mathcal{B}_1$. The branches in $\mathcal{B}_k$, $2 \leq k \leq \lfloor \frac{n}{2} \rfloor$, are generated by combining branches in $\mathcal{B}_l$ with branches in $\mathcal{B}_{k-l}$, $l = 1 \dots \lfloor k/2 \rfloor$. When $l = k - l$, care is taken to avoid generation of duplicate branches.

There are several criteria that can be used to reject a branch $B_k = B_l \oplus B_{k-l}$. First of all, B_l and B_{k-l} must be disjoint. This can be efficiently verified by bitwise AND on bit-strings representing subsets of terminals. Similarly, constructing the bitstring of B_k can be efficiently done using bitwise OR.

Assume next that t_i is a terminal in B_l while t_j is a terminal in B_{k-l}. Furthermore, assume that the depth of t_i in B_l is d_i and the depth of t_j in B_{k-l} is d_j. Let $d = d_i + d_j + 2$. Suppose that $|t_i t_j|/d > \beta(t_i, t_j)$ (see Fig. 3). Then B_k cannot be a part of $\mathcal{T}^*$ as it would contain an edge between t_i and t_j longer than the bottleneck distance between t_i and t_j.

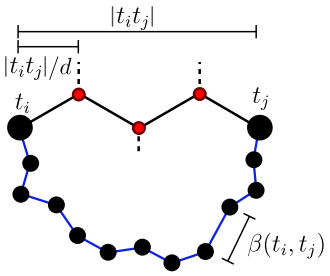


Fig. 3: A branch B_k with terminals t_i and t_j being $d = 4$ edges apart cannot be a part of $\mathcal{T}^*$ if any of these d edges are longer than the bottleneck distance $\beta(t_i, t_j)$. This will certainly be the case if $|t_i t_j|/d > \beta(t_i, t_j)$.

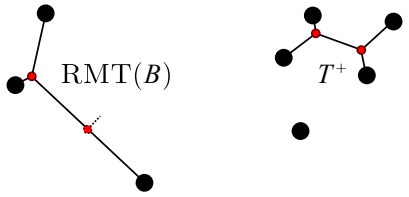


Fig. 4: The length of RMT(B) added to the EMST of any subset of terminals disjoint from $\mathcal{T}$ is a lower bound on $|T^*|$

Finally, suppose that RMT(B_k) spans the set of terminals N_k and the EMST T^+ for some non-empty subset of $N \setminus N_k$ satisfies

$$|\mathrm{RMT}(B_k)| + |T^+| \geq UB$$

where UB is an upper bound on the length of T^* (see Section 4), then B_k cannot be a part of $\mathcal{T}^*$. This is an immediate consequence of the following lemma and the fact that $UB \geq |T^*|$.

Lemma 1 *Let B_k be a branch of $\mathcal{T}^*$. Let N_k denote the terminals of B_k. Let T^+ be the EMST of a non-empty subset of $N \setminus N_k$. Then $|T^*| \geq |\mathrm{RMT}(B_k)| + |T^+|$.*

Proof. If $|N_k| = 1$ then $|\mathrm{RMT}(B_k)| = 0$ and the inequality trivially holds. Assume that $|N_k| \geq 2$. Let $B_k = B_i \oplus B_j$ with r_k, r_i and r_j being the roots of B_k, B_i and B_j respectively. Let T_k^*, T_i^* and T_j^* be the parts of T^* corresponding to B_k, B_i and B_j respectively. Then

$$|T_k^*| = |T_i^*| + |r_i r_k| + |r_j r_k| + |T_j^*| \geq |T_i^*| + |r_i r_j| + |T_j^*| \geq |\mathrm{RMT}(B_k)|$$

where the first inequality is due to the triangle inequality and the second inequality is due to the fact that RMT(B_k) is the RMT of the FST obtained by splicing away r_k from B_k. Hence,

$$|T^*| = |T_k^*| + |T^* \setminus T_k^*| \geq |\mathrm{RMT}(B_k)| + |T^+|$$

since $T^* \setminus T_k^*$ is a tree spanning $N \setminus N_k$. □

5.3 Concatenation of branches

Consider a FST $\mathcal{T}$ of a set N of n terminals, $n \geq 3$. Let s be any of its $n-2$ Steiner points. When s (and its three incident edges) are removed, $\mathcal{T}$ splits into three branches B_i, B_j and B_k with respectively n_i, n_j, and n_k terminals, $n = n_i + n_j + n_k$. Assume that $n_i \geq n_j \geq n_k$.

Lemma 2 *$\mathcal{T}$ has a splitting Steiner point s such that $n_i \leq n_j + n_k$.*

Algorithm 2 Generate branches containing up to $\lceil \frac{n}{2} \rceil$ terminals

Input: Set N of n terminals in $\mathbb{R}^d$.
Input: Sorted list $\mathcal{P}$ of EMSTs for small subsets of N.
Input: An upper bound UB on $|T^*|$
Output: $\mathcal{B}_k, k = 1 \ldots \lceil \frac{n}{2} \rceil$

$\mathcal{B}_1 \leftarrow N$
for $k = 2$ **to** $\lceil \frac{n}{2} \rceil$ **do**
  $\mathcal{B}_k \leftarrow \emptyset$
  for $l = 1$ **to** $\lfloor \frac{k}{2} \rfloor$ **do**
    for all $B_l \in \mathcal{B}_l$ **do**
      for all $B_{k-l} \in \mathcal{B}_{k-l}$ **do**
        if B_l and B_{k-l} have a common terminal **then**
          $B_l \oplus B_{k-l}$ cannot be a part of $\mathcal{T}^*$
        end if
        for all pairs of terminals (t_i, t_j), $t_i \in B_l, t_j \in B_{k-1}$ **do**
          Let d_i and d_j denote the depth of t_i and t_j in respectively B_l and B_{k-l}.
          if $\frac{|t_i t_j|}{d_i + d_j + 2} > \beta(t_i, t_j)$ **then**
            $B_l \oplus B_{k-l}$ cannot be a part of $\mathcal{T}^*$
          end if
        end for
        $B_k \leftarrow B_l \oplus B_{k-l}$
        Find the longest ESMT $T^+ \in \mathcal{P}$ disjoint from B_k
        if $|\mathrm{RMT}(B_k)| + |T^+| \geq UB$ **then**
          B_k cannot be a part of $\mathcal{T}^*$
        end if
        Add B_k to $\mathcal{B}_k$.
      end for
    end for
  end for
end for
return $\mathcal{B}_k, k = 1 \ldots \lceil \frac{n}{2} \rceil$

Proof. Assume that $n_i > n_j + n_k$ for every Steiner points in $\mathcal{T}$. Pick a Steiner point s minimizing n_i. Let $s' \in B_i$ denote the Steiner point adjacent to s in $\mathcal{T}$. It exists since $n_i \geq 2$. Let n'_i, n'_j, and n'_k denote the number of terminals obtained by splitting $\mathcal{T}$ at s'. Then $n'_i = n_j + n_k$, $n'_j = x$ for some x, $0 < x < n_i$, and $n'_k = n_i - x$. Hence, n'_i, n'_j, and n'_k are all less than n_i, contradicting the choice of s. □

A split of any FST with $n_i \geq n_j \geq n_k$, $n = n_i + n_j + n_k$ and $n_i \leq n_j + n_k$ is called a *canonical split*.

Lemma 3 *$\lceil \frac{n}{3} \rceil \leq n_i \leq \lfloor \frac{n}{2} \rfloor$ in a canonical split of any FST with n terminals.*

Proof. To obtain the first inequality, observe that $n = n_i + n_j + n_k \Rightarrow n \leq 3n_i \Rightarrow \lceil \frac{n}{3} \rceil \leq n_i$. To obtain the second inequality, observe that $n_i \leq n_j + n_k = n - n_i \Rightarrow 2n_i \leq n \Rightarrow n_i \leq \lfloor \frac{n}{2} \rfloor$. □

Lemma 4 *An FST $\mathcal{T}$ with n terminals has exactly one canonical split unless n is even and $n_i = n/2$ in which case $\mathcal{T}$ has two canonical splits at adjacent Steiner points.*

Proof. Let B_i, B_j and B_k denote three branches of the canonical split at a Steiner point s. Consider another Steiner point $s' \in \mathcal{T}$. Assume first that $s' \in B_j$. Let $n'_i \geq n'_j \geq n'_k$ denote the number of terminals in the branches of this split at s'. Hence, $n'_i \geq n_i + n_k$ and $n'_j + n'_k \leq n_j$. Now $n'_i \geq n_i + n_k > n_j \geq n'_j + n'_k$ implies that the split is not canonical. Similar argument applies if $s' \in B_k$. Assume finally that $s' \in B_i$. Hence, $n'_i \geq n_j + n_k$. Now $n'_i \geq n_j + n_k \geq n_i \geq n'_j + n'_k$ implies that this split is canonical iff $n'_i = n'_j + n'_k$. This can happen if and only if $n'_i = n_i = n/2$. Hence, n has to be even and s and s' must be adjacent in $\mathcal{T}$. □

It is therefore only necessary to concatenate triplets of disjoint branches (B_i, B_j, B_k) with $n = n_i + n_j + n_k$, and whose individual sizes satisfy $\lceil \frac{n}{3} \rceil \leq n_i \leq \lfloor \frac{n}{2} \rfloor$, $n_i \leq n_j + n_k$ and $n_i \geq n_j \geq n_k$. See Algorithm 3 for a detailed description of the concatenation of triplets of branches.

Algorithm 3 Concatenate triples of branches to obtain EMST T^*

Input: Sets of branches $\mathcal{B}_p, p = 1 \ldots \lfloor \frac{n}{2} \rfloor$
Input: An upper bound UB on $|T^*|$
Output: T^*

```
|T*| ← UB
for i = ⌊n/2⌋ to ⌈n/3⌉ do
  for j = min{i, n − i − 1} to ⌈(n−i)/2⌉ do
    k ← n − i − j
    for all disjoint triples of branches B_i, B_j, B_k with B_i ∈ B_i, B_j ∈ B_j and B_k ∈ B_k do
      T ← RMT(B_i ⊕ B_j ⊕ B_k).
      if |T| < |T*| then
        T* ← T
      end if
    end for
  end for
end for
return T*
```

6 Computational experiments

Smith's algorithm (Smith) described in Section 3, its modified version (Smith*) described in Section 4 and the branch enumeration algorithm (Branch) described in Section 5, were implemented in C++ and run on an Intel Xeon X5550 2.67GHz CPU. All runs were stopped after 24 hours if they had not terminated or if they consumed more than 10GB of memory.

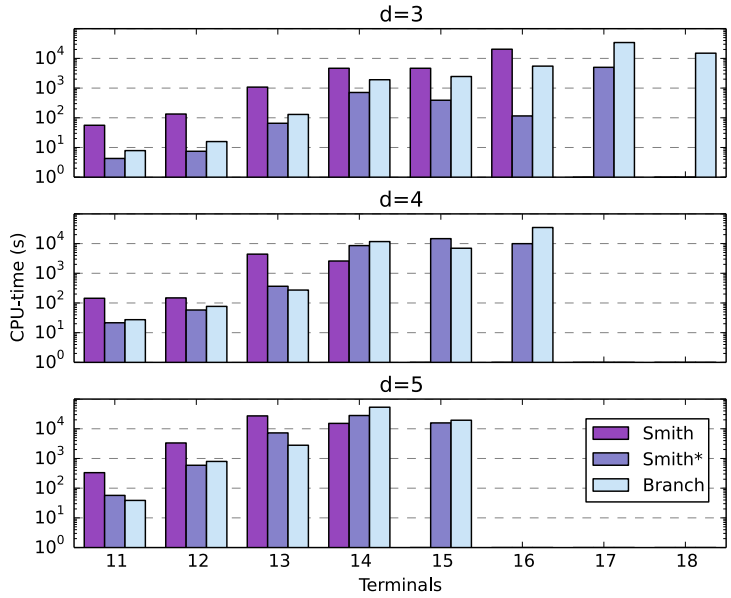


Fig. 5: Average CPU-times for SMITH, SMITH*, and BRANCH on instances from the Carioca set. Each column represents an average for each method run on five problem instances with a specific set of terminals (n) and dimensions (d). All values can be found in the appendix tables.

Figure 5 shows average CPU-times for the 40 Carioca instances for each $d = 3, 4, 5$ from the OR-library [2] (Carioca set). Tables 3 to 5 in the appendix shows the CPU times in more details as well as reported lengths of ESMTs and the total number of RMTs generated by each of the three exact algorithms. SMITH* and BRANCH consistently outperform the original SMITH algorithm[3]. In a few instances BRANCH is faster than SMITH* and for some large instances it terminates using the prespecified time and memory while SMITH* does not.

All three algorithms construct RMTs for various subsets of terminals in order to obtain reasonable lower bounds permitting pruning of non-optimal configurations as early as possible. In particular, BRANCH computes many such RMTs with few terminals in its preprocessing and generation phases. Note, that the very straightforward way that preprocessing is currently done, the RMT of a particular topology may be computed multiple times. Additionally, the RMT of two distinct branches may also be equivalent. Using appropriate look-up tables to avoid redundant RMT computations would no doubt speed up BRANCH significantly.

[3] For $n = 14, d = 4, 5$ the average of SMITH is lower because only one fast run terminated while SMITH* and BRANCH succesfully solved several instances.

Level	Nothing	Overlap	Bottleneck	Lowerbound	Preprocessing
8 random terminals in unit cube					
2	36	28	21	21	21
3	288	168	121	121	29
4	2 970	1 050	719	719	64
8	1 498 176	21 000	9 596	2 161	743
10 random terminals in unit cube					
2	55	45	32	32	32
3	550	360	246	246	50
4	7 040	3 150	2 034	2 034	77
5	100 650	26 460	15 810	15 810	108
10	12 560 296 100	3 508 785	1 183 316	65 609	7 274

Table 1: The effect of different pruning criteria on the number of generated branches/full topologies in BRANCH algorithm. The level of a branch is the number of terminals it contains. 'Nothing' is a theoretical approach where all combinations of branches are used in enumeration, including those that overlap. The Overlap-pruning discards all branches where the same terminal appears twice. Bottleneck-pruning additionally uses the bottleneck criterion to prune. Finally, Lowerbound and Preprocessing also calculates lower bounds and compares them with the upper bound even in the branch generation phase. The Preprocessing pruning strengthens the lower bound of each branch by adding the lengths of disjoint optimal solutions from the preprocessing phase.

The main reason why SMITH* and SMITH were terminated was because they consumed all memory of the machine. Even when switching to a depth-first-search, the size of the branch-and-bound tree grows extremely large. BRANCH never used more than 1GB of memory, but for large instances it was stopped because it exceeded the computation time.

Table 1 illustrates the strength of the different pruning criteria used in BRANCH. The 'Nothing' column indicates the number of branches that are explored if all pruning is disabled. The 'Preprocessing' column indicates the number of branches that are explored when all pruning methods are enabled. Using RMTs of branches as a lower bound is essential to be able to prune in the final concatenation phase (bottom rows in the table) but it is only when combined with tighter lower bounds obtained from the preprocessing stage that branches can be pruned while being generated.

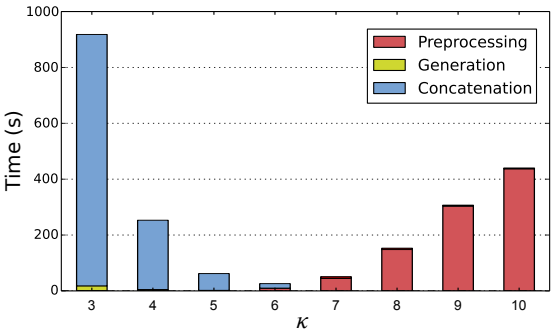
Table 2 compares our implementation with the three previous algorithms that solve the d-dimensional geometric Steiner tree problem. The values for SMITH-FAMPA and SMITH-LAARHOVEN are taken from [14]. For most instances both SMITH* and BRANCH are superior to the previously suggested algorithms.

Figure 6 illustrates the effects of changing the number of terminal subsets that are preprocessed. The BRANCH algorithm was run on three point sets with $d = 3$ and $n = 12, 13, 14$ and different values of κ. In all cases, setting $\kappa = \lceil n/2 \rceil$ gives the lowest CPU time. Lower values of κ result in very fast preprocessing,

	n	10	10	10
	d	3	4	5
SMITH	Nodes factor	1	1	1
	Time factor	1	1	1
SMITH-FAMPA	Nodes factor	33.4	26.9	50.8
	Time factor	3.1	2.8	4.4
SMITH-LAARHOVEN	Nodes factor	3.5	6.1	3.7
	Time factor	3.6	6.4	3.8
SMITH*	Nodes factor	19.9	17.5	6.6
	Time factor	17.9	17.2	8.7
BRANCHENUMERATION	Nodes factor	12.0	6.5	5.4
	Time factor	6.7	4.1	3.5

Table 2: Performance of known Steiner tree algorithms. Numbers indicate the improvement factor in nodes explored and CPU-time over Smith's original algorithm.

but weak lower bounds that makes the concatenation phase explode. Larger values of κ make the lower bound extremely strong, but forces us to spend a lot of time on preprocessing. As a lot of redundant computations are performed in the preprocessing phase, this indicates that speeding up preprocessing and increasing κ might substantially improve the BRANCH algorithm.



(a) CPU times for $n = 12$ and varying values of κ broken down by algorithm phase.

$n\backslash\kappa$	5	6	7	8
12	62	25	50	151
13	881	201	166	566
14	1734	467	320	1076

(b) CPU times for different values of n.

Fig. 6: The CPU time in seconds for 3D instances from the carioca set where preprocessing finds EMSTs for subsets up to different values of κ. No matter what n is, setting $\kappa = \lceil n/2 \rceil$ gives the lowest CPU time.

7 Conclusions

Two new exact algorithms for solving the Euclidean Steiner tree problem in dimensions $d \geq 3$ have been proposed. One is an extension of the seminal algorithm by Smith while the other constructs branches in a bottom-up fashion. Computational studies show that both methods are faster than the original but the improved version of Smith's algorithm is, in general, the fastest.

It is worth noting that the use of branches in the branch enumeration method provides more specific information about partial solutions than the expanding topologies in Smith's algorithm. This property has the potential to expose different and possibly stronger methods for fathoming partial solutions early. For instance, Laarhoven and Anstreicher [14] present a combination of lune-properties and bottleneck distances that could directly be employed and be used to prune in the very first steps of the enumeration. In general our findings suggest a critical need to identify tighter lower bounds and geometric criteria for excluding partial solutions in higher dimensions.

References

1. E. Althaus, J. Kupilas, and R. Naujoks. On the low-dimensional Steiner minimum tree problem in Hamming metric. *Theoretical Computer Science*, 505:2–10, 2013.
2. J. E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.
3. K. Bopp. *Üeber das kürzeste Verbindungssystem zwischen vier Punkten.* PhD thesis, Universität Göttingen, 1879.
4. M. Brazil, R. L. Graham, D. A. Thomas, and M. Zachariasen. On the history of the Euclidean Steiner tree problem. *Archive for History of Exact Sciences*, 68:327–354, 2014.
5. M. Brazil, D. A. Thomas, B. K. Nielsen, P. Winter, C. Wulff-Nilsen, and M. Zachariasen. A novel approach to phylogenetic trees: d-dimensional geometric Steiner trees. *Networks*, 53(2):104–111, 2009.
6. L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis: Models and estimation procedures. *Evolution*, 21:550–570, 1967.
7. D. Cieslik. *Shortest Connectivity — Introduction with Applications in Phylogeny*, volume 17 of *Combinatorial Optimization*. Springer, New York, 2004.
8. M. Fampa and K. M. Anstreicher. An improved algorithm for computing Steiner minimal trees in Euclidean d-space. *Discrete Optimization*, 5:530–540, 2008.
9. M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977.
10. E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.
11. T. Huang and E. F. Y. Young. Obsteiner: An exact algorithm for the construction of rectilinear Steiner minimum trees in the presence of complex rectilinear obstacles. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 32(6):882–893, 2013.
12. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* Annals of Discrete Mathematics 53. Elsevier Science Publishers, Netherlands, 1992.
13. V. Jarník and M. Kössler. O minimálních grafeth obeahujících n daných bodú. *Cas. Pest. Mat. a Fys.*, 63:223–235, 1934.
14. J. W. Van Laarhoven and K. M. Anstreicher. Geometric conditions for Euclidean Steiner trees in $\mathbb{R}^d$. *Computational Geometry*, 46:520–531, 2013.
15. B. K. Nielsen, P. Winter, and M. Zachariasen. An exact algorithm for the uniformly-oriented Steiner tree problem. In *Proceedings of the 10th European Symposium on Algorithms, Lecture Notes in Computer Science*, volume 2461, pages 760–772. Springer, 2002.
16. J. H. Rubinstein, D. A. Thomas, and N. C. Wormald. Steiner trees for terminals constrained to curves. *SIAM Journal on Discrete Mathematics*, 10(1):1–17, 1997.
17. J. H. Rubinstein, J. F. Weng, and N. Wormald. Approximations and lower bounds for the length of minimal Euclidean Steiner trees. *Journal of Global Optimization*, 35:573–592, 2006.
18. J. M. Smith, Y. Jang, and M. K. Kim. Steiner minimal trees, twist angles, and the protein folding problem. *PROTEINS: Structure, Function, and Bioinformatics*, 66(4):889–902, 2007.
19. W. D. Smith. How to find Steiner minimal trees in Euclidean d-space. *Algorithmica*, 7(2/3):137–177, 1992.

20. C. Stanton and J. M. Smith. Steiner trees and 3-d macromolecular conformation. *INFORMS Journal on Computing*, 16:470–485, 2004.
21. D. M. Warme. A New Exact Algorithm for Rectilinear Steiner Minimal Trees. Technical report, System Simulation Solutions, Inc., Alexandria, VA 22314, USA, 1997.
22. D. M. Warme, P. Winter, and M. Zachariasen. Exact solutions to large-scale plane Steiner tree problems. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 979–980, 1999.
23. D. M. Warme, P. Winter, and M. Zachariasen. GeoSteiner 3.1. Department of Computer Science, University of Copenhagen (DIKU), http://www.diku.dk/geosteiner/, 2001.
24. P. Winter. An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15:323–345, 1985.
25. P. Winter and M. Zachariasen. Euclidean Steiner minimum trees: An improved exact algorithm. *Networks*, 30:149–166, 1997.
26. M. Zachariasen and A. Rohe. Rectilinear group Steiner trees and applications in VLSI design. *Mathematical Programming*, 94:407–433, 2003.
27. M. Zachariasen and P. Winter. Obstacle-avoiding Euclidean Steiner trees in the plane: An exact algorithm. In *Workshop on Algorithm Engineering and Experimentation (ALENEX), Lecture Notes in Computer Science 1619*, pages 282–295. Springer, 1999.

A CPU-times and RMTs

		Smith		Smith*		Branch	
n	$\|EMST\|$	#RMT$_n$	Time	#RMT$_n$	Time	#RMT$_n$	Time
11	2.541	266461	12.45	35109	1.70	12220	4.22
11	3.546	381226	17.23	21112	0.95	21862	4.71
11	3.529	2666177	79.95	158970	8.27	148845	9.09
11	3.309	5009378	162.29	197303	9.75	265267	17.13
11	2.982	241340	7.88	13775	0.65	6542	4.11
12	3.875	860875	43.02	156264	7.85	475670	22.86
12	3.263	339825	10.53	246597	10.77		
12	3.321	3936545	105.01	24732	1.26	166962	9.85
12	3.693	8125279	375.75	86517	4.31	45448	9.97
12	3.661	4365068	134.49	252754	13.07	302071	20.61
13	3.564	1961970	102.62	71398	3.87	73115	78.29
13	3.724	54616518	2991.85	3187216	181.23	5669553	314.43
13	3.626	20353108	1134.69	1821346	99.37	660947	98.19
13	3.565	14464700	682.77	230368	13.08	284987	83.95
13	4.367	9092731	478.79	527653	29.34	133436	73.21
14	3.473	12966341	616.46	1665175	87.64	2637107	264.94
14	3.892	107259083	4089.41	2314918	130.17	2687391	283.92
14	3.860	42394933	1502.56	1520088	94.81	1515837	233.63
14	3.605	11348798	519.91	9123679	492.08	1494147	215.48
14	3.230	5406607	280.13	2487723	149.13	954247	189.22
15	4.665			47707291	1998.43	2130561	2446.88
15	4.055			10503611	452.01	2147076	2188.43
15	3.770	67485229	3732.51	13968282	821.38	1228642	1734.18
15	4.103			3413750	183.09	1361275	1385.96
15	4.139	94372427	5567.39	2940214	103.08	1586122	1805.75
16	4.179					13948310	5160.67
16	3.786	759350882	40671.94	1584348	108.33	5949149	3151.75
16	3.552	1221022	76.94	157898	8.75	212651	2830.75
16	4.268					53914726	11954.47
16	4.112			4115230	229.24	174497	4357.57
17	4.512					43870843	18104.04
17							
17	4.386			90512048	5944.10	40448925	16745.82
17	4.591			14748128	6422.63	104393953	33863.29
17	4.611			45414606	2651.16	27710754	15807.81
18	4.308					10114736	14975.10

Table 3: Results for 40 $\mathbb{R}^3$ instances from the Carioca set (OR-library). $|EMST|$ is the length of the ESMT, #RMT$_n$ is the number of RMTs spanning *all* n terminals, CPU-times are given in seconds. Runs were terminated if not completed within 24 hours or if more than 10GB of memory was consumed.

		Smith		Smith*		Branch	
n	$\|EMST\|$	#RMT$_n$	Time	#RMT$_n$	Time	#RMT$_n$	Time
11	4.196	2653541	106.21	84562	4.53	129898	11.96
11	4.572	1577114	60.15	238176	12.41	137842	12.73
11	4.400	6866738	269.13	933794	43.62	1066283	42.54
11	4.467	2179711	80.26	249430	12.61	372108	22.56
11	3.810	5401750	204.54	651595	34.65	742649	47.12
12	4.499	1358219	77.01	868184	45.58	718526	33.83
12	4.108	2844281	96.36	163953	8.20	485954	30.64
12	4.457	6394703	361.40	689008	40.31	1951142	78.97
12	3.956	2941309	106.91	410684	22.36	1664688	106.03
12	4.878	2927978	99.89	3107617	173.44	2539988	134.24
13	4.608	4217958	261.24	5952932	360.57	573853	167.14
13	4.954	10816253	667.42	2000521	118.75	1448689	185.54
13	4.682	7607968	458.00	6674205	400.90	926393	137.94
13	4.370	138575435	7650.75	1732209	97.52	689137	127.59
13	5.714	219847871	13005.2	14141777	848.55	13504454	743.19
14	4.695	30353322	1324.98	10853721	661.23	8911065	690.57
14	5.855			552626254	32571.56	698994929	45412.74
14	5.380			21074819	1298.95	101160633	6358.15
14	5.150			108992640	6501.49	69683339	4679.52
14	4.724	69179844	3851.82	28554566	1764.20	18667881	1276.67
15	6.368			193164286	11463.78	24916803	4768.98
15	5.450			50261733	2746.36	15900685	3876.42
15	5.481			687329526	41045.07	59948283	9698.79
15	4.899					68498588	9414.35
15	6.099			75309029	3427.54	33923065	7091.67
16							
16	6.001			99524746	6506.57	66656972	19353.38
16	5.473			21102561	1544.40	60277544	15059.90
16	6.063					384038792	69976.54
16	6.057			392290942	21782.06		

Table 4: Results for 40 $\mathbb{R}^4$ instances from the Carioca set (OR-library). $|EMST|$ is the length of the ESMT, #RMT$_n$ is the number of RMTs spanning *all* n terminals, CPU-times are given in seconds. Runs were terminated if not completed within 24 hours or if more than 10GB of memory was consumed.

		Smith		Smith*		Branch	
n	$\|EMST\|$	#RMT$_n$	Time	#RMT$_n$	Time	#RMT$_n$	Time
11	6.041	22061862	798.89	1577566	88.65	1707972	93.84
11	5.296	2192475	102.07	487525	26.64	145065	14.43
11	5.469	7015527	283.44	183401	10.32	349738	16.26
11	5.669	1824529	65.00	180453	10.30	113892	12.79
11	5.409	9175378	414.81	2558029	149.60	1113385	56.13
12	6.172	16493717	707.46	6253602	395.81	19205551	888.96
12	6.367	117516757	7541.55	17115895	1068.84	36061327	1750.61
12	5.230	41428379	2607.68	13446000	828.73	7925996	332.07
12	5.788	88731048	5367.68	10167397	635.79	18831564	968.08
12	5.148	9519295	414.74	384853	22.00	480960	34.68
13	5.913	500757535	32004.39	83968773	5243.99	58216112	3424.88
13	5.673	809127726	46876.66	64076185	4156.59	76465689	4063.95
13	6.675	320123483	21188.48	265897472	17061.29	54068880	3230.20
13	6.050	420308274	27330.79	131073163	8451.16	47090941	2688.19
13	5.554	125795826	8274.69	20432479	1377.52	7651268	535.88
14	6.855			592423734	39638.54	185109707	128554.4
14	7.009	74717871	15188.03	147314600	9759.52	135459705	9413.68
14	5.823			94747344	6556.67	369239423	23361.14
14	7.014					935384901	66659.38
14	6.578			844179767	55843.35	539736839	38110.09
15	6.389					211366906	25707.46
15							
15	6.941					254261617	28765.72
15	5.588			242292152	15752.26	24793442	7022.37
15	7.015					115045511	16142.57

Table 5: Results for 40 $\mathbb{R}^5$ instances from the Carioca set (OR-library). $|EMST|$ is the length of the ESMT, #RMT$_n$ is the number of RMTs spanning *all* n terminals, CPU-times are given in seconds. Runs were terminated if not completed within 24 hours or if more than 10GB of memory was consumed.