Steiner Tree Heuristic in the Euclidean d-Space Using Bottleneck Distances

Stephan S. Lorenzen and Pawel Winter(✉)

Department of Computer Science, University of Copenhagen,
Universitetsparken 5, 2100 Copenhagen O, Denmark
stephan.lorenzen@gmail.com, pawel@di.ku.dk

Abstract. Some of the most efficient heuristics for the Euclidean Steiner minimal tree problem in the d-dimensional space, $d \geq 2$, use Delaunay tessellations and minimum spanning trees to determine small subsets of geometrically close terminals. Their low-cost Steiner trees are determined and concatenated in a greedy fashion to obtain a low cost tree spanning all terminals. The weakness of this approach is that obtained solutions are topologically related to minimum spanning trees. To avoid this and to obtain even better solutions, bottleneck distances are utilized to determine good subsets of terminals without being constrained by the topologies of minimum spanning trees. Computational experiments show a significant solution quality improvement.

Keywords: Steiner minimal tree · d-dimensional Euclidean space · Heuristic · Bottleneck distances

1 Introduction

Given a set of points $N = \{t_1, t_2, ..., t_n\}$ in the Euclidean d-dimensional space $\mathcal{R}^d$, $d \geq 2$, the *Euclidean Steiner minimal tree* (ESMT) *problem* asks for a shortest connected network $T = (V, E)$, where $N \subseteq V$. The points of N are called *terminals* while the points of $S = V \setminus N$ are called *Steiner points*. The length $|uv|$ of an edge $(u, v) \in E$ is the Euclidean distance between u and v. The length $|T|$ of T is the sum of the lengths of the edges in T. Clearly, T must be a tree. It is called the *Euclidean Steiner minimal tree* and it is denoted by SMT(N). The ESMT problem has originally been suggested by Fermat in the 17-th century. Since then many variants with important applications in the design of transportation and communication networks and in the VLSI design have been investigated. While the ESMT problem is one of the oldest optimization problems, it remains an active research area due to its difficulty, many open questions and challenging applications. The reader is referred to [4] for the fascinating history of the ESMT problem.

The ESMT problem is NP-hard [5]. It has been studied extensively in $\mathcal{R}^2$ and a good exact method for solving problem instances with up to 50.000 terminals is available [13,23]. However, no analytical method can exist for $d \geq 3$ [1]. Furthermore, no numerical approximation seems to be able to solve instances with more

© Springer International Publishing Switzerland 2016
A.V. Goldberg and A.S. Kulikov (Eds.): SEA 2016, LNCS 9685, pp. 217–230, 2016.
DOI: 10.1007/978-3-319-38851-9_15

than 15–20 terminals [7–9,14,21]. It is therefore essential to develop good quality heuristics for the ESMT problem in $\mathcal{R}^d$, $d \geq 3$. Several such heuristics have been proposed in the literature [10,15,17,22]. In particular, [17] builds on the R^2-heuristics [2,20]. They use **D**elaunay tessellations and **M**inimum spanning trees and are therefore referred to as DM-heuristics. The randomized heuristic suggested in [15] also uses Delaunay tessellations. It randomly selects a predefined portion of simplices in the Delaunay tessellation. It adds a centroid for each selected simplex as a Steiner point. It then computes the minimum spanning tree for the terminals and added Steiner points. Local improvements of various kinds are then applied to improve the quality of the solution. It obtains good solutions in R^2. Only instances with $n \leq 100$ are tested and CPU times are around 40 sec. for $n = 100$. The randomized heuristic is also tested for very small problem instances ($n = 10, d = 3, 4, 5$) and for specially structured "sausage" instances ($n < 100, d = 3$). It can be expected that the CPU times increase significantly as d grows since the number of simplices in Delaunay tessellations then grows exponentially.

The goal of this paper is to improve the DM-heuristic in a deterministic manner so that the minimum spanning tree bondage is avoided and good quality solutions for large problem instances can be obtained. Some basic definitions and a resume of the DM-heuristic is given in the remainder of this section. Section 2 discusses how *bottleneck distances* can be utilized to improve the solutions produced by the DM-heuristic. The new heuristic is referred to as the DB-heuristic as it uses both **D**elaunay tessellations and **B**ottleneck distances. Section 3 describes data structures used for the determination of bottleneck distances while Sect. 4 gives computational results, including comparisons with the DM-heuristic.

1.1 Definitions

SMT(N) is a tree with $n - 2$ Steiner points, each incident with 3 edges [12]. Steiner points can overlap with adjacent Steiner points or terminals. Terminals are then incident with exactly 1 edge (possibly of zero-length). Non-zero-length edges meet at Steiner points at angles that are at least 120°. If a pair of Steiner points s_i and s_j is connected by a zero-length edge, then s_i or s_j are also be connected via a zero-length edge to a terminal and the three non-zero-length edges incident with s_i and s_j make 120° with each other. Any geometric network ST(N) satisfying the above degree conditions is called a *Steiner tree*. The underlying undirected graph $\mathcal{ST}(N)$ (where the coordinates of Steiner points are immaterial) is called a *Steiner topology*. The shortest network with a given Steiner topology is called a *relatively minimal Steiner tree*. If ST(N) has no zero-length edges, then it is called a *full Steiner tree* (FST). Every Steiner tree ST(N) can be decomposed into one or more full Steiner subtrees whose degree 1 points are either terminals or Steiner points overlapping with terminals. A reasonable approach to find a good suboptimal solution to the ESMT problem is therefore to identify few subsets $N_1, N_2, \ldots$ and their low cost Steiner trees ST(N_1), ST(N_2), ... such that a union of some of them, denoted by ST(N), will be a good approximation of SMT(N). The selection of the subsets $N_1, N_2, ...$ should in particular ensure that $|\mathrm{ST}(N)| \leq |\mathrm{MST}(N)|$ where MST(N) is the minimum spanning tree of N.

A Delaunay tessellation of N in $\mathcal{R}^d$, $d \geq 2$, is denoted by DT(N). DT(N) is a simplicial complex. All its k-simplices, $0 \leq k \leq d+1$, are also called *k-faces* or *faces* if k is not essential. In the following, we will only consider k-faces with $1 \leq k \leq d+1$, as 0-faces are terminals in N. It is well-known that the minimum spanning tree MST(N) of N is a subgraph of DT(N) [3]. A face σ of DT(N) is *covered* if the subgraph of MST(N) induced by the corners N_σ of σ is a tree. Corners of N_σ are then also said to be *covered*.

The *Steiner ratio* of a Steiner tree ST(N) is defined by

$$\rho(\mathrm{ST}(N)) = \frac{|\mathrm{ST}(N)|}{|\mathrm{MST}(N)|}$$

The Steiner ratio of N is defined by

$$\rho(N) = \frac{|\mathrm{SMT}(N)|}{|\mathrm{MST}(N)|}$$

It has been observed [23] that for uniformly distributed terminals in a unit square in $\mathcal{R}^2$, $\rho(N)$ typically is between 0.96 and 0.97 corresponding to 3 %–4 % length reduction of SMT(N) over MST(N). The reduction seems to increase as d grows. The smallest Steiner ratio over all sets N in $\mathcal{R}^d$ is defined by

$$\rho_d = \inf_N \{\rho(N)\}$$

It has been conjectured [11] that $\rho_2 = \sqrt{3}/2 = 0.866025....$ There are problem instances achieving this Steiner ratio; for example three corners of an equilateral triangle. Furthermore, ρ_d seems to decrease as $d \rightarrow \infty$. It has also been conjectured that ρ_d, $d \geq 3$ is achieved for infinite sets of terminals. In particular, a *regular 3-sausage* in R^3 is a sequence of regular d-simplices where consecutive ones share a regular 2-simplex (equilateral triangle). It has been conjectured that regular 3-sausages have Steiner ratios decreasing toward 0.7841903733771... as $n \rightarrow \infty$ [21].

Let $N_\sigma \subseteq N$ denote the corners of a face σ of DT(N). Let ST(N_σ) denote a Steiner tree spanning N_σ. Let F be a forest whose vertices are a superset of N. Suppose that terminals of N_σ are in different subtrees of F. The *concatenation* of F *with* ST(N_σ), denoted by $F \oplus \mathrm{ST}(N_\sigma)$, is a forest obtained by adding to F all Steiner points and all edges of ST(N_σ).

Let G be a complete weighted graph spanning N. The *contraction* of G by N_σ, denoted by $G \ominus N_\sigma$, is obtained by replacing the vertices in N_σ by a single vertex n_σ. Loops in $G \ominus N_\sigma$ are deleted. Among any parallel edges of $G \ominus N_\sigma$ incident with n_σ, all but the shortest ones are deleted.

Finally, let $T = \mathrm{MST}(N)$. The *bottleneck contraction* of T by N_σ, denoted by $T \ominus N_\sigma$, is obtained by replacing the vertices in N_σ by a single vertex n_σ. Any cycles in $T \ominus N_\sigma$ are destroyed by removing their longest edges. Hence, $T \ominus N_\sigma$ is a minimum spanning tree of $(N \backslash N_\sigma) \cup \{n_\sigma\}$. Instead of replacing N_σ by n_σ, the vertices of N_σ could be connected by a tree with zero-length edges spanning N_σ. Any cycles in the resulting tree are destroyed by removing their longest edges. We use the same notation, $T \ominus N_\sigma$, to denote the resulting MST on N.

1.2 DM-Heuristic in $\boldsymbol{R^d}$

The DM-heuristic constructs DT(N) and MST(N) in the preprocessing phase. For corners N_σ of every covered face σ of DT(N) in R^d (and for corners of some covered d-sausages), a low cost Steiner tree ST(N_σ) is determined using a heuristic [17] or a numerical approximation of SMT(N_σ) [21]. If full, ST(N_σ) is stored in a priority queue Q ordered by non-decreasing Steiner ratios. Greedy concatenation, starting with a forest F of isolated terminals in N, is then used to form a tree spanning N.

In the postprocessing phase of the DM-heuristic, a *fine-tuning* is performed. The topology of F is extended to the full Steiner topology $\mathcal{ST}(N)$ by adding Steiner points overlapping with terminals where needed. The numerical approximation of [21] is applied to $\mathcal{ST}(N)$ in order to approximate the relatively minimal Steiner tree ST(N) with the Steiner topology $\mathcal{ST}(N)$.

1.3 Improvement Motivation

The DM-heuristic returns better Steiner trees than its $\mathcal{R}^2$ predecessor [20]. It also performs well for $d \geq 3$. However, both the DM-heuristic and its predecessor rely on covered faces of DT(N) determined by the MST(N). The Steiner topology $\mathcal{ST}(N)$ of ST(N) is therefore dictated by the topology of the MST(N). This is a good strategy in many cases but there are also cases where this will exclude good solutions with Steiner topologies not related to the topology of the MST(N). Consider for example Steiner trees in Fig. 1. In T_{DM} (Fig. 1a) only covered faces of DT(N) are considered. By considering some uncovered faces (shaded), a better Steiner tree T_{DB} can be obtained (Fig. 1b).

We wish to detect useful uncovered faces and include them into the greedy concatenation. Consider for example the uncovered triangle σ of DT(N) in R^2 shown in Fig. 2a. If uncovered faces are excluded, the solution returned will be the MST(N) (red edges in Fig. 2a). The simplex σ is uncovered but it has a very good Steiner ratio. As a consequence, if permitted, ST(N_σ) = SMT(N_σ) should be in the solution yielding as much better ST(n) shown in Fig. 2b.

Some uncovered faces of DT(N) can however be harmful in the greedy concatenation even though they seem to be useful in a local sense. For example, use of the uncovered 2-simplex σ of DT(N) in $\mathcal{R}^2$ (Fig. 3a) will lead to a Steiner tree longer than MST(N) (Fig. 3b) while the ratio ρ(SMT(N_σ)) is lowest among all faces of DT(N). Hence, we cannot include all uncovered faces of DT(N).

Another issue arising in connection with using only uncovered faces is that the fraction of covered faces rapidly decreases as d grows. As a consequence, the number of excluded good Steiner trees increases as d grows.

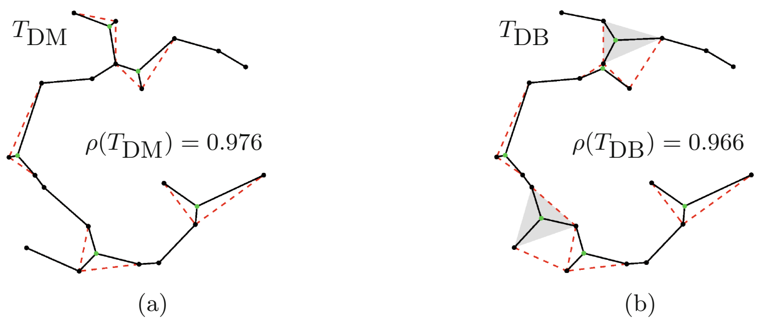

T_{DM}
$\rho(T_{\mathrm{DM}}) = 0.976$
(a)
T_{DB}
$\rho(T_{\mathrm{DB}}) = 0.966$
(b)

Fig. 1. Uncovered faces of DT(N) can improve solutions. Edges of MST(N) not in Steiner trees are dashed and red. (Color figure online)

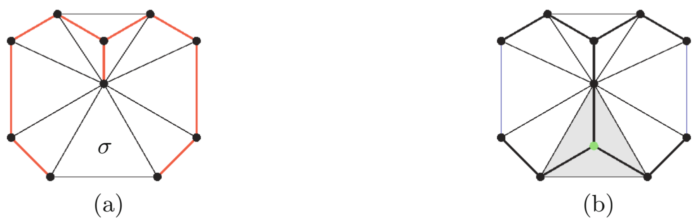


Fig. 2. $\rho(\mathrm{SMT}(N_\sigma))$ is very low and $\mathrm{SMT}(N_\sigma)$ should be included in ST(N). (Color figure online)

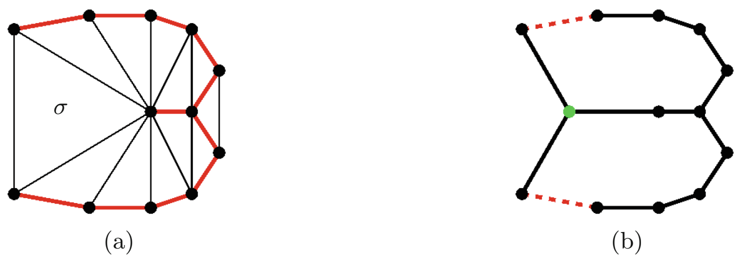


Fig. 3. $\rho(\mathrm{SMT}(N_\sigma))$ is very low but the inclusion of $\mathrm{SMT}(N_\sigma)$ into ST(N) increases the length of ST(N) beyond $|\mathrm{MST}(N)|$. (Color figure online)

2 DB-Heuristic in $\mathcal{R}^d$

Let $T = \mathrm{MST}(N)$. The *bottleneck distance* $|t_it_j|_T$ between two terminals $t_i, t_j \in N$ is the length of the longest edge on the path from t_i to t_j in T. Note that $|t_it_j|_T = |t_it_j|$ if $(t_i, t_j) \in T$.

The *bottleneck minimum spanning tree* $B_T(N_\sigma)$ of a set of points $N_\sigma \subseteq N$ is defined as the minimum spanning tree of the complete graph with N_σ as its vertices and with the cost of an edge $(t_i, t_j), t_i, t_j \in N_\sigma$, given by $|t_it_j|_T$. If N_σ

is covered by T, then $|B_T(N_\sigma)| = |\mathrm{MST}(N_\sigma)|$. Easy proof by induction on the size of N_σ is omitted. Note that N is covered. Hence, $|B_T(N)| = |T|$.

Consider a Steiner tree $\mathrm{ST}(N_\sigma)$ spanning $N_\sigma \subseteq N$. The *bottleneck Steiner ratio* $\beta_T(\mathrm{ST}(N_\sigma))$ is given by:

$$\beta_T(\mathrm{ST}(N_\sigma)) = \frac{|\mathrm{ST}(N_\sigma)|}{|B_T(N_\sigma)|}$$

If N_σ is covered by T, then $\beta_T(\mathrm{ST}(N_\sigma)) = \rho(\mathrm{ST}(N_\sigma))$. Note that $\rho_T(\mathrm{ST}(N_\sigma))$ for the 2-simplex σ in Fig. 3 is very high (even if $\mathrm{ST}(N_\sigma) = \mathrm{SMT}(N_\sigma)$) because $|B_T(N_\sigma)|$ is the sum of the lengths of the two red dashed edges shown in Fig. 3b. Hence, $\mathrm{ST}(N_\sigma)$ will be buried deep in the priority queue Q_B. In fact, it will never be extracted from Q_B as $\rho_T(\mathrm{ST}(N_\sigma)) > 1$.

The DB-heuristic consists of three phases: preprocessing, main loop and postprocessing, see Fig. 5. In the preprocessing phase, the DB-heuristic constructs $\mathrm{DT}(N)$ and $T = \mathrm{MST}(N)$. For corners N_σ of each k-face σ of $\mathrm{DT}(N)$, $2 \leq k \leq d+1$, a low cost Steiner tree $\mathrm{ST}(N_\sigma)$ is determined using a heuristic [17] or a numerical approximation of $\mathrm{SMT}(N_\sigma)$ [21]. Each full $\mathrm{ST}(N_\sigma)$ is stored in a priority queue Q_B ordered by non-decreasing bottleneck Steiner ratios. If σ is a 1-face, then $\mathrm{ST}(N_\sigma) = \mathrm{SMT}(N_\sigma)$ is the edge connecting the two corners of σ. Such $\mathrm{ST}(N_\sigma)$ is added to Q_B only if its edge is in T. Note that bottleneck Steiner ratios of these 1-faces are 1.

Let F be the forest of isolated terminals from N. Furthermore, let $N_0 = N$. In the main loop of the DB-heuristic, a greedy concatenation is applied repeatedly until F becomes a tree. Consider the i-th loop of the DB-heuristic, $i = 1, 2, ...$ Let $\mathrm{ST}(N_{\sigma_i})$ be a Steiner tree with *currently* smallest bottleneck Steiner ratio in Q_B. If a pair of terminals in N_{σ_i} is connected in F, $\mathrm{ST}(N_{\sigma_i})$ is thrown away. Otherwise, let $F = F \oplus \mathrm{ST}(N_{\sigma_i})$ and $T = T \ominus N_{\sigma_i}$, see Fig. 4(a) where the $|B_T(\mathrm{ST}(N_\sigma))| = |e_1| + |e_2|$. Such a bottleneck contraction of T (see Fig. 4(b)) may reduce bottleneck distances between up to $O(n^2)$ pairs of terminals. Hence, bottleneck Steiner ratios of some Steiner trees still in Q_B need to be updated

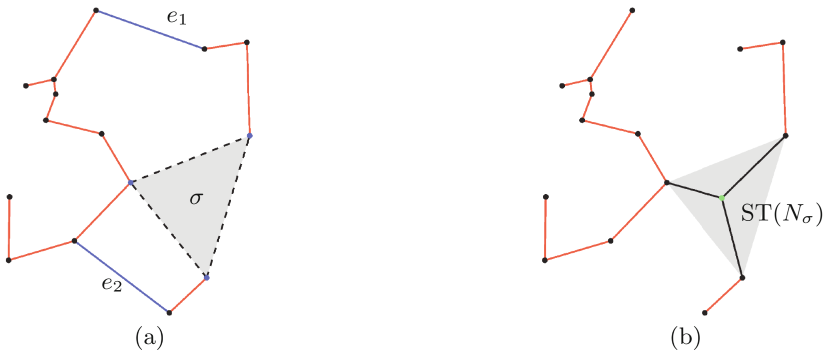


Fig. 4. The insertion of $ST(N_\sigma)$

```
// Preprocessing
      Construct DT(N) and T = MST(N);
      Push Steiner trees of all faces of DT(N) onto $Q_B$ (
          except 1-faces not in T)
      Let F be the forest on N with no edges.
// Main loop
      while (F is not a tree on N) {
         ST($N_\sigma$) = Steiner tree in $Q_B$ with smallest
             bottleneck Steiner ratio w.r.t. T;
         if (no pair of terminals in $N_\sigma$ is connected in F) {
            F = F $\oplus$ ST($N_\sigma$);
            T = T $\ominus$ $N_\sigma$;
         }
      }
 // Postprocessing
      Fine-tune F;
      return F
```

Fig. 5. DB-heuristic

either immediately or in a lazy fashion. Note that bottleneck Steiner ratios cannot decrease. If they increase beyond 1, the corresponding Steiner trees do not need to be placed back in Q_B. This is due to the fact that all 1-faces (edges) of the MST(N) are in Q_B and have bottleneck Steiner ratios equal to 1. We will return to the updating of bottleneck Steiner ratios in Sect. 3. Fine-tuning (as in the DM-heuristic) is applied in the postprocessing phase.

Unlike the DM-heuristic, d-sausages are not used in the DB-heuristic. In the DB-heuristic all faces of DT(N) are considered. As a consequence, fine-tuning in the postprocessing will in most cases indirectly generate Steiner trees spanning terminals in d-sausages if they are good candidates for subtrees of ST(N).

3 Contractions and Bottleneck Distances

As face-spanning Steiner trees are added to F in the main loop of the DB-heurstic, corners of these faces are bottleneck contracted in the current minimum spanning tree T. Bottleneck contractions will reduce bottleneck distances between some pairs of terminals. As a consequence, bottleneck Steiner ratios of face-spanning Steiner trees still in Q_B will increase. A face-spanning Steiner tree subsequently extracted from Q_B will not necessarily have the smallest bottleneck Steiner ratio (unless Q_B has been rearranged). Hence, appropriate lazy updating has to be carried out. To summarize, a data structure supporting the following operations is needed:

- `bc(`N_σ`)`: corners of a face $N_\sigma \in$ DT(N) are bottleneck contracted in T,
- `bd(p,q)`: bottleneck distance between p and q in current minimum spanning tree T is returned,
- Q_B is maintained as a priority queue ordered by non-decreasing bottleneck Steiner ratios.

Maintaining Q_B could be done by recomputing bottleneck Steiner ratios and rearranging the entire queue after each contraction. Since there may be as many as $O(n^{\lceil d/2 \rceil})$ faces in DT(N) [18], this will be too slow.

To obtain better CPU times, a slightly modified version of dynamic rooted trees [19] maintaining a changing forest of disjoint rooted trees is used. For our purposes, `bc`-operations and `bd`-queries require maintaining a changing minimum spanning tree T rather than a forest. Dynamic rooted trees support (among others) the following operations:

- `evert`(n_j): makes n_j the root of the tree containing n_j.
- `link`(n_j, n_i, x): links the tree rooted at n_j to a vertex n_i in another tree. The new edge (n_j, n_i) is assigned the cost x.
- `cut`(n_j): removes the edge from n_j to its parent. n_j cannot be a root.
- `mincost`(n_j): returns the vertex n_i on the path to the root and closest to its root such that the edge from n_i to its parent has minimum cost. n_j cannot be a root.
- `cost`(n_j): returns the cost of the edge from n_j to its parent. n_j cannot be a root.
- `update`(n_j, x): adds x to the weight of every edge on the path from n_j to the root.

For our purposes, a `maxcost` operation replaces `mincost`. Furthermore, the `update` operation is not needed. A root of the minimum spanning tree can be chosen arbitrarily.

Rooted trees are decomposed into paths (see Fig. 6) represented by balanced binary search trees or biased binary trees. The path decomposition can be changed by splitting or joining these binary trees.

By appropriate rearrangement of the paths, all above operations can be implemented using binary search tree operations [19]. Since the `update` operation is not needed, the values `cost` and `maxcost` can be stored directly with n_j. Depending on whether balanced binary search trees or biased binary trees are used for the paths, the operations require respectively $O(\log^2 n)$ and $O(\log n)$ amortized time.

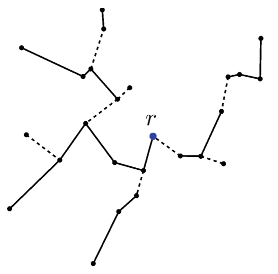


Fig. 6. A rooted tree decomposed into paths.

Using dynamic rooted trees to store the minimum spanning tree, `bd`-queries and bottleneck contractions can be implemented as shown in Fig. 7. The `bd`-query makes n_i the new root. Then it finds the vertex n_k closest to n_i such that the edge from n_k to its parent has maximum cost on the path from n_j to n_i. The cost of this edge is returned. The `bc`-operation starts by running through all pairs of vertices of N_σ. For each pair n_i, n_j, n_i is made the root of the tree (`evert`(n_i)) and then the edge with the maximum cost on the path from n_j to n_i is found. If n_i and n_j are connected, the edge is cut away. Having cut away all connecting edges with maximum cost, the vertices of N_σ are reconnected by zero-length edges.

```
// t$_i$ and t$_j$ are vertices of the minimum spanning tree T
bd(t$_i$,t$_j$) {
  evert(t$_i$)
  return cost(maxcost(t$_j$))
}

// N$_\sigma$ = { t$_1$, t$_2$, ..., t$_k$ }, $k \geq 2$ is a set of corners of a face
     of DT($N$)
bc(t$_\sigma$) {
  for(i = 1 .. k) {
    evert(t$_i$)
    for(j = i+1 .. k)
      if(t$_i$ and t$_j$ are connected) cut(maxcost(t$_j$));
  }
  evert(t$_1$)
  for(i = 2 .. k)  link(t$_{i-1}$, t$_i$, 0)
}
```

Fig. 7. `bd`-query and `bc`-operation

When using balanced binary trees, one `bd`-query takes $O((\log n)^2)$ amortized time. Since only faces of $\mathrm{DT}(N)$ are considered, the `bc`-operation performs $O(d)$ `everts` and `links`, $O(d^2)$ `maxcosts` and `cuts`. Hence, it takes $O((d \log n)^2)$ time.

In the main loop of the algorithm, Steiner trees of faces of $\mathrm{DT}(N)$ are extracted one by one. A face σ is rejected if some of its corners are already connected in F. Since the quality of the final solution depends on the quality of Steiner trees of faces, these trees should have smallest possible bottleneck Steiner ratios. When a Steiner tree $ST(N_\sigma)$ is extracted from Q_B, it is first checked if $ST(N_\sigma)$ spans terminals already connected in F. If so, $ST(N_\sigma)$ is thrown away. Otherwise, its bottleneck Steiner ratio may have changed since the last time it was pushed onto Q_B. Hence, bottleneck Steiner ratio of $ST(N_\sigma)$ is recomputed. If it increased since last but is still below 1, $ST(N_\sigma)$ is pushed back onto Q_B (with the new bottleneck Steiner ratio). If the bottleneck Steiner ratio did not change, $ST(N_\sigma)$ is used to update F and bottleneck contract T.

4 Computational Results

The DB-heuristic was tested against the DM-heuristic. Both Steiner ratios and CPU times were compared. To get reliable Steiner ratio and computational time comparisons, they were averaged over several runs whenever possible. Furthermore, the results in $\mathcal{R}^2$ were compared to the results achieved by the exact GeoSteiner algorithm [13].

To test and compare the DM- and the DB-heuristic, they were implemented in C++. The code and instructions on how to run the DM- and DB-heuristics can be found in the GitHub repository [16]. All tests have been run on a Lenovo ThinkPad S540 with a 2 GHz Intel Core i7-4510U processor and 8 GB RAM.

The heuristics were tested on randomly generated problem instances of different sizes in $\mathcal{R}^d$, $d = 2, 3, ..., 6$, as well as on library problem instances. Randomly generated instances were points uniformly distributed in $\mathcal{R}^d$-hypercubes.

The library problem instances consisted of the benchmark instances from the 11-th DIMACS Challenge [6]. More information about these problem instances can be found on the DIMACS website [6]. For comparing the heuristics with the GeoSteiner algorithm, we used ESTEIN instances in $\mathcal{R}^2$.

Dynamic rooted trees were implemented using AVL trees. The *restricted numerical optimisation* heuristic [17] for determining Steiner trees of DT(N) faces was used in the experiments.

In order to get a better idea of the improvement achieved when using bottleneck distances, the DM-heuristic does not consider covered d-sausages as proposed in [17]. Test runs of the DM-heuristic indicate that the saving when using d-sausages together with fine-tuning is only around 0.1 % for $d = 2$, 0.05 % for $d = 3$ and less than 0.01 % when $d > 3$. As will be seen below, the savings achieved by using bottleneck distances are more significant.

In terms of quality, the DB-heuristic outperforms the DM-heuristic. The Steiner ratios of obtained Steiner trees reduces by 0.2−0.3 % for $d = 2$, 0.4−0.5 % for $d = 3$, 0.6−0.7 % for $d = 4$, 0.7−0.8 % for $d = 5$ and 0.8−0.9 % for $d = 6$. This is a significant improvement for the ESMT problem as will be seen below, when comparing $\mathcal{R}^2$ results to the optimal solutions obtained by the exact GeoSteiner algorithm [13].

CPU times for both heuristics for $d = 2, 3, ..., 6$, are shown in Fig. 8. It can be seen that the improved quality comes at a cost for $d \geq 4$. This is due to the fact that the DB-heuristic constructs low cost Steiner trees for all $O(n^{\lceil d/2 \rceil})$ faces of DT(N) while the DM-heuristic does it for covered faces only. Later in this section it will be explored how the Steiner ratios and CPU times are affected if the DB-heuristic drops some of the faces.

Figure 9 shows how the heuristics and GeoSteiner (GS) performed on ESTEIN instances in $\mathcal{R}^2$. Steiner ratios and CPU times averaged over all 15 ESTEIN instances of the given size, except for $n = 10000$ which has only one instance. For the numerical comparisons, see Table 1 in the GitHub repository [16]. It can be seen that the DB-heuristic produces better solutions than the DM-heuristic without any significant increase of the computational time.

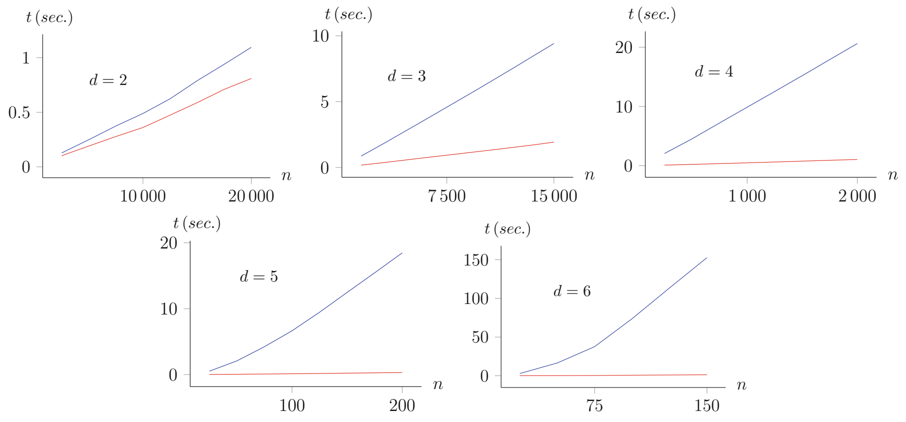


Fig. 8. Comparison of the CPU times for the DB-heuristic (blue) and the DM-heuristic (red) for $d = 2, 3, ..., 6$. (Color figure online)

It is also worth noticing that the DB-heuristic gets much closer to the optimal solutions. This may indicate that the DB-heuristic also produces high quality solutions when $d > 2$, where optimal solutions are only known for instances with at most 20 terminals. For the performance of the DB-heuristic on individual $\mathcal{R}^2$ instances, see Tables 3–7 in the GitHub repository [16].

The results for ESTEIN instances in $\mathcal{R}^3$ are presented in Fig. 10. The green plot for $n = 10$ is the average ratio and computational time achieved by numerical approximation [21]. Once again, the DB-heuristic outperforms the DM-heuristic when comparing Steiner ratios. However, the running times are

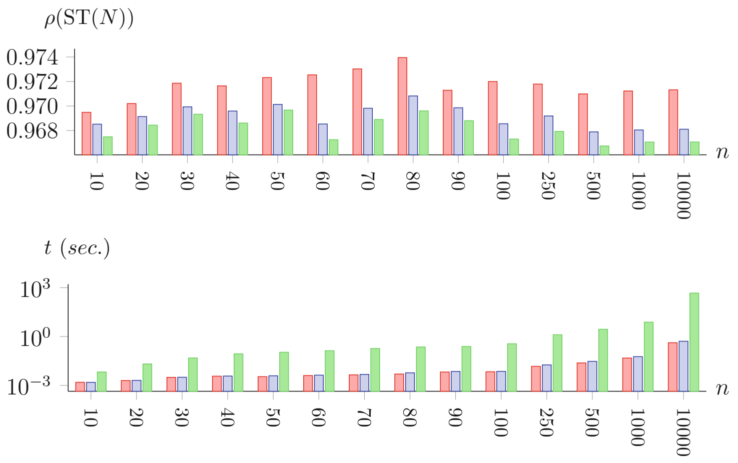


Fig. 9. Averaged ratios and CPU times for ESTEIN instances in $\mathcal{R}^2$. DM-heuristic (red), DB-heuristic (blue), GeoSteiner (green). (Color figure online)

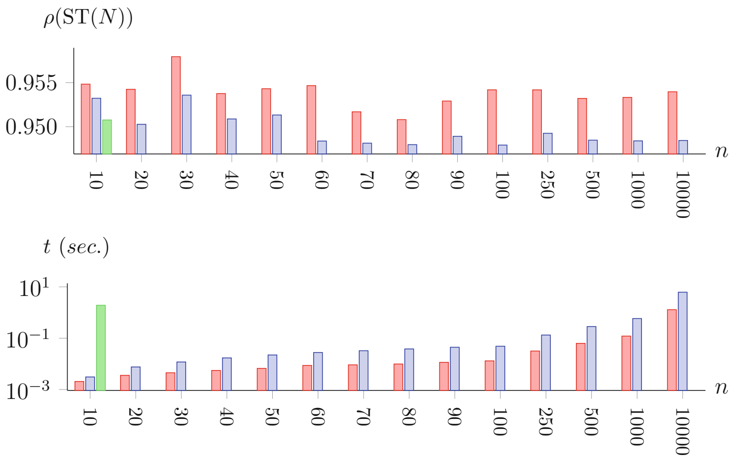
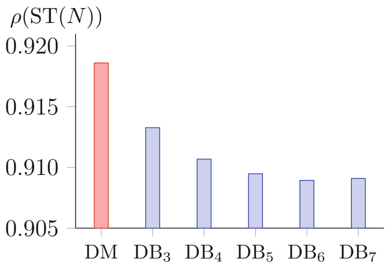


Fig. 10. Averaged ratio and CPU times for ESTEIN instances in $\mathcal{R}^3$. DM-heuristic (red), DB-heuristic (blue), numerical approximation (green). (Color figure online)

now up to four times worse. For the numerical comparisons, see Table 2 in the GitHub repository [16]. For the performance of the DB-heuristic on individual $\mathcal{R}^3$ instances, see Tables 8–12 in the GitHub repository [16].

The DB-heuristic starts to struggle when $d \geq 4$. This is caused by the number of faces of DT(N) for which low cost Steiner trees must be determined. The DB-heuristic was therefore modified to consider only faces with less than k terminals, for $k = 3, 4, ..., d + 1$. Figure 11 shows the performance of this modified DB$_k$-heuristic with $k = 3, 4, ..., 7$, on a set with 100 terminals in $\mathcal{R}^6$. Note that DB$_7$ = DB.

As expected, the DB$_k$-heuristic runs much faster when larger faces of DT(N) are disregarded. Already the DB$_4$-heuristic seems to be a reasonable alternative since solutions obtained by DB$_k$-heuristic, $5 \leq k \leq 7$, are not significantly better. Surprisingly, the DB$_6$-heuristic performs slightly better than the DB$_7$-heuristic.



Method	t
DM	0.4714
DB$_3$	0.6000
DB$_4$	6.0525
DB$_5$	26.2374
DB$_6$	51.3653
DB$_7$ = DB	62.8098

Fig. 11. Results achieved when considering faces of DT(N) with at most $k = 3, 4, ..., 7$ terminals in the concatenation for $d = 6$ and $n = 100$.

This is probably due to the fact that low cost Steiner trees of smaller faces have fewer Steiner points. This in turn causes the fine-tuning step of the DB_6-heuristic to perform better than is the case for DB_7.

5 Summary and Conclusions

The DM-heuristic in $\mathcal{R}^d$ [17] was extended to the DB-heuristic that uses bottleneck distances to determine good candidates for low cost Steiner trees. Computational results show a significant improvement in the quality of the Steiner trees produced by the DB-heuristic.

The CPU times of the DB-heuristic are comparable to the CPU times of the DM-heuristic in $\mathcal{R}^d$, $d = 2, 3$. It runs slower for $d \geq 4$. However, its CPU times can be significantly improved by skipping larger faces of $\mathrm{DT}(N)$. This results in only small decrease of the quality of solutions obtained.

References

1. Bajaj, C.: The algebraic degree of geometric optimization problems. Discrete Comput. Geom. **3**, 177–191 (1988)
2. Beasley, J.E., Goffinet, F.: A Delaunay triangulation-based heuristic for the Euclidean Steiner problem. Networks **24**(4), 215–224 (1994)
3. de Berg, M., Cheong, O., van Krevald, M., Overmars, M.: Computational Geometry - Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008)
4. Brazil, M., Graham, R.L., Thomas, D.A., Zachariasen, M.: On the history of the Euclidean Steiner tree problem. Arch. Hist. Exact Sci. **68**, 327–354 (2014)
5. Brazil, M., Zachariasen, M.: Optimal Interconnection Trees in the Plane. Springer, Cham (2015)
6. DIMACS, ICERM: 11th DIMACS Implementation Challenge: Steiner Tree Problems (2014). http://dimacs11.cs.princeton.edu/
7. Fampa, M., Anstreicher, K.M.: An improved algorithm for computing Steiner minimal trees in Euclidean d-space. Discrete Optim. **5**, 530–540 (2008)
8. Fampa, M., Lee, J., Maculan, N.: An overview of exact algorithms for the Euclidean Steiner tree problem in n-space, Int. Trans. OR (2015)
9. Fonseca, R., Brazil, M., Winter, P., Zachariasen, M.: Faster exact algorithms for computing Steiner trees in higher dimensional Euclidean spaces. In: Proceedings of the 11th DIMACS Implementation Challenge, Providence, Rhode Island, USA (2014). http://dimacs11.cs.princeton.edu/workshop.html
10. do Forte, V.L., Montenegro, F.M.T., de Moura Brito, J.A., Maculan, N.: Iterated local search algorithms for the Euclidean Steiner tree problem in n dimensions. Int. Trans. OR (2015)
11. Gilbert, E.N., Pollak, H.O.: Steiner minimal trees. SIAM J. Appl. Math. **16**(1), 1–29 (1968)
12. Hwang, F.K., Richards, D.S., Winter, P.: The Steiner Tree Problem. North-Holland, Amsterdam (1992)
13. Juhl, D., Warme, D.M., Winter, P., Zachariasen, M.: The GeoSteiner software package for computing Steiner trees in the plane: an updated computational study. In: Proceedings of the 11th DIMACS Implementation Challenge, Providence, Rhode Island, USA (2014). http://dimacs11.cs.princeton.edu/workshop.html

14. Laarhoven, J.W.V., Anstreicher, K.M.: Geometric conditions for Euclidean Steiner trees in R^d. Comput. Geom. Theor. Appl. **46**(5), 520–531 (2013)
15. Laarhoven, J.W.V., Ohlmann, J.W.: A randomized Delaunay triangulation heuristic for the Euclidean Steiner tree problem in R^d. J. Heuristics **17**(4), 353–372 (2011)
16. Lorenzen, S.S., Winter, P.: Code and Data Repository at Github (2016). https://github.com/StephanLorenzen/ESMT-heuristic-using-bottleneck-distances/blob/master/README.md
17. Olsen, A., Lorenzen, S. Fonseca, R., Winter, P.: Steiner tree heuristics in Euclidean d-space. In: Proceedings of the 11th DIMACS Implementation Challenge, Providence, Rhode Island, USA (2014). http://dimacs11.cs.princeton.edu/workshop.html
18. Seidel, R.: The upper bound theorem for polytopes: an easy proof of its asymptotic version. Comp. Geom.-Theor. Appl. **5**, 115–116 (1995)
19. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. J. Comput. Syst. Sci. **26**(3), 362–391 (1983)
20. Smith, J.M., Lee, D.T., Liebman, J.S.: An O($n \log n$) heuristic for Steiner minimal tree problems on the Euclidean metric. Networks **11**(1), 23–39 (1981)
21. Smith, W.D.: How to find Steiner minimal trees in Euclidean d-space. Algorithmica **7**, 137–177 (1992)
22. Toppur, B., Smith, J.M.: A sausage heuristic for Steiner minimal trees in three-dimensional Euclidean space. J. Math. Model. Algorithms **4**, 199–217 (2005)
23. Warme, D.M., Winter, P., Zachariasen, M.: Exact algorithms for plane Steiner tree problems: a computational study. In: Du, D.-Z., Smith, J., Rubinstein, J. (eds.) Advances in Steiner Trees, pp. 81–116. Springer, Dordrecht (2000)