UNIVERSITY OF COPENHAGEN

Steiner tree heuristic in the Euclidean d-space using bottleneck distances

Lorenzen, Stephan Sloth; Winter, Pawel

Published in:
Experimental Algorithms

DOI:
10.1007/978-3-319-38851-9_15

Publication date:
2016

Document version
Early version, also known as pre-print

Citation for published version (APA):
Lorenzen, S. S., & Winter, P. (2016). Steiner tree heuristic in the Euclidean d-space using bottleneck distances. In A. V. Goldberg, & A. S. Kulikov (Eds.), *Experimental Algorithms: 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings* (pp. 217-230). Springer. Lecture notes in computer science, Vol.. 9685 https://doi.org/10.1007/978-3-319-38851-9_15

Download date: 21. jan.. 2021

Bottleneck Distances and Steiner Trees in the Euclidean d-Space

Stephan S. Lorenzen Pawel Winter *

Abstract

Some of the most efficient heuristics for the Euclidean Steiner minimal trees 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 low cost trees spanning all terminals. The weakness of this approach is that obtained solutions are topologically related to minimum spanning trees. To obtain 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.

1 Introduction

Given a set of *terminals* $N = \{t_1, t_2, ..., t_n\}$ 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 in $S = V \setminus N$ 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. T must be a tree. It is called the *Euclidean Steiner minimal tree* and is denoted by $\mathrm{SMT}(N)$. 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 [3] for the fascinating history of the ESMT problem.

The ESMT problem is NP-hard [4]. A good exact method for solving problem instances with up to 50.000 terminals in R^2 is available [9]. However, no analytical method can exist for $d \geq 3$ [1]. Furthermore, no numerical approximation seems to be able to solve instances with more than 15-20 terminals [6]. It is therefore essential to develop good quality heuristics for $d \geq 3$. Several heuristics have been proposed in the literature [15, 7, 10]. In particular, the heuristic suggested in [10] builds on a R^2-heuristic [13]. Both use **D**elaunay tessellations and **M**inimum spanning trees and are therefore referred to as DM-heuristics.

$\mathrm{SMT}(N)$ must have $n - 2$ Steiner points, each incident with 3 edges [8]. Terminals must be incident with exactly 1 edge (possible of zero-length). Non-zero-length edges must meet at Steiner points at angles that are at least 120^o. If a pair of Steiner points s_i and s_j is connected by a zero-length edge, then s_i or s_j is connected via a zero-length edge to a terminal and the three non-zero-length edges incident with s_i and s_j must make 120^o with each other. Any geometric network $\mathrm{ST}(N)$ satisfying the above degree and angle conditions is called a *Steiner tree*. The underlying undirected graph (where the coordinates of Steiner points are immaterial) is called a *Steiner topology* of N. If $\mathrm{ST}(N)$ has no zero-length edges, then it is called a *full Steiner tree*. Every Steiner tree $\mathrm{ST}(N)$ can be decomposed into one or more full Steiner subtrees whose degree one 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 $N_1, N_2, ..., N_z$, and their low cost Steiner trees $\mathrm{ST}(N_1), \mathrm{ST}(N_2), ..., \mathrm{ST}(N_z)$, such that a union $\mathrm{ST}(N)$ of some of them will be a good approximation of $\mathrm{SMT}(N)$.

The Delaunay tessellation of N in $\mathcal{R}^d$ is denoted by $\mathrm{DT}(N)$ [2]. It is well-known that a minimum spanning tree of N, denoted by $\mathrm{MST}(N)$, is a subgraph of $\mathrm{DT}(N)$. A face σ of $\mathrm{DT}(N)$ is *covered* if the subgraph of $\mathrm{MST}(N)$ induced by the corners of σ is a tree.

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

Let $T = \mathrm{MST}(N)$. The *contraction* $T \ominus N_\sigma$ of T by N_σ is obtained by replacing the vertices in N_σ by a single vertex n_σ. Cycles in $T \ominus N_\sigma$ are destroyed by removing their longest edges.

The DM-heuristic constructs $\mathrm{DT}(N)$ and $\mathrm{MST}(N)$ in the preprocessing phase. For corners N_σ of every covered face σ of $\mathrm{DT}(N)$, a low cost Steiner tree $\mathrm{ST}(N_\sigma)$ is determined [10]. If full, it is stored in a priority queue Q ordered by non-decreasing Steiner

*Dept. of Computer Science, Univ. of Copenhagen, stephan.lorenzen@gmail.com, pawel@di.ku.dk, full version of the paper: http://www.diku.dk/~pawel/bottleneck.pdf

This is an extended abstract of a presentation given at EuroCG 2016. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear in a conference with formal proceedings and/or in a journal.

ratios $\rho(\mathrm{ST}(N_\sigma)) = |\mathrm{ST}(N_\sigma)|/|\mathrm{MST}(N_\sigma)|$. Greedy concatenation, starting with the forest F of isolated terminals in N, is then used to form $\mathrm{ST}(N)$.

The weakness of the DM-heuristic is that it relies on covered faces of $\mathrm{DT}(N)$. The Steiner topology of $\mathrm{ST}(N)$ is therefore dictated by the topology of T. This is a good strategy in many cases but there are also cases where this will exclude good solutions. Consider for example the two Steiner trees in Fig. 1. Only covered faces of $\mathrm{DT}(N)$ are considered in T_{DM}. By considering some uncovered faces (shaded), a better Steiner tree T_{DB} can be obtained.

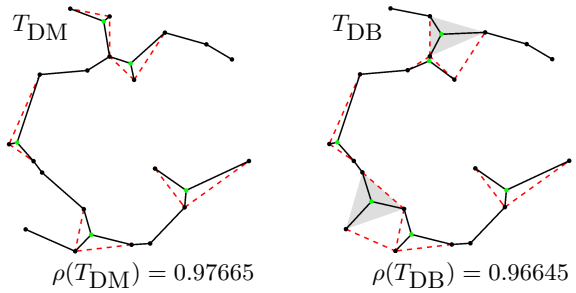


Figure 1: Uncovered faces of $\mathrm{DT}(N)$ can improve solutions. Edges of T not in Steiner trees are red.

We wish to detect useful uncovered faces and include them into the greedy concatenation. However, some uncovered faces of $\mathrm{DT}(N)$ can be harmful in the greedy concatenation even though they seem to be useful locally. As illustrated in Fig. 2, use of the uncovered face σ of $\mathrm{DT}(N)$ in $\mathcal{R}^2$ with the set of corners $N_\sigma = \{t_i, t_j, t_k\}$ will lead to a Steiner tree $\mathrm{ST}(N)$ longer than T while the ratio $\rho(\mathrm{ST}(N_\sigma))$ is lowest among all faces of $\mathrm{DT}(N)$.

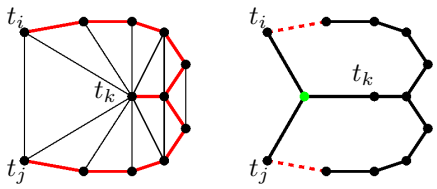


Figure 2: $\rho(\mathrm{ST}(N_\sigma))$ for $N_\sigma = \{t_i, t_j, t_k\}$ is low but the inclusion of $\mathrm{ST}(N_\sigma)$ increases its length beyond $|T|$.

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

The *bottleneck distance* $|t_i t_j|_T$ 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 $T = \mathrm{MST}(N)$. Note that $|t_i t_j|_T = |t_i t_j|$ 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 $|t_i t_j|_T$ being the cost of an edge $(t_i, t_j), t_i, t_j \in N_\sigma$. If N_σ is covered by T, then $|B_T(N_\sigma)| = |\mathrm{MST}(N_\sigma)|$.

Consider a Steiner tree $\mathrm{ST}(N_\sigma)$ spanning $N_\sigma \subseteq N$. Let the *bottleneck Steiner ratio* $\beta_T(\mathrm{ST}(N_\sigma)) = |\mathrm{ST}(N_\sigma)|/|B_T(N_\sigma)|$ If N_σ is covered by T, then $\beta_T(\mathrm{ST}(N_\sigma)) = \rho(\mathrm{ST}(N_\sigma))$.

The DB-heuristic constructs the **D**elaunay tesselation $\mathrm{DT}(N)$ and uses T to determine **B**ottleneck distances. For corners N_σ of each k-face σ of $\mathrm{DT}(N)$, $2 \le k \le d+1$, a low cost Steiner tree $\mathrm{ST}(N_\sigma)$ is determined using a heuristic [10]. 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 $\mathrm{ST}(N_\sigma)$ is the edge connecting the two corners of σ. Such $\mathrm{ST}(N_\sigma)$ is added to Q_B only if it is an edge in T.

Let F be the forest of isolated terminals from N. A greedy concatenation is then applied repeatedly until F becomes a tree. Let $\mathrm{ST}(N_\sigma)$ be a Steiner tree with *currently* smallest bottleneck Steiner ratio in Q_B. If any pair of terminals in N_σ is connected in F, $\mathrm{ST}(N_\sigma)$ is discarded. Otherwise, $F = F \oplus \mathrm{ST}(N_\sigma)$ and $T = T \ominus N_\sigma$, see Fig. 3. Such contraction of T 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, preferrably in a lazy fashion, see Section 3.

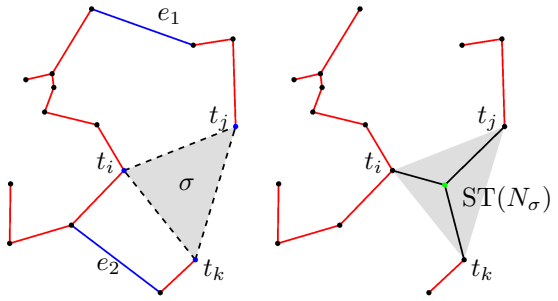


Figure 3: The insertion of $ST(N_\sigma)$, $N_\sigma = \{t_i, t_j, t_k\}$

3 Contractions and Bottleneck Distances

As face-spanning Steiner trees are added to F, their corners are contracted in the current minimum spanning tree T. 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 or appropriate lazy updating is carried out.

Steiner trees of faces of $\mathrm{DT}(N)$ are extracted from Q_B one by one. A face σ is discarded if some of its

corners are already connected in F. The bottleneck Steiner ratio $\beta_T(\mathrm{ST}(N_\sigma))$ of the extracted Steiner tree $\mathrm{ST}(N_\sigma)$ may have changed since the last time $\mathrm{ST}(N_\sigma)$ was pushed onto Q_B. Hence, $\beta_T(\mathrm{ST}(N_\sigma))$ has to be recomputed. If it increased, $\mathrm{ST}(N_\sigma)$ is pushed back onto Q_B (with the new bottleneck Steiner ratio). If not, $\mathrm{ST}(N_\sigma)$ is used to update F and to contract T.

A modified version of a *dynamic rooted tree* [12] to maintain a changing minimum spanning tree T (caused by contractions) and to answer bottleneck distance querries has been used. When using balanced binary trees to implement dynamic rooted trees, a bottleneck distance query takes $O((\log n)^2)$ amortized time. Since only faces of $\mathrm{DT}(N)$ are considered, a contraction takes $O((d \log n)^2)$ time.

4 Computational results

The DB-heuristic was compared with the DM-heuristic. Both Steiner ratios and CPU times were examined. To get reliable comparisons, they were averaged over several runs. Furthermore, the results in $\mathcal{R}^2$ were compared with the results achieved by the exact GeoSteiner algorithm [9].

The DM- and DB-heuristics were implemented in C++ [1] and run on a Lenovo ThinkPad S540 with a 2 GHz Intel Core i7-4510U processor and 8 GB RAM.

Both heuristics were tested on randomly generated problem instances 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 [5]. For comparing the DB-heuristic with the GeoSteiner algorithm, ESTEIN instances in $\mathcal{R}^2$ were used [5].

The new DB-heuristic outperforms the DM-heuristic 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 [9].

CPU times for the DM- and DB-heuristics for $d = 2, 3, ..., 6$, are shown in Fig. 4. 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 $\mathrm{DT}(N)$ [11] while the DM-heuristic does it for covered faces only.

Fig. 5 shows how DB-, DM-heuristic 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

[1] The DB-heuristic code and instructions on how to run it can be found at `https://github.com/StephanLorenzen/ESMT-heuristic-using-bottleneck-distances`

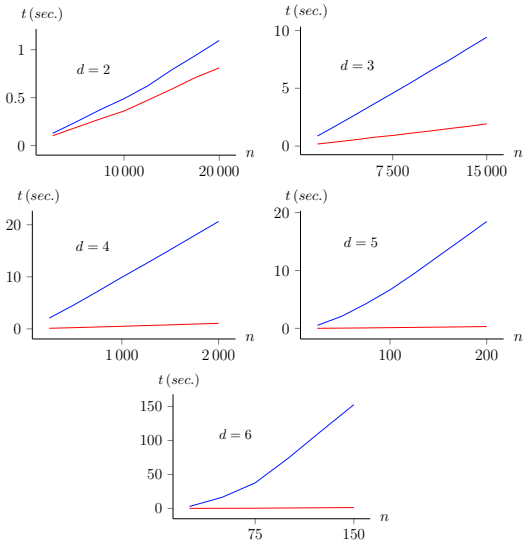


Figure 4: CPU times for DM (red) and DB (blue), $d = 2, 3, ..., 6$.

$n = 10.000$ which has only one instance. It can be seen that the DB-heuristic produces better solutions than the DM-heuristic without any significant increase of the CPU time. It is also worth noticing that the DB-heuristic gets very close 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.

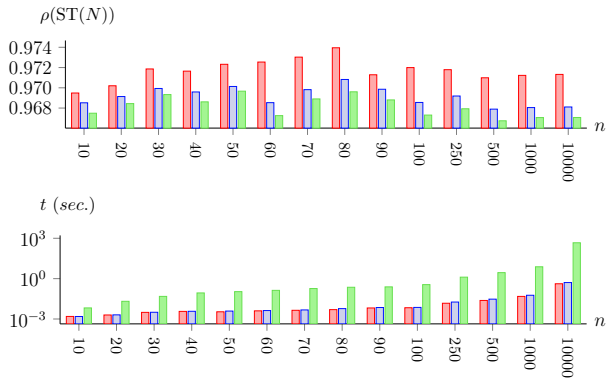


Figure 5: Averaged ratios and CPU times for ESTEIN instances in $\mathcal{R}^2$. DM (red), DB (blue), GeoSteiner (green).

The results for ESTEIN instances in $\mathcal{R}^3$ are presented in Fig. 6. The green plot for $n = 10$ is the average ratio and CPU time of the exact solutions [14]. Once again, the DB-heuristic outperforms the DM-heuristic when comparing the quality of solutions. However, the running times are now up to four times worse.

The DB-heuristic starts to struggle when $d \geq 4$.

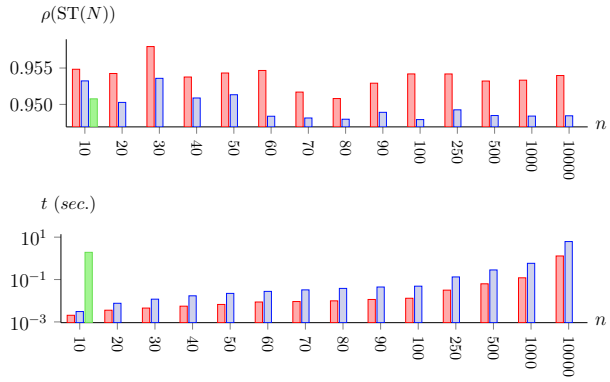
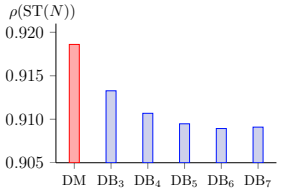


Figure 6: Averaged ratios and CPU times for ESTEIN instances in $\mathcal{R}^3$. DM (red), DB (blue), exact (green).

This is caused by the number of faces of $\mathrm{DT}(N)$ 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 $k = 3, 4, ..., d+1$. Fig. 7 shows the performance of this modified DB_k-heuristic for $k = 3, 4, ..., 7$, on a set with 100 terminals in $\mathcal{R}^6$. Note that $\mathrm{DB}_7 = \mathrm{DB}$.



Method	t
BM	0.4714
DB_3	0.6000
DB_4	6.0525
DB_5	26.2374
DB_6	51.3653
$\mathrm{DB}_7 = \mathrm{DB}$	62.8098

Figure 7: DB_k for $k = 3, 4, ..., 7$, $d = 6$ and $n = 100$.

As expected, the DB_k-heuristic runs much faster when larger faces of $\mathrm{DT}(N)$ 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.

5 Summary and conclusion

Computational results show a significant improvement in the quality of the Steiner trees produced by the DB-heuristic where the topologies of the solutions are no longer constrained by minimum spanning trees. Its CPU times 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, CPU times can be significantly improved by skipping larger faces of $\mathrm{DT}(N)$. This results in only small decrease of quality of solutions obtained.

References

[1] C. Bajaj, The algebraic degree of geometric optimization problems, *Discrete and Computational Geometry* **3** (1988), 177–191.

[2] M. de Berg, O. Cheong, M. van Krevald, and M. Overmars, *Computational Geometry - Algorithms and Applications* (3. ed.), Springer (2008).

[3] M. Brazil, R. Graham, D. Thomas, and M. Zachariasen, On the history of the Euclidean Steiner tree problem, *Archive for History of Exact Sciences* **68** (2014), 327–354.

[4] M. Brazil and M. Zachariasen, *Optimal Interconnection Trees in the Plane*, Springer (2015).

[5] DIMACS and ICERM, *11th DIMACS Implementation Challenge: Steiner Tree Problems*, `http://dimacs11.cs.princeton.edu/` (2014).

[6] M. Fampa, J. Lee, and N. Maculan, An overview of exact algorithms for the Euclidean Steiner tree problem in n-space, *Int. Trans. in OR* (2015).

[7] V. L. do Forte, F. M. T. Montenegro, J. A. de Moura Brito, and N. Maculan, Iterated local search algorithms for the Euclidean Steiner tree problem in n dimensions, *Int. Trans. in OR* (2015).

[8] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland (1992).

[9] D. Juhl, D. M. Warme, P. Winter, and M. Zachariasen, The GeoSteiner software package for computing Steiner trees in the plane: An updated computational study, *Proc. of the 11th DIMACS Implementation Challenge* (2014). `http://dimacs11.cs.princeton.edu/workshop.html`

[10] A. Olsen, S. Lorenzen, R. Fonseca, and P. Winter, Steiner tree heuristics in Euclidean d-space, *Proc. of the 11th DIMACS Implementation Challenge* (2014). `http://dimacs11.cs.princeton.edu/workshop.html`

[11] R. Seidel, The upper bound theorem for polytopes: an easy proof of its asymptotic version, *Comp. Geom.-Theor. Appl.* **5** (1995), 115–116.

[12] D. D. Sleator and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. and Syst. Sci.* **26**, 3 (1983), 362–391.

[13] J. M. Smith, An $O(n \log n)$ heuristic for Steiner minimal tree problems on the Euclidean metric, *Networks* **11**, 1 (1981), 23–39.

[14] W. D. Smith, How to find Steiner minimal trees in Euclidean d-space, *Algorithmica* **7** (1992), 137–177.

[15] B. Toppur and J. M. Smith, A sausage heuristic for Steiner minimal trees in three-dimensional Euclidean space, *J. Math. Model. and Algorithms* **4** (2005), 199–217.