Approximation Algorithms for Steiner Tree Based on Star Contractions: A Unified View

Radek Hušek
Computer Science Institute of Charles University, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
husek@iuuk.mff.cuni.cz

Dušan Knop
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
dusan.knop@fit.cvut.cz

Tomáš Masařík
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Czech Republic
http://tarken.krakonos.org/
masarik@kam.mff.cuni.cz

Abstract

In the Steiner Tree problem, we are given an edge-weighted undirected graph $G = (V, E)$ and a set of terminals $R \subseteq V$. The task is to find a connected subgraph of G containing R and minimizing the sum of weights of its edges. Steiner Tree is well known to be NP-complete and is undoubtedly one of the most studied problems in (applied) computer science.

We observe that many approximation algorithms for Steiner Tree follow a similar scheme (meta-algorithm) and perform (exhaustively) a similar routine which we call *star contraction*. Here, by a star contraction, we mean finding a star-like subgraph in (the metric closure of) the input graph minimizing the ratio of its weight to the number of contained terminals minus one; and contract. It is not hard to see that the well-known MST-approximation seeks the best star to contract among those containing two terminals only. Zelikovsky's approximation algorithm follows a similar workflow, finding the best star among those containing three terminals.

We perform an empirical study of star contractions with the relaxed condition on the number of terminals in each star contraction motivated by a recent result of Dvořák et al. [Parameterized Approximation Schemes for Steiner Trees with Small Number of Steiner Vertices, STACS 2018]. Furthermore, we propose two improvements of Zelikovsky's 11/6-approximation algorithm and we empirically confirm that the quality of the solution returned by any of these is better than the one returned by the former algorithm. However, such an improvement is exchanged for a slower running time (up to a multiplicative factor of the number of terminals).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Steiner tree, approximation, star contractions, minimum spanning tree

Digital Object Identifier 10.4230/LIPIcs.IPEC.2020.16

Related Version A full version [19] of this paper is available at https://arxiv.org/abs/2002.03583.

Supplementary Material
https://github.com/JohnNobody-3af744f30980b7458372/star-contractions

Funding Students were supported by Charles University student grant SVV-2017-260452 and GAUK 1514217.
Dušan Knop: Supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".
Tomáš Masařík: have received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme Grant Agreement 714704.

Acknowledgements We thank the authors of the Boost library [29], which we use in our code, for their work and effective implementation of many graph algorithms. We thank Tomáš Toufar for consultations in the early stages of preparation of the experiments in the paper as well as for the implementation of a part [20] which was partially used in our code. We thank the PACE challenge which was a good motivation and inspiration for the development of practical algorithms for the Steiner Tree problem. The results of this challenge motivated us to develop the experiments presented in this paper. We also thank anonymous referees for interesting feedback as well as for pointing us to several interesting directions for future improvements and related results. Part of the work was carried out while D. Knop and T. Masařík were at the University of Bergen.

1 Introduction

In the Steiner Tree problem an edge weighted graph $G = (V, E)$ is given together with the set of *terminal* vertices $R \subseteq V$; the non-terminal vertices are called *Steiner vertices*. The task is to find a connected subgraph of G containing all terminals and minimizing the sum of weights of its edges. Steiner Tree was among the first problems shown to be NP-complete [22] and is one of the most studied problems in computer science since then.

Steiner Tree	
Input:	A graph $G = (V, E)$, a set of terminals $R \subseteq V$, and a weight function $w\colon E \to \mathbb{N}$.
Solution:	A Steiner tree $F \subseteq G$ containing a path between any two terminals $s, t \in R$.

Steiner Tree is important not only as an interesting graph-theoretic problem, but it has many real-world applications e.g. in network design or VLSI design [21]. Thus, it is extensively studied by both theoreticians and practitioners. Many theoretical results are studying (approximation) algorithms for Steiner Tree; for an overview, see e.g. a survey [21]. We now discuss a few theoretical results important for our work.

Star Contraction and Approximation Algorithms. For an edge-weighted graph $G = (V, E)$ the *metric closure of* G, denoted $\mathrm{mc}(G)$, is the complete graph with the vertex set V with weight of an edge $\{u, v\}$ equal to the length of a shortest path between u and v in G. The most basic approximation algorithm for Steiner Tree is based on finding a minimum spanning tree (MST) in the metric closure of the input graph [25]. Improvements and variants of MST heuristics solving the Steiner tree problem were subsequently examined [9]. An MST heuristic was later improved by Zelikovsky [33] who used the finding of augmenting stars containing three terminals to improve the algorithm of Kou et al. [25] and was the first to beat the barrier of 2 for the approximation ratio. Here an augmenting star consists of a Steiner vertex and exactly three terminals such that if we contract the just defined star into a terminal and compute the weight of an MST, the weight of the thus obtained MST together with the weight of a contracted star is strictly smaller than the weight of the former MST. This approach was later improved by Borchers and Du [6]. Furthermore, the current best theoretical approximation algorithm of Byrka et al. [7] with approximation ratio $\ln(4) + \varepsilon$ is in fact based on star contractions as well.

We observe that the above-mentioned algorithms not only use star contractions as the main tool but, on top of this, most of these algorithms follow a very similar meta-algorithm – see Algorithm 1 and Example 1 below. There, we argue that the simplest algorithm we consider, the MST-approximation, can be described in the framework given in Algorithm 1. We justify the same for (our modification of) Zelikovsky's algorithm in Section 2, Example 2.

Algorithm 1 A unifying high-level framework of selected approximation algorithms for Steiner Tree. The `find_best_star()` function finds a star C with at most k terminals which among all such stars achieves the lowest value under the evaluation function `eval()`. The `contract()` function (usually) contracts the star C and assigns a partial (contracted) solution to S' (note that C and S' could possibly be different e.g. C can be in the metric closure of G').

Input: $G = (V, E)$, set of terminals R, parameters $k, \tau \in \mathbb{N}$, and functions `contract()`, `eval()`, `finish()`
Output: A Steiner tree
1 $G' \leftarrow G,\ R' \leftarrow R,\ S \leftarrow \emptyset$
2 **while** $|R'| > \tau$ **do**
3 $C \leftarrow$ `find_best_star(`G', R', k, `eval)`
4 **if** `eval(`C`)` $< \infty$ **then**
5 $G', R', S' \leftarrow$ `contract(`G', R', C`)`
6 $\mathcal{S} \leftarrow \mathcal{S} \cup \{S'\}$
7 **else break**
8 **return** `finish(`G, $\mathcal{S}$, R`)`

▶ **Example 1** (MST). Observe that in order to find a spanning tree of minimal weight it suffices to find the best star with two terminals, that is, the cheapest edge (between terminals) in the metric closure of the given graph. Furthermore, if we then contract such a path, we reduce the size of the terminal set by one. Thus, one can set the parameter $k = 2$ and $\tau = 1$ (as we perform star contractions exhaustively). The `eval()` function gives the length of the shortest path (i.e., the total weight of the proposed 2-star), the `contract()` function contracts, and the `finish()` function contracts the given collection of 2-stars.

Unbounded Size of the Best Star. A recent result of Dvořák et al. [15], which proposes a novel algorithm in the framework of parameterized approximations, also falls in the framework suggested in Algorithm 1. Surprisingly, their algorithm uses an unbounded value of the parameter k, the number of terminals in the best star. Their algorithm, given a parameter p and the desired approximation ratio $\varepsilon > 0$, runs in time $f(p, \varepsilon)\,\mathrm{poly}(|G|)$ and outputs a solution of cost at most $(1 + \varepsilon) \cdot \mathrm{OPT}(p)$, where $\mathrm{OPT}(p)$ is the value of an optimal solution that uses at most p Steiner vertices. Let us now discuss in more detail why the algorithm of Dvořák et al. follows the proposed framework; we discuss further technical details later (see Section 2). First we set the parameters $k = \infty$ and $\tau = c \cdot \frac{p^2}{\varepsilon^4}$ for a suitable constant c.

`eval()` Let C be a connected subgraph of G', let $w(C)$ be the total weight of edges in C, and let $R_C \subseteq R'$ be the set of terminals contained in C. The function `eval(`C`)` returns the value $\frac{w(C)}{|R_C|-1}$.

`find_best_star()` Since the parameter $k = \infty$, the function returns a connected subgraph C minimizing the value $\frac{w(C)}{|R_C|-1}$ among all connected subgraphs with at least two terminals.

`finish()` We first contract all subgraphs C obtained so far (i.e., we construct the graph G'). Then the algorithm of Fuchs et al. [16] is invoked on G'.

It is worth noting that the algorithm of Fuchs et al. computes an optimal solution in time $f(\tau) \cdot \mathrm{poly}(|G'|)$. Note that a similar running time has been achieved already by Dreyfus and Wagner [12]. We conclude that the best-star contraction is a popular and successful technique in the design of approximation algorithms for Steiner Tree. We refer to the work of Chimani and Woste 2011 [8] for an experimental comparison of contraction-based techniques known at that time. Recently, Beyer and Chimani 2019 [4] conducted another experimental study, where they compare approximation algorithms with approximation ratio

better than 2. The underlying technique within the compared algorithms (greedy as well as linear programming based approaches) is the contraction of *k-restricted full components*, components of at most k terminals where the set of leaves and terminals coincide, for some $k \geq 3$. For most of the considered algorithms, their strongest theoretical approximation bounds are only achieved for $k \to \infty$, but also, the running time is exponentially dependent on k, which makes it infeasible in practice.

Interestingly, in [8] conclude that the simplest and oldest algorithm with the weakest theoretical guarantee among the algorithms they considered has the best performance in practice. This was the already mentioned 11/6-approximation algorithm by Zelikovky [33] that we are considering in our study. They speculated in the conclusions that the reason might be that the larger values of k might not be feasible to compute in practice (A large k is also identified as the main practical obstacle in [4]). Instead, Chimani and Woste suggested that some clever algorithmic choices might help to overcome this issue in the future. In this light, we believe that our approach of best stars contractions might give such guidance. We are posing the following natural questions:

1. Do the star contractions behave well in practice? Specifically, is it possible to improve the total weight of a solution returned by well-studied heuristics (e.g. MST approximation) significantly when we first perform a few rounds of best star contractions?
2. Is it common to find large (nearly) best stars? That is, is there a significant fraction of all contracted stars containing more than e.g. 5 terminals?
3. Is there any point of Pareto optimality? For example, is it possible to reduce the number of terminals by 20% using only 10% of the total work? Ideally, while also improving the cost of the solution by 80%? Here 100% improvement is represented by performing best star contractions until only a single terminal is left?

Our Experiment. In what follows, we refer to Algorithm 1. We run the algorithm in steps – invocations of the while-loop – and each time we call the `contract()` function. We find a solution using all of the aforementioned methods. We collect the data and, e.g., for MST on public instances from PACE Challenge 2018 we aggregate statistics (see Figure 2). Furthermore, we measure the sizes of contracted stars and the performance of `find_best_star()`, since this is the most time-consuming step in the algorithm. It is worth pointing out that we implement some standard heuristics (see Section 2.1) which we use to preprocess the input, that is, all our data is collected on the already preprocessed instances.

Dataset. We evaluate the algorithm and present our results for the set of public instances of PACE Challenge 2018 in Section 3. According to the report from PACE Challenge 2018 [5], the set of instances in Track C consists of the hardest instances of Steinlib and from real-world telecommunication networks by Ivana Ljubic's group at the University of Vienna. It should be noted that similar sources were used for the DIMACS Challenge [1]. By that time, a majority of the selected instances cannot be solved within one hour by the state-of-the-art program and in several cases, the actual optimum was unknown. For more discussion about the chosen dataset, please consult the report from the PACE Challenge 2018 [5]. Furthermore, we evaluate our experiments on rectilinear instances from ORLib [3] in the full version of the paper.

Preliminaries. We give a brief recapitulation of graph theory terminology used in this work; for the basic notation, we refer the reader to monographs [26, 10]. All graphs are undirected without loops and multiple edges. If we argue about algorithmic complexity of a certain routine or the amount of memory needed in order to store some data for a graph G, by n we denote the number of vertices of G and by m we denote the number of edges of G. For

a graph $G = (V, E)$ and an edge $e = \{u, v\}$ if we *contract* e (denoted as G/e), we create a new graph with the vertex set $(V \setminus \{u, v\}) \cup \{z\}$, where z is a newly introduced vertex. The edge set of the resulting graph consists of all edges in E not incident to any end-vertex of e together with the newly introduced edges $\{w, z\}$ for every edge $\{w, u\}$ as well as for every edge $\{w, v\}$, where the weight of a newly created edge is the same as the weight of the edge $\{w, u\}$, $\{w, v\}$, respectively. Note that if the above operation is about to create multiple edges we simply keep the one with a lower weight. For a graph $G = (V, E)$ and a vertex v with exactly two neighbors in G by *suppressing* v we mean changing G into a new graph as follows. The new vertex set is $V \setminus \{v\}$ and we delete all edges incident to v. Finally, we insert edge $\{x, y\}$ if both $\{x, v\}, \{v, y\} \in E$ with weight $w(\{x, y\}) = w(\{x, v\}) + w(\{v, y\})$.

1.1 More Details on Past Implementation Challenges

Since Steiner Tree has many applications, it received attention among practitioners and in operations research. One particular example can be e.g. the specialized module SCIP-Jack [17] in the SCIP tool for solving (mixed) integer linear programs. In the 11$^{\text{th}}$ DIMACS Implementation Challenge [1] various variants of Steiner Tree formed the central topic of the challenge. Among others, e.g. the basic version Steiner Tree, geometrical versions (e.g., rectilinear instances), and prize-collecting variants were tackled. One can read in the description of the DIMACS Implementation Challenge:

> *DIMACS Implementation Challenges address questions of determining realistic algorithm performance where worst-case analysis is overly pessimistic and probabilistic models are too unrealistic: experimentation can provide guides to realistic algorithm performance where analysis fails.*

Last but not least, one track of the PACE Challenge 2018 [5, 2] was completely devoted to Steiner Tree with three specialized branches. In PACE Challenge 2018 there was an approximation branch and two exact branches – one with the additional promise of a small number of terminals and in the other, a tree-decomposition of the input graph of small tree-width was given.

2 Implementation Details and Heuristics

In this section, we describe our implementation and improvements of finding the Best Star as proposed in [15] (Section 2.2), the approximate algorithms used to finish the solution after the application of the Best Star Algorithm (Section 2.3), and also the heuristics we used to preprocess the instances (Section 2.1). Last but not least, we discuss a few simple yet in practice well-performing modification of Zelikovsky's algorithm (Section 2.3).

2.1 Heuristics

All heuristics we used are deterministic and ensure that the optimal value of instance before and after applying them is the same. Namely, we used the following well-known ones:

1. We contract all edges e with $w(e) = 0$ at the very beginning. Thus we assume in the rest that all weights are positive.
2. We remove all Steiner vertices of degree 1 and suppress Steiner vertices of degree 2.
3. We contract the edges incident to terminals of degree 1.
4. Contract an edge $e = \{s, t\}$ between two terminals if $w(e)$ is minimal among the edges incident to s or t.
5. *Shortest Path Test (SPT):* Delete an edge $e = \{u, v\}$ if $w(e)$ exceeds the length of the shortest path between u and v.
6. *Terminal Distance Test (TDT):* see below.

Algorithm 2 Pseudocode of the used preprocessing.

```
1  Function quick_heuristics(G):
2      run ← true
3      while run do
4          run ← false
5          run ← contract_zero_edges(G)
6          run ← run ∨ delete_degree_one_Steiner(G)
7          run ← run ∨ contract_the_only_edge_incident_to_term(G)
8          run ← run ∨ contract_the_cheapest_edge_between_two_terminals(G)
9  Function preprocessing(G):
10     quick_heuristics(G)
11     SPT(G)
12     quick_heuristics(G)
13     TDT(G)
14     quick_heuristics(G)
15     SPT(G)
16     quick_heuristics(G)
```

Heuristics (1) up to (4) are implemented using straightforward iteration over all edges or vertices of the graph, and if any of them succeeds, we again rerun all of these. This iteration blows up theoretical time complexity by a factor of n but in practice, only very few iterations yield an irreducible instance. In Algorithm 2 we encapsulate these into the `quick_heuristics()` function. For performance reasons, we split the SPT heuristic into two parts: first, it checks only paths consisting of two edges which is quite efficient ($O(n^2)$ because we store edges incident to every vertex in sorted order) and then we check paths of all lengths which requires to run Dijkstra's algorithm [11] from every vertex and is therefore much slower.

It is worth noting that both SPT and TDT can benefit from rerunning but due to their time complexity and usually lower number of improvements, we do not use these heuristics exhaustively. Instead, we run SPT, TDT, and SPT once more and execute `quick_heuristics()` at the beginning, between them, and at the end. See the `preprocessing()` function in Algorithm 2.

Terminal Distance Test. The TDT heuristic was introduced in [30]. The basic idea is the following: Let (W, W') be a partition of vertices such that both W and W' contain some terminal and each of them is connected, let be e and f the shortest and the second shortest edge of the cut induced by (W, W'). If there is a path connecting some terminal $t \in W \cap T$ and $t' \in W' \cap T$ which uses e and is no longer that f then there exists an optimal solution which uses edge e. Observe that while it is easy to verify the correctness of this heuristic, it does not give us directly an effective algorithm. Koch and Martin [23] point out that it is possible to implement TDT in time $O(|V|^3)$. Furthermore, they claim that the time complexity can be further improved to $O(|V|^2)$ as is described in the thesis [13]. However, we were unable to access the thesis.[1] Consequently, we describe our implementation in Algorithm 3 for the future reference.

[1] We thank to an anonymous referee for providing us with a reference [14] to an implementation of TDT heuristic running in time $O(|E| + |V| \log(|V|))$. It will be a part of the planned improvements in our experiments.

Our implementation is based on the data structure for dynamic edge 2-connectivity of Westbrook and Tarjan [32]. This structure has amortized time complexity $O(\alpha(m))$ per operation and supports edge addition and check whether two vertices belong to the same (2-connected) component. The time complexity of Algorithm 3 is $O(nm \log n)$ when Dijkstra's algorithm is implemented with d-regular heap.

Algorithm 3 Implementation of Terminal Distance Test.

```
1  Function test_edge(G, e, f, X):
2      u, v ← e
3      Remove e from G
4      t_1 ← terminal closest to u                    // Dijkstra's algorithm
5      t_2 ← terminal closest to v
6      if d(t_1, u) + w(e) + d(v, t_2) ≤ w(f) then
7          X ← X ∪ e
8      Add e back to G
9  Function TDT(G):
10     E' ← sort(E(G), w(a) < w(b))
11     C ← structure for 2-connectivity
12     X ← ∅
13     foreach e ∈ E' do
14         B ← add_edge(C, e)   // returns edges which were bridges before addition
           of e but no longer are
15         foreach b ∈ B do
16             test_edge(G, b, e, X)
17     Buy edges in X
```

2.2 Finding the Best Star

As already mentioned in Section 1, Algorithm 1 repeatedly finds and contracts a so called best star (according to a certain ratio) in G. Here, we first give the definition used by Dvořák et al. [15]. Later, we discuss a further additional practical extension of the former definition and describe our implementation in detail. Let G be a graph and R the set of terminals in G. For a vertex c and a set $R' \subseteq R$ we define a *star* (centered at c and a terminals set R') which we denote $\mathrm{st}(c, R')$. We define a *ratio of star* $\mathrm{st}(c, R')$ as

$$\mathrm{r}\left(\mathrm{st}(c, R')\right) = \frac{\sum_{t \in R'} \mathrm{dist}(c, t)}{|R'| - 1},$$

where $|R'| \geq 2$ and $\mathrm{dist}(c, t)$ is the weight of the edge $\{c, t\}$ in $\mathrm{mc}(G)$. The best ratio achievable for a star centered at c is $\mathrm{r}(c) = \min_{R' \subseteq R, |R'| \geq 2} \mathrm{r}(\mathrm{st}(c, R'))$ and a *best star centered at* c is any minimizer of the defining expression. The *best star in the graph* G is any star $\mathrm{st}(c, R')$ minimizing $\mathrm{r}(G) = \min_{c \in V} \mathrm{r}(c)$.

Clearly, the smaller the ratio is the better. On the other hand, in our experiments, there were many ties and thus we further extend this definition in the case there are more minimizers of the best ratio in G. We introduce a second measurement taking into account the number of terminals contained in a star. Intuitively, the more terminals it contains the better. Thus the *best star in the graph* G is any star $\mathrm{st}(c, R')$ with ratio $\min_{c \in V} \mathrm{r}(c)$ which maximizes $|R'|$.

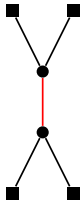

Figure 1 Simple graph containing four terminals (represented by squares) and two Steiner vertices (discs). Suppose all the edges have unit weight. When computing the weight of a star centered at any of the two Steiner vertices using the metric closure, we count the weight of the red edge twice. As a consequence, one gets two possible stars with ratio 2 – one containing only the two terminals connected directly to the assumed Steiner vertex and the other containing all four terminals. However, the second described star should better have a ratio of 5/3 (which is better than 2).

It is worth noting that if the best star with center c contains k terminals, then it contains k terminals that are closest to c in $\mathrm{mc}(G)$ [15, Lemma 6]. Even though the definition of the best star uses $\mathrm{mc}(G)$, in practice, we cannot afford to compute and store it due to its size (quadratic in n). Instead, we utilize Dijkstra's algorithm [11] to compute the best star centered in a given vertex. In total, we obtain running time $O(mn + n^2 \log n)$ per one round, that is, for one execution of the main loop in Algorithm 1. Furthermore, we use several heuristics to improve the running time significantly in practice. These heuristics employ memorization and early termination, among others; refer to Algorithm 4. Let $\mathrm{r}^{\mathrm{cur}}(G)$ denote the best so-far computed ratio, i.e., the best ratio among all already computed stars. By slightly overloading the notation when searching for the best star centered at c we let $\mathrm{r}^{\mathrm{cur}}(c)$ denote the ratio of the best so far computed star centered at c. We use this notation to describe practical heuristic improvements:

1. We stop the execution of Dijkstra's algorithm when the current distance from the source (center of the star) is strictly greater than $\mathrm{r}^{\mathrm{cur}}(c)$.
2. For every vertex $c \in V$ we store the best star centered at c in between the rounds.
3. We (re)compute the best star at c only if the stored one could have been affected by a star-contraction performed in the previous round. We can do this since a single star-contraction affects only a small (local) part of the graph.

Overestimating Star Weight. It is worth noting that the best star as described in [15] is a star in metric closure containing some number of terminals closest to its center (and minimizes the star ratio). Note that this clearly may overestimate the weight of such a star as well as its ratio (see Figure 1). Observe that if the best found star contains at most three terminals, then its weight is always estimated correctly. While this estimate is sufficient for the purpose of theoretical analysis, it may affect the overall behavior of the algorithm. We would like to point out that in our implementation of a star-contraction ($\mathrm{st}(c, R')$) we contract edges of an MST containing R' instead of contracting the star itself. Clearly, this modification can only decrease the cost of the single round. On the other hand, it is not clear how this "greedy" improvement affects the overall performance of the whole process. We propose a way to overcome the overcounting issue.

Improved Stars. Using Dijkstra's algorithm, we are recursively searching for a new terminal within the threshold distance and building the best star for each vertex. We start by setting the given vertex as the origin of Dijkstra's algorithm. Whenever we encounter a terminal

such that it forms a better star together with the so-far best star originating in the given vertex, we add it to the constructed star and start over while setting the whole star as the new origin for Dijkstra's algorithm.

Algorithm 4 A pseudocode for The Best Star function.

```
1  Function find_best_star_v(G,R,v,r):
2      weight ← 0                                          // sum of distances
3      nter ← 0                                            // current number of terminals
4      while (w,d) ← dijkstra_next(G,v) do
5          if nter ≥ 2 ∧ weight/(nter − 1) < d then return weight/(nter − 1)
6          if d > 2r then return lbound(d)
7          if w ∈ R then
8              weight ← weight + d
9              nter ← nter + 1
10 Function find_best_star(G,R):
11     r^cur(G) ← ∞
12     foreach v ∈ V do ratio[v] = ∞
13     foreach v ∈ V do
14         if invalid(v) ∨ (ratio[v] < r^cur(G) ∧ lbound(ratio[v])) then
15             ratio[v] ← find_best_star_v(G,R,v,r^cur(G))
16         if r^cur(G) > ratio[v] then
17             r^cur(G) ← ratio[v]
18             star_center ← v
19     return (star_center, r^cur(G))
```

2.3 Finishing the partial solution

The original algorithm of Dvořák et al. [15] performs star contractions until the number of terminals decreases under a threshold depending only on the desired approximation ratio ε and the number of Steiner vertices in (some) optimal solution. An exact algorithm is used to complete the solution when the number of terminals dropped below the threshold. While this is a very natural theoretical approach, it is not suitable for practical use for the following reasons:

1. Both discussed exact algorithms are based on the dynamic programming, which makes them quite slow in practice (mostly intractable for instances with more than 20 terminals).
2. The threshold depends on the number of Steiner vertices in (some) optimal solution and we, in general, have no good upper bound on it.

Instead, for the purposes of the evaluation, we choose the following algorithms which we run after every contraction:

- **MST:** The usual well-known minimum spanning tree approximation – we take a subgraph of the metric closure induced by terminals and find its minimum spanning tree. Note that the implementation does not compute whole metric closure but computes Voronoi regions of the terminals instead [28].. Then an auxiliary graph is constructed using terminals of the original graph as vertices and adding an edge for every edge $\{u, v\}$ crossing between Voronoi regions with length $d(u) + w(u, v) + d(v)$, where $w(\cdot, \cdot)$ is the length of an edge and $d(\cdot)$ is the distance to a closest terminal.

- **MST+:** We calculate MST and then improve its solution by taking its terminals and branching vertices (Steiner vertices with the degree at least 3 in the solution), marking them all as terminals and running MST on this modified instance. It is easy to see that solution of MST on this modified instance is never worse than the solution we began with because the original solution is a spanning tree of the modified instance. We repeat this while the solution is improving.
- **Zelikovsky:** Zelikovsky's algorithm [33], which augments the MST solution using stars with 3 terminals, was the first algorithm with a better approximation ratio than 2. The algorithm proceeds in rounds. Each round it looks at all 3-stars, selects the star s which maximizes the so-called "win" which is $mst(G) - mst(G/s) - d(s)$ (where $mst(G)$ is the weight of MST solution of instance G, G/s denotes G after contraction of terminals in s and $d(s)$ is the weight of star s), and if the win of the best star is strictly positive, it contracts it and starts another round. Otherwise, it stops, and returns MST on all terminals and selected star centers. (Computing the MST at the end is needed to ensure that the solution is really a tree.) The original version of the algorithm computes the best center for every triple of terminal and weight of such a star at the very beginning and runs in $O(n(m + n \log n + t^2) + t^4)$ time and requires $O(m + t^3)$ extra space.
- **Zelikovsky−:** This modification recomputes distances of triples in each round instead of precomputing them in advance. Unlike the usual Zelikovsky's algorithm where the triplets are only upper bounds, here we have optimal values in each round. This algorithm is slower ($O(nt(m + n \log n + t^2))$), but the memory requirement is $O(m)$ smaller.
- **Zelikovsky+:** This modification differs from Zelikovsky− only by the application of MST+ instead of MST at the end.

▶ **Example 2** (Zelikovsky-)**.** The Zelikovsky's algorithm also fits into the star-meta-algorithm framework described in the introduction: The parameters are $\tau = 2$ and $k = 3$, `eval()` function returns $-win$ (or ∞ for nonpositive win), the `contract()` just contracts the star and adds center of the star to S, and the `finish()` computes MST of $R \cup S$. Note that Zelikovsky− variant is more natural as it finds the best star each round, whereas the original Zelikovky's algorithm precomputes values of all stars with 3 terminals.

2.4 Comparison with state-of-the-art results

To put our study in the context, we compare its behavior with other programs competing in the PACE Challenge 2018. We describe the comparison system as was proposed for the PACE challenge 2018 [5]: A time-limit to output a solution was set to 30 minutes – we call this a run. Afterward, each run received points according to the fraction of the value of the returned solution to the best solution known. The results are aggregated over all instances, see Table 1. The implementation of best star contractions ended up at the 4th place in this comparison. It is important to note that the algorithm was enhanced with local search heuristics. We give a brief description of the particular implementation considered in the comparison (see [20] for the detailed description and the implementation).

After the initial heuristics described in Section 2.1 were exhausted, the program spends exactly 10 minutes performing the best star contraction algorithm. The rest of the available time was spent on the local search heuristics whose description follows.

Details on Local Search Heuristics. We implemented the two following randomized local search heuristic. Those heuristics run until the dedicated time was up. Then the best solution was returned. As the second heuristic is much more time consuming than the first one we run it only sparsely.

Table 1 An aggregated comparison of the overall performance of the algorithms participating in PACE Challenge 2018 Track C. We exchange the names of the teams with a summarizing name of the main method they used. For more implementation details of their algorithms, see the report [5] that also contains links to individual implementations and their comprehensive descriptions.
**Shortest Path Heuristic*: Pick one terminal as a root and repeat the following: Find the closest terminal to the root and contract the shortest path from this terminal to the root.

Algorithm	Score
Evolution Algorithm	99.91
MIP solver + Heuristics (SCIP-Jack [24])	99.89
Iterated Local Search	99.78
Best Star Contractions with Local Search [20]	**99.70**
Zelikovsky [33]	98.93
Simulated Annealing	98.27
Random Generation + Local Search	97.54
Shortest Path Heuristic*+ Local Search	97.15
Mehlhorn 2-approximation [27] + Watel and Weisser k-Approximation for the directed Steiner Tree Problem [31]	96.92
Shortest Path Heuristic*	94.57
Primal-Dual 2-approximation + Local Search	94.37
Contract Random 2-terminal Shortest Path + MST	82.61
Ant Colony Optimization	80.73

- **Local search using MST+-approximation.** We randomly select a couple of additional Steiner vertices to be added to the branching Steiner vertices of the current solution. Then we run the MST+ algorithm on them.
- **Local search using Dreyfus-Wagner partition.** Inspired by Dreyfus-Wagner FPT algorithm [12] we derive a heuristic that obtains the partition of the vertices given one of the promising solutions and then computes an optimal Steiner tree on this structure.

3 Outcomes of our Experiments

In this section, we perform our main tests that are carried out on the instances from the PACE Challenge 2018, Track C.[2][3] Aggregated data from the measurements are provided in Figure 2 and the corresponding data in the full version. There, star contractions are executed in rounds and in every round, all heuristics (from Section 2.3) are executed so

[2] It is worth noting that all the data, as well as the corresponding charts for all instances, are available in the repository containing our implementation code `https://github.com/JohnNobody-3af744f30980b7458372/star-contractions`.

[3] Due to time constraints the tests involving Zelikovski's algorithm were performed only on 123 out of 200 instances from the PACE Challenge 2018. The list of instances used in each experiment, as well as all the results and charts, are available in the above-mentioned git repository. The problem was that running Zelikovsky's algorithm after each best star contraction took too long on large instances. We provide figures where tests (excluding Zelikovsky's algorithm) were performed on almost all instances (excluding instances number 193, 196, 197, and 198 only) in the full version. Those four instances were still too large, even for the rest of the tests. We stress out that the outcomes of those tests are relatively similar to those on the limited number of instances.

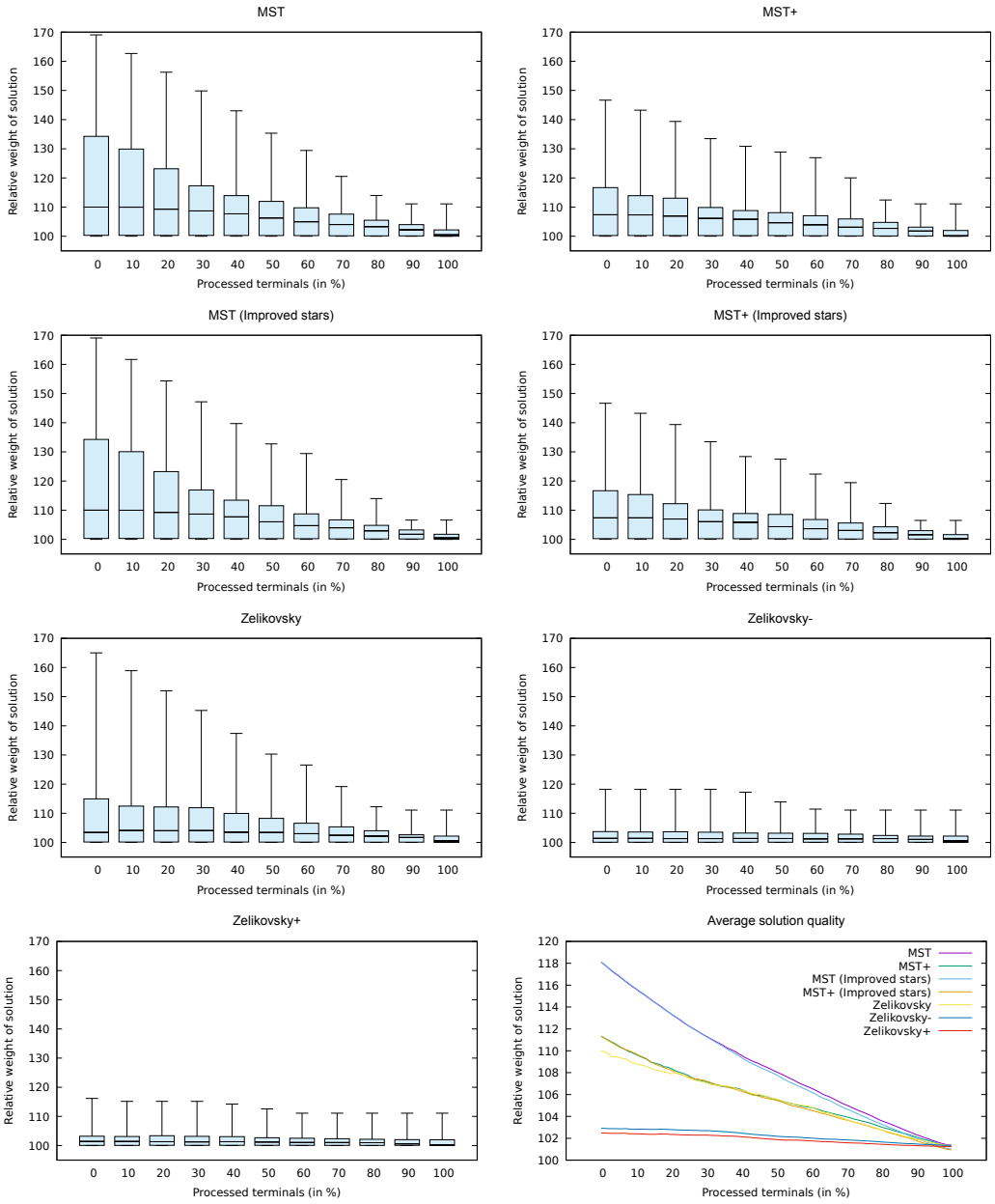

Figure 2 This chart shows the performance comparison of star contractions and MST heuristics on PACE Challenge 2018 instances. The x-axis represents the number of star contractions in percent before MST was computed. The zero value is MST heuristics after preprocessing only. Hundred denotes a result obtained by star contractions till one vertex remains in the graph. The y-axis represents the quality of the solution again in percentage where the hundred is the best solution we obtained during our experiments (not only in this comparison). As we pointed out, the best solution we are comparing to was derived using a local search algorithm, so optima are represented by more or less the current state-of-the-art results. The top line is the maximum in our dataset, the colored box represents data points from the first to the third quartile with the line in the middle denoting the median, and the line at the bottom is the minimum. The last plot shows arithmetic averages of the same data combined into a single plot for easier comparison.

Table 2 The total number of stars containing two to ten terminals contracted during the execution of the algorithm on all PACE Challenge 2018 instances.

# of terminals	2	3	4	5	6	7	8	9	10	> 10
# of basic stars	99965	23921	1721	683	246	135	197	126	149	1158
# of improved stars	91957	15346	4371	2070	969	539	410	263	218	1285

that the overall performance can be compared. Experiments are done separately for basic stars and only some of them are repeated for improved stars. An important thing to note is that the best solution for each input was derived out of the best of outcomes from all the performed experiments, including additional local search heuristics described in Subsection 2.4. Therefore, it represents more or less the current state-of-the-art. Also, "the worst" solution was obtained using the heuristics described in Section 2.1.

The basic outcome of our experiments is the chart for MST. It shows that the best star algorithm significantly improves the performance of an MST approximation. Here, the star contractions help to improve the quality of some solutions for more than 57%. The improvement is by more than 12% on average and it is worth pointing out that the number outliers is reduced significantly as well (see Figure 2) and thus we answered Question 1 positively.

We conclude that MST+ could replace MST for all purposes where a slow-down by a small multiplicative constant does not play a significant role. MST+ is not only better by definition, but also our experiments suggest that it outperforms MST quite significantly. In addition, it is still quite simple to code. Most importantly, Our measurements declare that it is approximately only 3 times slower than MST. This means it is (most of the time) negligible in practice since MST is computed within seconds on the current inputs. This is also supported by the fact that the computation of MST/MST+ takes only a fraction of the algorithms considered in this study. Moreover, star contraction combines very well with MST+ which improves not only the overall performance but, more importantly, a good solution is obtained much sooner. Besides, MST+ combines well with the local search algorithm presented in Subsection 2.4.

We answer Question 2 negatively. A vast majority of the contractions performed by Algorithm 1 on our instances are those of stars containing two or three terminals (see also Table 2 and Figure 3. It is worth noting that the best star containing only two terminals is found (on average) in more than 75% of all contractions performed during the execution of the algorithm.

As we have already observed, if during the algorithm's execution we only contract stars containing only two terminals, then the proposed algorithm returns a minimal spanning tree in the metric closure of the graph on terminals. Therefore, one should expect that if stars contracting more terminals are found and contracted during the execution, then the quality of the solution found should improve. Clearly, improved stars allow us to identify best stars containing more terminals; see Figure 3 and Table 2. We can see that the number of best stars containing more than 5 terminals increase from (roughly) 2% to 5%. Despite this improvement, overall performance is not much better than using the regular star contractions. However, the time consumption stays low (approximately three times as much, see Figure 4). In addition, improved stars improve the final solution quite significantly. They even cooperate well in combination with MST+. This is far the best method we studied when aiming at the smallest solution as it differs from the "best" solution by at most 6.49% and in the median by only 0.11%.

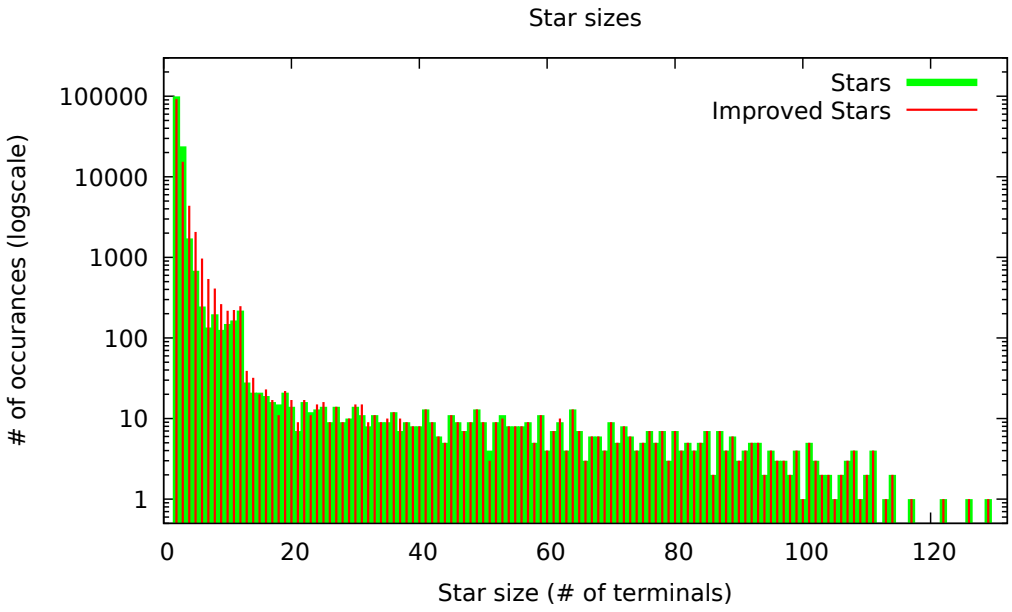

Figure 3 Star sizes on PACE Challenge 2018 instances with star contractions running until the end.

Despite our former beliefs that were supported by the results of Dvořák et al. [15], the answer to Question 3 seems to be also negative. Figure 5 indicates that there is no apparent threshold point for neither classical nor improved stars.

Driven by our experiments, we propose two variants of modifications of Zelikovsky's algorithm: Zelikovsky+, Zelikovsky-. These are based mainly on our insight that takes advantage of reformulating known algorithms in terms of star contractions. The key idea is to compute stars after each contraction as opposed to precomputing them. Of course, since we recompute a star to contract, the proposed variants are slower (roughly 3 times). However, they both achieve much better performance (i.e., the total weight of the solution found); see Figure 2. As it follows from the paragraph above, relaxed star contractions are not helping much since there are not so many large stars. A large improvement is achieved by combining it with the MST+ algorithm. Consult Figures 2 and 4 where one can compare MST+ with MST and our modification of Zelikovsky's algortihm. Surprisingly Star Contractions improve even the performance of the classical implementation of Zelikovsky's algorithm. However, this is easily outperformed by improved stars combined with the MST+ algorithm. On the other hand, our variants Zelikovsky− and Zelikovsky+ behaves reasonably well from the beginning and it seems that stars contractions have only a little to do with it. For example, Zelikovsky+ has a median that is only 1.04% worse even without any star contractions compared to 3.59% (for MST+). This good performance even without any star contractions is diminished by a slower running time which is comparable with many rounds of star contractions finished by MST+.

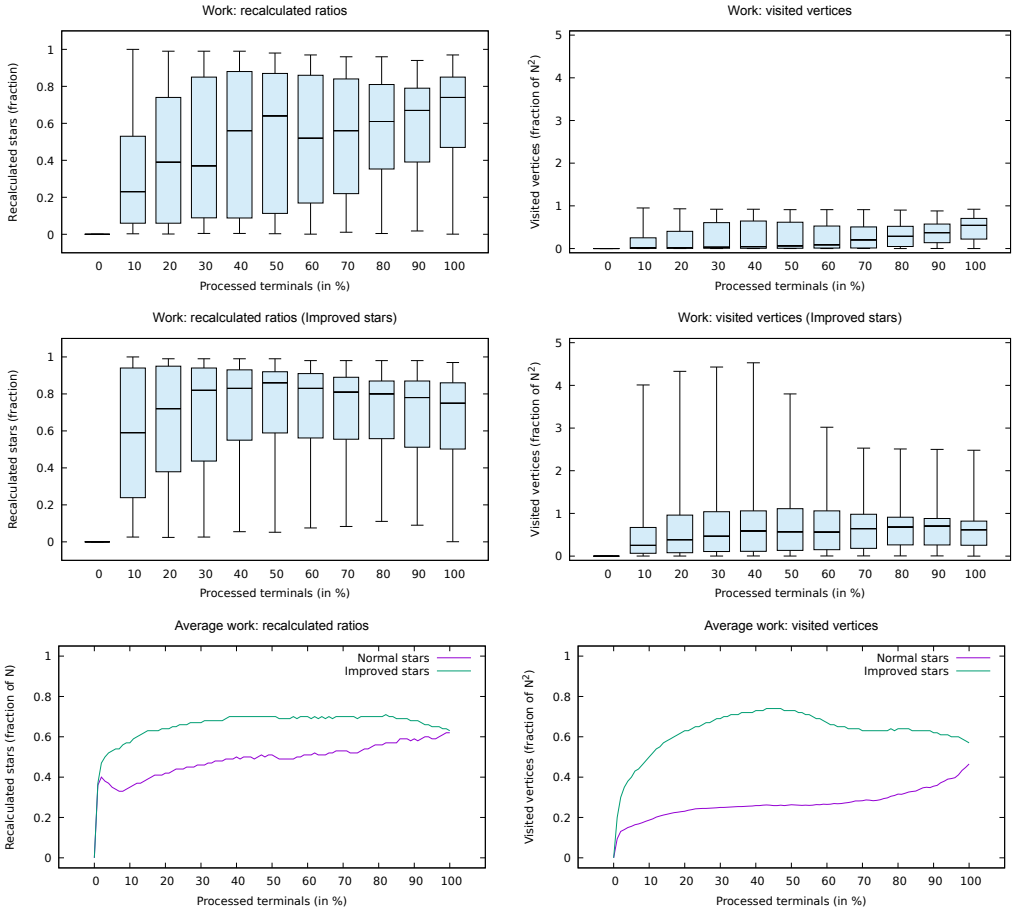

Figure 4 Work done on PACE Challenge 2018 instances.

4 Conclusions

In general, we have confirmed that contracting the best star improves the quality of the solution returned by the MST (MST+) algorithm. It seems that if we exhaustively apply contractions of best stars, we achieve a solution of slightly better quality than our modification of Zelikovsky's algorithm (i.e., Zelikovsky+ algorithm applied directly to the input). However, the running time of such approaches is comparable in practice. Unfortunately, unlike in classical Zelikovsky's algorithm, star contractions do not significantly help in our modifications. Importantly, MST+ heuristics should replace the classical MST since it outperforms it (by definition) without being much more complicated or time-consuming. Improved stars with MST+ do perform better when aiming for the best quality of the solution. They should be used whenever the slight increase in the running time is not important.

Future Work. Last but not least, our experiments suggest that our methods for lessening the time needed to compute the best star are useless when there are only a few terminals left in the graph since in such a case the computation is only local. Yet if this happens (according to Figure 4 this happens when about 30% of terminals are left), it is not possible to use the algorithm of Dreyfus and Wagner, since in such cases we still usually have more

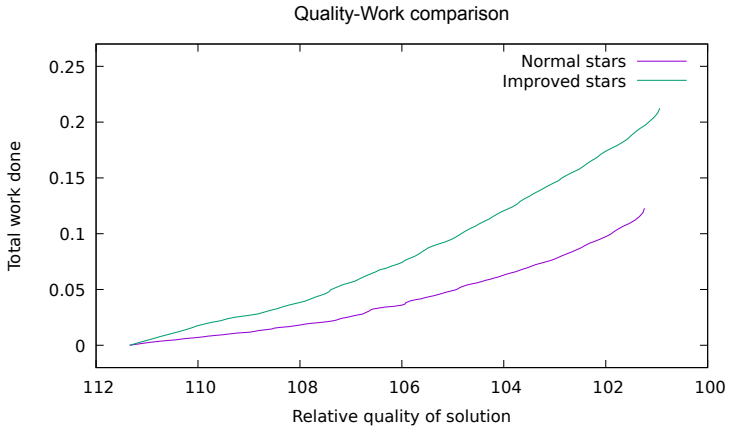

Figure 5 Work needed to get a solution of given quality using star contractions and MST+. (Y-axis is fraction of total maximum work done and work is measured as number of visited vertices during invocations of Dijkstra's algorithm, the work done by MST+ is negligible and hence ignored.)

than 30 terminals left—which is clearly intractable for larger instances. This brings us to the following question: Is it possible to use best stars to speed up (in both theory or practice) the algorithm of Dreyfus and Wagner while loosing only a bit in its precision? If yes, we hope that a suitable combination of the two algorithms can be used in practice. One can use recent improvements of Dreyfus and Wagner algorithm with the same worst-case running time but which behaves significantly well in practice on instances originated from VLSI design [18].

An interesting research direction is to augment the algorithm of Dvořák et al. [15] for Euclidean instances, since solutions to such instances should contain fewer Steiner vertices and thus the quality of the returned solution should increase. In a similar direction, we performed several basic tests for rectilinear instances from ORlib (see the full version). Interestingly, our approach works reasonably well on such specialized instances, significantly better than on general instances. However, we leave it for future work as the comparison with specialized heuristics for those instances is essential.

Yet another possibility is to, instead of contracting a subgraph with the best ratio, contract a subgraph with a slightly worse ratio which contains substantially many terminals. As our results suggest, the subgraph containing more terminals tends to improve the current as well as the final solution better. In a broader context, the algorithm of Dvořák et al. [15] cannot approximate Steiner Arborescence well, since even parameterized approximation is hard from parameterized complexity view [15]. Is this still true in practice?

References

1 11th DIMACS Implementation Challenge, 2011. URL: `http://dimacs11.zib.de/`.
2 PACE Challenge 2018, 2018. URL: `https://pacechallenge.wordpress.com/pace-2018/`.
3 John E. Beasley. Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. `doi:10.1057/jors.1990.166`.
4 Stephan Beyer and Markus Chimani. Strong steiner tree approximations in practice. *ACM J. Exp. Algorithmics*, 24(1), January 2019. `doi:10.1145/3299903`.
5 Édouard Bonnet and Florian Sikora. The PACE 2018 Parameterized Algorithms and Computational Experiments Challenge: The Third Iteration. In Christophe Paul and Michał Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages

26:1–26:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2018.26.

6 Al Borchers and Ding-Zhu Du. Thek-steiner ratio in graphs. *SIAM Journal on Computing*, 26(3):857–869, 1997. doi:10.1137/S0097539795281086.

7 Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *Journal of the ACM*, 60(1):1–33, February 2013. doi:10.1145/2432622.2432628.

8 Markus Chimani and Matthias Woste. Contraction-based steiner tree approximations in practice. In *Algorithms and Computation*, pages 40–49. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-25591-5_6.

9 Marcus Poggi de Aragão and Renato F. Werneck. On the implementation of MST-based heuristics for the steiner problem in graphs. In *Algorithm Engineering and Experiments*, pages 1–15. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45643-0_1.

10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

11 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.

12 Stuart E. Dreyfus and Robert A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971. doi:10.1002/net.3230010302.

13 Cees Duin. *Steiner's problem in graphs*. PhD thesis, University of Amsterdam, 1993.

14 Cees Duin. *Preprocessing the Steiner Problem in Graphs*, pages 175–233. Springer US, Boston, MA, 2000. doi:10.1007/978-1-4757-3171-2_10.

15 Pavel Dvořák, Andreas Emil Feldmann, Dušan Knop, Tomáš Masařík, Tomáš Toufar, and Pavel Veselý. Parameterized approximation schemes for steiner trees with small number of steiner vertices. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.26.

16 Bernhard Fuchs, Walter Kern, Daniel Mölle, Stefan Richter, Peter Rossmanith, and Xinhui Wang. Dynamic programming for minimum steiner trees. *Theory Comput. Syst.*, 41(3):493–500, 2007. doi:10.1007/s00224-007-1324-4.

17 Gerald Gamrath, Thorsten Koch, Stephen J. Maher, Daniel Rehfeldt, and Yuji Shinano. Scip-jack – a solver for STP and variants with parallelization extensions. *Math. Program. Comput.*, 9(2):231–296, 2017. doi:10.1007/s12532-016-0114-x.

18 Stefan Hougardy, Jannik Silvanus, and Jens Vygen. Dijkstra meets steiner: a fast exact goal-oriented steiner tree algorithm. *Mathematical Programming Computation*, 9(2):135–202, June 2017. doi:10.1007/s12532-016-0110-1.

19 Radek Hušek, Dušan Knop, and Tomáš Masařík. Approximation algorithms for steiner tree based on star contractions: A unified view, 2020. arXiv:2002.03583.

20 Radek Hušek, Tomáš Toufar, Dušan Knop, Tomáš Masařík, and Eduard Eiben. Steiner tree heuristics for PACE 2018 Challenge track C, 2018. URL: https://github.com/goderik01/PACE2018.

21 Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992. doi:10.1016/s0167-5060(08)x7008-6.

22 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Plenum, 1972. doi:10.1007/978-1-4684-2001-2_9.

23 Thorsten Koch and Alexander Martin. Solving steiner tree problems in graphs to optimality. *Networks*, 32(3):207–232, 1998. doi:10.1002/(SICI)1097-0037(199810)32:3<207::AID-NET5>3.0.CO;2-O.

24 Thorsten Koch and Daniel Rehfeldt. SCIP-jack, 2018. URL: https://github.com/dRehfeldt/scipjack/.

25 Lawrence Kou, George Markowsky, and Leonard Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15(2):141–145, June 1981. doi:10.1007/BF00288961.

26 Jiří Matoušek and Jaroslav Nešetřil. *Invitation to Discrete Mathematics*. Oxford University Press, Inc., New York, NY, USA, 1998.

27 Kurt Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988. doi:10.1016/0020-0190(88)90066-X.

28 Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 151–162, 1975. doi:10.1109/SFCS.1975.8.

29 Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

30 Eduardo Uchoa, Marcus Poggi de Aragão, and Celso C. Ribeiro. Preprocessing Steiner problems from VLSI layout. *Networks*, 40(1):38–50, 2002. doi:10.1002/net.10035.

31 Dimitri Watel and Marc-Antoine Weisser. A practical greedy approximation for the directed steiner tree problem. *Journal of Combinatorial Optimization*, 32(4):1327–1370, November 2016. doi:10.1007/s10878-016-0074-0.

32 Jeffery Westbrook and Robert E. Tarjan. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1):433–464, June 1992. doi:10.1007/BF01758773.

33 Alexander Z. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5), 1993. doi:10.1007/BF01187035.