STEINER TREES IN INDUSTRIES
D.-Z. Du and X. Cheng (Eds.) pp. – - –
©2000 Kluwer Academic Publishers

# Approximation Algorithms for the Steiner Tree Problem in Graphs

Clemens Gröpl, Stefan Hougardy, Till Nierhoff, Hans Jürgen Prömel
*Institut für Informatik*
*Humboldt-Universität zu Berlin, 10099 Berlin*
E-mail:
`{groepl,hougardy,nierhoff,proemel}@informatik.hu-berlin.de`

## Contents

Supported by Deutsche Forschungsgemeinschaft, Pr 296/6-2.

This paper appeared in: Steiner Trees in Industry, X. Cheng and D.-Z. Du (Eds.), 235-279, Kluwer Academic Publishers, 2001

## 1 Introduction

Given a graph $G = (V, E)$, a set $R \subseteq V$, and a length function on the edges, a *Steiner tree* is a connected subgraph of $G$ that spans all vertices in $R$. (It might use vertices in $V \setminus R$ as well.) The Steiner tree problem in graphs is to find a shortest Steiner tree, i.e., a Steiner tree whose total edge length is minimum. This problem is well known to be NP-hard [19] and therefore we cannot expect to find polynomial time algorithms for solving it exactly. This motivates the search for good approximation algorithms for the Steiner tree problem in graphs, i. e., algorithms that have polynomial running time and return solutions that are not far from an optimum solution.

In this paper we give a survey of the known approximation algorithms for the Steiner tree problem in graphs. After introducing the necessary notation in the next subsection, we will explain the minimum spanning tree heuristic. This algorithm was already known in 1968 (see page 24 in [14]) and finds a Steiner tree that is at most twice as long as an optimum solution. For more than twenty years, no better approximation algorithm had been found. In 1990 Zelikovsky proposed a simple greedy algorithm, using the idea of so called 3-Steiner trees. His approach was extended by Berman and Ramaiyer using $k$-Steiner trees. Since then, all approximation algorithms for the Steiner tree problem have used Zelikovsky's idea. This way, the approximation quality has been improved substantially over the last years. Table 1 summarizes the development.

Except for the randomized algorithm of Prömel and Steger [26], all these algorithms are based on simple greedy strategies. Analysing their performance ratio is a highly nontrivial task, however. In this survey we present several of these approximation algorithms along with an analysis of their performance ratio. They illustrate the fundamental concepts very nicely. Some general design principles and analysis tools that are common to these algorithms are presented prior to the algorithms.

| **Authors** | | **ratio** |
|---|---|---|
| Moore | 1968 | 2.0 |
| Zelikovsky | 1990 | 1.834 |
| Berman, Ramaiyer | 1991 | 1.734 |
| Zelikovsky | 1995 | 1.694 |
| Prömel, Steger | 1996 | 1.667 |
| Karpinski, Zelikovsky | 1996 | 1.644 |
| Hougardy, Prömel | 1999 | 1.598 |
| Robins, Zelikovsky | 2000 | 1.550 |

Table 1: Performance ratios for known approximation algorithms for the Steiner tree problem in graphs.

Berman and Ramaiyer [4] designed a family of algorithms $A_k$ which achieves a performance ratio of 1.734 for large $k$. The first algorithm we present is the algorithm $A_3$, which has a performance ratio of 1.834. Based on the analysis of $A_3$, we show that Zelikovsky's algorithm from 1990 has the same performance ratio.

The next algorithm we are going to describe is the relative greedy algorithm of Zelikovsky [34] which achieves a performance ratio of 1.694. Although the relative greedy algorithm is quite similar to algorithm $A_3$, a completely new kind of analysis is required. We will present this analysis in Section 3.

A powerful idea used by recent Steiner tree approximation algorithms is the concept of the so called *loss* of a Steiner tree, which was introduced by Karpinski and Zelikovsky in 1997 [20]. They obtained a performance ratio of 1.644 with an algorithm that minimizes the weighted sum of the length and the loss of a Steiner tree. Their idea was generalized by Hougardy and Prömel [18], resulting in an approximation algorithm that is at most a factor of 1.598 away from an optimum solution. Very recently, Robins and Zelikovsky [28] incorporated the loss of a Steiner tree into the relative greedy algorithm and were able to show that the resulting algorithm has a performance ratio of 1.550. At the time of writing this survey no better approximation algorithm for the Steiner tree problem in graphs is known. We will describe their algorithm and its analysis in Section 4.

A natural question about approximation algorithms is how small their performance ratio can get. Unless P = NP, the performance ratio of a polynomial time approximation algorithm for the Steiner tree problem in graphs cannot get arbitrarily close to 1. This is a consequence of the PCP-

Theorem [1] together with a reduction due to Bern and Plassmann [5]. The largest lower bound on the performance ratio, that can be achieved by an algorithm in polynomial time, is called the *approximation threshold.* In some effort over the last years it has been enlarged up to the currently best known value of 1.0074 [30]. We will describe these results in Section 6.

This lower bound is still fairly small compared to the best known performance ratio of 1.550. It is commonly believed that further improvement should be possible on both sides. The lower bound proofs are based on reductions which produce instances that are hard to solve by any polynomial time algorithm. However, the instances resulting from current reductions have a very special structure. There are several algorithms known which make use of this fact and obtain much better performance ratios than in the general case. A better understanding of such algorithms can on the one hand lead to better algorithms for the general case. On the other hand, the insight obtained from their analysis can help to extract the features of really hard instances and thus lead to better constructions for lower bound proofs in the future. We will present three analyses of algorithms for special instances in Section 5. None of the greedy algorithms in Table 1 is known to have a tight analysis in the general case, and therefore the actual performance might be much better than what has been proved so far. For two of the specialized algorithms we will present a worst-case instance which shows that the analysis is tight.

## 1.1 Basics

Given a graph $G = (V, E)$, a set $R \subseteq V$ of *terminals* and a length function $|\cdot|: E \to \mathbb{R}_+$, a *Steiner tree* is a connected subgraph of $G$ that spans all terminals. The Steiner tree problem in graphs asks for a shortest such subgraph, i. e., a tree that spans all vertices in $R$ and whose total edge length is minimum. It is called a *Steiner minimum tree* and denoted as $SMT$. We denote the length of $SMT$ by $smt$. We extend the definition of the length function $|\cdot|$ from single edges to arbitrary sets of edges by defining $|X| := \sum_{x \in X} |x|$ for $X \subseteq E$. Similarly we define $|G|$ for a graph $G = (V, E)$ as the total length of all its edges, i.e., $|E|$. This way we can write $smt = |SMT|$.

The Steiner tree problem in graphs is among the 21 problems for which Karp has shown NP-hardness in his seminal paper [19].

**Theorem 1.1 (Karp [19])** *The Steiner tree problem in graphs is NP-hard, even for unweighted graphs.*

The NP-hardness of the Steiner tree problem in graphs tells us that we cannot expect to find a polynomial time algorithm to solve it exactly. This motivates the search for good approximation algorithms for the Steiner tree problem in graphs, i.e., algorithms that have polynomial running time and return solutions that are not far from an optimum solution. The quality of an approximation algorithm $A$ is usually measured by its *performance ratio* $R_A$. This is the maximum ratio between the optimum and the solution returned by $A$. For a minimization problem like the Steiner tree problem in graphs the performance ratio is defined as

$$R_A := \sup \left\{ \frac{A(I)}{OPT(I)} \,\middle|\, \text{all instances } I \right\} .$$

Thus, the performance ratio of an algorithm is at least 1, and if it is exactly 1, then the algorithm can solve the problem optimally for all instances. The aim in designing approximation algorithms is to find polynomial time algorithms that have a performance ratio close to 1.

## 1.2 The Minimum Spanning Tree Heuristic

The history of Steiner tree approximation algorithms starts with a folklore result, which was rediscoverd by many researchers, but apparently first mentioned on p. 25 in [14], where it is attributed to Moore. This algorithm is called the *minimum spanning tree heuristic* and achieves a performance ratio of 2. The idea is simply to compute a *minimum spanning tree* instead of a Steiner minimum tree. A minimum spanning tree of a graph is a tree which is contained in the graph, connects all of its vertices and has minimum length among all such trees. We will denote a minimum spanning tree by $MST$ and its length by $mst$. Observe that a minimum spanning tree is a Steiner minimum tree if the set of terminals equals the set of all vertices. Whereas computing Steiner minimum trees is NP-hard, a minimum spanning tree can be found in polynomial time. The most famous algorithms for doing so are the algorithms of Kruskal [22] and Prim [25]. The latter algorithm can compute a minimum spanning tree for a graph with $n$ vertices and $m$ edges in time $\mathcal{O}(m + n \log n)$ if appropriate data structures are used (see eg. [10]).

The minimum spanning tree heuristic computes a Steiner tree in three steps: First it computes the so called *terminal distance graph*. This is a complete graph on the set $R$ of terminals. The edges are weighted by the length of a shortest path in $G$ between the two endpoints of the edge. In the second step a minimum spanning tree for the terminal distance graph is computed. This minimum spanning tree is translated in the final step to

This paper appeared in: Steiner Trees in Industry, X. Cheng and D.-Z. Du (Eds.), 235-279, Kluwer Academic Publishers, 2001

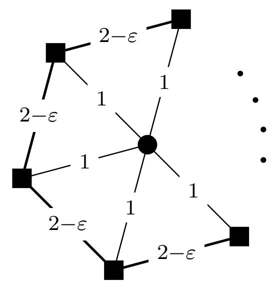


Figure 1: A worst-case instance for the *MST* heuristic.

a subgraph of $G$ by replacing its edges with the shortest paths which they represent. If the resulting subgraph of $G$ contains cycles or non-terminal vertices of degree one, they are removed iteratively to get a Steiner tree for $G$.

**Lemma 1.2 (Moore [14, p. 25])** *The performance ratio of the minimum spanning tree heuristic is* 2*, i. e., for all graphs* $mst \leq 2\,smt$ *holds.*

An example where the upper bound is asymptotically attained is shown in Figure 1. The square boxes are terminals. The *MST* heuristic will find a path around the border of the wheel, whereas the *SMT* uses the Steiner vertex in the middle. For a wheel with $k$ spokes, we obtain a lower bound of $(2-\varepsilon)(k-1)/k$ on the performance ratio, which converges to 2 as $k \to \infty$ and $\varepsilon \to 0$.

Slightly different versions of the minimum spanning tree heuristic have been rediscovered several times [9, 21, 29]. The fastest known implementation for the minimum spanning tree heuristic is due to Mehlhorn [23, 13]. It has a running time of $\mathcal{O}(m + n \log n)$.

## 1.3 $k$-Steiner Trees

For more than twenty years the performance ratio of the minimum spanning tree heuristic was not beaten by any other algorithm. The main reason for this was that for seemingly better algorithms it turned out to be difficult to analyze their perfomance ratio. The situation changed when in 1990 Zelikovsky suggested to use $k$-Steiner trees for the analysis of approximation algorithms. All currently known approximation algorithms for the Steiner tree problem make use of the idea of $k$-Steiner trees in an essential way.

A Steiner tree in which all terminals are leaves of the tree is called a *full Steiner tree.* Steiner trees that are not full can be decomposed into so

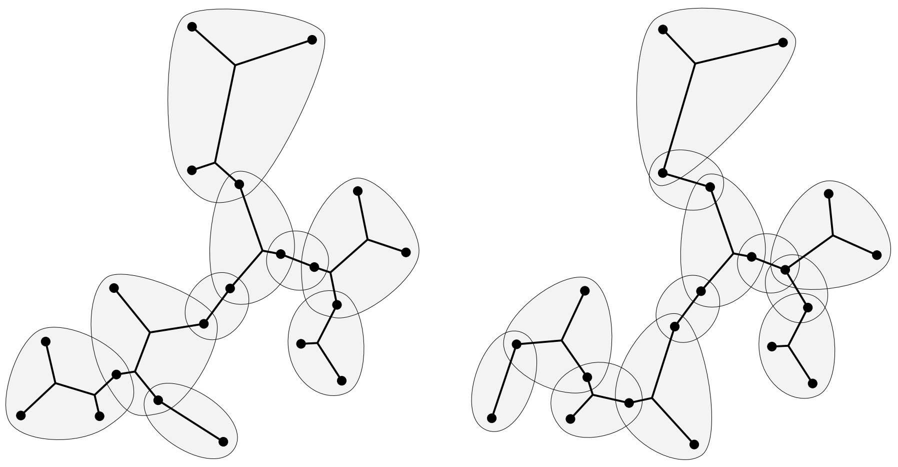


Figure 2: The full components of a Steiner tree

called *full components* by splitting terminals that are interior vertices of the Steiner tree. See Figure 2 for an illustration of the decomposition of a Steiner tree into full components. A *$k$-Steiner tree* is a Steiner tree all of whose full components contain at most $k$ terminals. There is a slight difficulty with this definition since a $k$-Steiner tree in this strict sense does not necessarily exist; consider e. g. a star with more than $k$ rays. Therefore we admit that a $k$-Steiner tree can use the edges and Steiner vertices of the graph in more than one full component. In this sense, a $k$-Steiner tree is a collection of full components with at most $k$ terminals that is connected and together spans the whole terminal set. An optimum $k$-Steiner tree is denoted by $SMT_k$ and its length by $smt_k$. Obviously, every Steiner tree is a $k$-Steiner tree for sufficiently large $k$, say $k = |R|$. In general the length of an optimum $k$-Steiner tree can be greater than that of a Steiner minimum tree. Figure 2 shows an optimum Steiner tree on the left and an optimum 3-Steiner tree on the right. The euclidean length is about 1% longer in the latter case.

Intuitively it seems reasonable that an optimum $k$-Steiner tree gives a good approximation for a Steiner minimum tree, if $k$ is sufficiently large. To make this intuition more precise consider the Steiner ratio $\rho_k$, which is defined as

$$\rho_k \;:=\; \sup_G \frac{smt_k(G)}{smt(G)}\,.$$

It follows from Lemma 1.2 that $\rho_2 = 2$. An upper bound for the value of $\rho_3$ was first obtained by Zelikovsky [32] and his result was generalized by Borchers and Du who proved the following explicit formula for $\rho_k$.

**Theorem 1.3 (Borchers, Du [6])** *For $k = 2^r + s$ with $0 \leq s < 2^r$ we have*

$$\rho_k = \frac{(r+1) \cdot 2^r + s}{r \cdot 2^r + s} .$$

Note that this result implies that $\rho_k \to 1$ for $k \to \infty$. This shows that optimum $k$-Steiner trees are good approximations for Steiner minimum trees if $k$ is sufficiently large. However the rate of convergence is not very fast. Some values of $\rho_k$ are shown in Table 2.

| $k$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 16 | 32 | 64 | $2^{100}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\rho_k$ | 2 | $\frac{5}{3}$ | $\frac{3}{2}$ | $\frac{13}{9}$ | $\frac{7}{5}$ | $\frac{15}{11}$ | $\frac{4}{3}$ | $\frac{33}{25}$ | $\frac{17}{13}$ | $\frac{5}{4}$ | $\frac{6}{5}$ | $\frac{7}{6}$ | 1.01 |

Table 2: Some values of the Steiner ratio $\rho_k$.

Based on Theorem 1.3 a natural idea is to compute an optimum $k$-Steiner tree instead of an optimum Steiner tree. However, it turns out that this problem is itself NP-hard. This follows by a simple reduction from the NP-completeness of the vertex cover problem in graphs of maximum degree 3.

**Lemma 1.4** *Finding optimum $k$-Steiner trees is NP-hard for $k \geq 4$.*

The situation is different for $k = 3$. Prömel and Steger [26] have shown that $SMT_3$ can be approximated with an error of only $1 + \varepsilon$ by a polynomial time randomized algorithm. Actually, they obtain their result as a corollary of a randomized algorithm for the minimum spanning tree problem in 3-uniform hypergraphs.

**Theorem 1.5 (Prömel, Steger [26])** *For every $\varepsilon > 0$ there exists a randomized polynomial time $1 + \varepsilon$ approximation algorithm for $SMT_3$ in weighted graphs.*

The proof technique of [26] does not extend to Steiner trees composed of larger full components. This is not surprising, since the results of Section 6 imply that for $k \geq 4$ there exists a constant $c > 1$ such that no algorithm can approximate $SMT_k$ with a performance ratio better than $c$, under reasonable complexity theoretical assumptions.

## 1.4 A General Framework for Greedy Algorithms

Most approximation algorithms for the Steiner tree problem that achieve a performance ratio less than 2 are simple greedy approaches. They fit into

```
Let $K$ be the set of full components of up to $k$ terminals.
$i \leftarrow 0$.
While an improving full component exists do:
    Choose $t_{i+1} \in K$ that minimizes the selection function $f_i$.
    $i \leftarrow i + 1$.
$i_{\max} \leftarrow i$.
Output a Steiner tree using $t_1, \ldots, t_{i_{\max}}$.
```

Figure 3: A general framework for greedy algorithms.

the general framework shown in Figure 3. Let $k \in \mathbb{N}$ be fixed. Consider the subsets $K'$ of $R$ with at most $k$ terminals and let $K$ be the collection of those $t \in K'$ for which $SMT(t)$ is a full Steiner tree. Note that $\#K \leq \#V^k$ and $SMT(t)$ for all $t \in K$ can be computed in polynomial time, since $k$ is fixed [11]. The algorithms start with a Steiner tree that is obtained by taking a minimum spanning tree in the distance graph. By Lemma 1.2 the total length of this solution is at most twice the length of a Steiner minimum tree. In each step the algorithms try to improve the current solution by using a Steiner minimum tree for an element of $K$. If there is more than one $t \in K$ that would improve the current solution, the algorithms use a selection function $f\colon K \to \mathbb{R}_+$ to decide which $t$ is to be chosen next. Therefore, to specify an algorithm fitting into this general framework it suffices to specify the function $f$, and how to improve the current solution using $t \in K$.

Let $A$ be an algorithm that fits into the framework and let $A(G)$ be its output on instance $G$. The performance ratio of $A$ is

$$\sup_G \frac{|A(G)|}{smt(G)} = \sup_G \left( \frac{smt_k(G)}{smt(G)} \cdot \frac{|A(G)|}{smt_k(G)} \right) \leq \rho_k \cdot \sup_G \frac{|A(G)|}{smt_k(G)} . \qquad (1)$$

If we can show that the ratio $|A(G)|/smt_k(G)$ is bounded from above by a constant $c_k$, then we immediately have a polynomial time approximation algorithm for the Steiner tree problem with performace ratio $\rho_k c_k$. If, moreover, $c = c_k$ is independent on $k$, then this gives a sequence of algorithms, whose performance ratio tends to $c$, as the Steiner ratio $\rho_k$ converges to 1. Algorithms that are specified via the framework and analyzed in this way are Zelikovsky's 11/6-approximation algorithm (Section 2.3), Zelikovsky's Relative Greedy Algorithm (Section 3), and the Loss Contraction Algorithm (Section 4).

Note, however, that for large $k$ the running time of such an algorithm may soon become impractical. Recent theoretical research has concentrated on improving the performance ratio instead of the running time, because it is commonly believed that the currently known approximation algorithms (or at least their analyses) are still far from optimal.

## 1.5 Contraction Lemma

The Contraction Lemma is one of the concepts that occur repeatedly in the context of algorithms that find minimum connecting structures incrementally. First we introduce some notation.

Let us denote by $MST(R)$ the minimum spanning tree in the terminal distance graph for a set of required vertices $R$. Now assume that we add a new edge $e$ between a pair of terminals. Parallel edges are allowed. Define

$$MST(R/e) \ := \ \text{a minimum spanning tree for } R \text{ in } G + e\,.$$

If $|e| = 0$, then $MST(R/e)$ can be viewed as a minimum spanning tree for $R$ after *contracting* $e$. In general the difference $mst(R) - mst(R/e)$ is what one can *gain* by adding $e$. We want to know how the gain of another new edge $f$ changes if $e$ is added.

A typical application of the Contraction Lemma to the general framework for greedy algorithms is as follows. Assume that we can model the 'effect' of inserting a certain full component by adding a set $E_1$ of new edges to the terminal distance graph. Moreover, let $E_0$ denote the set of edges which have already been added for earlier full components. The length of the current intermediate solution is denoted by $mst(R/E_0)$. Given yet another full component $E_2$, the question is: How does the improvement which can be achieved by $E_2$ depend on whether the algorithm will decide to include $E_1$ or not? The answer is: It will never increase.

**Contraction Lemma (Zelikovsky [32], Berman, Ramaiyer [4])** *Let $E_0$, $E_1$, $E_2$ be sets of (new) edges between terminals. Then*

$$mst(R/E_0) - mst(R/E_0E_2) \ \geq \ mst(R/E_0E_1) - mst(R/E_0E_1E_2)\,.$$

The proof of this Contraction Lemma needs some prerequisites. First let us see what happens if we just add a single new edge.

**Lemma 1.6** *Let $e$ be a new edge between terminals. Then*

$$MST(R/e) = MST(R) + e - e'\,,$$

*where $e'$ is the longest edge in the (unique) cycle in $MST(R) + e$.*

*Proof.* This is a consequence of the fact that Kruskal's algorithm for the minimum spanning tree problem is optimal. □

Using the notation of Lemma 1.6, the gain of $e$ is $mst(R) - mst(R/e) = |e'| - |e|$. Next we show that the gain of an edge never increases if we add another edge.

**Lemma 1.7** *Let $e_1$ and $e_2$ be edges between terminals. Then*

$$mst(R) - mst(R/e_2) \;\geq\; mst(R/e_1) - mst(R/e_1e_2)\,.$$

*Proof.* Let $E_{\leq\alpha}$ denote the set of edges of length at most $\alpha$, and denote by $T_{\leq\alpha}$ the intermediate solution of Kruskal's algorithm when all edges of length up to $\alpha$ have been considered. Then the components of $E_{\leq\alpha}$ and $T_{\leq\alpha}$ are identical for all $\alpha$, because an edge is included if and only if it joins two components. Obviously, $T_{\leq\alpha}$ is not affected by the new edge $e_2$, if $\alpha < |e_2|$. In that case both sides of the inequality are equal to 0. Otherwise, let $e_2'$ be the edge that is replaced with $e_2$. It is only a moment's thought to see that

$$\begin{aligned} mst(R) - mst(R/e_2) &= |e_2'| - |e_2| \\ &= \min\{\,\alpha > 0 \mid e_2 \text{ closes a cycle in } T_{\leq\alpha}\,\} - |e_2| \\ &= \min\{\,\alpha > 0 \mid e_2 \text{ closes a cycle in } E_{\leq\alpha}\,\} - |e_2| \end{aligned}$$

Now since $E_{\leq\alpha} \subseteq (E + e_1)_{\leq\alpha}$ for all $\alpha$, the lemma follows. □

***Proof of the Contraction Lemma.*** The special case $E_0 = \emptyset$, $E_1 = \{e_1\}$, $E_2 = \{e_2\}$ was proved in Lemma 1.7. Considering the graph with edge set $E \cup E_0$ we may assume w.l.o.g.that $E_0 = \emptyset$. The case where $E_1 = \{e_1^1, e_1^2, \dots\}$ and $E_2 = \{e_2\}$ follows easily by repeated application of Lemma 1.7. Finally, let $E_2 = \{e_2^1, e_2^2, \dots\}$. We claim that

$$\begin{aligned} mst(R) - mst(R/E_2) &= \sum_i mst(R/e_2^1 \dots e_2^{i-1}) - mst(R/e_2^1 \dots e_2^i) \\ &\geq \sum_i mst(R/E_1e_2^1 \dots e_2^{i-1}) - mst(R/E_1e_2^1 \dots e_2^i) \\ &= mst(R/E_1) - mst(R/E_1E_2)\,. \end{aligned}$$

To see that the inequality holds, take $E_0' := \{e_2^1, \dots, e_2^{i-1}\}$, $E_1' := E_1$, and $E_2' := \{e_2^i\}$, and apply induction on $|E_2|$. □

## 2 Improving the Factor of 2

In this section we present two algorithms which have a performance ratio of 11/6. The first appeared in 1990 and is due to Zelikovsky [32]. It uses full components with just three terminals in order to improve the solution of the minimum spanning tree heuristic. The algorithm fits into the general framework as we will see in Section 2.3.

The idea turned out to be very fruitful and marked the starting point of a series of algorithms by Zelikovsky and other authors with continuously improving performance ratios.

Unfortunately, the original analysis of Zelikovsky's algorithm [32] remains somewhat vague at a crucial point. The proof seems to be incomplete, and apparently it is not easy to fix it. Berman and Ramaiyer [4] described a family of algorithms $A_k$ working with full components of up to $k$ terminals which achieved a performance ratio of 1.734 for large $k$. Here we shall present their analysis of algorithm $A_3$ which attains the ratio of 11/6.

Based on the analysis of Berman and Ramaiyer it is possible to prove the performance ratio of Zelikovsky's algorithm rigorously. For completeness, we describe this argument in detail in Section 2.3. The main ingredients for this proof are taken from Zelikovsky [35], see also [8, 7]. Both algorithms allow fast implementations [33].

### 2.1 Algorithm $\mathbf{A_3}$

The algorithm $A_3$ of Berman and Ramaiyer has two essential phases, an evaluation and a construction phase. These are embedded in other phases. The initial solution is a spanning tree of the terminal distance graph as output by the *MST* heuristic, which is already a 2 approximation.

```
Initial phase.   Compute the terminal distance graph $(R, E_0)$ and let
                 $M_0 \subset E_0$ be a minimum spanning tree.
                 $i \leftarrow 0$.
```

The algorithm uses full components with three terminals to improve the current intermediate solution. Steiner trees for triples of terminals can be computed very fast. Potential triples are evaluated in terms of their *gain*. The gain is the saving of total length in the current intermediate solution *if* the triple *was* used. The precise definiton of gain will be given below. During the evaluation phase, triples are selected one by one. However, an important feature of algorithm $A_3$ is that not each of the selected triples needs to be used eventually. The decision is deferred to the construction

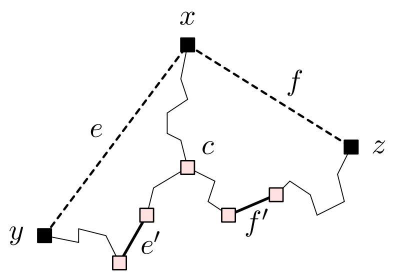


Figure 4: The gain of a triple.

phase. Meanwhile the triples are stacked using a special data structure. The algorithm builds multisets of artificial edges $E_i$ on the terminal set $R$. Each triple is represented by a pair of artificial edges. The edges of $E_0$ are also considered artificial.

In the $i$-th iteration, with every triple $t = \{x, y, z\} \subseteq R$ a quantity $gain_i(t)$ and two edges $e'$ and $f'$ from $E_i$ are associated in the following way (cf. Figure 4): Consider the tree $M_i \subset E_i$. There is a 'branching' terminal $c \in R$ such that the paths $p_x$ from $x$ to $c$, $p_y$ from $y$ to $c$, and $p_z$ from $z$ to $c$ are edge disjoint. One of these paths might be empty. Consider the longest edge on each of these paths (say its length is 0, if the path is empty) and assume w.l.o.g. that the longest among these edges, say $e'$ is on $p_y$ and the second longest, say $f'$ is on $p_z$. Let $Cut_i(t) := \{e', f'\}$ and $cut_i(t) := |e'| + |f'|$. Then the gain of the triple $t$ is defined as

$$gain_i(t) := cut_i(t) - smt(t).$$

Assume that there is a triple $t_{i+1}$ such that $gain_i(t_{i+1}) > 0$ and let $Cut_i(t_{i+1}) = \{e'_{i+1}, f'_{i+1}\}$. Using the notation above, let $e_{i+1} := \{x, y\}$ and $f_{i+1} := \{x, z\}$, where $|e_{i+1}| := |e'_{i+1}| - gain_i(t_{i+1})$ and $|f_{i+1}| := |f'_{i+1}| - gain_i(t_{i+1})$. Observe that

$$|e_{i+1}| = |e'_{i+1}| - |f'_{i+1}| - |e'_{i+1}| + smt(t_{i+1}) = smt(t_{i+1}) - |f'_{i+1}| \geq 0$$

and similarly, $|f_{i+1}| \geq 0$. With these definitions the evaluation phase of the algorithm can be stated as follows:

```
Evaluation.     While $\exists t_{i+1} : gain_i(t_{i+1}) > 0$ do:
                $E_{i+1} \leftarrow E_i + e_{i+1} + f_{i+1}$.
                $M_{i+1} \leftarrow M_i + e_{i+1} + f_{i+1} - e'_{i+1} - f'_{i+1}$.
                $i \leftarrow i + 1$.
```

Note that $M_{i+1}$ is a minimum spanning tree in $(R, E_{i+1})$. Once the gain of every potential triple has dropped to a non-positive value, the algorithm has chosen all triples to be considered for improvement of the terminal distance minimum spanning tree and the evaluation phase ends. The last of the $M_i$ is copied for the construction phase:

```
Intermediary   $i_{\max} \leftarrow i$.
phase.         $N_{i_{\max}} \leftarrow M_{i_{\max}}$.
```

In the construction phase the artificial edges are removed from $N_i$ in reverse order. Let $T_i := SMT(t_i)$. Only if both artificial edges that were introduced for the triple $t_i$ are still present in $N_i$, then $T_i$ is included into the eventual solution:

```
Construction.  While $i > 0$ do:
               $N'_i \leftarrow N_i - e_i - f_i$.
               $N_{i-1} \leftarrow \begin{cases} N'_i & \text{if } N'_i \text{ has only one component,} \\ N'_i + e & \text{if } N'_i \text{ has two components and } e \in E_{i-1} \text{ is a shortest edge between them,} \\ N'_i + T_i & \text{if } N'_i \text{ has three components.} \end{cases}$
               $i \leftarrow i - 1$.
```

At the end of the construction phase all of the remaining artificial edges in $N_0$ belong to $E_0$. These can easily be replaced by edges of the input graph:

```
Final phase.   Replace any remaining $e_0 \in N_0 \cap E_0$ with a shortest
               path in $E$.
               Output the result, i. e., $N_0$.
```

## 2.2 Analysis

**Theorem 2.1 (Berman, Ramaiyer [4])** *Algorithm $A_3$ computes an $11/6$ approximation for SMT.*

*Proof.* We first observe that the output of the algorithm is a feasible solution: As already observed above, $M_i$ is a minimum spanning tree for $R$ in $E_i$. Therefore $N_{i_{\max}} = M_{i_{\max}}$ connects $R$. By the construction step, $N_i$ is connected if $N_{i+1}$ is. Therefore $N_0$, and thus the output, connects $R$.

Let $gain := \sum_i gain_i(t_{i+1})$. The analysis of the performance ratio is based on the following (in)equalities:

$$|N_0| \;\leq\; |N_{i_{\max}}| + gain \tag{2}$$
$$|M_{i_{\max}}|/2 \;=\; |M_0|/2 - gain \tag{3}$$
$$|M_{i_{\max}}| \;\leq\; smt_3(R) \tag{4}$$

Before we verify them, let us first see how (2) – (4) together imply the claimed performance ratio. The length of the output is $|N_0|$. Using $N_{i_{\max}} = M_{i_{\max}}$, one half of $|N_{i_{\max}}|$ in (2) can be replaced via (3). The gain cancels out, and we obtain

$$|N_0| \;\leq\; \frac{|M_{i_{\max}}|}{2} + \frac{|M_0|}{2}\,.$$

By Lemma 1.2, $|M_0| \leq \rho_2 \cdot smt(R)$, and by (4), $|M_{i_{\max}}| \leq \rho_3 \cdot smt(R)$. Therefore the performance ratio is at most

$$\frac{\rho_3}{2} + \frac{\rho_2}{2} \;=\; \frac{5/3}{2} + \frac{2}{2} \;=\; \frac{11}{6}\,.$$

It remains to show the validity of (2) – (4). Equation (3) is the easiest. It follows by induction from the fact that

$$|M_{i+1}| \;=\; |M_i| + |e_{i+1}| + |f_{i+1}| - |e'_{i+1}| - |f'_{i+1}| \;=\; |M_i| - 2gain_i(t_{i+1})\,.$$

Equation (2) is proved by a similar inductive argument, using

$$|N_{i-1}| \;\leq\; |N_i| + gain_{i-1}(t_i)\,. \tag{5}$$

To show (5) we distinguish the three cases of the contruction phase. The case that $N'_i$ has only one component is trivial. In the case where it has three components, (5) follows from

$$\begin{aligned} |N_{i-1}| \;&=\; |N_i| - |e_i| - |f_i| + |T_i| \\ &=\; |N_i| - |e'_i| + gain_{i-1}(t_i) - |f'_i| + gain_{i-1}(t_i) + |T_i| \\ &=\; |N_i| + gain_{i-1}(t_i). \end{aligned}$$

The remaining case is when $N'_i$ has two components. Assume w.l.o.g. that the two components result from removing the edge $e_i$. The edge $e'_i$ is the longest edge on the path in $M_{i-1}$ connecting the endpoints of $e_i$. Since the path crosses the cut between the two components at least once, we have $|e| \leq |e'_i|$, where $e$ is the edge chosen by the algorithm. Thus,

$$\begin{aligned} |N_{i-1}| \;&=\; |N_i| - |e_i| - |f_i| + |e| \;\leq\; |N_i| - |e_i| + |e| \\ &\leq\; |N_i| - |e_i| + |e'_i| = \; |N_i| + gain_{i-1}(t_i)\,, \end{aligned}$$

and the induction step for (2) is complete.

To show (4) we compare $M_{i_{\max}}$ with a minimum 3-Steiner tree. By construction we have $M_{i_{\max}} = MST_{i_{\max}}(R)$, where the subscript $i_{\max}$ indicates that the underlying graph includes all the artificial edges which were added during the evaluation phase. On the other hand, let $s_1, \ldots, s_{j_{\max}}$ be terminal sets of the full components of $SMT_3$. Each $s_j$ is a pair or a triple, and for pairs we have $smt(s_j) = |s_j|$. Using the convention that contracting a set of vertices means connecting them by zero length edges, we have $mst_{i_{\max}}(R/s_1 \ldots s_{j_{\max}}) = 0$, since $SMT_3$ is already connected. Writing a telescoping sum, we can apply the Contraction Lemma in the following way.

$$\begin{aligned}
|M_{i_{\max}}| &= mst_{i_{\max}}(R) - mst_{i_{\max}}(R/s_1 \ldots s_{j_{\max}}) \\
&= \sum_{j=1}^{j_{\max}} mst_{i_{\max}}(R/s_1 \ldots s_{j-1}) - mst_{i_{\max}}(R/s_1 \ldots s_j) \\
&\leq \sum_{j=1}^{j_{\max}} mst_{i_{\max}}(R) - mst_{i_{\max}}(R/s_j)
\end{aligned}$$

If $s_j$ is a pair, then the $j$-th summand is clearly bounded by $|s_j|$. If $s_j$ is a triple, then $mst_{i_{\max}}(R) - mst_{i_{\max}}(R/s_j) = gain_{i_{\max}}(s_j) + smt(s_j) \leq smt(s_j)$, because at the end of the evaluation phase there are no more improving triples. Hence $|M_{i_{\max}}| \leq \sum_j smt(s_j) = smt_3(R)$, and (4) is verified. $\Box$

## 2.3 Zelikovsky's Algorithm

Zelikovsky's algorithm [32] fits into the general framework (Figure 3) where $k = 3$ and in each step $gain_i$ is maximized. But it can also be viewed as a modification of Berman and Ramaiyer's Algorithm $A_3$. Building on their analysis, one can prove a bound on the performance ratio of Zelikovsky's algorithm.

**Theorem 2.2** *Zelikovsky's algorithm computes an* $11/6$ *approximation for* $SMT$.

Note that $gain_i$ is not necessarily maximized by $t_{i+1}$ in the evaluation phase of Algorithm $A_3$, since the proof of the performance ratio requires only that the gain is positive. It turns out, however, that the construction phase can be simplified if the triples are chosen greedily. Assume that some triple chosen in the evaluation phase is not used in the contruction phase. This can happen only if one of the edges introduced for the first triple is removed by a second triple later on. Intuition advocates that, at the time when the first triple was chosen, the gain of the second triple should have been larger than that of the first triple, contradicting the greedy selection. We are going to prove the following lemma.

**Lemma 2.3** *Assume that in each step $i$ of the evaluation phase of Berman and Ramaiyer's algorithm $A_3$ the triple $t_{i+1}$ is chosen to maximize $gain_i$. Then $Cut_i(t_{i+1}) \subseteq E_0$, i. e., the removed edges belong to the initial terminal-distance graph.*

What does this mean for the 'greedy-$A_3$' algorithm? The artificial edges $e_{i+1}, f_{i+1}$ introduced during the evaluation phase are assigned non-negative lengths but never removed again. Therefore the result of the algorithm does not change if we assign them any shorter length, for instance the length 0. This is equivalent to contracting the triple $t_{i+1}$. Further, the construction phase can be merged into the evaluation phase, since we know that $N'_i$ will always have three components.

In this way, we obtain Zelikovsky's algorithm: In each step, the terminals of the chosen triple are contracted. After the evaluation phase we replace the remaining edges from $E_0$ with shortest paths in $E$ and connect the contracted triples by their corresponding Steiner trees.

## 2.4 Analysis

In this section we prove Lemma 2.3. As we have just seen, this lemma will imply Theorem 2.2.

First we introduce some additional notation. Let $M$ be a set of edges without cycles that connects two terminals $x$ and $z$. Then we denote by $Cut_M\{x,z\}$ the longest edge on a path in $M$ connecting $x$ and $z$, where ties are broken in an arbitrary way, and let $cut_M = |Cut_M|$. We write $M : xz \mid y$ to indicate that the terminals $x$ and $z$ are in the same connected component of $M$, but $y$ is in another one.

If $M$ is a tree and $t = \{x, y, z\}$ is a triple of its vertices, we denote by $M[t]$ the least part of $M$ that connects $t$. Recall that this is the situation of Figure 4. We add the following to the notation defined in Sections 2.1 and 2.2: For $e' \in M[t]$, we define the *relative cut* $Cut_M(t, e') := Cut_M\{x, z\}$, where $M - e' : xz \mid y$. Let $cut_M(t, e') := |Cut_M(t, e')|$ and observe that $cut_M(t, e') + |e'| \leq cut_M(t)$. We also define the *relative gain*

$$gain_M(t, e') \;:=\; cut_M(t, e') + |e'| - smt(t)\,. \qquad (6)$$

Observe that $gain_M(t, e') \leq gain_M(t)$. If in particular $e' \in Cut_M(t)$, then $Cut_M(t, e') = Cut_M(t) - e'$ and $gain_M(t, e') = gain_M(t)$.

In the rest of this section we suppose that in each step $i$ of Algorithm $A_3$ the triple $t_{i+1}$ is chosen to maximize $gain_i$. We subdivide step $i$ into half-steps as follows: In the first half-step, $e'_{i+1}$ is replaced with $e_{i+1}$, and in the second one, $f'_{i+1}$ with $f_{i+1}$. We have $|e'_{i+1}| \geq |f'_{i+1}|$, and $e'_{i+1}$ is replaced first. So let $M'_i := M_i - e_{i+1} + e'_{i+1}$. For the half-steps, we use the notation $Cut_i := Cut_{M_i}$, $Cut'_i := Cut_{M'_i}$, etc. Observe that

$$cut_i\{u, v\} \geq cut'_i\{u, v\} \geq cut_{i+1}\{u, v\} \tag{7}$$

holds for all $u, v$ and $i$ by the Contraction Lemma.

The next lemma implies Lemma 2.3 by a simple inductive argument.

**Lemma 2.4** *Let $i \geq 1$ and $a \in M_i$. If there exists a triple $t$ such that $a \in M_i[t]$ and $gain_i(t, a) > 0$, then there also exists a triple $t' \in \{t, t_i\}$ such that $a \in M_{i-1}[t']$ and $gain_{i-1}(t', a) \geq gain_i(t, a)$.*

***Proof of Lemma 2.3.*** Let $a \in Cut_i(t_{i+1})$. Then we have $a \in M_i[t_{i+1}]$ and $gain_i(t_{i+1}, a) = gain_i(t_{i+1}) > 0$. By repeated application of Lemma 2.4, there is a triple $t'$ such that $a \in M_0[t'] \subset E_0$. □

In the proof of Lemma 2.4 we may assume that $a \notin \{e_i, f_i\}$ due to the following Proposition.

**Proposition 2.5** *If $a \in \{e_i, f_i\} \cap M_i[t]$ for some triple $t$, then $gain_i(t, a) \leq 0$.*

*Proof.* If $a = e_i$ then let $a' = e'_i$, otherwise let $a' = f'_i$. Let $\{x, y, z\}$ be the vertices of $t$, and assume w.l.o.g. that $M_i[t] - a : xy \mid z$, and hence also $M_{i-1}[t] - a' : xy \mid z$. Then

$$gain_{i-1}(t) \;\geq\; gain_{i-1}(t, a') \;=\; cut_{i-1}(t, a') + |a'| - smt(t)\,,$$

and by (7),

$$cut_{i-1}(t, a') \;=\; cut_{i-1}\{x, y\} \;\geq\; cut_i\{x, y\} \;=\; cut_i(t, a)\,.$$

Using $|a| = |a'| - gain_{i-1}(t_i)$ we obtain the inequality

$$\begin{aligned} 0 &\geq gain_{i-1}(t) - gain_{i-1}(t_i) \\ &\geq cut_i(t, a) + |a| - smt(t) \\ &= gain_i(t, a)\,. \end{aligned}$$

□

We distinguish several cases to prove Lemma 2.4, using Lemma 2.6 and Lemma 2.7.

**Lemma 2.6** *Let $M$ be a terminal-spanning tree and $t = \{x, y, z\}$ be a triple. Let $\{e', f'\} := Cut_M(t)$ and $e$, $f$ be the corresponding artificial edges as shown in Figure 4, and assume $a \neq e$ is an edge from the cycle in $M + e$. Then we have*

$$gain_M(t, a) \;\geq\; gain_M(t) - |e'| + |a|\,.$$

*Proof.* Let $c$ be the branching vertex of $M[t]$. If $a$ is on the path $p_x$ from $c$ to $x$ in $M[t]$, then $cut_M(t, a) = \max\{|e'|, |f'|\} \geq |f'|$. If $a$ is on $p_y$, then $cut_M(t, a) = |f'|$. In both cases, we have

$$\begin{aligned} gain_M(t, a) &= cut_M(t, a) + |a| - smt(t) \\ &\geq \big(|f'| + |e'| - smt(t)\big) - |e'| + |a| \\ &= gain_M(t) - |e'| + |a| \end{aligned}$$

as claimed. □

**Lemma 2.7** *Let $M \ni e$ and $M' \ni e'$ be terminal-spanning trees such that $M = M \cap M' + e$, $M' = M \cap M' + e'$, and assume that $e'$ is the longest edge on the cycle in $M \cup M'$. Moreover, let $a \in M \cap M'$ and $t$ be a triple such that $a \in M[t]$ and $gain_M(t, a) > 0$. Then at least one of the following holds.*

- ($\alpha$) *$a \in M'[t]$ and $gain_{M'}(t, a) \geq gain_M(t, a) > 0$.*
- ($\beta$) *$e' \in M'[t]$, $gain_{M'}(t, e') - |e'| + |a| \geq gain_M(t, a)$, and $a$ lies on the cycle in $M \cup M'$.*

*Proof.* Let us denote the three connected components of $M \cap M' - a$ by $A$, $B$, and $C$. We can assume w. l. o. g. that $a$ connects $A$ and $B$ and that $e$ connects $B$ and $C$. Let $t = \{u, v, w\}$ and assume w. l. o. g. that $M[t] - a : uv \mid w$ and hence, $M - a : uv \mid w$.

*Case 1:* $M' - a : uv \mid w$. Then we have $cut_M(t, a) = cut_M\{u, v\}$ and $cut_{M'}(t, a) = cut_{M'}\{u, v\}$. By (7) we have $0 \leq cut_{M'}\{u, v\} - cut_M\{u, v\} = gain_{M'}(t, a) - gain_M(t, a)$, proving ($\alpha$).

*Case 2:* $C \cap t \neq \emptyset$ and $e'$ connects $A$ and $C$, since otherwise Case 1 holds. This further implies that $A \cap t \neq \emptyset$, $e' \in M'[t]$, and $a$ lies on the cycle in $M \cup M'$. Observe that ($\beta$) holds, if we can show that $cut_{M'}(t, e') \geq cut_M(t, a)$.

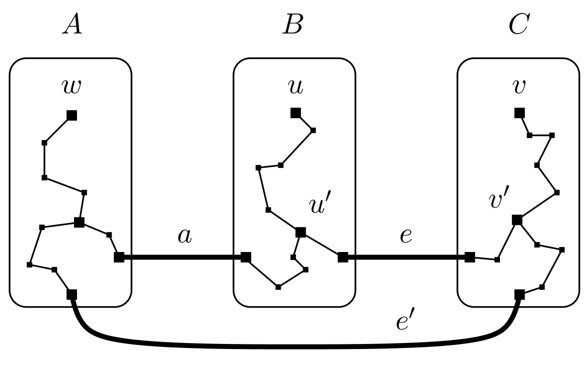


Figure 5: Case 2.2 in the proof of Lemma 2.6.

*Case 2.1:* If $u$ and $v$ are both in the same component of $M \cap M'$, then $Cut_{M'}(t, e') = Cut_{M'}\{u, v\} = Cut_M\{u, v\} = Cut_M(t, a)$. (The path remains the same.) So ($\beta$) holds.

*Case 2.2:* Otherwise $C \cap \{u, v\} \neq \emptyset$ and $w \notin C$, since $M - a : uv \mid w$. Assume w.l.o.g. that $v \in C$. Then $M - a : uv \mid w$ implies that $w \in A$ and $u \in B$. Consider the path in $M$ connecting $u$ to $v$. Let $u'$ be the vertex where it enters and $v'$ be the vertex where it leaves the cycle in $M \cup M'$. Again we distinguish cases, depending on the position of the edge $b := Cut_M(t, a) = Cut_M\{u, v\}$. See Figure 5.

*Case 2.2.1:* If $b$ lies between $u$ and $u'$, then $cut_{M'}(t, e') = cut_{M'}\{u, w\} \geq |b|$ and we have ($\beta$). (Actually, this is the only case where the inequality in ($\beta$) can be strict.)

*Case 2.2.2:* If $b$ lies between $v$ and $v'$, then $cut_{M'}(t, a) = cut_{M'}\{v, w\} \geq |b|$. We get ($\alpha$).

*Case 2.2.3:* Finally, if $b$ lies between $u'$ and $v'$, then we know that $|b| \leq |e'| \leq cut_{M'}(t, a)$ as $e'$ is the longest edge on the cycle in $M \cup M'$, and ($\alpha$) follows. □

***Proof of Lemma 2.4.*** We can apply Lemma 2.7 with $M = M_i$, $M' = M'_{i-1} = M_i - f_i + f'_i$, $e = f_i$, and $e' = f'_i$, so one of the following must hold:

($\alpha$) $a \in M'_{i-1}[t]$ and $gain'_{i-1}(t, a) \geq gain_i(t, a) > 0$.

($\beta$) $f'_i \in M'_{i-1}[t]$, $gain'_{i-1}(t, f'_i) - |f'_i| + |a| \geq gain_i(t, a)$, and $a$ lies on the cycle in $M_i + f'_i$.

This paper appeared in: Steiner Trees in Industry, X. Cheng and D.-Z. Du (Eds.), 235-279, Kluwer Academic Publishers, 2001

Case ($\beta$) can be settled fairly easily. We will show that $t' = t_i$ works. We have $gain'_{i-1}(t, f'_i) \leq gain'_{i-1}(t)$, and the Contraction Lemma implies that $gain'_{i-1}(t) \leq gain_{i-1}(t)$. By the greedy criterion, we have $gain_{i-1}(t) \leq gain_{i-1}(t_i)$. It remains to show that $gain_{i-1}(t_i) + |a| - |f'_i| \leq gain_{i-1}(t_i, a)$. This is just Lemma 2.6 if we set $M = M_{i-1}$, $t = t_i$, $e = f_i$, $e' = f'_i$, $f = e_i$, and $f' = e'_i$.

In case ($\alpha$) we invoke Lemma 2.7 once more, this time with $M = M'_{i-1}$, $M' = M_{i-1}$, $e = e_i$, and $e' = e'_i$. We find ourselves in one of the following situations.

($\alpha\alpha$) $a \in M_{i-1}[t]$ and $gain_{i-1}(t, a) \geq gain'_{i-1}(t, a) > 0$.

($\alpha\beta$) $e'_i \in M_{i-1}[t]$, $gain_{i-1}(t, e'_i) - |e'_i| + |a| \geq gain'_{i-1}(t, a)$, and $a$ lies on the cycle in $M_{i-1} + e_i$.

In case ($\alpha\alpha$) we are done with $t' = t$. — In case ($\alpha\beta$) we will show that $t' = t_i$ is a good choice. From the case distinctions, we already know that

$$gain_i(t, a) \;\leq\; gain'_{i-1}(t, a) \;\leq\; gain_{i-1}(t, e'_i) - |e'_i| + |a|\,.$$

We have $gain_{i-1}(t, e'_i) \leq gain_{i-1}(t)$, and by the greedy criterion, $gain_{i-1}(t) \leq gain_{i-1}(t_i)$. The remaining inequality $gain_{i-1}(t_i) - |e'_i| + |a| \leq gain_{i-1}(t_i, a)$ follows from Lemma 2.6 if we set $M = M_{i-1}$, $t = t_i$, $e = e_i$, $e' = e'_i$, $f = f_i$, and $f' = f'_i$. □

## 3 Relative Greedy Algorithm

The relative greedy algorithm due to Zelikovsky [34] is another example of an algorithm that fits into the general framework. It has a performance ratio of 1.694. The main idea of the relative greedy algorithm (and the loss contracting algorithm, which we will consider in the next section) is to use a 'relative' difference in the selection function instead of the absolute difference, as was the case in the algorithms of Berman and Ramaiyer and Zelikovsky.

### 3.1 Relative Greedy Algorithm

The relative greedy algorithm uses a minimum terminal spanning tree (i. e., a minimum spanning tree in the terminal distance graph) as its initial solution. When a full component $T \in K$ is chosen, its terminals are connected by a set of zero length edges. A spanning tree for $R$, where this set of edges is added, is denoted by $MST(R/T)$. Choosing $T$ in the $i$-th step reduces the length of the terminal spanning tree by $mst(R/T_1 \dots T_i) - mst(R/T_1 \dots T_i T)$. In order to relate the length $|T|$ of the full component $T$ to its benefit, the relative greedy algorithm uses the following selection function.

$$f_i(T) \;:=\; \frac{|T|}{mst(R/T_1 \dots T_i) - mst(R/T_1 \dots T_i T)}$$

The Contraction Lemma implies that $f_{i-1}(T) \le f_i(T)$ for all $T$. Once $f_i(T) \ge 1$ for all $T \in K$, no further improving full components can be selected and the algorithm stops. At this point, it has found a solution of size

$$|T_1| + \dots + |T_i| + mst(R/T_1 \dots T_i)\,.$$

The edges of $MST(R/T_1 \dots T_i)$ can be considered as full components with just two terminals. To simplify the notation for the analysis, we include these edges into the final solution set. Note that $f_{i-1}(T_i) \le 1$ holds for all chosen full components.

## 3.2 Analysis

By our remarks on the Steiner ratio and the general framework (Section 1.4), the following theorem implies that the relative greedy algorithm computes an approximation of $SMT$ whose error is bounded by $1 + \ln 2 + \varepsilon \le 1.694$ for large enough $k = k(\varepsilon)$.

**Theorem 3.1 (Zelikovsky [34])** *The relative greedy algorithm computes a* $1 + \ln 2$ *approximation for* $SMT_k$.

*Proof.* Let $T_1, \dots, T_{i_{\max}}$ be the Steiner tree found by the relative greedy algorithm. We have to show that $\sum_{i=1}^{i_{\max}} |T_i| \le smt_k(1 + \ln 2)$. As remarked above

$$f_i(T_{i+1}) \le 1, \tag{8}$$

holds for all $i < i_{\max}$. Let $T_1^*, \dots, T_{j_{\max}}^*$ be a $k$-Steiner minimum tree. Since the algorithm chooses the full component $T_{i+1}$ such that $f_i$ is minimized, we have

$$f_i(T_{i+1}) \le \min_j f_i(T_j^*)\,. \tag{9}$$

We will use the following inequality, valid for $a_j \ge 0$ and $b_j > 0$.

$$\min_j \frac{a_j}{b_j} \;\le\; \frac{\sum_j a_j}{\sum_j b_j}. \tag{10}$$

With $a_j = |T_j^*|$ and $b_j = mst(R/T_1 \ldots T_i) - mst(R/T_1 \ldots T_i T_j^*)$ using (9) and (10) we get

$$f_i(T_{i+1}) \;\le\; \frac{\sum_j |T_j^*|}{\sum_j mst(R/T_1 \ldots T_i) - mst(R/T_1 \ldots T_i T_j^*)} \,. \tag{11}$$

By the Contraction Lemma the denominator of (11) can be replaced with

$$\sum_j mst(R/T_1 \ldots T_i T_1^* \ldots T_{j-1}^*) - mst(R/T_1 \ldots T_i T_1^* \ldots T_j^*) \,. \tag{12}$$

This is a telescoping sum in which all but the first and the last term cancel. Since $T_1^*, \ldots, T_{j_{\max}}^*$ is an $SMT_k$ the last term is $mst(R/T_1 \ldots T_i T_1^* \ldots T_{j_{\max}}^*) \le mst(R/T_1^* \ldots T_{j_{\max}}^*) = 0$ and $\sum_j |T_j^*| = smt_k$. So

$$f_i(T_{i+1}) \;\le\; \frac{smt_k}{mst(R/T_1 \ldots T_i)} \,. \tag{13}$$

Now the desired bound on the approximation ratio follows from elementary calculus. Define $M_i := mst(R/T_1 \ldots T_i)$. Using the definition of $f_i$ and applying the inequalities (8) and (13) we find that

$$\begin{aligned}\sum_{i=1}^{i_{\max}} |T_i| &= \sum_{i=1}^{i_{\max}} f_{i-1}(T_i) \cdot (M_{i-1} - M_i) \\ &\le \sum_{i=1}^{i_{\max}} \min\left(1, \frac{smt_k}{M_{i-1}}\right) \cdot (M_{i-1} - M_i) \,.\end{aligned}$$

The sequence $M_0, M_1, \ldots, M_{i_{\max}}$ is monotone decreasing with $M_0 = mst$ and $M_{i_{\max}} = 0$. Therefore we can estimate the sum by an integral as follows.

$$\begin{aligned}\sum_{i=1}^{i_{\max}} \min\left(1, \frac{smt_k}{M_{i-1}}\right) \cdot (M_{i-1} - M_i) &\le \int_{M_{i_{\max}}}^{M_0} \min\left(1, \frac{smt_k}{x}\right) dx \\ &= \int_0^{smt_k} 1\, dx + smt_k \int_{smt_k}^{mst} \frac{1}{x}\, dx \\ &= smt_k + smt_k \cdot \ln \frac{mst}{smt_k}\end{aligned}$$

Using $mst \le 2smt$ we obtain

$$\sum_{i=1}^{i_{\max}} |T_i| \;\le\; smt_k\,(1 + \ln 2) \;,$$

the desired bound for the length of the solution. □

It is not known wether the above analysis of the relative greedy algorithm is tight, i.e., whether its performance ratio is $1 + \ln 2$. The best lower bound on the performance ratio of the relative greedy algorithm was obtained in [16] and has a value of 1.330.

## 4 Loss Contracting Algorithm

The *loss* of a Steiner tree was introduced by Karpinski and Zelikovsky in [20]. It measures how much length is needed to connect the Steiner points of a full component to its terminals. The idea behind this concept is that we would like to choose only Steiner points that are also contained in an optimum solution. Of course, this is not possible for an approximation algorithm. By penalizing the choice of Steiner points that require long edges to connect them to a terminal, one tries to avoid at least bad choices.

Karpinski and Zelikovsky [20] use the general framework with a selection function that minimizes the weighted sum of the length and the loss of a Steiner tree. In a second step they take the output of this algorithm as input for the relative greedy algorithm and are able to prove a performance ratio of 1.644 for this algorithm.

The idea of Karpinski and Zelikovsky was generalized by Hougardy and Prömel [18]. They designed a seqence of algorithms each of which gets the output of its predecessor as its input. All algorithms in the sequence use the weighted sum of the length and the loss of a Steiner tree for greedy selection, but with different weights in each round. Hougardy and Prömel prove that by choosing the weights appropriately one obtains an approximation algorithm with performance ratio 1.598.

Very recently, Robins and Zelikovsky [28] incorporated the loss of a Steiner tree into a new selection function for the relative greedy algorithm and were able to show that the resulting algorithm has a performance ratio of 1.550. We are going to describe their algorithm in this section.

### 4.1 The Loss of a Steiner Tree

The loss of a set of Steiner vertices $A \subseteq S$ is a minimum length forest $Loss(A) \subseteq E$ in which every Steiner vertex $v \in S$ is connected to a terminal $r \in R$. The loss of a Steiner tree or a collection of full components is defined with respect to the corresponding tree edges. *Contracting* the loss of a full component means that for every edge between the loss components, a new edge with the same weight is inserted between the corresponding terminals. This is shown in Figure 6. We write $loss := |Loss|$.

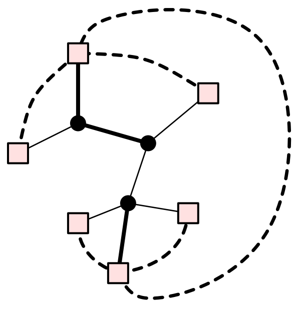


Figure 6: Contracting the loss of a Steiner tree. Terminal vertices are shown as square boxes and Steiner vertices as dots. Dark edges belong to the loss. For every thin edge between loss components, a dashed new edge with the same weight is inserted between the corresponding terminals.

We want to apply the Contraction Lemma in this setting as well. Therefore we have to make sure that the lengths of the newly inserted edges do not depend on previous loss contractions involving the same Steiner vertices. By a simple preprocessing (duplicating Steiner vertices), we can achieve that no two full components of the graph share a Steiner vertex. While the length of $SMT_k$ does not change, the instance grows by a factor which is at most a polynomial in the input size. The set $K$ from the general framework will refer to the preprocessed instance.

**Lemma 4.1 (Karpinski, Zelikovsky [20])** *The length of the loss of a Steiner tree is at most half of its total length.*

*Proof.* It suffices to prove the inequality $loss \leq smt/2$ for full components. It is easily seen that any full component can be transformed into a complete binary tree where the leaves of the tree are exactly the terminals. This can be achieved by adding new terminals and edges of length 0. Now for each internal vertex choose from the two edges leading to its children the cheapest one. This way one gets a subgraph that includes the loss of the full component with length at most half of the total length. □

There are examples for which Lemma 4.1 is essentially best possible. Consider an unweighted binary tree with $2^k$ terminals with an extra terminal attached to the root, as shown in Figure 7. Then $loss = 2^k - 1$ as we need one edge for every Steiner vertex and $smt = 2^{k+1} - 1$, so $\frac{loss}{smt} \to \frac{1}{2}$ as $k \to \infty$.

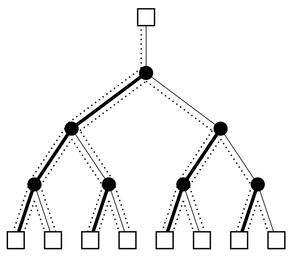


Figure 7: An example from a family of graphs with $mst \sim 2smt \sim 4loss$. Loss edges are dark. Dotted lines indicate edges used by $MST$.

## 4.2 Loss Contracting Algorithm

We are now prepared to describe the loss contracting algorithm of Robins and Zelikovsky [28]. It fits into the general framework for greedy algorithms. Unlike the relative greedy algorithm it does not contract the selected full component entirely, but only their loss.

Throughout the algorithm a terminal spanning tree is maintained. We denote its length by

$$m(\cdot) \ := \ mst(R/Loss(\cdot))\,.$$

$m$ is the length of a minimum spanning tree after the loss of certain full components has been contracted. Due to the preprocessing, we can model the effect of a loss contraction by adding new edges between terminals. The analysis does not rely on details the implementation of loss contractions. Therefore the use of the Contraction Lemma is justified.

Assume that the algorithm has already chosen some full components $T_1, \ldots, T_i$. Then the length of the corresponding Steiner tree is

$$cost(T_1, \ldots, T_i) \ = \ m(T_1, \ldots, T_i) + loss(T_1, \ldots, T_i) \qquad (14)$$

by the preprocessing step and the definition of $m$. The selection function

$$f_i(T) \ := \ \frac{loss(T)}{m(T_1 \ldots T_i) - m(T_1 \ldots T_i T)}$$

is applied to compare the *loss* of a new full component $T$ with its reduction of $m$. Thus the loss contracting algorithm fits into the general framework. We will see that $f_i(T_{i+1}) \leq 1$ for all $i$.

**Theorem 4.2 (Robins, Zelikovsky [28])** *The loss contracting algorithm computes a $1 + \frac{\ln 3}{2}$ approximation for $SMT_k$.*

We need some more notation for the proof. Equation (14) can be written shortly as $cost_i = m_i + loss_i$. Let $T_1^*, \ldots, T_{j_{\max}}^*$ be the full components of a Steiner minimum tree. Then $smt_k = m^* + loss^*$, where $m^* := m(T_1^* \ldots T_{j_{\max}}^*)$ and $loss^* := loss(T_1^* \ldots T_{j_{\max}}^*)$. The following lemma is the heart of the proof.

**Lemma 4.3** *The Steiner tree with full components $T_1, \ldots, T_{i_{\max}}$ returned by the loss contracting algorithm satisfies*

$$cost\,(T_1, \ldots, T_{i_{\max}}) \;\leq\; smt_k + loss^* \cdot \ln\left(1 + \frac{mst - smt_k}{loss^*}\right).$$

*Proof.* A full component $T$ reduces the length of the current intermediate solution if and only if $f_i(T) < 1$, because

$$\begin{aligned} & cost(T_1 \ldots T_i) - cost(T_1 \ldots T_i T) \\ &\quad = \; m(T_1 \ldots T_i) + loss(T_1 \ldots T_i) - m(T_1 \ldots T_i T) - loss(T_1 \ldots T_i T) \\ &\quad = \; m(T_1 \ldots T_i) - m(T_1 \ldots T_i T) - loss(T). \qquad (15) \end{aligned}$$

Following the lines of the proof for the relative greedy algorithm (Theorem 3.1), the next step is to bound the value of $f_i(T_{i+1})$ from above. Let $T_1^*, \ldots, T_{j_{\max}}^*$ be the full components of an optimal Steiner tree. Again, the greedy choice of the algorithm implies that

$$f_i(T_{i+1}) \;\leq\; \min_j f_i(T_j^*).$$

Using (10) we get

$$f_i(T_{i+1}) \;\leq\; \frac{\sum_j loss(T_j^*)}{\sum_j m(T_1 \ldots T_i) - m(T_1 \ldots T_i T_j^*)}.$$

Due to the Contraction Lemma the denominator is bounded from below by

$$\sum_j m(T_1 \ldots T_i T_1^* \ldots T_{j-1}^*) - m(T_1 \ldots T_i T_1^* \ldots T_{j-1}^* T_j^*), \qquad (16)$$

a telescoping sum equal to $m(T_1 \ldots T_i) - m(T_1 \ldots T_i T_1^* \ldots T_{j_{\max}}^*)$. By monotonicity

$$m(T_1 \ldots T_i T_1^* \ldots T_{j_{\max}}^*) \leq m(T_1^* \ldots T_{j_{\max}}^*), \qquad (17)$$

and we obtain the inequality

$$f_i(T_{i+1}) \;\leq\; \frac{loss(T_1^* \ldots T_{j_{\max}}^*)}{m(T_1 \ldots T_i) - m(T_1^* \ldots T_{j_{\max}}^*)} \;=\; \frac{loss^*}{m_i - m^*}. \qquad (18)$$

Using $f_i(T_{i+1}) \le 1$ and (18) we can now estimate

$$\begin{aligned} \mathit{loss}(T_1 \dots T_{i_{\max}}) &= \sum_i \mathit{loss}(T_i) = \sum_i f_{i-1}(T_i)(m_{i-1} - m_i) \\ &\le \sum_i \min\left(1, \frac{\mathit{loss}^*}{m_{i-1} - m^*}\right)(m_{i-1} - m_i). \qquad (19) \end{aligned}$$

Clearly $m_0 = \mathit{mst} \ge \mathit{smt}_k$, and we will show that $\mathit{smt}_k \ge m_{i_{\max}}$ in Lemma 4.4 below. Therefore (19) is bounded by

$$\begin{aligned} \int_{m_{i_{\max}}}^{\mathit{mst}} \min\left(1, \frac{\mathit{loss}^*}{x - m^*}\right) dx &= \int_{m_{i_{\max}} - m^*}^{\mathit{mst} - m^*} \min\left(1, \frac{\mathit{loss}^*}{x}\right) dx \\ = \int_{m_{i_{\max}} - m^*}^{\mathit{loss}^*} 1\, dx &+ \mathit{loss}^* \cdot \int_{\mathit{loss}^*}^{\mathit{mst} - m^*} \frac{dx}{x} \\ = \mathit{loss}^* - m_{i_{\max}} &+ m^* + \mathit{loss}^* \cdot \ln\left(\frac{\mathit{mst} - m^*}{\mathit{loss}^*}\right) \\ = \mathit{smt}_k - m_{i_{\max}} &+ \mathit{loss}^* \cdot \ln\left(1 + \frac{\mathit{mst} - \mathit{smt}_k}{\mathit{loss}^*}\right), \end{aligned}$$

and the lemma follows. □

***Proof of Theorem 4.2.*** Since $\mathit{smt}_k \ge \mathit{mst}/2$, we have $\mathit{mst} - \mathit{smt}_k \le \mathit{smt}_k$. It follows that

$$\mathit{cost}(T_1 \dots T_{i_{\max}}) \le \mathit{smt}_k \left(1 + \frac{\mathit{loss}^*}{\mathit{smt}_k} \cdot \ln\left(1 + \frac{\mathit{smt}_k}{\mathit{loss}^*}\right)\right).$$

Now we apply the inequality $\mathit{loss}^* \le \mathit{smt}_k/2$. Elementary calculus shows that $\max\left\{x \cdot \ln\left(1 + \frac{1}{x}\right) \,\middle|\, 0 \le x \le \frac{1}{2}\right\}$ is attained for $x = \frac{1}{2}$. Therefore

$$\mathit{cost}(T_1 \dots T_{i_{\max}}) \le \mathit{smt}_k \left(1 + \frac{\ln 3}{2}\right),$$

which concludes the proof of the theorem.

□

It remains to show that $m_{i_{\max}} \le \mathit{smt}_k$, which is a consequence of the following Lemma 4.4. The proof resembles somewhat the argumentation for equation (4) in the analysis of Algorithm $A_3$ of Berman and Ramaiyer. Lemma 4.4 will also be useful in Section 5, which is the reason why we prove a more general statement.

**Lemma 4.4** *If $T_1, \dots, T_i$ are full components such that adding another full component with at most $k$ terminals yields no further improvement, then $m_{i_{\max}} \le smt_k$. In this case, $loss(T_1, \dots, T_i) \le \lambda\, cost(T_1, \dots, T_i)$ implies $cost(T_1, \dots, T_i) \le \frac{1}{1-\lambda} smt_k$.*

*Proof.* Let $T_1^*, \dots, T_{j_{\max}}^*$ be a minimum $k$-Steiner tree. By (15) we have

$$m(T_1, \dots, T_{i_{\max}}) - m(T_1, \dots, T_{i_{\max}}, T_j^*) \;\le\; loss(T_j^*)$$

for all $j = 1, \dots, j_{\max}$. Using the Contraction Lemma in the second inequality, we find that

$$\begin{aligned}
m_{i_{\max}} - m^* \;&=\; m(T_1, \dots, T_{i_{\max}}) - m(T_1^*, \dots, T_{j_{\max}}^*) \\
&\le\; \sum_j m(T_1, \dots, T_{i_{\max}}, T_1^*, \dots, T_{j-1}^*) - m(T_1, \dots, T_{i_{\max}}, T_1^*, \dots, T_j^*) \\
&\le\; \sum_j m(T_1, \dots, T_{i_{\max}}) - m(T_1, \dots, T_{i_{\max}}, T_j^*) \\
&\le\; \sum_j loss(T_j^*) \;=\; loss^*,
\end{aligned}$$

that is, $m_{i_{\max}} \le smt_k$. Therefore

$$cost_{i_{\max}} \;=\; m_{i_{\max}} + loss_{i_{\max}} \;\le\; smt_k + loss_{i_{\max}} \;\le\; smt_k + \lambda cost_{i_{\max}},$$

and the lemma follows. □

As for the relative greedy algorithm, it is not known wether the analysis of Robin's and Zelikovsky's algorithm is tight. In [16] it is shown that 1.2 is a lower bound on the performance ratio of the loss contracting algorithm.

## 5 Special Instances of the Steiner Tree Problem

An instance of the Steiner tree problem is called *quasi-bipartite*, if the set $V \setminus R$ of (possible) Steiner vertices is stable, i. e., contains no edges. Quasi-bipartite instances appear in all known lower bound proofs for the approximation threshold of Steiner tree approximation algorithms (see Section 6). For such proofs one needs to construct instances to the Steiner tree problem that are in some sense the most difficult ones to solve. As we will see in this section, for quasi-bipartite instances there exist algorithms with better performance ratio as in the general case. This may have two reasons: Either quasi-bipartite instances are easier to solve than general instances of the Steiner tree problem. Then one should try to find better lower bound proofs based on other constructions. On the other hand, if quasi-bipartite instances are as hard to solve as general instances, then algorithms for these special instances should be useful as a basis to design better approximation algorithms in the general case.

In a quasi-bipartite instance all full components are stars, i. e., they have a single Steiner point and the loss is just one of its shortest edges. Since Steiner vertices of degree two can always be eliminated using the triangle inequality, we will also require w. l. o. g. that every full component has at least three edges. Thus, we have the following

**Proposition 5.1** *In quasi-bipartite instances, the length of the loss of a Steiner tree is at most one third of its total length.*

Rajagopalan and Vazirani [27] gave a $\frac{3}{2} + \varepsilon$ approximation algorithm for quasi-bipartite graphs based on the primal-dual method. The primal-dual method has been applied successfully to many network design problems (see e. g. [15]). Nevertheless, their result was surprising because in general it is considered difficult to obtain performance ratios better than 2 for Steiner tree like problems using this method. However, their algorithm is outperformed by a simple combinatorial algorithm, which we describe next.

## 5.1 Iterated 1-Steiner Heuristic

The iterated 1-Steiner heuristic is a simple local search heuristic. Recall that the Steiner minimum tree for a set of required vertices $R$ can be reconstructed if we know the set $I$ of its Steiner vertices since $SMT(R) = MST(R \cup I)$. Here the argument of $SMT(\cdot)$ and $MST(\cdot)$ denotes the set of vertices which has to be connected. Therefore the main problem is to find a good collection of Steiner vertices.

The heuristic starts from a spanning tree for the terminal set, i. e. $I = \emptyset$. In each step, we check whether the current solution can be improved by adding a single Steiner vertex $v$ in the following way: compute a minimum spanning tree on $R \cup I \cup \{v\}$, and remove all Steiner vertices of degree one and two. These are dispensable because of the triangle inequality. If the resulting Steiner tree is shorter, then let $I$ be the new set of its Steiner vertices, otherwise $I$ remains the same. The algorithm stops when no single Steiner vertex leads to an improvement in this way. See Figure 8.

```
Iterated 1-Steiner Heuristic

$I \leftarrow \emptyset$.

Repeat

    For every $v \in V \setminus (R \cup I)$ do:
        $I' \leftarrow I \cup \{v\}$.
        Remove vertices from $I'$ having degree 1 or 2 in $MST(R \cup I')$.
        If $mst(R \cup I') < mst(R \cup I)$ then $I \leftarrow I'$.

until no improvement found during last loop

Output $MST(R \cup I)$.
```

Figure 8: Iterated 1-Steiner Heuristic

**Theorem 5.2 (Robins, Zelikovsky [28])** *The iterated 1-Steiner heuristic achieves an approximation ratio of $\frac{3}{2}$ on quasi-bipartite instances.*

*Proof.* Due to Proposition 5.1 we can apply Lemma 4.4 with $\lambda = \frac{1}{3}$. □

A family of instances for which this upper bound on the performance ratio is asymptotically tight is shown in Figure 9. Here $SMT(R) = MST(R + s)$, so $smt = (2k+1)(1+\varepsilon)$. Assume that the iterated 1-Steiner heuristic has selected all Steiner vertices except $s$ so far. The edges incident to these vertices have length 1 and each of them was chosen as it reduced the $mst$ value by one. Since $s$ has degree 1 in $MST(V)$, the iterated 1-Steiner heuristic does not include $s$ into its current solution and stops with a Steiner tree of length $3k$. The resulting lower bound is $\frac{3k}{(2k+1)(1+\varepsilon)} \sim \frac{3}{2}$.

Note that it is natural to do the choice of the next Steiner vertex in a greedy way. The lower bound example of Figure 9 does not apply to this greedy version of the iterated 1-Steiner heuristic, because $s$ would be chosen in the first step. Minoux [24] showed how to speed up the greedy version of the iterated 1-Steiner heuristic for quasi-bipartite instances.

## 5.2 The Loss Contracting Algorithm in Special Graphs

The loss contracting algorithm has a performance ratio $\leq 1.279$ for two interesting classes of special instances [28]. Note that this is much better

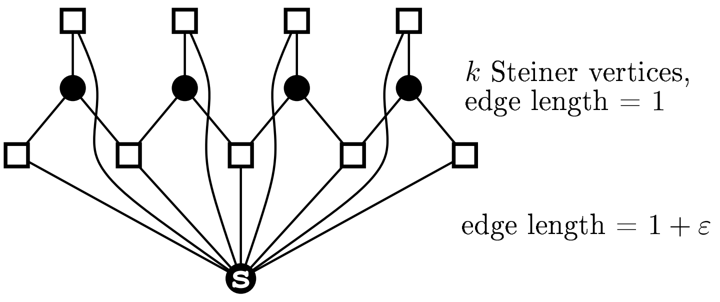


Figure 9: A worst-case example for the iterated 1-Steiner heuristic with vertex removal.

than the performance ratio of 1.550 that holds for the general case. The proof is based on an inequality which holds for these special instances. We use the notation of Section 4.

**Proposition 5.3 (Robins, Zelikovsky [28])** *In quasi-bipartite instances, the inequality $mst \leq 2(smt - loss^*)$ holds.*

*Proof.* It suffices to prove the inequality for full components. Let $r_1, r_2, \ldots$ be the required vertices and $v$ the Steiner vertex and assume w.l.o.g. that $Loss = \{r_1 v\}$. For $i \geq 2$, a shortest path between $r_i$ and $r_1$ has length at most $|r_i v| + |r_1 v| \leq 2|r_i v|$. □

In their proof that there exists a constant $\varepsilon > 0$ such that the Steiner tree problem in graphs cannot be approximated up to a factor $1 + \varepsilon$, Bern and Plassmann used a reduction from vertex cover [5]. The resulting instances are quasi-bipartite and have the property that the shortest distance between any two vertices is either 1 or 2. This makes the following special case interesting.

**Proposition 5.4** *If the instance is a complete graph and all edge lengths are either 1 or 2, then the inequality $mst \leq 2(smt - loss^*)$ holds.*

*Proof.* Denote by $r$ (resp. $s$) the number of required (resp. Steiner) vertices in $SMT_k$. Clearly $mst \leq 2(r-1)$ since all edge lengths are at most 2, and $smt_k \geq r + s - 1$ since $SMT_k$ contains $r + s$ vertices. We can assume that all loss edges have length 1, which implies $loss^* = s$. Therefore,

$$mst \leq 2(r-1) = 2\big((r+s-1) - s\big) \leq 2(smt_k - loss^*)$$

as claimed. □

**Theorem 5.5 (Robins, Zelikovsky [28])** *If the instance is quasi-bipartite, or the shortest distance between any two vertices is either* 1 *or* 2*, then the loss contracting algorithm computes a* 1.279 *approximation for* $SMT_k$.

*Proof.* In either case, we have $mst \leq 2(smt_k - loss^*)$ by Propositions 5.3 and 5.4, and this leads to a better estimate for the right hand side of Lemma 4.3. Since $1 + \frac{mst - smt_k}{loss^*} \leq \frac{smt_k}{loss^*} - 1$, we get

$$cost\,(T_1, \ldots, T_{i_{\max}}) \;\leq\; smt_k + loss^* \cdot \ln\left(\frac{smt_k}{loss^*} - 1\right) .$$

Now we use the inequality $loss^* \leq smt_k/2$. Numerically one can show that $\max\{x\big(\ln\big(\frac{1}{x} - 1\big)\big) \bigm| 0 \leq x \leq \frac{1}{2}\} \doteq 0.278465$. (This value is attained for $x \doteq 0.217812 < 1/3$.) The theorem follows. □

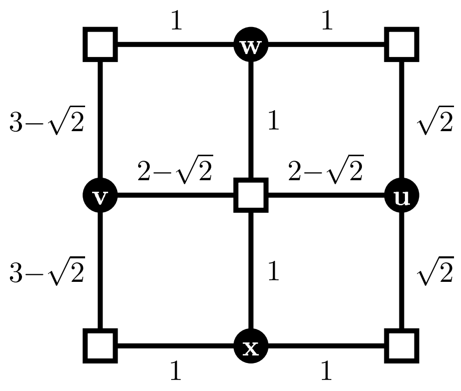
In quasi-bipartite graphs a full component that minimizes the function $f_i$ can be found in polynomial time even without the usual restriction on the terminal number, see [28] for details. Therefore, we can replace $smt_k$ with $smt$ in Theorem 5.5 for quasi-bipartite graphs. The total running time is $O(s^2 r)$, where $r$ is the number of terminals and $s$ is the number of non-terminals.

The performance ratio of the loss contracting algorithm in quasi-bipartite graphs is at least $(5 - \sqrt{2})/3 > 1.195$. This follows from the instance shown in Figure 10. The Steiner minimum tree consists of $T_w$ and $T_x$, whereas the algorithm chooses $T_u$ and $T_v$. After that, $T_w$ and $T_x$ are not improving anymore, so the algorithm stops. The resulting lower bound is $\big(|T_u| + |T_v|\big)/\big(|T_w| + |T_x|\big) = (10 - 2\sqrt{2})/6 \doteq 1.195$. The instance shown in Figure 11 implies a lower bound of 1.2 for the performance ratio of the loss contracting algorithm in general graphs. One can check that the Steiner minimum tree consists of the 'row trees', but the algorithm will choose the Steiner vertices $s$ and $t$. Detailed calculations can be found in [16].

## 5.3 Quasi-Bipartite Graphs with Uniform Edge Lengths

We call an instance of the Steiner tree problem *uniformly quasi-bipartite* if it is quasi-bipartite and edges incident to the same Steiner vertex have the same length. This class of instances contains in particular the unweighted instances and, more interestingly, the instances produced by all known reductions [5, 30] that prove lower bounds for the approximation threshold of the Steiner tree problem.

We present in this section an algorithm for such special instances which has a performance ratio of 1.217 for uniformly quasi-bipartite instances. This



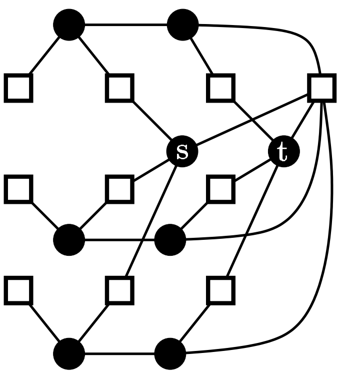


Figure 10: (left) A quasi-bipartite 1.195 lower bound example for the loss contracting algorithm.

Figure 11: (right) A non-quasi-bipartite 1.2 lower bound instance for the loss contracting algorithm. All edges of the row trees have length 5. The edges incident to $s$ and $t$ have weights 9 and 6, respectively.

algorithm uses the idea that the Steiner tree problem is similar to the set cover problem in some respect. Moreover, we can show that the analysis of the algorithm is tight.

For an instance of the Steiner tree problem and a constant $k$, we consider the following hypergraph on the set of terminals. A set of at most $k$ terminals is a hyperedge if the Steiner minimum tree for it is full. The length of this hyperedge is given by the length of the Steiner minimum tree. This hypergraph can be constructed in polynomial time. Finding a shortest $k$-Steiner tree is equivalent to finding a minimum spanning subgraph in this hypergraph. In general hypergraphs this problem is known as the *minimum spanning subset problem (MSS)*. It is a generalization of the set cover problem. If the hyperedges have size at most $k$, the greedy algorithm for this problem (Greedy-MSS) has a performance ratio of $H_k$, where $H_k = \sum_{i=1}^{k} 1/i = \ln k + O(1)$ [31, 2]. This is best possible [12]. A simple combination of this algorithm with the Steiner ratio analysis yields a performance ratio of $\rho_k \cdot H_{k-1} \geq 2$.

In case of the instances induced by the $k$-Steiner tree problem, we shall use the following property: If $t'$ is a subset of a set $t$ of terminals, then the 'cost' to connect $t'$ tends to be smaller than the cost to connect $t$. While this property is far from being valid in the general MSS problem, in uniformly quasi-bipartite instances the cost is even proportional to the number of terminals. This implies in particular that the solution contains no cycles.

Algorithm Greedy-MSS as described in [2] applied to uniformly quasi-bipartite instances fits into the general framework. Assume that the hyperedges $t_1, \ldots, t_{i-1}$ have already been selected and denote by $c_{i-1}(t)$ the decrease in the number of components if the hyperedge $t$ is added. Then the selection function is the 'cost per connection' ratio

$$f_{i-1}(t) := \frac{|t|}{c_{i-1}(t)}\,.$$

**Theorem 5.6 (Gröpl, Hougardy, Nierhoff, Prömel [16])** *The general framework with the function* $f_{i-1}(t) := \frac{|t|}{c_{i-1}(t)}$ *gives a* 1.217 *approximation for the Steiner tree problem in uniformly quasi-bipartite graphs.*

*Proof.* Let $t^*$ be a hyperedge of the optimal solution. We know that when the algorithm decided to select $t_i$, there was a competing minimal subhyperedge $t^{*\prime} \subset t^*$ with $c_{i-1}(t^{*\prime}) = c_{i-1}(t^*)$. As $t^{*\prime}$ corresponds to a star with $c_{i-1}(t^*)+1$ edges, we know that

$$|t^{*\prime}| = \frac{c_{i-1}(t^*)+1}{c_0(t^*)+1} \cdot |t^*|\,. \tag{20}$$

So the length of a subhyperedge is roughly proportional to the number of connections it yields (especially for large stars) and we get the following inequality.

$$\frac{|t_i|}{c_{i-1}(t_i)} \le \frac{|t^{*\prime}|}{c_{i-1}(t^{*\prime})} = \frac{\frac{c_{i-1}(t^*)+1}{c_0(t^*)+1}|t^*|}{c_{i-1}(t^*)} = \left(1 + \frac{1}{c_{i-1}(t^*)}\right)\frac{|t^*|}{c_0(t^*)+1}\,. \tag{21}$$

Let $ALG := \{t_1, \ldots, t_{i_{\max}}\}$ and $OPT$ be the hyperedges of an optimal solution. In order to estimate the length of $ALG$ in terms of the length of $OPT$, we introduce artificial edge sets $A$ resp. $A^*$ for the algorithmic and for the optimal solution to mimic the connections accomplished by the hyperedges of both solutions. Then we distribute the length of the edges in $A$ among those in $A^*$ using a matroid-style exchange argument.

Let $ALG_i := \{t_1, \ldots, t_i\}$. We define $\ell_i \in \mathbb{N}$ and $a_1, \ldots, a_{\ell_i} \in \binom{R}{2}$ such that $A_i := \{a_1, \ldots, a_{\ell_i}\}$ has the same connected components as $ALG_i$. We start with $\ell_0 := 0$ and $A_0 := ALG_0 = \emptyset$ and set $\ell_i := \ell_{i-1} + c_{i-1}(t_i) = \ell_{i-1} + c_0(t_i)$. The new edges $a_{\ell_{i-1}+1}, \ldots, a_{\ell_i} \subseteq t_i$ form an arbitrary spanning tree for the vertices of $t_i$. The length of $t_i$ is distributed uniformly among the new edges, i.e., their lengths are $|a_{\ell_{i-1}+1}|, \ldots, |a_{\ell_i}| := |t_i|/(\ell_i - \ell_{i-1})$. Let $A := A_{\ell_{i_{\max}}}$. (Clearly, $\ell_{i_{\max}}$ is the number of terminals minus one.)

Let $A_0^* := \emptyset$ and $T_0 := OPT$. We successively define trees $A_j^* \subseteq \binom{R}{2}$ and hypertrees $T_j$, $j > 0$. To define $A_j^*$ we insert the edges of $A$ one after another. For each $j = 1, \dots, \ell_{i_{\max}}$ there are two possibilities. If $a_j \in T_{j-1}$ then we let $a_j^* := a_j$ and $T_j := T_{j-1}$. Otherwise, $T_{j-1} + a_j$ contains a cycle. As $A$ is a tree, this cycle must contain a hyperedge $t^* \in T_{j-1} \setminus A$. We let $T'_{j-1} := T_{j-1} - t^*$ and choose an edge $a_j^* \subseteq t^*$ which connects precisely those two components of $T'_{j-1}$ which contain the endpoints of $a_j$. We remove one of the endpoints of $a_j^*$ from $t^*$ to obtain $t^{*\prime}$. Finally, we let $T_j := T'_{j-1} + t^{*\prime} + a_j$, which is again a spanning subhypergraph without cycles. In this way, we have replaced one of the connections in $OPT$ with one from $ALG$, or, more precisely, $A$. In both cases we set $A_j^* := A_{j-1}^* + a_j^*$. Let $A^* := A^*_{\ell_{i_{\max}}}$. The replacement process defines a bijection $\varphi\colon A \leftrightarrow A^*$ by $\varphi(a_j) := a_j^*$.

Note that for every edge $a^* \in A^*$ there is a unique hyperedge $t^* \in OPT$ with $a^* \subseteq t^*$, because $OPT$ contains no cycle. For every hyperedge $t^* \in OPT$, the set of connections which have not yet been removed from it during the replacement steps $j = 1, \dots, \ell_i$ is $U_i(t^*) := \{\, a^* \in A^* \setminus A^*_{\ell_i} \mid a^* \subseteq t^* \,\}$. Let $u_i(t^*) := \#U_i(t^*)$. Then the number of connections removed from $t^*$ during the replacement steps $j = \ell_{i-1} + 1, \dots, \ell_i$ is $u_{i-1}(t^*) - u_i(t^*)$.

The above definitions imply that

$$\begin{aligned}
|ALG| &= \sum_{i=1}^{i_{\max}} |t_i| \;=\; \sum_{a \in A} |a| \;=\; \sum_{a^* \in A^*} |\varphi^{-1}(a^*)| \\
&= \sum_{t^* \in OPT} \sum_{\substack{a^* \subseteq t^* \\ a^* \in A^*}} |\varphi^{-1}(a^*)| \\
&= \sum_{t^* \in OPT} \sum_{i=1}^{i_{\max}} \sum_{\substack{a^* \subseteq t^* \\ a^* \in A^*_{\ell_i} \setminus A^*_{\ell_{i-1}}}} |\varphi^{-1}(a^*)| \\
&= \sum_{t^* \in OPT} \sum_{i=1}^{i_{\max}} \big(u_{i-1}(t^*) - u_i(t^*)\big) \frac{|t_i|}{c_{i-1}(t_i)} \,. \qquad (22)
\end{aligned}$$

We can estimate the inner sum using (21) in the following way.

$$\begin{aligned}
&\sum_{i=1}^{i_{\max}} \big(u_{i-1}(t^*) - u_i(t^*)\big) \frac{|t_i|}{c_{i-1}(t_i)} \\
\le\; &\sum_{i=1}^{i_{\max}} \big(u_{i-1}(t^*) - u_i(t^*)\big) \left(1 + \frac{1}{c_{i-1}(t^*)}\right) \frac{|t^*|}{c_0(t^*) + 1}
\end{aligned}$$

$$= |t^*| \cdot \frac{u_0(t^*) - u_{i_{\max}}(t^*) + \sum_{i=1}^{i_{\max}} \frac{u_{i-1}(t^*) - u_i(t^*)}{c_{i-1}(t^*)}}{c_0(t^*) + 1} . \tag{23}$$

In the numerator we have $u_0(t^*) = c_0(t^*)$ and $u_{i_{\max}}(t^*) = 0$. The inequality $c_{i-1}(t^*) \geq u_{i-1}(t^*)$ is valid for all $i$, since $U_{i-1}(t^*) \subseteq A^*$ contains no cycles and its edges are contained in $t^*$, but, by definition, adding the whole $t^*$ to $ALG_{i-1}$ would yield $c_{i-1}(t^*)$ new connections. As $u_0(t^*), \ldots, u_{i_{\max}}(t^*)$ is a non-increasing sequence of natural numbers,

$$\begin{aligned}\sum_{i=1}^{i_{\max}} \frac{u_{i-1}(t^*) - u_i(t^*)}{c_{i-1}(t^*)} &\leq \sum_{i=1}^{i_{\max}} \frac{u_{i-1}(t^*) - u_i(t^*)}{u_{i-1}(t^*)} \\ &\leq H(u_0(t^*)) - H(u_{i_{\max}}(t^*)) \;=\; H(c_0(t^*)) .\end{aligned}$$

We have

$$\frac{c_0(t^*) + H(c_0(t^*))}{c_0(t^*) + 1} \leq \max_{x \in \mathbb{N}} \frac{x + H(x)}{x + 1} = \frac{4 + H(4)}{4 + 1} = \frac{73}{60} ,$$

so (23) is bounded by $\frac{73}{60}|t^*|$. Putting things together, we obtain

$$\begin{aligned}|ALG| &= \sum_{t^* \in OPT} \sum_{i=1}^{i_{\max}} \left(u_{i-1}(t^*) - u_i(t^*)\right) \frac{|t_i|}{c_{i-1}(t_i)} \\ &\leq \sum_{t^* \in OPT} |t^*| \cdot \frac{73}{60} \\ &= \frac{73}{60} \cdot |OPT| ,\end{aligned}$$

as desired. □

An interesting feature of the Greedy-MSS algorithm is that we have a matching lower bound instance, that is, the analysis is tight.

Let $p \in \mathbb{N}$. The terminals $\{r_{i,j}\}_{i,j=1,\ldots,p}$ of our instance are arranged in form of a $p \times p$ grid. We also have a terminal $r_0$. The set of potential Steiner vertices is $U \cup W$, where $U = \{u_1, \ldots, u_p\}$ and $W := \{w_1, \ldots, w_p\}$. Each $u_i$ is adjacent to $r_0$ and to $r_{i,j}$ for all $j$, and each $w_j$ is adjacent to $r_0$ and to $r_{i,j}$ for all $i$. The optimal solution uses the Steiner vertices $W$. All edges incident to $w_i \in W$ have length $\frac{1}{p+1}$, so $smt \leq p$. See Figure 12 for the case $p = 4$.

Let $v \in U \cup W$ and let $f_i(v) := \min_t f_i(t)$, where $t$ ranges over all subsets of the neighborhood of $v$. Note that the minimum is attained for all maximal

$t$ that intersect with each component of $(R, \{t_1, \dots, t_i\})$ at most once. As all these $t$ connect the same components, their effect is the same. Therefore, in uniformly quasi-bipartite instances the algorithm effectively chooses Steiner vertices $v_i$ minimizing $f_{i-1}(v_i)$.

We will force Greedy-MSS to choose the Steiner vertices $u_1, \dots, u_p$ in this order by defining the lengths of the edges incident to the Steiner vertices $u_i \in U$ in such a way that $f_{i-1}(u_i) \le f_{i-1}(w_j)$ for all $i$ and $j$. Note that adding an $\varepsilon > 0$ to the length of the optimal edges would break the tie in our favor. By symmetry, $f_{i-1}(w_j)$ is independent of $j$ in each step. Clearly the edge lengths for $u_1$ must be $\frac{1}{p+1}$, the same as for $w_1, \dots, w_p$. Given that the algorithm has already picked $u_1, \dots, u_i$, we calculate the edge lengths for $u_{i+1}$ as follows. For all $j$, the terminals $r_{1,j}, \dots, r_{i,j}$ are already connected to $r_0$, so we have $f_i(w_j) = \frac{p+1-i}{(p+1)(p-i)}$. On the other hand, $u_{i+1}$ will still connect $p$ terminals to $r_0$. To be competitive with $w_j$, its edge lengths can add up to $p \cdot f_i(w_j) = \frac{p(p+1-i)}{(p+1)(p-i)}$. Therefore we let the length of the edges incident to $u_i$ be $\frac{p(p+1-i)}{(p+1)^2(p-i)}$.

The ratio between the lengths of the algorithmic and the optimal solution is

$$\left( \sum_{i=0}^{p-1} \frac{p(p+1-i)}{(p+1)(p-i)} \right) \Bigg/ p = \sum_{i=0}^{p-1} \frac{(p-i)+1}{(p+1)(p-i)} = \frac{p + H(p)}{p+1} .$$

This fraction is maximized for $p = 4$ where it matches the guaranteed performance ratio of 73/60 proved above. The worst-case instance is shown in Figure 12.

An unweighted worst-case instance has been presented in [16].

## 6 Nonapproximability Results

A natural question about approximation algorithms for the Steiner tree problem is how small their performance ratio can get. In this section we survey some results that deal with this question.

As already mentioned in the introduction, it is known that the performance ratio of a polynomial time approximation algorithm for the Steiner tree problem in graphs can not get arbitrarily close to 1, unless $P = NP$. This is a consequence of the famous PCP-Theorem due to Arora, Lund, Motwani, Sudan and Szegedy [1] together with a reduction due to Bern and Plassmann [5].

**Theorem 6.1 (Arora et al. [1], Bern, Plassmann [5])** *There exists some constant $c > 1$ such that no polynomial time approximation algorithm*

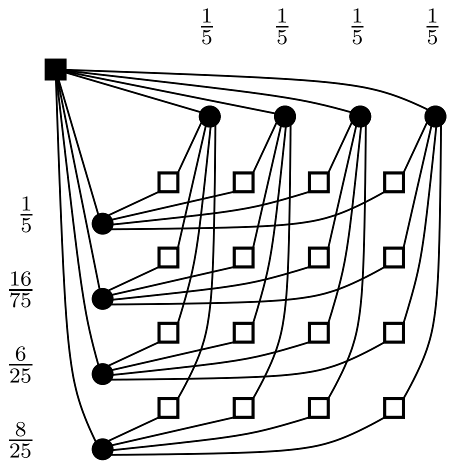


Figure 12: A worst-case instance for the algorithm for uniformly quasi-bipartite graphs.

*for the Steiner tree problem in graphs can have a performance ratio smaller than c, unless $P = NP$.*

The constant $c$ in the above theorem could in principle be extracted from the proofs, however it would turn out to be extremely close to 1. The reason for this is mainly that the constants involved in the first version of the PCP-Theorem are very large. Additionally, the reduction of Bern and Plassmann which reduces the problem Node-Cover-B to the Steiner tree problem in graphs, also loses in the non-approximability constant. Nevertheless, their reduction is still the tightest reduction known between these two problems and improved non-approximability results for the Node-Cover-B problem also improve the non-approximability results for the Steiner tree problem in graphs. Therefore we explain their reduction in more detail below.

The Node-Cover problem asks for a minimum set of nodes in a graph, such that every edge in the graph is incident to at least one node from this set. Node-Cover-B is the same problem, but restricted to graphs where the maximum degree of every node is bounded by a constant $B$. Bern and Plassmann have shown that good approximation algorithms for the Steiner tree problem in graphs also imply good approximation algorithms for Node-Cover-B.

**Lemma 6.2 (Bern, Plassmann [5])** *If there exists a polynomial time approximation algorithm for the Steiner tree problem in graphs with perfor-*

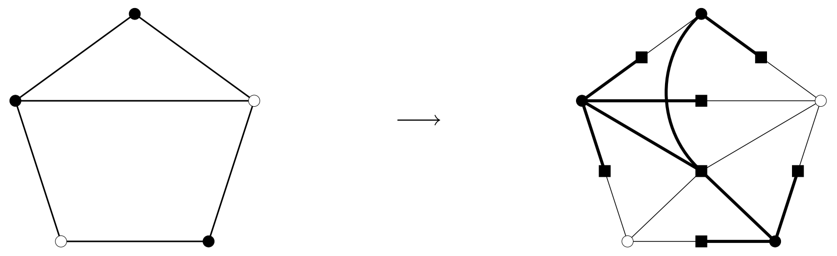


Figure 13: A reduction from Node-Cover-B to the Steiner tree problem in graphs

*mance ratio* $1+\varepsilon$*, then there exists a polynomial time approximation algorithm for Node-Cover-B with performance ratio* $1+(B+1)\varepsilon$*.*

*Proof.* Let $G=(V,E)$ be an instance to the Node-Cover-B problem. We will show how an $(1+\varepsilon)$-approximation algorithm for the Steiner tree problem in graphs can be used to get a $(1+(B+1)\varepsilon)$-approximation algorithm for Node-Cover-B.

We first transform the Graph $G$ into a graph $H$ that is an instance to the Steiner tree problem in graphs as follows: On each edge $e$ of $G$ place a terminal $t_e$. Furthermore add a terminal $t$ that is connected to all vertices in $G$. See Figure 13 for an example of this reduction.

We denote the length of a Steiner minimum tree in $H$ by $smt$ and the size of a minimum node-cover in $G$ by $nc$. First we show that $smt$ and $nc$ differ exactly by the number $m$ of edges in $G$:

$$smt = nc + m \tag{24}$$

To see this note first that any node cover in $G$ can be transformed into a Steiner tree for $H$ as follows. Connect terminal $t$ to the vertices from the node cover. Now each of the remaining $m$ terminals has a neighbor in the node cover. Thus, a Steiner tree is found of length $nc+m$.

Next we show how to get a node cover of size $smt-m$ from a Steiner tree $SMT$ of length $smt$. First we normalize the Steiner tree $SMT$ as follows. If there exists a terminal $t_e$ that has degree 2, then at least one of its two neighbors is not connected to $t$. Add this edge and destroy the resulting cycle by removing an edge that is not incident to $t$. This decreases the number of terminals different from $t$ that have degree 2 by one without increasing the length of the Steiner tree. By iterating this process we may assume that $SMT$ contains no terminals $t_e$ of degree 2. Now it is easily seen that the

Steiner vertices in $SMT$ constitute a node cover in $G$ of size $smt - m$. This proves Equation (24).

Suppose we have an approximation algorithm for the Steiner tree problem in graphs that returns a solution $SMT'$ of length $smt' \leq (1+\varepsilon) \cdot smt$ for the graph $H$. From this we can construct a node cover for $G$ of size $nc' = smt' - m$ as described in the proof of Equation (24). Therefore we get:

$$\begin{aligned} nc' = smt' - m &\leq (1+\varepsilon) \cdot smt - m \\ &= (1+\varepsilon) \cdot (nc + m) - m \\ &= (1+\varepsilon) \cdot nc + \varepsilon \cdot m \end{aligned}$$

Since $G$ is a graph with maximum degree $B$, any node cover in $G$ must have size at least $m/B$, i.e., we have $m \leq nc \cdot B$. Thus we get

$$nc' \leq (1+\varepsilon) \cdot nc + \varepsilon \cdot nc \cdot B = nc \cdot (1 + (B+1) \cdot \varepsilon),$$

proving the lemma. □

Lemma 6.2 immediately gives non-approximability results for the Steiner tree problem in graphs, as soon as one knows non-approximability results for node-cover-B. Currently, the strongest non-approximability result for node-cover-B is due to Berman and Karpinski:

**Theorem 6.3 (Berman, Karpinski [3])** *No polynomial time approximation algorithm for Node-Cover-4 can have a performance ratio below* 1.0128*, unless NP=coRP.*

From this result and Lemma 6.2 one immediately gets

**Lemma 6.4** *No polynomial time approximation algorithm for the Steiner tree problem in graphs can have a performance ratio below* 1.0025*, unless NP=coRP.*

This lower bound has been improved recently by Thimm [30] who used a non-approximability result of Håstad [17] for Max-E3-Lin-2 to obtain the following bound.

**Lemma 6.5 (Thimm [30])** *No polynomial time approximation algorithm for the Steiner tree problem in graphs can have a performance ratio below* 1.0074*, unless NP=coRP.*

To conclude, it remains a big gap between the currently known lower bound 1.0074 and the upper bound 1.550 on the approximation threshold for the Steiner tree problems in graphs. As the reduction by Thimm produces uniformly quasi-bipartite instances, we get that the approximation threshold in this case is between 1.0074 and 1.217.

## References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM 45 (1998), 501–555.

[2] G. Baudis, C. Gröpl, S. Hougardy, T. Nierhoff, H.J. Prömel, *Approximating minimum spanning sets in hypergraphs and polymatroids*, technical report, Humboldt-Universität zu Berlin, 2000.

[3] P. Berman and M. Karpinski, *On some tighter inapproximability results*, Electronic Colloquium on Computational Complexity, report TR98-065, 1998.

[4] P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, J. Algorithms 17 (1994), 381–408.

[5] M. Bern, P. Plassmann, *The Steiner problem with edge lengths 1 and 2*, Inform. Process. Lett. 32 (1989), 171–176.

[6] A. Borchers, D.-Z. Du, *The k-Steiner ratio in graphs*, SIAM J. Comput. 26 (1997), 857–869.

[7] G. Calinescu, C.G. Fernandes, H. Karloff, A. Zelikovsky, *A new approximation algorithm for finding heavy planar subgraphs*, technical report, 1998.

[8] J. Cheriyan, R. Ravi, *Approximation algorithms for network problems*, lecture notes, 1998.

[9] E. Choukhmane, *Une heuristique pour le problème de l'arbre de Steiner*, RAIRO Rech. Opér. 12 (1978), 207–212.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to algorithms*, McGraw-Hill, 1989.

[11] S. Dreyfus, R. Wagner, *The Steiner problem in graphs*, Networks 1 (1972), 195–207.

[12] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, J. ACM 45 (1998), 634–652.

[13] R. Floren, *A note on "A faster approximation algorithm for the Steiner problem in graphs"*, Inform. Process. Lett. 38 (1991), 177–178.

[14] E.N. Gilbert, H.O. Pollak, *Steiner minimal trees*, SIAM J. Appl. Math. 16 (1968), 1–29.

[15] M.X. Goemans, D.P. Williamson, *The primal-dual method for approximation algorithms and its applcations to network design problems*, In: D.S. Hochbaum (ed.), *Approximation algorithms for NP-hard problems*, PWS, Boston, Mass., 1995, 144–191.

[16] C. Gröpl, S. Hougardy, T. Nierhoff, H.J. Prömel, *Lower bounds for approximation algorithms for the Steiner tree problem*, In: Proceedings of 27th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2001, Springer LNCS.

[17] J. Håstad, *Some optimal inapproximability results*, In: Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, STOC 1997, 1–10.

[18] S. Hougardy and H.J. Prömel, *A 1.598 approximation algorithm for the Steiner problem in graphs*, In: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1999, 448–453.

[19] R.M. Karp, *Reducibility among combinatorial problems*, In: Complexity of Computer Computations, (Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972). New York: Plenum 1972, 85-103.

[20] M. Karpinski and A. Zelikovsky, *New approximation algorithms for the Steiner tree problems*, J. Comb. Optim. 1 (1997), 47–65.

[21] L. Kou, G. Markowsky, L. Berman, *A fast algorithm for Steiner trees*, Acta Inform. 15 (1981), 141–145.

[22] J.B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc. 7 (1956), 48–50.

This paper appeared in: Steiner Trees in Industry, X. Cheng and D.-Z. Du (Eds.), 235-279, Kluwer Academic Publishers, 2001

[23] K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Inform. Process. Lett. 27 (1988), 125–128.

[24] M. Minoux, *Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity*, INFOR 28 (1990), 221–233.

[25] R.C. Prim, *Shortest connection networks and some generalizations*, Bell System Technical Journal 36 (1957), 1389–1401.

[26] H.J. Prömel, A. Steger, *A new approximation algorithm for the Steiner tree problem with performance ratio 5/3*, J. Algorithms 36 (2000), 89–101.

[27] S. Rajagopalan, V.V. Vazirani, *On the bidirected cut relaxation for the metric Steiner problem*, In: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1999, 742–757.

[28] G. Robins, A. Zelikovsky, *Improved Steiner tree approximation in graphs*, In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000, 770–779.

[29] H. Takahashi, A. Matsuyama, *An approximate solution for the Steiner problem in graphs*, Math. Japon. 24 (1980), 573–577.

[30] M. Thimm, *On the approximability of the Steiner tree problem*, In: Mathematical Foundations of Computer Science 2001, MFCS 2001, Springer LNCS.

[31] L.A. Wolsey, *An analysis of the greedy algorithm for the submodular set covering problem*, Combinatorica 2 (1982), 385–393.

[32] A. Zelikovsky, *An 11/6-approximation algorithm for the network Steiner problem*, Algorithmica 9 (1993), 463–470.

[33] A. Zelikovsky, *A faster approximation algorithm for the Steiner tree problem in graphs*, Inform. Process. Lett. 46 (1993), 79–83.

[34] A. Zelikovsky, *Better approximation bounds for the network and Euclidean Steiner tree problems*, technical report CS-96-06, University of Virginia, 1996.

[35] A. Zelikovsky, *Improved approximations of maximum planar subgraph*, technical report, University of Virginia.