Approximating Steiner trees

Anders Schack-Nielsen Christian Wulff-Nilsen

January 2006

Abstract

We give a presentation of Robins and Zelikovsky's 1.55 approximation algorithm to the Steiner Tree Problem and a thorough proof of its approximation ratio. Furthermore we sketch a proof by Thimm bounding the approximability of the Steiner Tree Problem.

1 Introduction

Given a weighted undirected graph $G = (V, E)$ and a subset $Z \subseteq V$ of the vertices called *terminals*, a *Steiner tree* is a tree in G spanning all the terminals. Any non-terminals spanned by a Steiner tree are called *Steiner points*. Now the *Steiner Tree Problem* asks for a minimum cost Steiner tree.

Notice that we may trivially assume that the cost-function is a metric as we can always replace the edge costs by the cost of a shortest path.

The Steiner Tree Problem is known to be NP-hard even in the Euclidean and rectilinear metric.

When restricted to the Euclidean or rectilinear metric, the Steiner Tree Problem admits a PTAS, but in general graphs it does not. It does however admit an approximation factor. A simple approximation algorithm is to compute a minimum spanning tree, which is a factor 2 algorithm [9]. The best known approximation factor has been gradually decreased from 2 to the 1.55 presented below. The following table shows some of the development:

Year	Ratio	Authors
1993	1.834	Zelikovsky [10]
1994	1.746	Berman, Ramaiyer [1]
1996	1.693	Zelikovsky [11]
1997	1.667	Prömel, Steger [6]
1997	1.644	Karpinski, Zelikovsky [5]
1998	1.598	Hougardy, Prömel [3]
2000	1.550	Robins, Zelikovsky [7]

The final paper by Robins and Zelikovsky and their algorithm is the main topic of this paper.

The non-existence of a PTAS (given the assumption co-RP$\neq$NP) and a lower bound on the best possible approximation ratio is given in section 6.

2 Basic definitions, notation and properties

Let us start off by introducing some notational conveniences. In this paper we will in general be working with a weighted undirected graph $G = (V, E)$ in which a set of terminals $Z \subseteq V$ is given. We will often need to refer to various subgraphs of G and it will be convenient to represent these as sets of edges. Unless explicitly stated otherwise, we will consider the vertex sets of these subgraphs as the collection of endpoints of the edges, i.e. a subset of edges $H \subseteq E$ will be considered equal to the subgraph (V_H, H) where $V_H = \{v \mid \exists u.(v, u) \in H\}$.

We will assume that the cost-function is injective, as this implies uniqueness of minimum spanning trees and the like. This is no serious restriction, as it easily can be accomplished by a small perturbation of the edge weights or by any kind of secondary ordering simulating a perturbation.

It is convenient to extend the domain of the cost-function on the edges of G to arbitrary subgraphs such that $cost(H)$ is equal to the sum of the edge costs of H.

Spanning trees are central to our analysis, so we will need some notation for minimum spanning trees. Let $MST(H)$ be the minimum spanning tree on the graph H and let $mst(H)$ be its cost, i.e. $mst(H) = cost(MST(H))$. Let $G_{V'}$ denote the subgraph of G induced by the set of vertices $V' \subseteq V$. We write MST as a shorthand for $MST(G_Z)$ and mst for $mst(G_Z)$. A *terminal-spanning tree* is a Steiner tree without any Steiner points. Hence, MST is the minimum terminal-spanning tree.

The cost of a minimum Steiner tree will be denoted opt.

A *full component* is a Steiner tree w.r.t. a subset of the terminals $Z' \subseteq Z$ in which all the terminals in Z' are leaves. Any Steiner tree can be decomposed into its full components by splitting it at any non-leaf terminals. In a terminal-spanning tree, all full components are single edges.

During the course of the algorithm by Robins and Zelikovsky we will aim to reduce the cost of the current Steiner tree by adding full components. Adding a full component may reduce the cost of the Steiner tree and in this case it gives an immediate gain through the reduced cost, but it also gives a potential loss, since the choices that are made may be incompatible with the optimal solution.

We will give the formal definitions of *gain* and *loss* below.

We will need to define *gain* on a slightly larger set of subgraphs than the full components. Also note that we are only defining *gain* with respect to a terminal-spanning tree as opposed to an arbitrary Steiner tree; the reason for this will become clear later on.

Definition 1. *Let T be a terminal-spanning tree, and let H be an acyclic subgraph of G containing at least one terminal in each connected component (H does not need to be connected). Then the* augmentation*, $T[H]$, is defined to be the minimum cost tree in $H \cup T$ containing all edges of H and spanning Z (see Figure 1). Now the* gain *of H with respect to T is defined as $gain_T(H) = cost(T) - cost(T[H])$.*

The requirement that H should include at least one terminal in each connected component ensures that $H \cup T$ is a connected graph and thus has a spanning tree.[1]

Some useful properties of *gain* are immediate. If H is a Steiner tree then $T[H]$ equals H and therefore $gain_T(H) = cost(T) - cost(H)$. We also have the inequality $gain_T(H) \leq cost(T) - mst(T \cup H)$ since $T[H]$ cannot cost less than $mst(T \cup H)$. The following property is more involved and we state it as a lemma:

[1] Robins and Zelikovsky does not seem to notice this, as they define *gain* on arbitrary graphs.

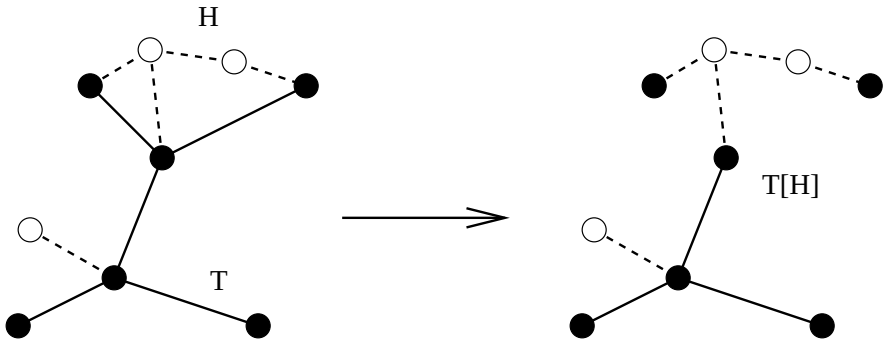

Figure 1: On the left: A terminal-spanning tree T (solid edges) along with a graph H (dashed edges). On the right: The augmentation $T[H]$.

Lemma 1. *For any terminal-spanning tree T and edge-disjoint Steiner point-disjoint trees, H and H', sharing at most a single terminal,*

$$gain_T(H \cup H') \leq gain_T(H) + gain_T(H').$$

(H and H' are assumed to contain at least one terminal each, as gain otherwise would not be defined.)

Proof. By definition,

$$gain_T(H \cup H') = cost(T) - cost(T[H \cup H'])$$

and

$$gain_T(H) + gain_T(H') = 2cost(T) - cost(T[H]) - cost(T[H']).$$

Thus we need to show

$$cost(T) - cost(T[H \cup H']) \leq 2cost(T) - cost(T[H]) - cost(T[H'])$$

or equivalently,

$$cost(T[H]) + cost(T[H']) \leq cost(T) + cost(T[H \cup H']).$$

To do this, we will move some edges from T to $T[H \cup H']$ and vice versa so that T becomes a tree T_H in $T \cup H$ spanning Z and containing H and $T[H \cup H']$ becomes a tree $T_{H'}$ in $T \cup H'$ spanning Z and containing H'. Since we only move edges between the two trees, this procedure will not change the value of the right hand side above, i.e.

$$cost(T_H) + cost(T_{H'}) = cost(T) + cost(T[H \cup H']).$$

By definition, $cost(T[H]) \leq cost(T_H)$ and $cost(T[H']) \leq cost(T_{H'})$, showing our claim.

To construct T_H and $T_{H'}$ we will find a partition of T into disjoint edge sets T_1 and T_2 such that $H \cup T_1 = T_H$ and $(T[H \cup H'] \setminus H) \cup T_2 = T_{H'}$ each connect all terminals (since H and H' are edge-disjoint, T_H and $T_{H'}$ trivially contain H and H' respectively).

Consider the forest $T[H \cup H'] \setminus H$ consisting of the components $C_1, \ldots, C_c$ (see Figure 2) where we include all terminals but exclude any Steiner points occuring in H, i.e. the C_i include

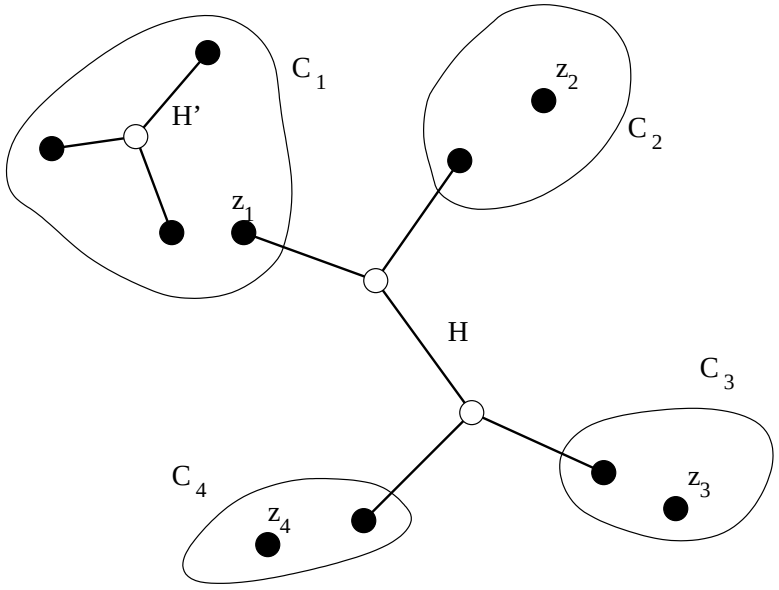

Figure 2: The tree $T[H \cup H']$ is shown along with the placement of H and H' relative to the components $C_1, \ldots, C_c$. The terminal z_1 is the terminal connecting C_1 and H, while $z_2, \ldots, z_c$ are the first terminals in $C_2, \ldots, C_c$ encountered when traversing T from z_1.

isolated points if and only if they are terminals. Notice that the terminals in H therefore are in one to one correspondance with the listed components with one terminal of H included in each of the C_i. Now since H' is connected, it must be included in one of these components, say C_1. Let z_1 be the terminal from H in C_1. Now we root the tree T at z_1 and consider a depth-first traversal of T. For $1 < i \leq c$ we let z_i be the terminal encountered first among the terminals of C_i. Now we define the partition of T as follows: Let $T_2 = \{(a_i, z_i) \mid 1 < i \leq c\}$ and $T_1 = T \setminus T_2$ where a_i is the ancestor of z_i for $1 < i \leq c$. First consider $T_{H'}$. Since the traversal of T starts in the component C_1 and visits all terminals adding the edges (a_i, z_i) to $T_{H'}$, this exactly connects all the components, thus making $T_{H'}$ span Z as needed. Next consider T_H. To prove that T_H spans Z, we will consider each terminal in the order of the traversal and prove that it is connected to z_1 in T_H, i.e. we do an induction over the tree T rooted at z_1. The root of the traversal is trivial; z_1 is certainly connected to itself. Now let $z \neq z_1$ be some terminal and let a be its ancestor. The induction hypothesis tells us that a is connected to z_1 in T_H. If z is not any of the z_i, it means that (a, z) is included in T_1 and we are home free. If $z = z_i$, we have a path contained in $C_i \subseteq T$ from z_i to the terminal shared by C_i and H. Since this path does not cross between components, its edges cannot be in T_2 and therefore connects z to z_1 in T_H through H as needed. □

Consider the course of the algorithm once more. Starting from a terminal-spanning tree and adding a full component based on its *gain* we arrive at a tree which is no longer a terminal-spanning tree (it contains Steiner points). We could continue by either contracting the entire full component or expanding the definition of *gain*. But since it is not clear how to evaluate the gain of full components which conflicts with previous chosen full components, both approaches eliminate the possibility of changing previous choices should better alternatives appear. Therefore we would like to represent our current best Steiner tree as a

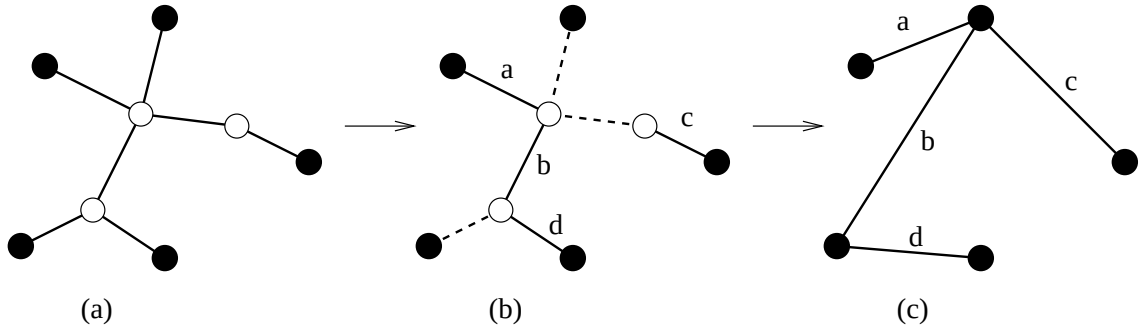

Figure 3: (a) A full component K. (b) $Loss(K)$ is marked by the dashed edges. (c) The corresponding tree $\mathcal{C}[K]$ with contracted $Loss(K)$.

terminal-spanning tree in which all full components may be considered.

To give some intuition, the idea of this representation is to reduce the cost of certain edges to zero and update the cost of the edges corresponding to the new shortest paths in such a way that the current Steiner tree may equally well be represented as a terminal-spanning tree. This reduction of the edge costs can be viewed as choosing certain edges, recording their cost and then contracting them.

Definition 2. *Given a full component, K, we define $Loss(K)$ to be the minimum cost forest in K such that each connected component contains at least one terminal. The cost of $Loss(K)$ is denoted $loss(K)$, i.e. $loss(K) = cost(Loss(K))$. The Loss (loss) of a disjoint union of full components, e.g. a Steiner tree, is the union (sum) of their individual losses.*

Note that even though the above definition just states that the connected components of $Loss(K)$ should contain at least one terminal, they will in fact always contain exactly one terminal, since otherwise $Loss(K)$ would not be minimal.

When we contract the edges of $Loss(K)$ we get a terminal-spanning tree which will be denoted $\mathcal{C}[K]$ (see Figure 3).

Definition 3. *Given a full component, K, we define $\mathcal{C}[K]$ to be the spanning tree of the terminals of K in which two terminals are connected if K has an edge between the two corresponding connected components in $Loss(K)$. The edge costs in $\mathcal{C}[K]$ are given by the edge costs of the corresponding edges in $K \setminus Loss(K)$. Similarly, for any Steiner tree H, $\mathcal{C}[H]$ is the terminal-spanning tree in which the losses of all full components are contracted.*

We note that the definition of $\mathcal{C}[K]$ implies that $cost(\mathcal{C}[K]) = cost(K) - loss(K)$.

Since the number of full components may be exponential we will consider certain classes of full components.

Definition 4. *A Steiner tree is* k-restricted *if each of its full components has at most k terminals.*

Since there always exists a minimum Steiner tree in which each Steiner point has degree at least three, we can disregard full components containing Steiner points of degree less than three. Hence, the number of relevant k-restricted full components is polynomial for fixed k.

We denote the optimal k-restriced Steiner tree by Opt_k (if there are more[2], we choose one) and its cost by opt_k. The loss of Opt_k will be denoted $loss_k$.

The following lemma will be useful when analyzing the final iteration of the algorithm.

Lemma 2. *Let H be a Steiner tree. If $gain_{\mathcal{C}[H]}(K) \leq 0$ for every k-restricted full component K then, by Lemma 1,*

$$cost(H) - loss(H) = cost(\mathcal{C}[H]) \leq opt_k.$$

Proof. If $K_1, \ldots, K_p$ are the full components of Opt_k then

$$\begin{aligned} cost(\mathcal{C}[H]) - opt_k &= gain_{\mathcal{C}[H]}(Opt_k) \\ &= gain_{\mathcal{C}[H]}(K_1 \cup \cdots \cup K_p) \\ &\leq gain_{\mathcal{C}[H]}(K_1) + \cdots + gain_{\mathcal{C}[H]}(K_p) \\ &\leq 0 \end{aligned}$$

□

3 The k-LCA algorithm

In this section we present the Loss-Contracting Algorithm (k-LCA) by Robins and Zelikovsky [7].

The algorithm maintains the current solution represented as a terminal-spanning tree in T, which is initialized to the minimum spanning tree of the terminals. One may view the algorithm as follows: Every time the algorithm adds a full component, the *Loss* of the component is added to the final tree and then the selected edges have their weight reduced to 0 (since we have locked the choice of these edges their cost no longer matter). After the reduction of the edge costs, the tree is represented as a terminal-spanning tree of the same cost. Now another full components is found, its *Loss* selected, and so on. At the end, when no full component can reduce the cost of the tree, we have a terminal-spanning tree and a large collection of selected edges. The solution represented at this point is thus some Steiner tree which includes all the endpoints of the selected edges. If we therefore return the minimum spanning tree of all these points and all terminals, we end up with a solution which could be better and is guaranteed to be no worse.[3] Thus there is no need to remember the selected edges during the course of the algorithm as long as we remember the Steiner points included. The set of included Steiner points and the terminals is maintained as the set H.

The components chosen by the algorithm are based on a greedy gain-over-loss objective. The intuition behind this is as follows. The cost of the approximate solution lies somewhere between *mst* and opt_k (actually we cannot rule out the possibility that we might end up with an even better solution, since the final step may produce a solution, which is not k-restricted). When we accept a component K with a positive gain, we increase the gap between *mst* and the cost of the approximation by this amount. On the other hand, if K does not belong to Opt_k, then we can no longer reach Opt_k, because we would have to pay for the incorrectly chosen edges. Therefore the *loss* of K is an upper bound on the increase in the gap between opt_k and the best achievable solution after accepting K.

[2]Consider for instance K_3 with two terminals and edge weights 1 and 2, and 3 between the terminals.

[3]The algorithm in the original article [7] does not seem to perform this obvious optimization, as it claims that the output of the algorithm is a k-restricted Steiner tree, but the line of thought of the original authors is a bit unclear on this point.

Presented in pseudo-code the algorithm is as follows:

$T := MST(G_Z)$
$H := Z$
Repeat
  Find a k-restricted full component K maximizing $r = gain_T(K)/loss(K)$
  If $r \leq 0$ **then exit repeat**
  $H := H \cup V_K$
  $T := MST(T \cup \mathcal{C}[K])$
Return the tree $MST(G_H)$

Note that $T \cup \mathcal{C}[K]$ may contain multiple copies of the same edge, but in this case the cheapest takes precedence, i.e. the edge from $\mathcal{C}[K]$.

When a component is accepted, its gain becomes 0 in subsequent iterations. Thus each component can be added only once and since there is only a polynomial number of relevant k-restricted components, the algorithm terminates in polynomial time.

We will prove (Corollary 1) that the approximation ratio of the algorithm converges to $1 + \frac{\ln 3}{2} < 1.55$ when $k \to \infty$. In certain special cases we can prove an even stronger approximation ratio and this will be the topic of section 5.

4 General graphs

In this section, we analyze the k-LCA algorithm on general graphs. Letting $1 \leq i \leq last$ range over the iterations of the algorithm, we let K_i be the full component chosen in the i-th iteration and T_i be the trees T constructed after the i-th iteration. Correspondingly T_0 is set to be the initial tree $MST(G_Z)$ chosen before the first iteration.

We will need the following inequality on fractions.

Lemma 3. *For finite sequences a_i and b_i of positive numbers, the following holds:*

$$\frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \left\{ \frac{a_i}{b_i} \right\}$$

Proof. The proof simply consists of noticing that the left hand side is a convex combination of the fractions a_i/b_i. Let $d_j = b_j / \sum_i b_i$. Notice that $\sum_i d_i = 1$ and therefore

$$\frac{\sum_i a_i}{\sum_i b_i} = \sum_i \frac{a_i}{b_i} d_i \leq \max_i \left\{ \frac{a_i}{b_i} \right\}$$ □

We will need to relate the cost of the trees T_i in different iterations. To do this we start by defining the *supergain* of a graph H with respect to a terminal-spanning tree T.

Definition 5. *Let T be a terminal-spanning tree and let H be a Steiner tree or a full component. Then*

$$supergain_T(H) = gain_T(H) + loss(H).$$

Now we get the following relation between the cost of the trees T_i:

Lemma 4. *The supergain of the chosen components equals the cost-reduction of the constructed trees:*

$$supergain_{T_{i-1}}(K_i) = cost(T_{i-1}) - cost(T_i)$$

Proof. Assume that $T_{i-1}[K_i] = MST(T_{i-1} \cup K_i)$. Then

$$\begin{aligned} supergain_{T_{i-1}}(K_i) &= gain_{T_{i-1}}(K_i) + loss(K_i) \\ &= cost(T_{i-1}) - mst(T_{i-1} \cup K_i) + mst(T_{i-1} \cup K_i) - mst(T_{i-1} \cup \mathcal{C}[K_i]) \\ &= cost(T_{i-1}) - cost(T_i) \end{aligned}$$

Thus, we will just have to prove our assumption. Assume to the contrary that some edge e of K_i does not belong to $MST(T_{i-1} \cup K_i)$. Removing e from K_i splits it into two connected components, A and B. We will show that either A or B has a larger gain-over-loss ratio, which contradicts the choice of K_i.

Since $e \notin MST(T_{i-1} \cup K_i)$, we have $cost(T_{i-1}[A \cup B]) < cost(T_{i-1}[K_i])$ and therefore $gain_{T_{i-1}}(K_i) < gain_{T_{i-1}}(A \cup B)$. Since A and B fulfills the requirements of Lemma 1, this gives $gain_{T_{i-1}}(K_i) < gain_{T_{i-1}}(A) + gain_{T_{i-1}}(B)$. Adding e to $MST(T_{i-1} \cup K_i)$ creates a cycle on which e is the longest edge. This implies that e is the longest edge on a path in K_i between some pair of terminals, and therefore cannot belong to $Loss(K_i)$. Thus $Loss(K_i) = Loss(A) \cup Loss(B)$ and $loss(K_i) = loss(A) + loss(B)$. Putting the parts together and using Lemma 3, we finally get

$$\frac{gain_{T_{i-1}}(K_i)}{loss(K_i)} < \frac{gain_{T_{i-1}}(A) + gain_{T_{i-1}}(B)}{loss(A) + loss(B)} \leq \max\left\{ \frac{gain_{T_{i-1}}(A)}{loss(A)}, \frac{gain_{T_{i-1}}(B)}{loss(B)} \right\}. \qquad \square$$

Let $G_i = supergain_{T_i}(Opt_k)$ for $0 \leq i \leq last$. Let $loss(n)$ be the sum of the losses of the n first accepted full components, i.e. $loss(n) = \sum_{i=1}^{n} loss(K_i)$. We will now prove the following bound on the loss of the selected components.

Lemma 5. *Assume that* $cost(T_{n+1}) \leq opt_k < cost(T_n)$. *Then*

$$\frac{loss(n) + a \cdot loss(K_{n+1})}{loss_k} \leq \ln \frac{G_0}{loss_k}$$

where $a = \frac{cost(T_n) - opt_k}{cost(T_n) - cost(T_{n+1})}$.

Proof. To shorten the notation in the proof we set $l_i = loss(K_i)$ and $g_i = supergain_{T_{i-1}}(K_i)$. Let Opt_k consist of full components X_j. Now for any $1 \leq i \leq n+1$ we have

$$\frac{G_{i-1}}{loss_k} \leq \frac{\sum_j supergain_{T_{i-1}}(X_j)}{\sum_j loss(X_j)} \leq 1 + \max_j \left\{ \frac{gain_{T_{i-1}}(X_j)}{loss(X_j)} \right\} \leq 1 + \frac{gain_{T_{i-1}}(K_i)}{loss(K_i)} = \frac{g_i}{l_i}$$

where the first inequality follows from Lemma 1, the second follows from Lemma 3 and the third from the definition of K_i.

Let $G_n^+ = loss_k$. From the assumption $cost(T_{n+1}) \leq opt_k < cost(T_n)$ we get $G_{n+1} \leq G_n^+ < G_n$ by subtracting opt_k and adding $loss_k$. Now since $G_{i-1} - G_i = cost(T_{i-1}) - cost(T_i) = g_i$ (Lemma 4) and $G_n - G_n^+ = cost(T_n) - opt_k = a \cdot g_{n+1}$ (Lemma 4), we have $G_i = G_{i-1} - g_i \leq$

$G_{i-1}\left(1-\frac{l_i}{loss_k}\right)$ and $G_n^+ = G_n - a \cdot g_{n+1} \leq G_n\left(1-a\frac{l_{n+1}}{loss_k}\right)$. Multiplying these inequalities yields

$$\frac{G_n^+}{G_0} \leq \left(1-a\frac{l_{n+1}}{loss_k}\right)\prod_{i=1}^{n}\left(1-\frac{l_i}{loss_k}\right)$$

Taking the logarithm and using $x \geq \ln(1+x)$ we get

$$\ln\frac{G_0}{G_n^+} \geq \sum_{i=1}^{n}\frac{l_i}{loss_k} + a\frac{l_{n+1}}{loss_k} = \frac{loss(n)+a\cdot l_{n+1}}{loss_k}$$ □

We are now able to prove an upper bound on the approximate solution produced by the algorithm.

Theorem 1. *For any instance of the Steiner Tree Problem, the cost, Approx, of the Steiner tree produced by algorithm k-LCA is at most:*

$$Approx \leq loss_k \cdot \ln\left(1+\frac{mst-opt_k}{loss_k}\right) + opt_k.$$

Proof. By Lemma 2, when the algorithm terminates, the cost of the last tree T_{last} will be at most opt_k. We can therefore consider the iteration n in which $cost(T_{n+1}) \leq opt_k < cost(T_n)$. Take a to be $\frac{cost(T_n)-opt_k}{cost(T_n)-cost(T_{n+1})}$. Now since in each iteration $cost(T_i)+loss(i)$ is an upper bound on $Approx$,

$$\begin{aligned}
Approx &\leq a(cost(T_{n+1})+loss(n+1)) + (1-a)(cost(T_n)+loss(n)) \\
&= \frac{(cost(T_n)-opt_k)cost(T_{n+1}) + (opt_k - cost(T_{n+1}))cost(T_n)}{cost(T_n)-cost(T_{n+1})} + a\cdot l_{n+1} + loss(n) \\
&= opt_k + a\cdot l_{n+1} + loss(n) \\
&\leq opt_k + loss_k \cdot \ln\frac{G_0}{loss_k} \\
&= opt_k + loss_k \cdot \ln\frac{mst-opt_k+loss_k}{loss_k}
\end{aligned}$$ □

The estimate of the approximation factor of k-LCA is based on optimal k-restricted Steiner trees. Thus, the results we are able to obtain, critically depend on the ratio $\frac{opt_k}{opt}$. Let ρ_k be the worst-case ratio of $\frac{opt_k}{opt}$. It has been shown in [2] that

$$\rho_k \leq 1 + \frac{1}{\lfloor \log_2 k \rfloor + 1}. \tag{1}$$

Theorem 2. *Algorithm k-LCA has an approximation ratio of at most* $(1+\frac{1}{2}\ln(\frac{4}{\rho_k}-1))\rho_k$.

Proof. Since our distance function is metric, $mst \leq 2opt$, see [9]. Thus, by Theorem 1,

$$Approx \leq loss_k \cdot \ln\left(1+\frac{mst-opt_k}{loss_k}\right) + opt_k \leq loss_k \cdot \ln\left(1+\frac{2opt-opt_k}{loss_k}\right) + opt_k.$$

Now, consider the function $f :]0,\infty[\rightarrow \mathbb{R}_+$, defined by

$$f(x) = x\ln\left(1+\frac{2opt-opt_k}{x}\right) + opt_k.$$

By the above, f is well-defined and the derivative f' is

$$\begin{aligned} f'(x) &= \ln\left(1 + \frac{2opt - opt_k}{x}\right) + x \cdot \left(\frac{x}{x + 2opt - opt_k}\right)\left(-\frac{2opt - opt_k}{x^2}\right) \\ &= \ln\left(1 + \frac{2opt - opt_k}{x}\right) - \frac{2opt - opt_k}{x + 2opt - opt_k} \end{aligned}$$

The second order derivative f'' of f is

$$\begin{aligned} f''(x) &= \frac{x}{x + 2opt - opt_k}\left(-\frac{2opt - opt_k}{x^2}\right) + \frac{2opt - opt_k}{(x + 2opt - opt_k)^2} \\ &= \frac{2opt - opt_k}{x + 2opt - opt_k}\left(\frac{1}{x + 2opt - opt_k} - \frac{1}{x}\right). \end{aligned}$$

Since $2opt - opt_k \geq mst - opt_k \geq 0$ it follows that $f''(x) \leq 0$ for all $x > 0$. Thus, f' is a non-increasing function. Since

$$\lim_{x \to \infty} f'(x) = \ln 1 = 0,$$

the derivative of f is always non-negative. As proven in [5], for any Steiner tree T, $loss(T) \leq \frac{1}{2} cost(T)$. In particular, this holds for an optimal k-restricted Steiner tree, i.e. $loss_k \leq \frac{1}{2} opt_k$. Since f is non-decreasing, the maximum value of the upper bound on $Approx$ is therefore achieved when $loss_k = \frac{1}{2} opt_k$ and we obtain

$$\frac{Approx}{opt} \leq \frac{opt_k}{opt} \cdot \left(1 + \frac{\ln\left(\frac{4opt}{opt_k} - 1\right)}{2}\right).$$

The right-hand side above increases when opt_k/opt increases. To see this, it suffices to show that the function $g :]0, 2] \to \mathbb{R}$, defined by

$$g(x) = x + \frac{x}{2} \ln\left(\frac{4}{x} - 1\right)$$

is a non-decreasing function. The derivative g' of g is

$$g'(x) = 1 + \frac{1}{2} \ln\left(\frac{4}{x} - 1\right) + \frac{x}{2} \frac{x}{4 - x}\left(-\frac{4}{x^2}\right) = 1 + \frac{1}{2} \ln\left(\frac{4}{x} - 1\right) - \frac{2}{4 - x}.$$

Note that g' is a decreasing function. Since

$$g'(2) = 1 + \frac{1}{2} \ln 1 - 1 = 0,$$

$g'(x) \geq 0$ for all $x \in]0, 2]$. Thus, g is a non-decreasing function.

Since ρ_k is the worst-case ratio of $\frac{opt_k}{opt}$, we obtain from the above

$$\frac{Approx}{opt} \leq \left(1 + \frac{1}{2} \ln\left(\frac{4}{\rho_k} - 1\right)\right) \rho_k,$$

which shows the theorem. □

Theorem 2 combined with the bound (1) immediately implies the following corollary:

Corollary 1. *The approximation ratio of k-LCA converges to $1 + \frac{\ln 3}{2} < 1.55$ when $k \to \infty$.*

5 Quasi-bipartite graphs and 1-2 graphs

In this section we will consider certain restricted classes of graphs, namely quasi-bipartite graphs and 1-2 graphs.

Definition 6. *A* quasi-bipartite graph *is a graph in which no two non-terminals are adjacent.*

For the definition of quasi-bipartite graph to make sense in this context, we will further require that such a graph is connected and that the subgraph induced by the set of terminals is complete.

Definition 7. *A* 1-2 graph *is a complete graph with edge costs 1 and 2.*

In these restricted graphs we are able to prove a better approximation ratio of the k-LCA.

Note that in quasi-bipartite graphs we may choose k arbitrarily high and still have polynomial running time. To see this, consider a full component, H with maximum gain-over-loss containing some Steiner point v. Let $e = (z, v)$ be the least-cost edge incident to v in G and let $e' = (v, z') \in H$ be the first edge on the path in $T[H]$ from v to z. If we assume that $e' \neq e$ then removing e' from H and inserting e decreases $loss(H)$ and $cost(T[H])$. But this yields a larger gain over loss for H, a contradiction. It follows that $e \in H$ and hence that the loss is determined by v. Among full components containing v, k-LCA therefore needs to find the component with maximum gain. But this can be done in polynomial time by finding all neighbours of v in $MST(T \cup v)$.

Lemma 6. *For the Steiner Tree Problem in quasi-bipartite graphs and 1-2 graphs,*

$$mst \leq 2(opt_k - loss_k). \tag{2}$$

Proof. First, consider quasi-bipartite graphs. Let K be any full component of a Steiner tree T such that K contains exactly one Steiner point, say s, and p terminals, $z_0, \ldots, z_{p-1}$. Then K is a star with center s having the p edges, (s, z_i), $0 \leq i < p$. Let d_i be the length of edge (s, z_i). We know that $loss(K)$ is the length of a minimal length edge, say (s, z_0) of K. Then $loss(K) = d_0 = \min_{0 \leq i < p}\{d_i\}$. Let $mst(Z_K)$ be the cost of a minimum spanning tree of G_{Z_K} where $Z_K = \{z_0, \ldots, z_{p-1}\}$. The edges (z_0, z_i), $1 \leq i < p$, define a spanning tree, T, of G_{Z_K}. Recall that the cost function is metric. Hence, the length of edge (z_0, z_i) is at most the length of edge (z_0, s) plus the length of edge (s, z_i), i.e. $d_0 + d_i$. Since the length of T is an upper bound on the length of a minimum spanning tree of G_{Z_K}, we have

$$\begin{aligned} mst(Z_K) &\leq \sum_{i=1}^{p-1}(d_0 + d_i) \\ &= p \cdot d_0 + cost(K) - 2 \cdot d_0 \\ &\leq \sum_{i=0}^{p-1} d_i + cost(K) - 2 \cdot loss(K) \\ &= 2 \cdot cost(K) - 2 \cdot loss(K). \end{aligned}$$

If $\mathcal{K}$ is the collection of full components of Opt_k then $\cup_{K \in \mathcal{K}} MST(Z_K)$ spans Z and is connected. Hence,

$$mst \leq \sum_{K \in \mathcal{K}} mst(Z_K) \leq 2(opt_k - loss_k).$$

Now we consider complete graphs with edge weights 1 and 2. Let m be the number of terminals in Opt_k. Any spanning tree of the terminals must contain $m-1$ edges. Since any edge has cost at most 2, we have $mst \leq 2m-2$. Since $\mathcal{C}[Opt_k]$ is a terminal-spanning tree it has cost at least $m-1$. Thus $mst \leq 2(m-1) \leq 2cost(\mathcal{C}[Opt_k]) = 2(opt_k - loss_k)$. □

Theorem 3. *Algorithm k-LCA has an approximation ratio of at most* 1.279 *for quasi-bipartite graphs and an approximation ratio approaching* 1.279 *for complete graphs with edge weights* 1 *and* 2.

Proof. By inserting (2) into the inequality of Theorem 1, we obtain

$$\begin{aligned} Approx &\leq loss_k \cdot \ln\left(1 + \frac{mst - opt_k}{loss_k}\right) + opt_k \\ &\leq loss_k \cdot \ln\left(1 + \frac{2(opt_k - loss_k) - opt_k}{loss_k}\right) + opt_k \\ &= loss_k \cdot \ln\left(\frac{opt_k}{loss_k} - 1\right) + opt_k. \end{aligned} \tag{3}$$

To get rid of $loss_k$ in the upper bound, consider the function $f :]0, opt_k[\to \mathbb{R}$, defined by

$$f(x) = x \ln\left(\frac{opt_k}{x} - 1\right) + opt_k.$$

The derivative of f is

$$f'(x) = \ln\left(\frac{opt_k}{x} - 1\right) + x \cdot \frac{x}{opt_k - x} \cdot \left(-\frac{opt_k}{x^2}\right) = \ln\left(\frac{opt_k}{x} - 1\right) - \frac{opt_k}{opt_k - x}$$

and the second-order derivative of f is

$$f''(x) = \frac{x}{opt_k - x} \cdot \left(-\frac{opt_k}{x^2}\right) - \frac{opt_k}{(opt_k - x)^2} = -\frac{opt_k}{opt_k - x} \cdot \left(\frac{1}{x} + \frac{1}{opt_k - x}\right) < 0.$$

Thus, f' is a decreasing function and so has at most one root. Since

$$\lim_{x \to 0} f'(x) = \infty$$

and

$$\lim_{x \to opt_k} f'(x) = -\infty$$

f' has exactly one root, x_0. Furthermore, $f'(x) > 0$ for $0 < x < x_0$ and $f'(x) < 0$ for $x_0 < x < opt_k$. It follows that x_0 is the single maximum of f and hence by (3) we get $Approx \leq f(loss_k) \leq f(x_0)$. Letting $y = \frac{x}{opt_k - x}$, we have

$$f'(x) = \ln\left(\frac{opt_k}{x} - 1\right) - \frac{opt_k}{opt_k - x} = \ln(y^{-1}) - \frac{x - (x - opt_k)}{opt_k - x} = -\ln y - y - 1$$

Set $r = \frac{x_0}{opt_k - x_0}$. Then r is root of the equation $1 + \ln y + y = 0$. Solving numerically, we get $r \approx 0.279$. Finally, since

$$\begin{aligned} f(x) &= x \ln\left(\frac{opt_k}{x} - 1\right) + opt_k \\ &= \frac{x}{opt_k - x}\left(\frac{opt_k}{opt_k - x}\right)^{-1} opt_k \cdot \ln(y^{-1}) + opt_k \\ &= y(1+y)^{-1} opt_k \cdot \ln(y^{-1}) + opt_k, \end{aligned}$$

and since

$$-\ln r = 1 + r \Leftrightarrow -\frac{1}{1+r}\ln r = 1 \Leftrightarrow -\frac{r}{1+r}\ln r = r$$

we have

$$\begin{aligned} \mathit{Approx} &\leq f(x_0) \\ &= \frac{r}{1+r} \cdot \mathit{opt}_k \cdot \ln(r^{-1}) + \mathit{opt}_k \\ &= \left(-\frac{r}{1+r} \cdot \ln r + 1\right) \cdot \mathit{opt}_k \\ &= (r+1) \cdot \mathit{opt}_k \approx 1.279 \cdot \mathit{opt}_k \end{aligned}$$

Since any full component in a Steiner tree contains at most $|Z|$ terminals, we have $\mathit{opt} = \mathit{opt}_k$ for $k = |Z|$. Thus, for quasi-bipartite graphs, $\mathit{Approx} \leq 1.279 \cdot \mathit{opt}$. For complete graphs with edge weights 1 and 2, opt_k converges to opt, and so the approximation ratio of algorithm k-LCA converges to 1.279 when $k \to \infty$. □

The Iterated 1-Steiner Heuristic A simple heuristic for the Steiner Tree Problem in general graphs is the Iterated 1-Steiner Heuristic, which adds Steiner points to a minimum spanning tree one by one if they decrease the cost and remove them again if they become useless, i.e. their degrees become 1 or 2 in the minimum spanning tree. In quasi-bipartite graphs we are able to give an approximation guarantee of 1.5 for this heuristic.

Theorem 4. *Given an instance of the Steiner Tree Problem in a quasi-bipartite graph G, let H be a Steiner tree in G such that*

(i) any Steiner point has degree at least 3 and

(ii) for any vertex v in G, $\mathit{mst}(H \cup v) \geq \mathit{cost}(H)$.

Then the cost of H is at most 1.5 times the optimal.

Proof. In a quasi-bipartite graph, there can be no edge between any two Steiner points. Hence, any full component in such a graph has at most one Steiner point.

Suppose F is a full component containing a Steiner point s. The loss of F is the cost of a minimum-cost forest containing s such that each connected component contains at least one terminal. Since F is a star with center s, the loss of F equals the cost of the least-cost edge connecting s to a terminal. By condition (i), s has degree at least 3 and so the loss of F is at most one third of the cost of F.

If F is a full component containing no Steiner points then F consists of a single edge connecting two terminals. In this case, the loss of F is 0 which is trivially at most one third of the cost of F. Summing over all full components in H, we get $\mathit{loss}(H) \leq \frac{1}{3}\mathit{cost}(H)$.

Next, we show that $\mathit{gain}_{\mathcal{C}[H]}(K) \leq 0$ for any full component K. Lemma 2 will then give us what we want.

We claim that the inequality $\mathit{mst}(\mathcal{C}[H] \cup K) \geq \mathit{mst}(H \cup K) - \mathit{loss}(H)$ holds. Each edge of $\mathit{MST}(\mathcal{C}[H] \cup K)$ corresponds to a unique edge in $H \cup K$ with the same cost. This defines a subset of edges of $H \cup K$ which, together with $\mathit{Loss}(H)$, span $H \cup K$ and have a total cost of $\mathit{mst}(\mathcal{C}[H] \cup K) + \mathit{loss}(H)$. Since $\mathit{MST}(H \cup K)$ is a minimum tree spanning $H \cup K$, the inequality follows.

Now, condition (ii) implies that

$$mst(\mathcal{C}[H] \cup K) \geq mst(H \cup K) - loss(H) \geq cost(H) - loss(H) = cost(\mathcal{C}[H]).$$

It follows that

$$\begin{aligned} gain_{\mathcal{C}[H]}(K) &\leq cost(\mathcal{C}[H]) - mst(\mathcal{C}[H] \cup K) \\ &\leq 0 \end{aligned}$$

By Lemma 2, $cost(H) - loss(H) \leq opt_k$ for any k, hence $cost(H) - loss(H) \leq opt$. Since $loss(H) \leq \frac{1}{3} cost(H)$, we have

$$cost(H) \leq opt + loss(H) \leq opt + \frac{1}{3} cost(H)$$

which implies $cost(H) \leq \frac{3}{2} opt$. □

6 Lower bound on the approximability of the Steiner tree problem

In the preceding sections, we have presented a polynomial time approximation algorithm for the minimum Steiner tree problem which has a factor of ≈ 1.55. In this section, we prove that we cannot hope to find a polynomial time approximation algorithm with a factor of less than $\frac{136}{135} \approx 1.0074$ unless co-RP = NP (see [9] for a definition of co-RP).

To find this lower bound, the strategy is to show that any approximation algorithm for the minimum Steiner tree problem with a factor of less than $\frac{136}{135}$ gives a randomized polynomial time algorithm which solves a particular NP-hard problem. Consider the maximum satisfiability problem of linear equations modulo 2 with three variables per equation, *MAX-E3-LIN-2*. The theorem below, due to Håstad [4], describes the NP-hard problem that we consider and is defined by an instance of *MAX-E3-LIN-2*.

Theorem 5. *For every $\epsilon > 0$ there is an integer k such that it is NP-hard to tell whether a set of n linear equations modulo 2 with three variables per equation and with $2k$ occurences of each variable has an assignment that satisfies $n(1-\epsilon)$ equations, or has no assignment that satisfies more than $n(\frac{1}{2}+\epsilon)$ equations.*[4]

Given such an instance of *MAX-E3-LIN-2*, we wish to construct an instance of the minimum Steiner tree problem.

First, some simplifying assumptions. We may assume that all equations are of the form $x + y + z = 1$. For if we have an equation of the form $x + y + z = 0$ then we may flip say z, obtaining the equivalent equation $x + y + \overline{z} = 1$. Furthermore, we may assume that each variable appears exactly k times negated and k times unnegated in the set of equations. For consider an equation, say $x + y + z = 1$. Then the equations $x + \overline{y} + \overline{z} = 1$, $\overline{x} + y + \overline{z} = 1$, and $\overline{x} + \overline{y} + z = 1$ are all equivalent to this since we have changed the value of the left hand side by ± 2 or 0 which is equal to 0 (mod 2). Adding these to our instance, x, y, and z each appear two times negated and two times unnegated in the four equations. Doing this for all equations, each variable appears k times negated and k times unnegated for some k.

[4] This theorem builds on the fact that it is NP-hard to approximate *MAX-E3-LIN-2* within a factor $2 - \epsilon$.

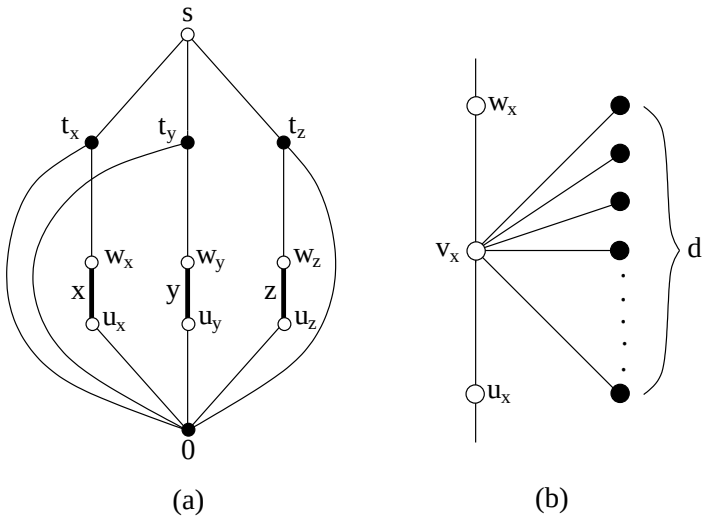

Figure 4: An equation gadget (a) and an edge gadget (b) for literal x. Bold line segments in (a) represent edge gadgets for each of the three literals of the equation.

We are now ready to describe the construction of the corresponding minimum Steiner tree problem instance. Disregarding edge weights for now, the graph consists of two types of *gadgets*: *equation gadgets* and *edge gadgets*. There is exactly one equation gadget for each equation and exactly one edge gadget for each literal, hence n equation gadgets and $3n$ edge gadgets in total. For an equation $x+y+z=1$, the corresponding equation gadget[5] is defined as shown in Figure 4 (a) and for a literal x, the edge gadget of x is defined as shown in Figure 4 (b). Note that edge gadgets are substructures of equation gadgets and that each equation gadget contains three edge gadgets, one for each of the three literals of the equation. The number d of terminals of an edge gadget will be specified later.

Before describing how the gadgets are interconnected, we need the following definition.

Definition 8. *A* d-regular graph *is a graph where each vertex has degree* d.

The gadgets are interconnected as follows. All equation gadgets share the terminal **0**. Now, suppose a bipartite d-regular graph, $H=(A\cup B,E)$, $|A|=|B|=k$, is given and let x be a variable. Given some ordering of the edge gadgets of x and $\overline{x}$, a terminal in the ith edge gadget corresponding to an occurence of x is identified with a terminal in the jth edge gadget corresponding to an occurence of $\overline{x}$ if and only if the ith vertex of A is connected to the jth vertex of B, $1\leq i,j\leq k$, see Figure 5. This is well-defined since a variable and its negation each appear exactly k times. Furthermore, the fact that each vertex of H has degree d implies that, for any edge gadget, each of its d terminals is identified with a terminal in some other edge gadget.

Graph H needs to have a special property which is captured by the following definition.

Definition 9. *Let* G *be a bipartite* d*-regular graph with vertex set* $V_1\cup V_2$, $|V_1|=|V_2|=k$. *Suppose that for every* $S\subset V_1$ *with* $|S|\leq\frac{k}{2}$, *the number of vertices in* V_2 *incident to vertices in* S *is at least* $c|S|$ *for some* $c>1$. *Then* G *is called a* (k,d,c)-expander, *or just an* expander.

[5]This equation gadget is slightly simplified compared to the one in [8], which included three superfluous edges.

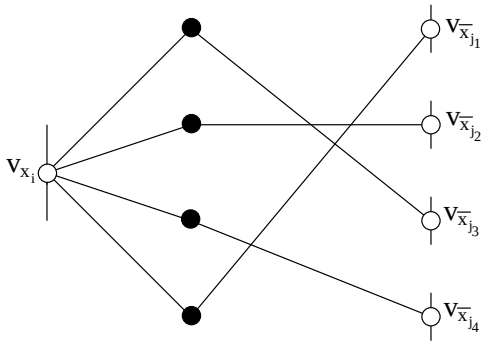

Figure 5: Identification of the d terminals in an edge gadget of x with terminals in d distinct edge gadgets of $\overline{x}$.

We assume that H is a (k, d, c)-expander.

Finally, we define the edge weights of our constructed graph. We do it in such a way that we have the following useful properties:

1. Each truth assignment for the given *MAX-E3-LIN-2* instance yields a solution to the corresponding Steiner tree problem instance whose weight reflects the number of satisfied/not satisfied equations (we specify below how to obtain such a solution).

2. There exists an optimal solution to the Steiner tree problem instance from which it is easy to derive an optimal solution to the corresponding *MAX-E3-LIN-2* instance.

More specifically, we will define the weights such that, for some solution, the subgraphs induced by satisfied equation gadgets all have the same weight, w_s, the subgraphs induced by not satisfied equation gadgets all have the same weight, w_{ns}, and the sum of weights of edge gadgets is independent of the number of satisfied/unsatisfied equations. Since we know that the total number of equations is n, this immediately gives us the first of the above properties and, as we shall see, also the second property.

Assume that we are given constants a and b. We will specify the value of these later. We set the weight of edges in the equation gadgets and edge gadgets as shown in Figure 6.

Suppose we are given a truth assignment to our *MAX-E3-LIN-2* instance. Then in our Steiner tree problem instance, we define the corresponding Steiner tree, which we shall refer to as a *standard tree*, as follows. For all nodes v_x in edge gadgets corresponding to literals with truth value 1, connect v_x to all of its d adjacent terminals. Connect all these v_x nodes to the remaining nodes in the equation gadgets to get a Steiner tree with smallest possible weight. If two edge gadgets share a terminal then one of them corresponds to a literal x and the other to $\overline{x}$. By construction, such a terminal can only have degree one. Thus, all terminals in edge gadgets have degree one. This implies that equation gadgets are only interconnected through terminal $\mathbf{0}$. Hence, the Steiner tree that we have constructed is locally optimal when restricted to an equation gadget. Since each equation gadget is of constant size (disregarding its three edge gadgets) it is easy to see that the subgraphs corresponding to equation gadgets are one of the four possibilites shown in Figure 7, depending on the number of edge gadgets that must be connected to the tree.

The subtrees in Figure 7 (a) and (c) correspond to equations with exactly zero and two variables having truth value 1 respectively. Such equations cannot be satisfied since $2 \equiv 0 \pmod 2$. Similarly, the subtrees in Figure 7 (b) and (d) correspond to satisfied equations.

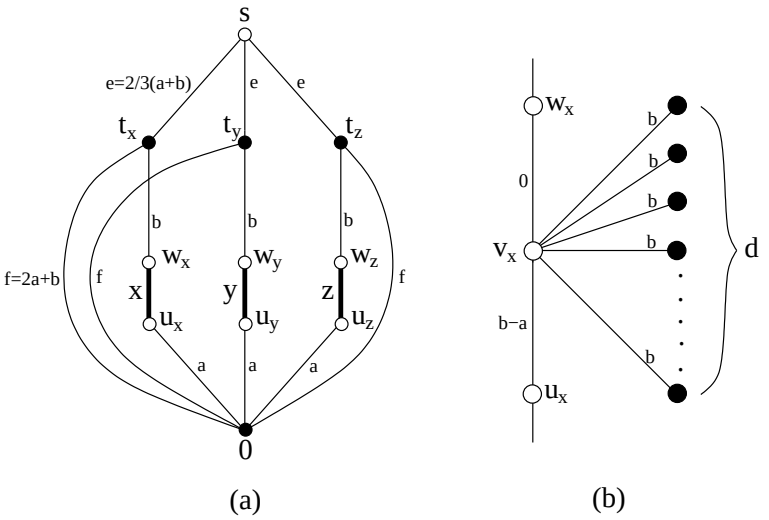

Figure 6: Edge weights of an equation gadget (a) and an edge gadget (b).

In the main theorem below, we will need to add up edge weights of equation gadgets and edge gadgets separately. Note that, in all four cases, if a node v_x is a Steiner point then the path $(v_x, u_x, \mathbf{0})$ is part of the solution. We may therefore assume that the edge gadget containing v_x pays $b-a$ (the weight of edge (v_x, u_x)) of the cost of connecting v_x to the tree, whereas the cost a of edge $(u_x, \mathbf{0})$ is paid for in the equation gadget.

It follows that subtrees corresponding to satisfied equations each have weight $a+b+3e = 3a+3b$ and subtrees corresponding to not satisfied equations each have weight $f+3e = 2a+b+3e = 4a+3b$. An edge gadget corresponding to a literal x with truth value 1 has weight $b-a+db$ since all its d terminals are connected to v_x and the cost of (v_x, u_x) is $b-a$.

The above stated property 1 is then satisfied with $w_s = 3a+3b$ and $w_{ns} = 4a+3b$. The following lemma (which we will not prove) shows that the second property above is satisfied as well with these edge weights.

Lemma 7. *Every optimal Steiner tree can be transformed into a standard tree without increasing the weight.*

Recall that we need an expander in order to construct a Steiner tree instance from a given instance of *MAX-E3-LIN-2*. The following lemma shows that we can do this with a randomized algorithm for sufficiently large k.

Lemma 8. *For sufficiently large k there exist (k, d, c)-expanders for*

$$d > \max\left\{c + \frac{3}{2}, \frac{2}{2-c}, \frac{\frac{c}{2}\ln(\frac{c}{2}) + (1-\frac{c}{2})\ln(1-\frac{c}{2}) - \ln 2}{\frac{c}{2}\ln c - \frac{c-1}{2}\ln(c-1) - \ln 2}\right\}.$$

Furthermore, we can find such an expander with high probability with a randomized polynomial time algorithm.

Now we are ready for the main theorem of this section.

Theorem 6. *No polynomial time approximation algorithm for the minimum Steiner tree problem can have a performance ratio below $\frac{136}{135}$, unless co-RP = NP.*

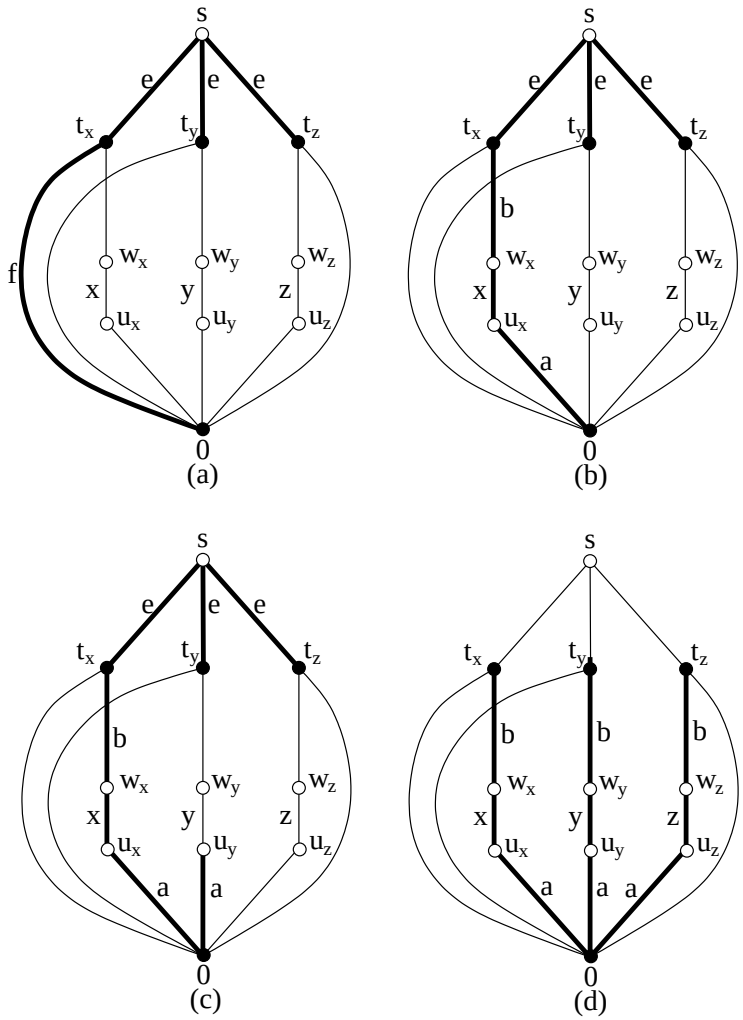

Figure 7: The four types of subgraphs induced by an equation gadget. Bold lines show the chosen edges.

Proof. Suppose there exists a polynomial time approximation algorithm for the minimum Steiner tree problem which has an approximation factor below $\frac{136}{135}$. We will show that an algorithm, which runs in expected polynomial time, exists that solves the NP-hard problem defined in Theorem 5, implying that co-RP = NP.

First, we show how to determine the number of satisfied equations of an instance of *MAX-E3-LIN-2* by considering the weight of the corresponding standard Steiner tree.

The corresponding Steiner tree has $3n$ edge gadgets. Since each variable occurs k times negated and k times unnegated, half, i.e. $\frac{3}{2}n$, of the edge gadgets correspond to literals with truth value 1. Recall that the weight such an edge gadget is $b - a + db$. Since edge gadgets corresponding to false literals have zero weight, the total weight of all edge gadgets is $\frac{3}{2}n(b - a + db)$. Also recall that each equation gadget corresponding to a satisfied equation has weight $3a + 3b$, whereas each equation gadget corresponding to an unsatisfied equation has weight $4a + 3b$. Thus, if M is the number of unsatisfied equations, the standard Steiner tree corresponding to our instance of *MAX-E3-LIN-2* has weight

$$\frac{3}{2}n(b - a + db) + (n - M)(3a + 3b) + M(4a + 3b) = \frac{3}{2}n(b - a + db) + n(3a + 3b) + Ma.$$

We see how the weight of this Steiner tree reveals the number of satisfied and unsatisfied equations.

Let $p = \frac{3}{2}(db + b - a) + (3a + 3b)$ and define

$$r = \frac{p + \frac{1}{2}a}{p}.$$

We claim that a polynomial time approximation algorithm for the minimum Steiner tree problem cannot have an approximation factor below r unless co-RP = NP. Assume otherwise. Let $\mathcal{A}$ be the approximation algorithm with factor below r. The function $x \mapsto (p+\frac{1}{2}a-x)/(p+2x)$, $x \geq 0$, is continuous and tends to $-\infty$ as $x \to \infty$. Hence, the approximation factor of $\mathcal{A}$ can be written as

$$r_\epsilon = \frac{p + \frac{1}{2}a - a\epsilon}{p + 2a\epsilon}$$

for a suitable $\epsilon > 0$. We claim that with this ϵ, we can solve the corresponding NP-hard problem of Theorem 5 with an algorithm running in expected polynomial time. This will imply co-RP = NP.

Given a set of n linear equations modulo 2 with three variables per equation, we construct the corresponding Steiner tree problem instance. This can be done with an algorithm running in expected polynomial time since we may pick k arbitrarily large in Theorem 5 so that Lemma 8 applies. We then find a solution of weight Sol, using algorithm $\mathcal{A}$ on this instance. We consider two cases: $Sol < n(p+\frac{1}{2}a-a\epsilon)$ and $Sol \geq n(p+\frac{1}{2}a-a\epsilon)$. If $Sol < n(p+\frac{1}{2}a-a\epsilon)$ then also $OPT < n(p + \frac{1}{2}a - a\epsilon)$. By Lemma 7, there is a standard tree of weight less than $n(p + \frac{1}{2}a - a\epsilon)$. Thus, there is an assignment for which less than $n(\frac{1}{2} - \epsilon)$ equations are unsatisfied or, equivalently, more than $n(\frac{1}{2} + \epsilon)$ equations are satisfied.

Now, if $Sol \geq n(p + \frac{1}{2}a - a\epsilon)$ then

$$OPT \geq \frac{Sol}{r_\epsilon} \geq \frac{n(p + \frac{1}{2}a - a\epsilon)}{r_\epsilon} = n(p + 2a\epsilon).$$

Therefore, no assignment can have less than $2n\epsilon$ unsatisfied equations or, equivalently, more than $n(1-2\epsilon)$ satisfied equations. Since $n(1-2\epsilon) < n(1-\epsilon)$, no assignment satisfies $n(1-\epsilon)$ equations.

What remains is to show that we can find suitable values of a, b, c, and d so that $r = \frac{136}{135}$. If we let $a = tb$ then

$$r = 1 + \frac{\frac{1}{2}a}{\frac{3}{2}(db + b - a) + (3a + 3b)} = 1 + \frac{tb}{3(db + b - tb) + 6tb + 6b} = 1 + \frac{t}{3(d + 3 + t)}.$$

We may set $c = \frac{b+a}{b-a}$ (this follows from the proof of Lemma 7 which we have chosen not to include) and so

$$c = \frac{b + a}{b - a} = \frac{b + tb}{b - tb} = \frac{1 + t}{1 - t}$$

and

$$t = c(1 - t) - 1 \Rightarrow t(c + 1) = c - 1 \Rightarrow t = \frac{c - 1}{c + 1}.$$

Inserting this, we get

$$r = 1 + \frac{\frac{c-1}{c+1}}{3(d + 3 + \frac{c-1}{c+1})}$$

By Lemma 8, we may set $d = 6$ and $c = \frac{53}{35}$. With these values, $r = \frac{136}{135}$. □

7 Concluding remarks

Our presentation of the k-LCA differs slightly from [7]. We will highlight some of the differences.

Lemma 1 is adopted from [7, Lemma 1] which is given without proof. The two references given for the proof both use different formalisms and it is therefore very hard to distill a proof in the context of [7]. Furthermore [7, Lemma 1] is missing important assumptions, so several counterexamples can be given. The version presented in this paper along with its proof is therefore entirely our contribution.

Lemma 5 is adopted from [7, Lemma 5]. The use of the lemma in [7] requires "partially" performing an iteration, which is a very ill-defined notion, and using [7, Lemma 5] as a blackbox simply is not strong enough. Therefore we have expanded Lemma 5 and its proof to cover the needed "partial" iteration.

Furthermore, all through this paper we have given lots of additional detail in the various proofs.

Modifying the k-LCA When the algorithm selects a component, the contracted edges affect the edge costs of the maintained terminal-spanning tree, but the subsequent components are based on the original edge costs. It would seem that doing an actual contraction and edge cost modification in the graph could improve the algorithm. The argument supporting this is that any subsequently considered components would then have better gain and still represent legal additions to the represented Steiner tree. It seems (but we have not checked the details) that the proof of the approximation ratio of k-LCA still holds for this altered algorithm, and if this indeed is the case then the modified algorithm could have a better approximation factor.

References

[1] P. Berman and V. Ramaiyer, "Improved Approximations for the Steiner Tree Problem", J. of Algorithms, 17 (1994), 381–408.

[2] A. Borchers and D.-Z. Du, "The k-Steiner Ratio in Graphs", SIAM J. Computing 26 (1997), 857–869.

[3] S. Hougardy and H. J. Prömmel, "A 1.598 Approximation Algorithm for the Steiner Tree Problem in Graphs", Proceedings of ACM-SIAM Symposium on Discrete Algorithms (1999), 448–453.

[4] J. Håstad, "Some optimal inapproximability results", Proceedings of the 28th Annual Symposium on Theory of Computing, ACM, 1997.

[5] M. Karpinski and A. Zelikovsky, "New Approximation Algorithms for the Steiner Tree Problem", Journal of Combinatorial Optimization, 1 (1997), 47–65.

[6] H. J. Prömmel and A. Steger, "RNC-Approximation Algorithms for the Steiner Problem", Proceedings 14th Annual Symposium on Theoretical Aspects of Computer Science (1997), 559–570.

[7] G. Robins and A. Zelikovsky, "Improved Steiner Tree Approximation in Graphs", Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (2000), 770–779.

[8] M. Thimm, "On the Approximability of the Steiner Tree Problem", Theor. Comput. Sci. 295, 1–3 (Feb. 2003), 387–402.

[9] V. V. Vazirani, "Approximation Algorithms", Springer-Verlag, 2002.

[10] A. Zelikovsky, "An 11/6-Approximation Algorithm for the Network Steiner Problem", Algorithmica 9 (1993), 463–470.

[11] A. Zelikovsky, "Better Approximation Bounds for the Network and Euclidean Steiner Tree Problems", Technical report CS-96-06, University of Virginia, 1996.