A Local-Search Algorithm for Steiner Forest*†

Martin Groß[1], Anupam Gupta[2], Amit Kumar[3], Jannik Matuschke[4], Daniel R. Schmidt[5], Melanie Schmidt[6], and José Verschae[7]

1 **Institut für Mathematik, Technische Universität Berlin, Germany**
gross@math.tu-berlin.de
2 **Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, USA**
anupamg@cs.cmu.edu
3 **Dept. of Comp.Sci. and Engg., Indian Institute of Technology, Delhi, India**
amitk@cse.iitd.ernet.in
4 **TUM School of Management, Technische Universität München, Germany**
jannik.matuschke@tum.de
5 **Institut für Informatik, Universität zu Köln, Germany**
schmidt@informatik.uni-koeln.de
6 **Institut für Informatik, Universität Bonn, Germany**
melanieschmidt@uni-bonn.de
7 **Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile**
jverschae@uc.cl

Abstract

In the *Steiner Forest* problem, we are given a graph and a collection of source-sink pairs, and the goal is to find a subgraph of minimum total length such that all pairs are connected. The problem is APX-Hard and can be 2-approximated by, e.g., the elegant primal-dual algorithm of Agrawal, Klein, and Ravi from 1995.

We give a local-search-based constant-factor approximation for the problem. Local search brings in new techniques to an area that has for long not seen any improvements and might be a step towards a combinatorial algorithm for the more general survivable network design problem. Moreover, local search was an essential tool to tackle the dynamic MST/Steiner Tree problem, whereas dynamic Steiner Forest is still wide open.

It is easy to see that any constant factor local search algorithm requires steps that add/drop many edges together. We propose natural local moves which, at each step, either (a) add a shortest path in the current graph and then drop a bunch of inessential edges, or (b) add a set of edges to the current solution. This second type of moves is motivated by the potential function we use to measure progress, combining the cost of the solution with a penalty for each connected component. Our carefully-chosen local moves and potential function work in tandem to eliminate bad local minima that arise when using more traditional local moves.

Our analysis first considers the case where the local optimum is a single tree, and shows optimality w.r.t. moves that add a single edge (and drop a set of edges) is enough to bound the locality gap. For the general case, we show how to "project" the optimal solution onto the different trees of the local optimum without incurring too much cost (and this argument uses optimality w.r.t. both kinds of moves), followed by a tree-by-tree argument. We hope both the potential function, and our analysis techniques will be useful to develop and analyze local-search algorithms in other contexts.

* This work was partially supported by the DFG within project A07 of CRC TRR 154, by the German Academic Exchange Service (DAAD), by the Alexander von Humboldt Foundation with funds of the German Federal Ministry of Education and Research (BMBF), by Nucleo Milenio Información y Coordinación en Redes ICM/FIC P10-024F, and by NSF awards CCF-1536002, CCF-1540541, and CCF-1617790.

† A full version of this paper is available at https://arxiv.org/abs/1707.02753

9th Innovations in Theoretical Computer Science Conference (ITCS 2018).
Editor: Anna R. Karlin; Article No. 31; pp. 31:1–31:17

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Local Search, Steiner Forest, Approximation Algorithms, Network Design

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.31

1 Introduction

The Steiner Forest problem is the following basic network design problem: given a graph $G = (V, E)$ with edge-lengths d_e, and a set of source-sink pairs $\{\{s_i, t_i\}\}_{i=1}^{k}$, find a subgraph H of minimum total length such that each $\{s_i, t_i\}$ pair lies in the same connected component of H. This problem generalizes the Steiner Tree problem, and hence is APX-hard. The Steiner Tree problem has a simple 2-approximation, namely the minimum spanning tree on the terminals in the metric completion; however, the forest version does not have such obvious algorithms.

Indeed, the first approximation algorithm for this problem was a sophisticated and elegant primal-dual 2-approximation due to Agrawal, Klein, and Ravi [1]. Subsequently, Goemans and Williamson streamlined and generalized these ideas to many other constrained network design problems [15]. These results prove an integrality gap of 2 for the natural cut-covering LP. Other proofs of this integrality gap were given in [21, 7]. No better LP relaxations are currently known (despite attempts in, e.g., [24, 25]), and improving the approximation guarantee of 2 remains an outstanding open problem. Note that all known constant-factor approximation algorithms for Steiner Forest were based on linear programming relaxations, until a recent greedy algorithm [18]. In this paper, we add to the body of techniques that give constant-factor approximations for Steiner Forest. The main result of this paper is the following:

▶ **Theorem 1.** *There is a (non-oblivious) local search algorithm for Steiner Forest with a constant locality gap. It can be implemented to run in polynomial time.*

The Steiner Forest problem is a basic network problem whose approximability has not seen any improvements in some time. We explore new techniques to attacking the problem, with the hope that these will give us more insights into its structure. Moreover, for many problems solved using the constrained forest approach of [15], the only constant factor approximations known are via the primal-dual/local-ratio approach, and it seems useful to bring in new possible techniques. Another motivation for our work is to make progress towards obtaining combinatorial algorithms for the survivable network design problem. In this problem, we are given connectivity requirements between various source-sink pairs, and we need to find a minimum cost subset of edges which provide this desired connectivity. Although we know a 2-approximation algorithm for the survivable network design problem [21] based on iterative rounding, obtaining a combinatorial constant-factor approximation algorithm for this problem remains a central open problem [34]. So far, all approaches of extending primal-dual or greedy algorithms to survivable network design have only had limited success. Local search algorithms are more versatile in the sense that one can easily *propose* algorithms based on local search for various network design problems. Therefore, it is important to understand the power of such algorithms in such settings. We take a step towards this goal by showing that such ideas can give constant-factor approximation algorithms for the Steiner Forest problem.

Finally, we hope this is a step towards solving the *dynamic Steiner Forest* problem. In this problem, terminal pairs arrive online and we want to maintain a constant-approximate Steiner Forest while changing the solution by only a few edges in each update. Several of the

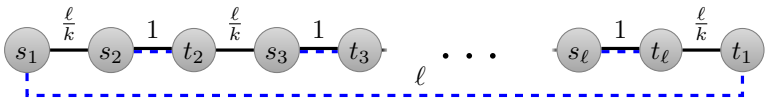


Figure 1 The black edges (continuous lines) are the current solution. If $\ell \gg k$, we should move to the blue forest (dashed lines), but any improving move must change $\Omega(k)$ edges. Details can be found in Section A.1.

approaches used for the Steiner *Tree* case (e.g., in [30, 16, 27]) are based on local-search, and we hope our local-search algorithm for Steiner Forest in the offline setting will help solve the dynamic Steiner Forest problem, too.

1.1 Our Techniques

One of the challenges with giving a local-search algorithm for Steiner Forest is to find the right set of moves. Indeed, it is easy to see that simpler approaches like just adding and dropping a constant number of edges at each step is not enough. E.g., in the example of Figure 1, the improving moves must add an edge and remove multiple edges. (This holds even if we take the metric completion of the graph.) We therefore consider a natural generalization of simple edge swaps in which we allow to add paths and remove multiple edges from the induced cycle.

Local Moves: Our first task is to find the "right" moves that add/remove many edges in each "local" step. At any step of the algorithm, our algorithm has a feasible forest, and performs one of these local moves (which are explained in more detail in Section 3):

- **edge/set swaps**: Add an edge to a tree in the current forest, and remove one or more edges from the cycle created.
- **path/set swaps**: Instead of one edge, add a set of edges to connect two vertices from the same tree T in the current forest, creating exactly one cycle, then remove edges from the cycle. The set of edges shall be a shortest path in the graph where all trees except T are contracted.
- **connecting moves**: Connect some trees of the current forest by adding edges between them.

At the end of the algorithm, we apply the following post-processing step to the local optimum:

- **clean-up**: Delete all inessential edges. (An edge is *inessential* if dropping it does not alter the feasibility of the solution.)

Given these local moves, the challenge is to bound the locality gap of the problem: the ratio between the cost of a local optimum and that of the global optimum.

The Potential. The connecting moves may seem odd, since they only increase the length of the solution. However, a crucial insight behind our algorithm is that we do not perform local search with respect to the total length of the solution. Instead we look to improve a different potential ϕ. (In the terminology of [3, 23], our algorithm is a *non-oblivious* local search.) The potential $\phi(T)$ of a tree T is the total length of its edges, plus the distance between the furthest source-sink pair in it, which we call its *width*. The potential of the forest $\mathcal{A}$ is the sum of the potentials of its trees. We only perform moves that cause the potential of the resulting solution to decrease.

In Section A.2 we give an example where performing the above moves with respect to the total length of the solution gives us local optima with cost $\Omega(\log n) \cdot \mathrm{OPT}$ — this example is useful for intuition for why using this potential helps. Indeed, if we have a forest where the distance between two trees in the forest is much less than both their widths, we can merge them and reduce the potential (even though we increase the total length). So the trees in a local optimum are "well-separated" compared to their widths, an important property for our analysis.

The Proof. We prove the constant locality gap in two conceptual steps.

As the first step, we assume that the local optimum happens to be a single tree. In this case we show that the essential edges of this tree T have cost at most $\mathcal{O}(\mathrm{OPT})$—hence the final removal of inessential edges gives a good solution. To prove this, we need to charge our edges to OPT's edges. However, we cannot hope to charge single edges in our solution to single edges in OPT—we need to charge multiple edges in our solution to edges of OPT. (We may just have more edges than OPT does. More concretely, this happens in the example from Figure 1, when $\ell = \Theta(k)$ and we are at the black tree and OPT is the blue forest.) So we consider edge/set swaps that try to swap some subset S of T's edges for an edge f of OPT. Since such a swap is non-improving at a local optimum, the cost of S is no more than that of f. Hence, we would like to partition T's edges into groups and find an $O(1)$-to-1 map of groups to edges of OPT of no less cost. Even if we cannot find an explicit such map, it turns out that Hall's theorem is the key to showing its existence.

Indeed, the intuition outlined above works out quite nicely if we imagine doing the local search with respect to the total length instead of the potential. The main technical ingredient is a partitioning of our edges into equivalence classes that behave (for our purposes) "like single edges", allowing us to apply a Hall-type argument. This idea is further elaborated in Section 4.1. However, if we go back to considering the potential, an edge/set swap adding f and removing S may create multiple components, and thus increase the width part of the potential. Hence we give a more delicate argument showing that similar charging arguments work out: basically we now have to charge to the width of the globally optimal solution as well. A detailed synopsis is presented in Section 4.2.

The second conceptual step is to extend this argument to the case where we can perform all possible local moves, and the local optimum is a forest $\mathcal{A}$. If OPT's connected components are contained in those of $\mathcal{A}$, we can do the above analysis for each $\mathcal{A}$-component separately. So imagine that OPT has edges that go between vertices in different components of $\mathcal{A}$. We simply give an algorithm that takes OPT and "projects" it down to another solution OPT' of comparable cost, such that the new projected solution OPT' has connected components that are contained in the components of $\mathcal{A}$. We find the existence of a cheap projected solution quite surprising; our proof crucially uses the optimality of the algorithm's solution under both path/set swaps and connecting moves. A summary of our approach is in Section 5.

Polynomial-time Algorithm. The locality gap with respect to the above moves is at most 46. The swap moves can be implemented in polynomial time, and connecting moves can be approximated to within constant factors. Indeed, a c-approximation for *weighted* k-MST gives a $23(1 + c) + \varepsilon$-guarantee for the local search algorithm. Applying a weighted version of Garg's 2-approximation [13, 14] yields $c = 2$. The resulting approximation guarantee is 69 (compared to 96 for [18]).

1.2 Related Work

Local search techniques have been very successful for providing good approximation guarantees for a variety of problems: e.g., network design problems such as low-degree spanning trees [12], min-leaf spanning trees [29, 33], facility location and related problems, both uncapacitated [26, 4] and capacitated [31], geometric k-means [22], mobile facility location [2], and scheduling problems [32]. Other examples can be found in, e.g., the book of Williamson and Shmoys [34]. More recent are applications to optimization problems on planar and low-dimensional instances [10, 6]. In particular, the new PTAS for low dimensional k-means in is based on local search [9, 11].

Local search algorithms have also been very successful in practice – e.g., the widely used Lin-Kernighan heuristic [28] for the travelling salesman problem, which has been experimentally shown to perform extremely well [19].

Imase and Waxman [20] defined the dynamic Steiner tree problem where vertices arrive/depart online, and a few edge changes are performed to maintain a near-optimal solution. Their analysis was improved by [30, 16, 17, 27], but extending it to Steiner Forest remains wide open.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with non-negative edge weights $d_e \in \mathbb{R}_{\geq 0}$. Let $n := |V|$. For $W \subseteq V$, let $G[W] = (W, E[W])$ be the vertex-induced subgraph, and for $F \subseteq E$, $G[F] = (V[F], F)$ the edge-induced subgraph, namely the graph consisting of the edges in F and the vertices contained in them. A forest is a set of edges $F \subseteq E$ such that $G[F]$ is acyclic.

For a node set $W \subseteq V$ and an edge set $F \subseteq E$, let $\delta_F(W)$ denote the edges of F leaving W. Let $\delta_F(A : B) := \delta_F(A) \cap \delta_F(B)$ for two disjoint node sets $A, B \subseteq V$ be the set of edges that go between A and B. For forests $F_1, F_2 \subseteq E$ we use $\delta_F(F_1 : F_2) := \delta_F(V[F_1] : V[F_2])$. We may drop the subscript if it is clear from the context.

Let $\mathfrak{T} \subseteq \{\{v, \bar{v}\} \mid v, \bar{v} \in V\}$ be a set of terminal pairs. Denote the shortest-path distance between u and $\bar{u}$ in (G, d) by $\mathrm{dist}_d(u, \bar{u})$. Let n_t be the number of terminal pairs. We number the pairs according to non-decreasing shortest path distance (ties broken arbitrarily). Thus, $\mathfrak{T} = \{\{u_1, \bar{u}_1\}, \ldots, \{u_{n_t}, \bar{u}_{n_t}\}\}$ and $i < j$ implies $\mathrm{dist}_d(u_i, \bar{u}_i) \leq \mathrm{dist}_d(u_j, \bar{u}_j)$. This numbering ensures consistent tie-breaking throughout the paper. We say that $G = (V, E)$, the weights d and $\mathfrak{T}$ form a *Steiner Forest instance*. We often use $\mathcal{A}$ to denote a feasible Steiner forest held by our algorithm and $\mathcal{F}$ to denote an optimal/good feasible solution to which we compare $\mathcal{A}$.

Width. Given a connected set of edges E', the *width* $w(E')$ of E' is the maximum distance (in the original graph) of any terminal pair connected by E': i.e., $w(E') = \max_{\{u,\bar{u}\} \in \mathfrak{T}, u, \bar{u} \in V[E']} \mathrm{dist}_d(u, \bar{u})$. Notice that $w(E')$ is the width of the pair $\{u_i, \bar{u}_i\}$ with the largest i among all pairs in $V[E']$. We set $\mathtt{index}(E') := \max\{i \mid u_i, \bar{u}_i \in V[E']\}$, i.e., $w(E') = \mathrm{dist}_d(u_{\mathtt{index}(E')}, \bar{u}_{\mathtt{index}(E')})$.

For a subgraph $G[F] = (V[F], F)$ given by $F \subseteq E$ with connected components $E_1, \ldots, E_l \subseteq F$, we define the *total width* of F to be the sum $w(F) := \sum_{i=1}^{l} w(E_i)$ of the widths of its connected components. Let $d(F) := \sum_{e \in F} d_e$ be the sum of edge lengths of edges in F and define $\phi(F) := d(F) + w(F)$. By the definition of the width, it follows that $d(F) \leq \phi(F) \leq 2d(F)$.

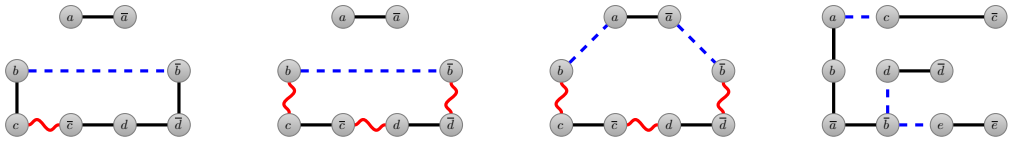


Figure 2 Our different moves. Black solid edges are not changed by the move. Blue dashed edges are added by the move. Red curled edges are removed by the move.

3 The Local Search Algorithm

Our local-search algorithm starts with a feasible solution $\mathcal{A}$, and iteratively tries to improve it. Instead of looking at the actual edge cost $d(\mathcal{A})$, we work with the potential $\phi(\mathcal{A})$ and decrease it over time.

In the rest of the paper, we say a move changing $\mathcal{A}$ into $\mathcal{A}'$ is *improving* if $\phi(\mathcal{A}') < \phi(\mathcal{A})$. A solution $\mathcal{A}$ is `<move>`-*optimal* with respect to a certain kind of move and with respect to a set of edges $\mathcal{F}$ if no move of that kind consisting of edges from $\mathcal{F}$ is improving.

Swaps. Swaps are moves that start with a cycle-free feasible solution $\mathcal{A}$, add some edges and remove others to get to another cycle-free feasible solution $\mathcal{A}'$.

- The most basic swap is: *add an edge e creating a cycle, remove an edge f from this cycle.* This is called an *edge/edge swap* (e, f).
- We can slightly generalize this: *add an edge e creating a cycle, and remove a subset S of edges from this cycle $C(e)$.* This is called the *edge/set swap* (e, S). Edge/edge swaps are a special case of edge/set swaps, so edge/set swap-optimality implies edge/edge swap-optimality.
  There may be many different subsets of $C(e)$ we could remove. A useful fact is that if we fix some edge $f \in C(e)$ to remove, this uniquely gives a maximal set $R(e, f) \subseteq C(e)$ of edges that can be removed along with f after adding e without violating feasibility. Indeed, $R(e, f)$ contains f, and also all edges on $C(e)$ that can be removed in $\mathcal{A} \cup \{e\} \setminus \{f\}$ without destroying feasibility. (See Lemma 13 in the full version for a formalization.)
  Moreover, given a particular $R(e, f)$, we could remove any subset $S \subseteq R(e, f)$. If we were doing local search w.r.t. $d(\mathcal{A})$, there would be no reason to remove a proper subset. But since the local moves try to reduce $\phi(\mathcal{A})$, removing a subset of $R(e, f)$ may be useful. If $e_1, \ldots, e_\ell$ are the edges in $R(e, f)$ in the order they appear on $C(e)$, we only need swaps where S consists of edges $e_i, \ldots, e_j$ that are consecutive in the above order. There are $\mathcal{O}(n^2)$ sets $S \subseteq R(e, f)$ that are consecutive.[1] Moreover, there are at most $n - 1$ choices for e and $O(n)$ choices for f, so the number of edge/set swaps is polynomial.
- A further generalization: we can pick two vertices u, v lying in some component T, add a shortest-path between them (in the current solution, where all other components are shrunk down to single points, and the vertices/edges in $T \setminus \{u, v\}$ are removed). This creates a cycle, and we want to remove some edges. We now imagine that we added a "virtual" edge $\{u, v\}$, and remove a subset of consecutive edges from some

[1] In fact, we only need five different swaps (e, S) for the following choices of consecutive sets S: The set $S = \{f\}$, the complete set $S = R(e, f)$, and three sets of the form $S = \{e_1, \ldots, e_i\}$, $S = \{e_{i+1}, \ldots, e_j\}$ and $S = \{e_{j+1}, \ldots, e_\ell\}$ for specific indices i and j. How to obtain the values for i and j is explained in Section 4.1.

$R(\{u,v\},f) \subseteq C(\{u,v\})$, just as if we'd have executed an edge/set swap with the "virtual" edge $\{u,v\}$. We call such a swap a *path/set swap* (u,v,S).
Some subtleties: Firstly, the current solution $\mathcal{A}$ may already contain an edge $\{u,v\}$, but the uv-shortest-path we find may be shorter because of other components being shrunk. So this move would add this shortest-path and remove the direct edge $\{u,v\}$—indeed, the cycle $C(uv)$ would consist of two parallel edges, and we'd remove the actual edge $\{u,v\}$. Secondly, although the cycle contains edges from many components, only edges within T are removed. Finally, there are a polynomial number of such moves, since there are $O(n^2)$ choices for u,v, $O(n)$ choices for f, and $O(n^2)$ consecutive removal sets S.

Note that edge/set swaps never decrease the number of connected components of $\mathcal{A}$, but path/set swaps may increase or decrease the number of connected components.

Connecting moves. Connecting moves reduce the number of connected components by adding a set of edges that connect some of the current components. Formally, let $G^{\text{all}}_{\mathcal{A}}$ be the (multi)graph that results from contracting all connected components of $\mathcal{A}$ in G, deleting loops and keeping parallel edges. A *connecting move* (denoted $conn(T)$) consists of picking a tree in $G^{\text{all}}_{\mathcal{A}}$, and adding the corresponding edges to $\mathcal{A}$. The number of possible connecting moves can be large, but we can show that an approximation for k-MST is sufficient to obtain an approximate connecting move that works appropriately (see full version).

Note that connecting moves cause $d(\mathcal{A}') > d(\mathcal{A})$, but since our notion of improvement is with respect to the potential ϕ, such a move may still cause the potential to decrease.

In addition to the above moves, the algorithm runs the following post-processing step at the end.

Clean-up. Remove the unique maximal edge set $S \subseteq \mathcal{A}$ such that $\mathcal{A} \setminus S$ is feasible, i.e., erase all unnecessary edges. This might increase $\phi(\mathcal{A})$, but it will never increase $d(\mathcal{A})$.

Checking whether an improving move exists is polynomial except for connecting moves, which we can do approximately. Thus, the local search algorithm can be made to run in polynomial time by using standard methods (see full version).

4 In Which the Local Optimum is a Single Tree

We want to bound the cost of a forest that is locally optimal with respect to the moves defined above. To start, let us consider a simpler case: suppose we were to find a single tree T that is optimal with respect to just the *edge/edge* and *edge/set* swaps. (Recall that edge/set swaps add an edge and remove a consecutive subset of the edges on the resulting cycle, while maintaining feasibility. Also, recall that optimality means that no such moves cause the potential ϕ to decrease.) Our main result of this section is the following:

▶ **Corollary 2.** *Let $G=(V,E)$ be a graph, let d_e be the cost of edge $e \in E$ and let $\mathfrak{T} \subseteq V \times V$ be a set of terminal pairs. Let $\mathcal{A},\mathcal{F} \subseteq E$ be two feasible Steiner forests for $(G,d,\mathfrak{T})$ with $V[\mathcal{A}] = V[\mathcal{F}]$. Assume that $\mathcal{A}$ is a tree and that $\mathcal{A}$ is swap-optimal with respect to $\mathcal{F}$ and ϕ under edge/edge and edge/set swaps. Denote by $\mathcal{A}'$ the modified solution where all inessential edges have been dropped from $\mathcal{A}$. Then,*

$$d(\mathcal{A}') \leq 10.5 \cdot d(\mathcal{F}) + w(\mathcal{F}) \leq 11.5 \cdot d(\mathcal{F}).$$

The actual approximation guarantee is 42 for this case: indeed, Corollary 2 assumes $V[\mathcal{A}] = V[\mathcal{F}]$, which can be achieved (by taking the metric completion on the terminals) at the cost of a factor 2.

The intuition here comes from a proof for the optimality of edge/edge swaps for the Minimum Spanning tree problem. Let $\mathcal{A}$ be the tree produced by the algorithm, and $\mathcal{F}$ the reference (i.e., optimal or near-optimal) solution, with $V[\mathcal{A}] = V[\mathcal{F}]$. Suppose we were looking for a minimum spanning tree instead of a Steiner forest: one way to show that edge/edge swaps lead to a global optimum is to build a bipartite graph whose vertices are the edges of $\mathcal{A}$ and $\mathcal{F}$, and which contains edge (e, f) when $f \in \mathcal{F}$ can be swapped for $e \in \mathcal{A}$ and $d_e \leq d_f$. Using the fact that all edge/edge swaps are non-improving, we can show that there exists a perfect matching between the edges in $\mathcal{A}$ and $\mathcal{F}$, and hence the cost of $\mathcal{A}$ is at most that of $\mathcal{F}$.

Our analysis is similar in spirit. Of course, we now have to (a) consider edge/*set* swaps, (b) do the analysis with respect to the potential ϕ instead of just edge-lengths, and (c) we cannot hope to find a perfect matching because the problem is NP-hard. These issues make the proofs more complicated, but the analogies still show through.

4.1 An approximation guarantee for trees and d

In this section, we conduct a thought-experiment where we imagine that we get a connected tree on the terminals which is optimal for edge/set swaps *with respect to just the edge lengths, not the potential*. In very broad strokes, we define an equivalence relation on the edges of $\mathcal{A}$, and show a constant-to-1 cost-increasing map from the resulting equivalence classes to edges of $\mathcal{F}$—again mirroring the MST analysis—and hence bounding the cost of $\mathcal{A}$ by a constant times the cost of $\mathcal{F}$. The analysis of the real algorithm in Section 4.2 builds on the insights we develop here.

Some Definitions. The crucial equivalence relation is defined as follows: For edges $e, f \in \mathcal{A}$, let $T_{e,f}$ be the connected component of $\mathcal{A} \setminus \{e, f\}$ that contains the unique e-f-path in $\mathcal{A}$. We say e and f are *compatible w.r.t.* $\mathcal{F}$ if $e = f$ or if there are no $\mathcal{F}$-edges leaving $T_{e,f}$, and denote it by $e \sim_{cp} f$. One can show that $\sim_{cp}$ is an equivalence relation (see full version). We denote the set of its equivalence classes by $\mathfrak{S}$.

An edge is *essential* if dropping it makes the solution infeasible. If T_1, T_2 are the connected components of $\mathcal{A} \setminus \{e\}$, then e is called *safe* if at least one edge from $\mathcal{F}$ crosses between T_1 and T_2. Observe that any essential edge is safe, but the converse is not true: safe edges can be essential or inessential. However, it turns out that the set S_u of all unsafe edges in $\mathcal{A}$ forms an equivalence class of $\sim_{cp}$. Hence, all other equivalence classes in $\mathfrak{S}$ contain only safe edges. Moreover, these equivalence classes containing safe edges behave like single edges in the sense of the following lemma. For a proof of the lemma, see Lemma 14 in the full version.

▶ **Lemma 3.** *Let $S \in \mathfrak{S} \setminus \{S_u\}$ be an equivalence class of safe edges. It holds that:*

1. *S lies on a path in $\mathcal{A}$.*
2. *For any edge $f \in \mathcal{F}$, either S is completely contained in the fundamental cycle $C_{\mathcal{A}}(f)$ obtained by adding f to $\mathcal{A}$, or $S \cap C_{\mathcal{A}}(f) = \emptyset$.*
3. *If $(\mathcal{A} \setminus \{e\}) \cup \{f\}$ is feasible, and e belongs to equivalence class S, then $(\mathcal{A} \setminus S) \cup \{f\}$ is feasible. (This last property also trivially holds for $S = S_u$.)*

Charging. We can now give the bipartite-graph-based charging argument sketched above.

▶ **Theorem 4.** *Let $I = (V, E, \mathfrak{T}, d)$ be a Steiner Forest instance and let $\mathcal{F}$ be a feasible solution for I. Furthermore, let $\mathcal{A} \subseteq E$ be a feasible* tree *solution for I. Assume that $V[\mathcal{F}] = V[\mathcal{A}]$. Let $\Delta : \mathfrak{S} \to \mathbb{R}$ be a cost function that assigns a cost to all $S \in \mathfrak{S}$. Suppose that $\Delta(S) \leq d_f$*

for all pairs of $S \in \mathfrak{S} \setminus \{S_u\}$ and $f \in \mathcal{F}$ such that the cycle in $\mathcal{A} \cup \{f\}$ contains S. Then,

$$\sum_{S \in \mathfrak{S} \setminus \{S_u\}} \Delta(S) \leq \frac{7}{2} \cdot \sum_{f \in \mathcal{F}} d_f.$$

Proof. Construct a bipartite graph $H = (A \cup B, E(H))$ with nodes $A := \{a_S \mid S \in \mathfrak{S} \setminus \{S_u\}\}$ and $B := \{b_f \mid f \in \mathcal{F}\}$. Add an edge $\{a_S, b_f\}$ whenever f closes a cycle in $\mathcal{A}$ that contains S. By our assumption, if $\{a_S, b_f\} \in E(H)$ then $\Delta(S) \leq d_f$. Suppose that we can show that $\frac{7}{2} \cdot |N(X)| \geq |X|$ for all $X \subseteq A$, where $N(X) \subseteq B$ is the set of neighbors of nodes in X[2]. By a generalization of Hall's Theorem, this condition implies that there is an assignment $\alpha : E \to \mathbb{R}_+$ such that $\sum_{e \in \delta_H(a)} \alpha(e) \geq 1$ for all $a \in A$ and $\sum_{e \in \delta_H(b)} \alpha(e) \leq \frac{7}{2}$ for all $b \in B$. Hence

$$\begin{aligned}\sum_{S \in \mathfrak{S} \setminus \{S_u\}} \Delta(S) &\leq \sum_{S \in \mathfrak{S} \setminus \{S_u\}} \sum_{e \in \delta_H(a_S)} \alpha(e) \Delta(S) \\ &= \sum_{f \in \mathcal{F}} \sum_{e \in \delta_H(b_f)} \alpha(e) \Delta(S) \leq \sum_{f \in \mathcal{F}} \sum_{e \in \delta_H(b_f)} \alpha(e) d_f \leq \frac{7}{2} \sum_{f \in \mathcal{F}} d_f.\end{aligned}$$

It remains to show that $\frac{7}{2} \cdot |N(X)| \geq |X|$ for all $X \subseteq A$. To that aim, fix $X \subseteq A$ and define $\mathfrak{S}' := \{S \mid a_S \in X\}$. In a first step, contract all $e \in U := \bigcup_{S \in \mathfrak{S} \setminus \mathfrak{S}'} S$ in $\mathcal{A}$, and denote the resulting tree by $\mathcal{A}' := \mathcal{A} / U$.[3] Note that edges in each equivalence class are either all contracted or none are contracted. Also note that all unsafe edges are contracted, as $S_u \notin \mathfrak{S}'$. Apply the same contraction to $\mathcal{F}$ to obtain $\mathcal{F}' := \mathcal{F} / U$, from which we remove all loops and parallel edges. Notice that $\mathcal{A}'$ does not contain loops and parallel edges, since we contracted a subset of $\mathcal{A}$. Furthermore, $\mathcal{A}'$ is a tree, while $\mathcal{F}'$ can contain cycles.

Let $f \in \mathcal{F}'$. Since $\mathcal{A}'$ is a tree, f closes a cycle C in $\mathcal{A}'$ containing at least one edge $e \in \mathcal{A}'$. Denoting the equivalence class of e by S_e, observing that all edges in $\mathcal{A}'$ are safe, and using Lemma 3, statement 2, we get that cycle C contains S_e. Hence the node $b_f \in B$ corresponding to f belongs to $N(a_{S_e}) \subseteq N(X)$. Thus, $|N(X)| \geq |\mathcal{F}'|$ and it remains to show that $\frac{7}{2}|\mathcal{F}'| \geq |X|$.

We want to find a unique representative for each $a_S \in X$. So we select an arbitrary root vertex $r \in V[\mathcal{A}']$ and orient all edges in $\mathcal{A}'$ away from r. Every non-root vertex now has exactly one incoming edge. Every equivalence class $S \in \mathfrak{S}'$ consists only of safe edges, so it lies on a path. Consider the two well-defined *endpoints* which are the outermost vertices of S on this path. For at least one of them, the unique incoming edge must be an edge in S. We represent S by one of the endpoints which has this property and call this representative r_S. Let $R \subseteq V[\mathcal{A}']$ be the set of all representative nodes. Since every vertex has an unique incoming edge, $S \neq S'$ implies that $r_S \neq r_{S'}$. Hence $|R| = |\mathfrak{S}'| = |X|$. Moreover, let R_1 and R_2 be the representatives with degrees 1 and 2 in $\mathcal{A}'$, and L be the set of leaves of $\mathcal{A}'$. As the number of vertices of degree at least 3 in a tree is bounded by the number of its leaves, the number of representatives of degree at least 3 in $\mathcal{A}'$ is bounded by $|L|$. So $|X| \leq |R_1| + |R_2| + |L|$.

We now show that every $v \in R_2 \cup L$ is incident to an edge in $\mathcal{F}'$. First, consider any $v \in L$ and let e be the only edge in $\mathcal{A}'$ incident to v. As e is safe, there must be an edge $f \in \mathcal{F}'$

[2] Notice that $N(X)$ is a set of nodes, in contrast to $\delta(X)$, which is the set of edges leaving X.

[3] Formally, we define the graph $G[T]/e = (V[T]/e, T/e)$ for a tree T by $V[T]/e := V[T] \cup \{uv\} \setminus \{u, v\}$ and $T/e := T \setminus \delta(\{u, v\}) \cup \{\{w, uv\} \mid \{u, w\} \in T \vee \{v, w\} \in T\}$ for an edge $e = \{u, v\} \in E$, then set $G/U := G/e_1/e_2/\ldots/e_k$ for $U = \{e_1, \ldots, e_k\}$ and let T/U be the edge set of this graph. If $U \subseteq T$, then the contraction causes no loops or parallel edges, otherwise, we delete all loops or parallel edges.

incident to v. Now consider any $r_S \in R_2$ and let $e_1, e_2 \in \mathcal{A}'$ be the unique edges incident to r_S. Because r_S is the endpoint of the path corresponding to the equivalence class S, the edges e_1 and e_2 are not compatible. Hence there must be an edge $f \in \mathcal{F}'$ incident to r_S. Because R_2 and L are disjoint and every edge is incident to at most two vertices, we conclude that $|\mathcal{F}'| \geq (|R_2| + |L|)/2$. This implies that $|X| \leq |R_1| + |R_2| + |L| \leq 2L + |R_2| \leq 4|\mathcal{F}|$. We can get a slightly better bound below by showing that $|\mathcal{F}'| \geq \frac{2}{3}|R_1|$.

Let $\mathcal{C}$ be the set of connected components of $\mathcal{F}'$ in G/U. Let $\mathcal{C}' := \{T \in \mathcal{C} \mid |V[T] \cap R_1| \leq 2\}$ and $\mathcal{C}'' := \{T \in \mathcal{C} \mid |V[T] \cap R_1| > 2\}$. Note that no representative $r_S \in R_1$ is a singleton as every leaf of $\mathcal{A}'$ is incident to an edge of $\mathcal{F}'$. We claim that $|T| \geq |V[T] \cap R_1|$ for every $T \in \mathcal{C}'$. Assume by contradiction that this was not true and let $T \in \mathcal{C}'$ with $|T| < |V(T) \cap R_1|$. This means that $V[T] \cap R_1$ contains exactly two representatives $r_S, r_{S'} \in R_1$ and T contains only the edge $\{r_S, r_{S'}\}$. Let $e \in S$ and $e' \in S'$ be the edges of $\mathcal{A}'$ incident to r_S and $r_{S'}$, respectively. As e and e' are not compatible, there must be an edge $f \in \mathcal{F}'$ with exactly one endpoint in $\{r_S, r_{S'}\}$, a contradiction as this edge would be part of the connected component T. We conclude that $|T| \geq |V[T] \cap R_1|$ for every $T \in \mathcal{C}'$. Additionally, we have that $|T| \geq |V[T]| - 1 \geq \frac{2}{3}|V[T]|$ for all $T \in \mathcal{C}''$ as $|V[T]| > 2$. Therefore,

$$|\mathcal{F}'| = \sum_{T \in \mathcal{C}} |T| \geq \sum_{T \in \mathcal{C}'} |V[T] \cap R_1| + \sum_{T \in \mathcal{C}''} \frac{2}{3}|V[T] \cap R_1| \geq \frac{2}{3}|R_1|.$$

The three bounds together imply $|X| \leq |R_1| + |R_2| + |L| \leq \frac{3}{2}|\mathcal{F}'| + 2|\mathcal{F}'| = \frac{7}{2}|\mathcal{F}'|$. ◀

We obtain the following corollary of Theorem 4.

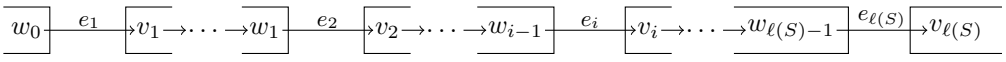
▶ **Corollary 5.** *Let $I = (V, E, \mathfrak{T}, d)$ be a Steiner Forest instance and let OPT be a solution for I that minimizes $d(OPT) = \sum_{e \in OPT} d_e$. Let $\mathcal{A} \subseteq E$ be feasible tree solution for I that does not contain inessential edges. Assume $V[\mathcal{A}] = V[OPT]$. If $\mathcal{A}$ is edge/edge and edge/set swap-optimal with respect to OPT and d, then it holds that $\sum_{e \in \mathcal{A}} d_e \leq (7/2) \cdot \sum_{e \in OPT} d_e$.*

Proof. Since there are no inessential edges, $S_u = \emptyset$. We set $\Delta(S) := \sum_{e \in S} d_e$ for all $S \in \mathfrak{S}$. Let $f \in \mathrm{OPT}$ be an edge that closes a cycle in $\mathcal{A}$ that contains S. Then, $(\mathcal{A} \setminus \{e\}) \cup \{f\}$ is feasible for any single edge $e \in S$ because it is still a tree. Statement 3 of Lemma 3 implies that $(\mathcal{A} \setminus S) \cup \{f\}$ is also feasible. Thus, we consider the swap that adds f and deletes S. It was not improving with respect to d, because $\mathcal{A}$ is edge/set swap-optimal with respect to edges from OPT and d. Thus, $\Delta(S) = \sum_{e \in S} d_e \leq d_f$, and we can apply Theorem 4 to obtain the result. ◀

4.2 An approximation guarantee for trees and ϕ

We now consider the case where a connected tree $\mathcal{A}$ is output by the algorithm when considering the edge/set swaps, but now with respect to the potential ϕ (instead of just the total length as in the previous section). These swaps may increase the number of components, which may have large widths, and hence edge/set swaps that are improving purely from the lengths may not be improving any more. This requires a more nuanced analysis, though relying on ideas we have developed in the previous section.

Here is the high-level summary of this section. Consider some equivalence class $S \in \mathfrak{S} \setminus \{S_u\}$ of safe edges, let $\ell(S)$ be the number of edges in S. The edges lie on a path, hence for an appropriate numbering $e_1, \ldots, e_{\ell(S)}$, the situation looks like this:



Notice that removing the $\ell(S)$ edges forms $\ell(S)+1$ connected components. We let In_S be set of the "inner" components (the ones containing $v_1, \ldots, v_{\ell(S)-1}$), and $\text{In}_{S'}$ be the inner components except the two with the highest widths. Just taking the definition of ϕ, and adding and subtracting the widths of these "not-the-two-largest" inner components, we get

$$\phi(\mathcal{A}) = w(\mathcal{A}) + \sum_{e \in S_u} d_e + \underbrace{\sum_{S \in \mathfrak{S} \setminus \{S_u\}} \Big(\sum_{i=1}^{\ell(S)} d_{e_i} - \sum_{K \in \text{In}_{S'}} w(K) \Big)}_{\leq 10.5 \cdot d(\mathcal{F}) \text{ (first proof step)}} + \underbrace{\sum_{S \in \mathfrak{S} \setminus \{S_u\}} \sum_{K \in \text{In}_{S'}} w(K)}_{\leq w(\mathcal{F}) \text{ (second proof step)}}.$$

As indicated above, the argument has two parts. For the first summation, look at the cycle created by adding edge $f \in \mathcal{F}$ to our solution $\mathcal{A}$. Suppose class S is contained in this cycle. We prove that edge/set swap optimality implies that $\sum_{i=1}^{\ell(S)} d_{e_i} - \sum_{K \in \text{In}_{S'}} w(K)$ is at most $3d_f$. (Think of this bound as being a weak version of the facts in the previous section, which did not have a factor of 3 but did not consider weights in the analysis.) Using this bound in Theorem 4 from the previous section gives us a matching that bounds the first summation by $3 \cdot (7/2) \cdot d(\mathcal{F})$. A couple of words about the proofs: the bound above follows from showing that three different swaps must be non-improving, hence the factor of 3. Basically, we break the above path into three at the positions of the two components of highest width, since for technical reasons we do not want to disconnect these high-width components. Details are in §7.1 in the full version.

For the second summation, we want to sum up the widths of all the "all-but-two-widest" inner components, over all these equivalence classes, and argue this is at most $w(\mathcal{F})$. This is where our notions of safe and compatible edges comes into play. The crucial observations are that (a) given the inner components corresponding to some class S, the edges of some class S' either avoid all these inner components, or lie completely within some inner component; (b) the notion of compatibility ensures that these inner components correspond to distinct components of $\mathcal{F}$, so we can get more width to charge to; and (c) since we don't charge to the two largest widths, we don't double-charge these widths. The details are in §7.3 in the full version.

5 In Which the Local Optimum may be a Forest

The main theorem. In the general case, both $\mathcal{A}$ and $\mathcal{F}$ may have multiple connected components. We assume that the distance function d is a metric. The first thing that comes to mind is to apply the previous analysis to the components individually. Morally, the main obstacle in doing so is in the proof of Theorem 4: There, we assume implicitly that no edge from $\mathcal{F}$ goes between connected components of $\mathcal{A}$.[4] This is vacuously true if $\mathcal{A}$ is a single tree, but may be false if $\mathcal{A}$ is disconnected. In the following, our underlying idea is to replace $\mathcal{F}$-edges that cross between the components of $\mathcal{A}$ by edges that lie within the components of $\mathcal{A}$, thereby re-establishing the preconditions of Theorem 4. We do this in a way that $\mathcal{F}$ stays feasible, and moreover, its cost increases by at most a constant factor. This allows us to prove that the local search has a constant locality gap.

Reducing to local tree optima. Suppose $\mathcal{F}$ has no inessential edges to start. Then we convert $\mathcal{F}$ into a collection of cycles (shortcutting non-terminals), losing a factor of 2 in

[4] More precisely, we need the slightly weaker condition that for each node $t \in V[\mathcal{A}]$, there is an $\mathcal{F}$-edge incident to t that does not leave the connected component of $\mathcal{A}$ containing t.

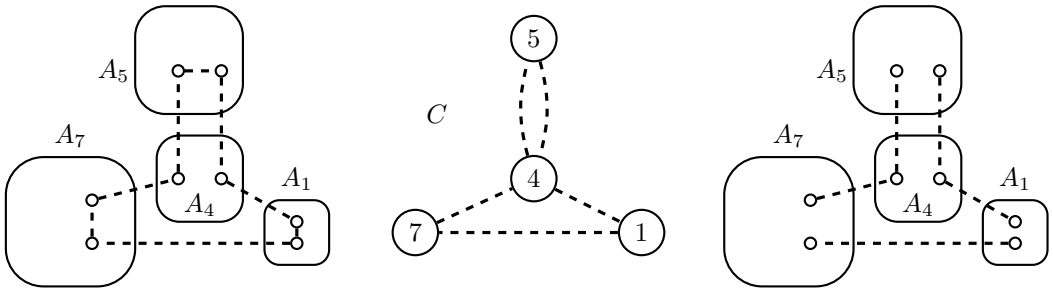


Figure 3 The charging argument with four components A_1, A_4, A_5 and A_7 of $\mathcal{A}$. The area of each component corresponds to its width. *On the left.* A cycle in $\mathcal{F}$. *In the middle.* The corresponding circuit (non-simple cycle) in $G_{\mathcal{A}}$. *On the right.* A suitable decomposition into connecting moves.

the cost. Now observe that each "offending" $\mathcal{F}$-edge (i.e., one that goes between different components of $\mathcal{A}$) must be part of a path P in $\mathcal{F}$ that connects some $s, \bar{s}$, and hence starts and ends in the same component of $\mathcal{A}$. This path P may connect several terminal pairs, and for each such pair $s, \bar{s}$, there is a component of $\mathcal{A}$ that contains s and $\bar{s}$. Thus, P could be replaced by direct connections between $s, \bar{s}$ within the components of $\mathcal{A}$. This would get rid of these "offending" edges, since the new connections would stay within components of $\mathcal{A}$. The worry is, however, that this replacement is too expensive. We show how to use connecting tree moves to bound the cost of the replacement.

Consider one cycle from $\mathcal{F}$, regarded as a circuit C in the graph $G_{\mathcal{A}}$ where the connected components $A_1, \ldots, A_p$ of $\mathcal{A}$ are shrunk to single nodes, i.e., C consists of offending edges. The graph $G_{\mathcal{A}}$ might contain parallel edges and C might have repeated vertices. So C is a circuit, meaning that it is a potentially non-simple cycle, or, in other words, a Eulerian multigraph. The left and middle of Figure 3 are an example.

Index the A_j's such that $w(A_1) \leq \cdots \leq w(A_p)$ and say that node j in $G_{\mathcal{A}}$ corresponds to A_j. Suppose C visits the nodes $v_1, \ldots, v_{|C|}, v_1$ (where several of these nodes may correspond to the same component A_j) and that component A_j is visited n_j times by C. In the worst case, we may need to insert n_j different $s, \bar{s}$ connections into component A_j of $\mathcal{A}$, for all j. The key observation is that the total cost of our direct connections is at most $\sum_{i=1}^{|C|} n_i w(A_i)$. We show how to pay for this using the length of C.[5]

To do so, we use optimality with respect to all moves, in particular connecting moves. The idea is simple: We cut C into a set of trees that each define a valid connecting move. For each tree, the connecting move optimality bounds the widths of some components of $\mathcal{A}$ by the length of the tree. E.g., $w(A_1)+w(A_4)$ is at most the length of the tree connecting A_1, A_4, A_5 in Figure 3. Observe that we did not list $w(A_5)$: Optimality against a connecting move with tree T relates the length of T to the width of all the components that T connects, *except* for the component with maximum width. We say a tree *pays* for A_j if it hits A_j, and also hits another A_j of higher width. So we need three properties: (a) the trees should collectively pack into the edges of the Eulerian multigraph C, (b) each tree hits each component A_j at most once, and (c) the number of trees that pay for A_j is at least n_j.

Assume that we found such a tree packing. For circuit C, if $A_{j^\star}$ is the component with greatest width hit by C, then using connecting move optimality for all the trees shows that

$$\sum_{j:A_j \text{ hit by } C, j \neq j^\star} n_j \, w(A_j) \leq d(C).$$

[5] We also need to take care of the additional width of the modified solution, but this is the easier part.

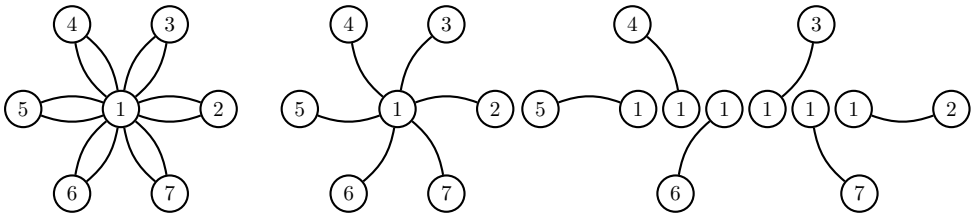


Figure 4 A flower graph. Even though the graph is a non-simple cycle, we can easily decompose it into trees that pay for each $j \neq 6$ at least n_j times (1 is payed for $7 = n_1 + 1$ times).

In fact, even if we have c-approximate connection-move optimality, the right-hand side just gets multiplied by c. But what about $n_{j^\star} w(A_{j^\star})$? We can cut C into sub-circuits, such that each subcircuit C' hits $A_{j^\star}$ exactly once. To get this one extra copy of $w(A_{j^\star})$, we use path/set swap optimality which tells us that the missing connection cannot be more expensive than the length of C. Thus, collecting all our bounds (see Lemma 36 in the full version), adding all the extra connections to $\mathcal{F}$ increases the cost to at most $2(1+c)d(\mathcal{F})$: the factor 2 to make $\mathcal{F}$ Eulerian, $(1+c)$ to add the direct connections, using c-approximate optimality with respect to connecting moves and optimality with respect to path/set swaps. §B.2 in the full version discusses that $c \leq 2$.

Now each component A_j of $\mathcal{A}$ can be treated separately, i.e., we can use Corollary 2 on each A_j and the portion of $\mathcal{F}$ that falls into A_j. By combining the conclusions for all connected components, we get that

$$d(\mathcal{A}') \overset{\text{Cor. 2}}{\leq} 10.5 d(\mathcal{F}') + w(\mathcal{F}') \leq 11.5 d(\mathcal{F}') \leq 23(1+c) \cdot d(\mathcal{F}) \leq 69 \cdot d(\mathcal{F})$$

for any feasible solution $\mathcal{F}$. This proves Theorem 1.

Obtaining a decomposition into connecting moves. It remains to show how to take C and construct the set of trees. If C had no repeated vertices (it is a simple cycle) then taking a spanning subtree would suffice. And even if C has some repeated vertices, decomposing it into suitable trees can be easy: E.g., if C is the "flower" graph on n vertices, with vertex 1 having two edges to each vertex $2, \ldots, n$. Even though 1 appears multiple times, we find a good decomposition (see Figure 4). Observe, however, that breaking C into simple cycles and than doing something on each simple cycle would not work, since it would only pay 1 multiple times and none of the others.

The flower graph has a property that is a generalization of a simple cycle: We say that C is minimally guarded if (a) the largest vertex is visited only once (b) between two occurrences of the same (non-maximal) vertex, there is at least one larger number. The flower graph and the circuit at the top of Figure 5 have this property. Indeed, every minimally guarded circuit can be decomposed suitably. The full algorithm is provided as Algorithm 1 in the full version. It iteratively finds trees that pay for all (non maximal) j with $j \leq z$ for increasing z. Figure 5 shows how the set of trees $\mathfrak{M}_5$ is converted into $\mathfrak{M}_6$ in order to pay for all occurrences of 6. Intuitively, we look where 6 falls into the trees in $\mathfrak{M}_5$. Up to one occurrence can be included in a tree. If there are more occurrences, the tree has to be split into multiple trees appropriately. §8.1.2 in the full version contains the details of the algorithm and its correctness.

Finally, we go from minimally guarded circuits to arbitrary C by extracting subcircuits in a recursive fashion (see Lemma 35 in the full version).

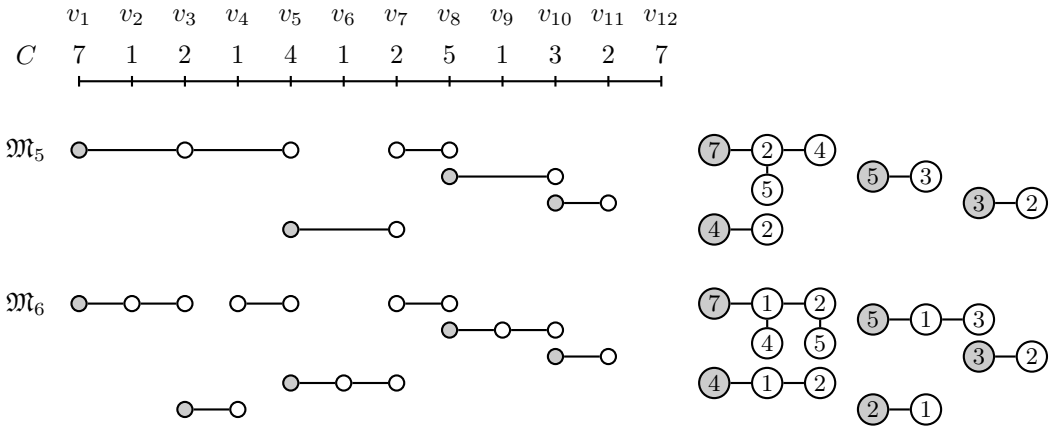


Figure 5 Two iterations of an example run of the Algorithm 1 in the full version.

References

1 Ajit Agrawal, Philip N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995. doi:10.1137/S0097539792236237.

2 Sara Ahmadian, Zachary Friggstad, and Chaitanya Swamy. Local-search Based Approximation Algorithms for Mobile Facility Location Problems. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1607–1621. SIAM, 2013. URL: http://dl.acm.org/citation.cfm?id=2627817.2627932.

3 Paola Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. In Maurizio A. Bonuccelli, Pierluigi Crescenzi, and Rossella Petreschi, editors, *Algorithms and Complexity, Second Italian Conference, CIAC '94, Rome, Italy, February 23-25, 1994, Proceedings*, volume 778 of *Lecture Notes in Computer Science*, pages 40–53. Springer, 1994. doi:10.1007/3-540-57811-0_5.

4 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004. doi:10.1137/S0097539702416402.

5 Norman Biggs. Constructions for cubic graphs with large girth. *The Electronic Journal of Combinatorics*, 5(1):A1:1–A1:25, 1998.

6 Sergio Cabello and David Gajser. Simple ptas's for families of graphs excluding a minor. *Discrete Applied Mathematics*, 189:41–48, 2015. doi:10.1016/j.dam.2015.03.004.

7 Chandra Chekuri and F. Bruce Shepherd. Approximate integer decompositions for undirected network design problems. *SIAM J. Discrete Math.*, 23(1):163–177, 2008. doi:10.1137/040617339.

8 Ho-Lin Chen, Tim Roughgarden, and Gregory Valiant. Designing network protocols for good equilibria. *SIAM J. Comput.*, 39(5):1799–1832, 2010. doi:10.1137/08072721X.

9 Vincent Cohen-Addad, Philip N. Klein, and Claire Mathieu. Local search yields approximation schemes for k-means and k-median in euclidean and minor-free metrics. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science*, 2016. to appear.

10 Vincent Cohen-Addad and Claire Mathieu. Effectiveness of local search for geometric optimization. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 329–343. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.SOCG.2015.329.

11 Zachary Friggstad, Mohsen Rezapour, and Mohammad R. Salavatipour. Local search yields a PTAS for k-means in doubling metrics. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science*, volume abs/1603.08976, 2016. to appear.

12 Martin Fürer and Balaji Raghavachari. Approximating the minimum-degree steiner tree to within one of optimal. *J. Algorithms*, 17(3):409–423, 1994. doi:10.1006/jagm.1994.1042.

13 Naveen Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 396–402. ACM, 2005. doi:10.1145/1060590.1060650.

14 Naveen Garg, 2016. Personal Communication.

15 Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24(2):296–317, 1995. doi:10.1137/S0097539793242618.

16 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 525–534. ACM, 2013. doi:10.1145/2488608.2488674.

17 Anupam Gupta and Amit Kumar. Online steiner tree with deletions. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 455–467. SIAM, 2014. doi:10.1137/1.9781611973402.34.

18 Anupam Gupta and Amit Kumar. Greedy Algorithms for Steiner Forest. In Ronitt Rubinfeld and Rocco Servedio, editors, *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 871–878. ACM, 2015.

19 Keld Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi:10.1016/S0377-2217(99)00284-2.

20 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

21 Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

22 Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. *Comput. Geom.*, 28(2-3):89–112, 2004. doi:10.1016/j.comgeo.2004.03.003.

23 Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comput.*, 28(1):164–191, 1998. doi:10.1137/S0097539795286612.

24 Jochen Könemann, Stefano Leonardi, and Guido Schäfer. A Group-Strategyproof Mechanism for Steiner Forests. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 612–619. Society for Industrial and Applied Mathematics, 2005. doi:10.1.1.126.4369.

25 Jochen Könemann, Stefano Leonardi, Guido Schäfer, and Stefan H. M. van Zwam. A group-strategyproof cost sharing mechanism for the steiner forest game. *SIAM J. Comput.*, 37(5):1319–1341, 2008. doi:10.1137/050646408.

26 Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *J. Algorithms*, 37(1):146–188, 2000. doi:10.1006/jagm.2000.1100.

27 Jakub Lacki, Jakub Ocwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual*

ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015, pages 11–20. ACM, 2015. doi:10.1145/2746539.2746615.

28 S. Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. doi:10.1287/opre.21.2.498.

29 Hsueh-I Lu and R. Ravi. The Power of Local Optimization: Approximation Algorithms for Maximum-leaf Spanning Tree. In *In Proceedings, Thirtieth Annual Allerton Conference on Communication, Control and Computing*, pages 533–542, 1996.

30 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 689–700. Springer, 2012. doi:10.1007/978-3-642-31594-7_58.

31 Martin Pál, Éva Tardos, and Tom Wexler. Facility location with nonuniform hard capacities. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 329–338. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959907.

32 Lukás Polácek and Ola Svensson. Quasi-polynomial local search for restricted max-min fair allocation. *ACM Trans. Algorithms*, 12(2):13:1–13:13, 2016. doi:10.1145/2818695.

33 Roberto Solis-Oba. 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina, and Geppino Pucci, editors, *Algorithms - ESA '98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, volume 1461 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 1998. doi:10.1007/3-540-68530-8_37.

34 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

A Notes on simpler local search algorithms

A.1 Adding an edge and removing a constant number of edges

Let ℓ and $k < \ell$ be integers and consider Figure 6. Notice that adding a single edge and removing k edges does not improve the solution. However, the current solution costs more than ℓ^2/k and the optimal solution costs less than 2ℓ, which is a factor of $\ell/(2k)$ better.

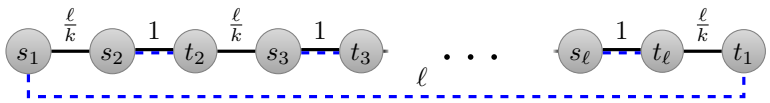


Figure 6 A bad example for edge/set swaps that remove a constant number of edges.

A.2 Regular graphs with high girth and low degree

Assume that G is a degree-3 graph with girth $g = c \log n$ like the graph used in Chen et al. [8]. Such graphs can be constructed, see [5]. Select a spanning tree $\mathcal{F}$ in G which will be the optimal solution. Let E' be the non-tree edges, notice that $|E'| \geq n/2$, and let M be a maximum matching in E'. Because of the degrees, we know that $|M| \geq n/10$. The endpoints of the edges in M form the terminal pairs $\mathfrak{T}$. Set the length of all edges in $\mathcal{F}$ to 1 and the

length of the remaining edges to $g/4$. The solution $\mathcal{F}$ is feasible and costs $n-1$. The solution M costs $\Omega(\log n)$.

Assume we want to remove an edge $e = \{v, w\} \in M$ and our swap even allows us to add a path to reconnect v and w (in the graph where $M \backslash \{e\}$ is contracted). Let P be such a path. Since M is a matching, at most every alternating edge on P is in M. Thus, we have to add $|P|/2 - 1 \geq g/2 - 1$ edges of length one at a total cost that is larger than the cost $g/4$ of e. Thus, no d-improving swap of this type exists (note that, in particular, path/set swaps are not d-improving for M). As a consequence, any oblivious local search with constant locality gap needs to sport a move that removes edges from multiple components of the current solution. In order to restrict to local moves that only remove edges from a single component, we therefore introduced the potential ϕ.