LA-UR 96-2719

CONF-970142--1

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: **ON OPTIMAL STRATEGIES FOR UPGRADING NETWORKS**

AUTHOR(S): S. O. Krumke, H. Noltemeier, M. V. Marathe, S. S. Ravi, R. Ravi, R. Sundaram

SUBMITTED TO: Symposium on Discrete Algorithms(SODA)
January, 1997
New Orleans, LA

MASTER

Los Alamos

Los Alamos National Laboratory
Los Alamos New Mexico 87545

DISCLAIMER

On Optimal Strategies for Upgrading Networks
(Extended Abstract)

S. O. Krumke [1] H. Noltemeier [1] M. V. Marathe [2] S. S. Ravi [3] R. Ravi [4] R. Sundaram [5]

July 2, 1996

Abstract

We study *budget constrained optimal network upgrading problems*. Such problems aim at finding optimal strategies for improving a network under some cost measure subject to certain budget constraints. Given a edge weighted graph $G(V, E)$, in the *edge based upgrading model*, it is assumed that each edge e of the given network has an associated function $c(e)$ that specifies the cost of upgrading the edge by a given amount. A reduction strategy specifies for each edge e the amount by which the length $\ell(e)$ is to be reduced. In the *node based upgrading model* a node v can be upgraded at an expense of cost(v). Such an upgrade reduces the cost of each edge incident on v by a fixed factor ϱ, where $0 < \varrho < 1$. For a given budget B, the goal is to find an improvement strategy such that the total cost of reduction is at most the given budget B and the cost of a subgraph (e.g. minimum spanning tree) under the modified edge lengths is the best over all possible strategies which obey the budget constraint. Define an (α, β)-approximation algorithm as a polynomial-time algorithm that produces a solution within α times the optimal function value, violating the budget constraint by a factor of at most β.

The results obtained in this abstract include the following.

1. We show that in general the problem of computing optimal reduction strategy for modifying the network as above is NP-hard.
2. In the node based model, we show how to devise a near optimal strategy for improving the bottleneck spanning tree. The algorithms has a performance guarantee of $(2 \ln n, 1)$.
3. For the edge based improvement problems we present improved (in terms of performance and time) approximation algorithms.
4. We also present pseudo-polynomial time algorithms (extendable to polynomial time approximation schemes) for a number of edge/node based improvement problems when restricted to the class of treewidth-bounded graphs.

[1]Department of Computer Science, University of Würzburg, Am Hubland, 97074 Würzburg, Germany. Email: {krumke,noltemei}@informatik.uni-wuerzburg.de.

[2]Los Alamos National Laboratory, P.O. Box 1663, MS K990, Los Alamos, NM 87545, USA. Email: madhav@c3.lanl.gov. The work is supported by the Department of Energy under Contract W-7405-ENG-36.

[3]Department of Computer Science, University at Albany – SUNY, Albany, NY 12222, USA. Email: ravi@cs.albany.edu.

[4]GSIA, Carnegie Mellon University, Pittsburgh, PA 15213. Email: ravi+@cmu.edu.

[5]Delta Trading Co. Work done while at MIT, Cambridge MA 02139. Email: koods@theory.lcs.mit.edu. Research supported by DARPA contract N0014-92-J-1799 and NSF CCR 92-12184.

1 Introduction

Several problems arising in areas such as communication networks and VLSI design can be expressed in the following general form: Enhance the performance of an underlying network by carrying out upgrades at certain nodes and/or edges of the network [32, 31, 25].

Consider the following scenario which best illustrates the type of problems we investigate. A large communication company is approached by a client with the requirement to interconnect a set of cities housing the client's offices (e.g. banks with high transaction rates between branches). The company has a list of feasible links that it can use to construct a network to connect these cities. Each link has a construction cost associated with it. One of the main concerns of the client is to build a communication network of minimum cost. This is the ubiquitous minimum spanning tree problem. With the advent of optical communication technology, the client would like to upgrade the communication network and has allocated a fixed budget to do so. In communication networks, upgrading a node corresponds to installing faster communication equipment at that node. Such an upgrade reduces the communication delay along each edge emanating from the node. Similarly upgrading an edge can be achieved by replacing the old line with a new optical cable. In general, there is a cost for improving each link (node) in the existing network by a unit amount. The goal is to design a strategy to upgrade the links of the network so that the total cost of upgrading the links (nodes) is no more than the fixed budget, and the cost of a minimum spanning tree for the upgraded network is the least over all the possible improvements of the network satisfying the budget constraint.

Although substantial work has been done in findng optimal networks (e.g. spanning trees) in graphs, there has been little work on how to *modify* a graph so as to optimize the cost of the network in the resulting graph. In this paper, we formulate and study such network upgrade problems and call them *budget constrained optimal network upgrading problems.*

The paper is organized as follows. Section 2 introduces the node and edge based upgrading models. In Section 3 we formally define the problems under study. Section 4 briefly summarizes our results. In Section 5 we briefly justify our claims that our formulation is indeed general and robust. In Section 6 we present our approximation algorithm for the bottleneck node upgrading problem on general graphs and establish its performance guarantee. In Section 7 give pseudo-polynomial time algorithms for node upgrading problems. It is shown in Appendix B how these algorithms can be converted into fully polynomial approximation schemes. In Section 8 we treat the edge upgrading problem under study. Appendix C contains the hardness results.

2 Node versus Edge Based Models for Network Upgrade

Throughout the presentation we assume that $G = (V, E)$ is a connected undirected graph. Let d be a nonnegative edge-weight function defined on G. For a spanning tree $T = (V, E_T)$ of G, we the *bottleneck-delay* of T under d is defined to be the weight of the heaviest edge in T. The *total weight* of T under the cost function d is the sum of the weights $d(e)$ of the edges $e \in T$. Finally, the *diameter* of T (with respect to d) is the length of a longest simple path in T. We now describe our node based and the edge based upgrade model.

In the *node based upgrading model* we are given the following situation: With each edge $e \in E$ from the graph G, the nonnegative number $\ell(e)$ represents the *length* or *delay* of the link e. When a node v is upgraded, the delay of each edge incident on v decreases by a fixed factor ϱ, where $0 < \varrho < 1$. Thus, if $e = (v, u)$ is an edge, its delay after upgrading exactly one of v and u is $\varrho\ell(e)$; the delay of e falls to $\varrho^2\ell(e)$, if both v and u are upgraded. The cost of upgrading a node v is denoted by cost(v). For a subset V' of V, the cost of upgrading all the nodes in V', denoted by cost(V'), is equal to $\sum_{v \in V'}$ cost(v).

In the *edge based upgrading model*, with each edge $e \in E$, there are associated three nonnegative values as follows: $\ell(e)$ denotes the *length* of the edge e and $\ell_{\min}(e)$ denotes the *minimum length* to which the edge e can be reduced. Consequently, we assume throughout the presentation that $\ell_{\min}(e) \leq \ell(e)$. The nonnegative value $c(e)$ indicates how expensive it is to reduce the length of e by a certain amount: shortening e by t units will involve a cost of $tc(e)$.

Given a budget B, we define a *feasible reduction* to be a nonnegative function r defined on E with the following properties: For all edges $e \in E$, $\ell(e) - r(e) \geq \ell_{\min}(e)$ and $\sum_{e \in E} c(e) \cdot r(e) \leq B$. If r is a (feasible) reduction, in G we can consider the graph G with edge weights given by the "reduced lengths", namely $(\ell - r)(e) := \ell(e) - r(e)$ $(e \in E)$. We denote the total weight of a minimum total length spanning tree with respect to the weight function ℓ by $\mathrm{MST}_G(\ell)$. Similarly, if r is a reduction in G then $\mathrm{MST}_G(\ell - r)$ denotes the weight of a MST with respect to the reduced lengths $\ell(e) - r(e)$ $(e \in E)$.

3 Problem Formulations and Notion of Approximation

We are now ready to define the problems studied in this paper. Our formulation of these problems is based on the work of [29]. A generic node/edge based network upgrade problem $(f_1, f_2, \mathcal{S})$, is defined by identifying two minimization objectives , - f_1 and f_2, - from a set of possible objectives, and specifying a membership requirement in a class of subgraphs, - $\mathcal{S}$. The problem specifies a budget value on the first objective, f_1, under one cost function, and seeks to find a network having minimum possible value for the second objective, f_2, under another cost function, such that this network is within the budget on the first objective. The solution network must belong to the subgraph-class $\mathcal{S}$.

For example, the node based upgrading problems studied here can be formulated as follows. Given a node and edge weighted graph G as above, a speedup factor $0 < \varrho < 1$ and a bound B, (Node-cost, Bottleneck, Spanning Tree) is to upgrade a set $V' \subseteq V$ of nodes of cost cost(V') at most B such that the bottleneck delay of a bottleneck spanning tree in the resulting graph is minimized. The problems (Node-cost, Total-weight, Spanning Tree) and (Node-cost, Diameter, Spanning Tree) are defined similarly. Similarly, the edge based upgrading problem can be stated as: The (Edge-cost, Total-weight, Spanning Tree)is to find an (edge-) reduction r of cost at most B such that $\mathrm{MST}_G(\ell - r)$ has the least possible value.

We argue in Section 5 that this approach for modeling network upgrade problems is both general as well as robust. Next, we now discuss what we mean by finding approximation algorithms for such upgrade problems.

Definition 3.1 We say that a polynomial-time algorithm is an (α, β)-*approximation algorithm* for one of the problems $(f_1, f_2, \mathcal{S})$ defined above, if for each instance of the problem, it produces a solution in which the first objective (f_1) value, is at most α times the budget, and the second objective (f_2) value, is at most β times the minimum for any solution that is within the budget on f_1. The solution produced must belong to the subgraph-class $\mathcal{S}$.

For example, an (α, β)-approximation algorithm for (Edge-cost, Total-weight, Spanning Tree) finds a reduction r of cost at most β times the budget B such that $\frac{\mathrm{MST}_G(\ell - r)}{\mathrm{MST}_G(\ell - r^*)} \leq \alpha$, where r^* denotes an optimal edge-reduction on G of cost at most B.

4 Summary of Results

For the first time, we study the complexity and approximability of a number of node weighted and edge weighted upgrade network improvement problems. We consider three objectives to evaluate the cost of the spanning tree in the modified network: the bottleneck delay, the diameter and the total cost. We show that the problems are hard even for very restricted classes of graphs. The hardness results contrast with the results in [25] about the complexity of *edge based upgrading problems*. For instance, while (Edge-cost, Total-weight, Spanning Tree) is polynomial time solvable on trees [25], we show that (Node-cost, Total-weight, Spanning Tree) and (Node-cost, Diameter, Spanning Tree) are NP-hard *even* on a chain. Given the hardness of finding optimal solutions, we focus on devising approximation algorithms with good performance guarantees.

1. We believe that the discussion in the previous section provides a sound formulation for studying network improvement problems. Following [29], we can show that the formalism is both robust and general. It is more general because it subsumes the case where one wishes to minimize some

Cost Measures	Bottleneck	Diameter	Total Cost
Node-Cost	polynomial-time	(weakly NP-hard) $(1, 1+\varepsilon)$	(weakly NP-hard) $(1, 1+\varepsilon)$

Table 1: Results for node based spanning tree upgrade problems restricted to treewidth-bounded graphs. The row is indexed by the budgeted objective. As mentioned all the results directly extend to finding Steiner trees instead of spanning trees. Similar results also hold for edge based problems.

functional combination of the two criteria. It is more robust because the quality of approximation is independent of which of the two criteria we impose the budget on. Section 5 provides justification for these claims.

2. For the edge based improvement problems we present improved (in terms of performance and time) approximation algorithms. The algorithms are based on an elegant technique introduced by Megiddo [30] and can be extended to obtain approximation algorithms for more general network design problems such as those considered in [15, 16]. This includes problems such as generalized Steiner trees, k-connected subgraphs, etc.

3. We show that the bottleneck upgrading problem (Bottleneck, Node-cost, Spanning Tree) is NP-hard for any fixed $0 < \varrho < 1$ even when there are unit costs on the nodes, i.e. $\text{cost}(v) = 1$ for all $v \in V$ and even for bipartite graphs.

   We provide a polynomial time approximation algorithm for (Bottleneck, Node-cost, Spanning Tree) with a performance guarantee of $(2 \ln n, 1)$. We counterbalance this approximation result with the following lower bound result: Unless NP $\subseteq$ DTIME$(n^{\log \log n})$, there can be no polynomial time approximation algorithm for (Node-cost, Bottleneck, Spanning Tree) with a performance guarantee of (α, β) for any $\alpha < \ln n$ and $\beta < 1/\varrho$, where $0 < \varrho < 1$ is the speedup factor given in the instance.

   Our results constitute the first approximation algorithms for node weighted network improvement problems in the literature. The technique for establishing the logarithmic performance is of independent interest and might be useful in obtaining bounds for other node based improvement problems.

4. For the class of treewidth-bounded graphs we give algorithms with improved time bounds and performance guarantees for each of the three performance measures of a tree mentioned above. This is done in two steps. First we develop pseudopolynomial-time algorithms based on dynamic programming. We then present a general method for deriving fully polynomial-time approximation schemes (FPAS) from the pseudopolynomial-time algorithms. The results for treewidth-bounded graphs are summarized in Table 1.

   The FPAS for a number of node/edge based network improvement problems restricted treewidth-bounded graphs are based on fairly general techniques. Our research in this direction is motivated by the fact that communication networks encountered in practice usually have a small treewidth (e.g. rings, trees, near-trees, series parallel graphs, outerplanar graphs, etc).

4.1 Related Work

To the best of our knowledge, the problems considered in this paper have not been previously studied. The node upgrading model used in this paper was introduced in a recent paper by Paik and Sahni [31], although they considered different problems than the ones considered here. Frederickson and Solis-Oba [12] considered the problem of increasing the weight of the minimum spanning tree in a graph subject to a budget constraint where the cost functions are assumed to be linear in the weight increase. In contrast to the work presented here, they showed that the problem is solvable in strongly polynomial time. Berman [3]

considers the problem of upgrading edges in a given tree to minimize its shortest path tree weight and shows that the problem can be solved in polynomial time by a greedy algorithm. Phillips [32] studies the problem of finding an optimal strategy for reducing the capacity of the network so that the residual capacity in the modified network is minimized. Reference [25] considers network improvement problems under a different model where there are cost functions associated with improving edge weights.

Finally, some important questions remain unsolved. these include approximation algorithms for the node based minimum total cost spanning trees, minimum diameter spanning trees, etc.

5 Formulation: General and Robust

In Section 3, we claimed that our formulation for bicriteria problems is robust and general. In this section, we justify these claims.

We claimed that our formulation is robust because the quality of approximation is independent of which of the two criteria we impose the budget on. To see this note that there are two natural ways to formulate a bicriteria problem: (i) $(f_1, f_2, \mathcal{S})$- find a subgraph in $\mathcal{S}$ whose f_1-objective value is at most B and which has minimum f_2-objective value, (ii) $(f_2, f_1, \mathcal{S})$- find a subgraph in $\mathcal{S}$ whose f_2-objective value is at most B and which has minimum f_1-objective value. Using ideas similar to the ones in [29], we can show that

Theorem 5.1 Any (α, β)-approximation algorithm for $(f_1, f_2, \mathcal{S})$ can be transformed in polynomial time into a (β, α)-approximation algorithm for $(f_2, f_1, \mathcal{S})$.

Thus our approximation results for $(f_1, f_2, \mathcal{S})$ problems in the following sections will also yield approximation algorithms for the symmetric problem $(f_2, f_1, \mathcal{S})$.

For justifying our claims of generality, let f_1 and f_2 be two objective functions and let us say that we wish to minimize the *sum* of the two objectives f_1 and f_2. Call this an $(f_1+f_2, \mathcal{S})$ problem. Let BiAlg(G, B) be any (α, β)-approximation algorithm for $(f_1, f_2, \mathcal{S})$ on graph G with budget B specified for the objective f_1. Using a binary search on the range of values of f_1 with an application of the given approximation algorithm, BiAlg, at each step of this search we obtain the following theorem.

Theorem 5.2 Let BiAlg(G, B) be any (α, β)-approximation algorithm for $(f_1, f_2, \mathcal{S})$ on graph G with budget B under A. Then, there is a polynomial time $\max\{\alpha, \beta\}$-approximation algorithm for the $(f_1 + f_2, \mathcal{S})$ problem.

A similar argument shows that an (α, β)-approximation algorithm BiAlg(G, B), for a $(f_1, f_2, \mathcal{S})$ problem can be used to find devise a polynomial time $\alpha \cdot \beta$ approximation algorithm for the $(f_1 \cdot f_2, \mathcal{S})$ problem. A similar argument can also be given for other basic functional combinations.

The above discussion points out that a good solution to the $(f_1, f_2, \mathcal{S})$-network upgrade problem yields a "good" solution to any unicriterion version (the converse is not necessarily true). It is in this sense that we say our formulation of network upgrade network design problems is general and subsumes other functional combinations.

6 Approximation Algorithm for (Bottleneck, Node-cost, Spanning Tree)

In this section, we present our approximation algorithm for (Bottleneck, Node-cost, Spanning Tree). Recall that in the (Bottleneck, Node-cost, Spanning Tree) problem we are given a bound δ on the bottleneck-delay of a tree and the goal is to upgrade a set V' of the vertices of minimum cost such that the upgraded graph contains a bottleneck spanning tree of delay at most δ.

6.1 Overview

We can assume without loss of generality that all the delays on the edges of the given network are taken from the three element set $\{\delta/\varrho^2, \delta/\varrho, \delta\}$. If the delay of an edge is greater than δ/ϱ^2, then vertex upgrading cannot reduce its delay value to δ. Thus, in the sequel we will assume that the delay of each edge is one of the three above values.

1 Heuristic-(Bottleneck, Node-cost, Spanning Tree)

2 Let $G' :=$ bottleneck(G, ℓ, δ) and let $C_1, \ldots, C_t$ be the connected components of G'.

3 Initialize S to empty and F to the set of edges in G'.

4 Repeat while we have more than one component

5 Let $\mathcal{C} = \{C_1 \ldots, C_q\}$ be the set of clusters, where $q = |\mathcal{C}|$ is number of remaining components.

6 Find a node $v \in V$ in the graph G minimizing the ratio

$$\min_{2 \le r' \le q} \min_{\{C_1, \ldots, C_{r'}\} \subseteq \mathcal{C}} \frac{\text{cost}(v) + \sum_{j=1}^{r'} c(v, C_j)}{r'}.$$

Here, the cost $c(v, C_j)$ is defined in the following way: If $v \in C_j$ or v is adjacent to a node in C_j via an edge of delay δ/ϱ, then $c(v, C_j) := 0$. If all the edges from v to C_j are of delay δ/ϱ^2, then $c(v, C_j)$ is defined to be the minimum cost of a vertex in C_j adjacent to v. If there is no edge between v and any node in C_j, then $c(v, C_j) := +\infty$.

7 Let v be the node and $C_1, \ldots, C_r$ be the components in $\mathcal{C}$ chosen in Step 6 above, where w.l.o.g. $v \in C_1$. Let $f(v) = \text{cost}(v) + \sum_{j=1}^{r} c(v, C_j)$.

8 Let $e_2, \ldots, e_r$ be a set of edges in G connecting v to $C_2, \ldots, C_r$ respectively.

9 Add the edges $e_2, \ldots, e_r$ to F so as to merge $C_1, C_2, \ldots, C_r$ into one component. Add v and the other endpoint of each edge from $\{e_2, \ldots, e_r\}$ whose delay is δ/ϱ^2 to S.

Note that the total cost of the nodes added to the solution S is exactly $f(v)$.

10 Output S as the solution.

Figure 1: The approximation algorithm for (Bottleneck, Node-cost, Spanning Tree).

We first give a brief overview of our algorithm. The algorithm maintains a set S of nodes, a set F of edges and a set $\mathcal{C}$ of clusters which partition the vertex set V of the given graph G. The set $\mathcal{C}$ of clusters is initialized to be the set of connected components of the *bottleneck graph* bottleneck(G, ℓ, δ), which is defined to be the edge-subgraph of G containing only those edges e which have a delay $\ell(e)$ of at most δ. The set S of upgrading nodes is initially empty.

The algorithm iteratively merges clusters until only one cluster remains. To this end, in each iteration it determines a node v of minimum *quotient cost*.

The algorithm Heuristic-(Bottleneck, Node-cost, Spanning Tree) is shown in Figure 1. Step 6 can be implemented in polynomial time by using ideas similar to those in [21]. We omit the details due to lack of space.

It is easy to see that the set S output by algorithm Heuristic-(Bottleneck, Node-cost, Spanning Tree) is indeed a valid upgrading set, since all the edges added to F in Step 9 will be of delay at most δ after upgrading the nodes in S. In the sequel, we use V^* to denote an optimal upgrading set; i.e. an upgrading set of minimal cost OPT $:=$ cost(V^*). We now proceed to prove the performance guarantee provided by the algorithm. Our proof (of Theorem 6.4) relies on several lemmas, which are presented below. We estimate the cost of the nodes added by the heuristic in each iteration by first establishing an averaging lemma and then using a potential function argument. The notion of a *claw decomposition* which is introduced below will be a crucial tool in the analysis.

Definition 6.1 A *claw* is either a single node or a $K_{1,r}$ graph for some $r \ge 1$. If there are at most two nodes in the claw then we can choose any of the nodes as its *center*. Otherwise, the node with degree greater than 1 is the unique *center*. The vertices in the claw different from the center are said to be the *fingers* of the claw. A claw with at least two nodes is called a *non-trivial* claw.

Let G be a graph with node set V. A *claw decomposition* of V in G is a collection of node-disjoint nontrivial claws, which are all subgraphs of G and whose vertices form a partition of V.

The following theorem can be proven by induction on the number $|V|$ of nodes.

Theorem 6.2 Let G be a connected graph with node set V, where $|V| \geq 2$. Then there is a claw decomposition of V in G.

6.2 An Averaging Lemma

Lemma 6.3 Let v be a node chosen in Step 6 and let C denote the total cost of the nodes added to the solution set S in this iteration. Let there be q clusters before v is chosen and assume that in this iteration r clusters are merged. Then: $C/r \leq \mathrm{OPT}/q$.

Proof: Let T^* be an optimal tree with the nodes V^* be the upgraded nodes. Let $\mathcal{C} = C_1, \ldots, C_q$ be the clusters when the node v was chosen and let $T^*(v)$ be the graph obtained from T^* by contracting each C_j to a supernode. $T^*(v)$ is connected and contains all supernodes. We then remove edges (if necessary) from $T^*(v)$ so as to make it a spanning tree. Note that all the edges in this tree have original delay *at least* δ/ϱ.

Let $H \subseteq V^*$ be the set of nodes in the optimal solution that are adjacent to another cluster in $T^*(v)$. Clearly, the cost of these nodes is no more than OPT. Take a claw decomposition of $T^*(v)$. We now obtain a set of claws in the graph G itself in the following way: Initialize E' to be the empty set. For each claw in the decomposition with center C'_1 and fingers $C'_2, \ldots, C'_l$ we do the following: For each edge (C'_1, C'_j) the optimal tree T^* must have contained an edge (u, w) with $u \in C'_1$ and $w \in C'_i$. Notice that since this edge was of original delay at least δ/ϱ, at least one of the vertices u and w must belong to $H \subseteq V^*$. We add (u, w) to E'.

It is easy to see that the subgraph of G induced by the edges in E' consists of disjoint nontrivial claws. Also, all edges in the claws were of original delay at least δ/ϱ and the total number of nodes in the claws is at least q. We need one more useful observation: If a claw center is not contained in H, then *all* the fingers of the claw must be contained in H, since the edges in the claw were of original delay at least δ/ϱ.

Let H_c be the set of nodes from H acting as centers in the just generated claws. Let $H_f^{\delta/\varrho}$ denote the fingers of the claws contained in H which are connected to their claw center via an edge of delay δ/ϱ, whereas H_f^{δ/ϱ^2} stands for the set of fingers adjacent to the center via an edge of delay δ/ϱ^2 and also contained in H. For each claw with exactly two nodes we designate an arbitrary one of the nodes to be the center. Then by construction, H_c, $H_f^{\delta/\varrho}$ and H_f^{δ/ϱ^2} are disjoint. Therefore,

$$\mathrm{OPT} \geq \sum_{u \in H_c \cup H_f^{\delta/\varrho^2}} \mathrm{cost}(u) + \sum_{u \in H_f^{\delta/\varrho}} \mathrm{cost}(u). \tag{1}$$

For a node $u \in H_c$, let N_u denote the number of vertices in the claw centered at u. We have seen that if a center is not in H, then all the fingers belong to the optimal solution. Thus, we can estimate the total number of nodes in the claws from above by summing up the cardinalities of the claws with centers in H and for all other claws adding twice the number of fingers. Hence

$$\sum_{u \in H_c} N_u + 2|H_f^{\delta/\varrho}| \;\geq\; |\{w : w \text{ belongs to some claw}\}| \;\geq\; q, \tag{2}$$

since the total number of nodes in the claws is at least q.

We now estimate the first sum in (1). If $u \in H_c$, then the quotient cost of u is *at most* the cost of u plus the cost of the fingers in the claw that are in H_f^{δ/ϱ^2} divided by the total number of nodes in the claw. This in turn is *at least* C/r by the choice of the algorithm in Step 6. By summing up over all those centers, this leads to

$$\sum_{u \in H_c \cup H_f^{\delta/\varrho^2}} \mathrm{cost}(u) \;\geq\; \frac{C}{r} \sum_{u \in H_c} N_u. \tag{3}$$

Now, for a node u in $H_f^{\delta/\varrho}$, its quotient cost is at most $\text{cost}(u)/2$, which again is at least C/r. Thus

$$\sum_{u \in H_f^{\delta/\varrho}} \text{cost}(u) \geq \sum_{u \in H_f^{\delta/\varrho}} 2\frac{C}{r} = 2|H_f^{\delta/\varrho}| \cdot \frac{C}{r}. \tag{4}$$

Using (3) and (4) in (1) yields

$$\text{OPT} \geq \sum_{u \in H_c \cup H_f^{\delta/\varrho^2}} \text{cost}(u) + \sum_{u \in H_f^{\delta/\varrho}} \text{cost}(u) \geq \frac{C}{r}\left(\sum_{u \in H_c} N_u + 2|H_f^{\delta/\varrho}|\right) \overset{(2)}{\geq} \frac{C}{r} \cdot q. \tag{5}$$

This proves the claim. □

Theorem 6.4 Algorithm Heuristic-(Bottleneck, Node-cost, Spanning Tree) is a polynomial time $(2\ln n, 1)$-approximation algorithm for (Bottleneck, Node-cost, Spanning Tree).

Proof: Let OPT denote the cost of an optimal upgrading set. Assume that the algorithm uses f iterations of the loop and denote by $v_1, \ldots v_f$ the vertices chosen in Step 6 of the algorithm.

Let ϕ_j denote the number of clusters *after* choosing vertex v_j in this iteration. Thus, for instance, $\phi_0 = q$, the number of components at the beginning of this iteration in (S, F). Let the number of clusters merged using vertex v_j be r_j and the total cost of the vertices added in that iteration be c_j. Then we have $\phi_j = \phi_{j-1} - (r_j - 1)$. Since $r_j \geq 2$, we have $\phi_j \leq \phi_{j-1} - \frac{1}{2}r_j$. By Lemma 6.3: $r_j \geq \frac{c_j \phi_{j-1}}{\text{OPT}}$ for all $0 \leq j \leq f$. Thus, we obtain the recurrence

$$\phi_j \leq \phi_{j-1} - \frac{1}{2} \cdot \frac{c_j \phi_{j-1}}{\text{OPT}} = \phi_{j-1} \cdot \left(1 - \frac{c_j}{2 \cdot \text{OPT}}\right). \tag{6}$$

Observe that $\phi_j \geq 2$ for $j = 0, \ldots, f-1$, since the algorithm does not stop before the f-th iteration. Notice also that $\phi_f \geq 1$. We now use an analysis technique due to Leighton and Rao [27] to complete the proof as in [21]. Using the recurrence (6), we obtain

$$\phi_f \leq \phi_0 \prod_{j=1}^{f} \left(1 - \frac{c_j}{2 \cdot \text{OPT}}\right). \tag{7}$$

Taking natural logarithms on both sides and simplifying using the estimate $\ln(1+x) \leq x$, we get

$$2 \cdot \text{OPT} \cdot \ln(\frac{\phi_0}{\phi_f}) \geq \sum_{j=1}^{f} c_j. \tag{8}$$

Notice that by Lemma 6.3 we have $c_j \leq \text{OPT} \cdot \frac{r_j}{\phi_{j-1}} \leq \text{OPT} < 2 \cdot \text{OPT}$, and so the logarithms of all the terms in the product of (7) are well defined. Note also that $\phi_0 \leq n$ and $\phi_f = 1$ and hence from (8) we get $\sum_{j=1}^{f} c_j \leq 2 \cdot \text{OPT} \cdot \ln(n)$. Notice that the total cost of the nodes chosen by the algorithm is exactly the sum $\sum_{j=1}^{f} c_j$. This completes the proof. □

7 Treewidth-Bounded Graphs

In this section we will provide improved algorithms for the node upgrading problems under study if restricted to the class of treewidth-bounded graphs. Treewidth-bounded graphs were introduced by Robertson and Seymour (see [34, 2] and the references therein). Independently, Bern, Lawler and Wong [4] introduced the notion of decomposable graphs. Later, it was shown [2] that the class of decomposable graphs and the class of treewidth-bounded graphs coincide. A class of treewidth-bounded graphs can be specified using a

finite number of primitive graphs and a finite collection of binary composition rules. We use this characterization for proving our results. A class of treewidth-bounded graphs Γ is defined in [4]. For completeness the definition is also given in the appendix. Let Γ be any class of decomposable graphs. Let the maximum number of terminals associated with any graph G in Γ be k. Following [4], it is assumed that a given graph G is accompanied by a parse tree specifying how G is constructed using the rules and that the size of the parse tree is linear in the number of nodes. Moreover, we may assume without loss of generality that the parse tree is a binary tree. The first main result of this section is the following theorem which states the existence of pseudopolynomial-time algorithms for the node weighted network improvement problems given in the Table 1, when restricted to the class of treewidth-bounded graphs. Note that the theorem is symmetric in that we could interchange the budget and objective values.

Theorem 7.1 Every problem in Table 1 can be solved exactly in $\mathcal{O}((n \cdot B)^{\mathcal{O}(1)})$-time for any class of treewidth bounded graphs with no more than k terminals, for fixed k and a budget B on the first objective.

Proof: Bottleneck Problem: Suppose that the maximum cost of nodes that can be upgraded is B. Let π be a partition of the terminals of G. We keep the following information along with each partition π of terminals of G and each $0 \leq i \leq B$:

$Cost_i^\pi$:= *Minimum bottleneck cost of a tree for each block of π, such that the terminal nodes occurring in each tree are exactly the members of the corresponding block of π, no pair of trees is connected, every vertex in G appears in exactly one tree, and the cost of nodes updated in the tree is exactly i.*

For the above defined cost, if there is no forest satisfying the required conditions the value of $Cost$ is defined to be $+\infty$.

Note that the number of cost values associated with any graph in Γ is $\mathcal{O}(k^k B)$. We now show how the cost values can be computed in a bottom-up manner given the parse tree for G. To begin with, since Γ is fixed, the number of primitive graphs is finite. For a primitive graph, each cost value can be computed in constant time, since the number of forests to be examined is fixed. Now consider computing the cost values for a graph G constructed from subgraphs G_1 and G_2, where the cost values for G_1 and G_2 have already been computed. Notice that any forest realizing a particular cost value for G decomposes into two forests, one for G_1 and one for G_2 with some cost values. Since we have maintained the best cost values for all possibilities for G_1 and G_2, we can reconstruct for each partition of the terminals of G the forest that has minimum cost value among all the forests for this partition obeying the diameter constraints. We can do this in time independent of the sizes of G_1 and G_2 because they interact only at the terminals to form G, and we have maintained all relevant information.

Hence we can generate all possible cost values for G by considering combinations of all relevant pairs of cost values for G_1 and G_2. This takes time $\mathcal{O}(k^4)$ per combination for a total time of $\mathcal{O}(k^k B^2)$. As in [4], we assume that the size of the given parse tree for G is $\mathcal{O}(n)$. Thus the dynamic programming algorithm takes time $\mathcal{O}(k^k n B^2)$. This completes the proof of the bottleneck problem.

The case of total cost spanning tree is similar to the bottleneck spanning tree and is omitted. The only difference is that we need to keep track of the total cost of the spanning tree instead of bottleneck cost. In case of the diameter problem, we need to keep more information about each subtrees. Specifically, we need to keep information about the distance of each node from every other node in the tree in a particular partition as well certain other distances. We omit the discussion due to lack of space.

The pseudopolynomial-time algorithms described in the previous section can be used to design fully polynomial-time approximation schemes (FPAS) for these problems for the class of treewidth-bounded graphs. We describe this in the Appendix.

8 Fast Approximation Algorithms for Edge-Improvement Problems

In this section we are going to present a fast approximation algorithm for the (Edge-cost, Total-weight, Spanning Tree) problem. This algorithms improves on the results in [25] in terms of performance and

running time. Recall that in the (Edge-cost, Total-weight, Spanning Tree) problem, the task is to find an edge-improvement strategy r of cost at most B such that $\mathrm{MST}_G(\ell - r)$ is as small as possible. In [25] (Edge-cost, Total-weight, Spanning Tree) has been shown NP-hard.

8.1 The Basic Ideas for an Improved Improvement Algorithm

Let $\gamma > 0$ be an accuracy parameter. Define an interval by $\mathcal{I} := [\frac{(n-1)}{\gamma} \min_{e \in E} \ell_{\min}(e), \frac{(n-1)}{\gamma} \max_{e \in E} \ell(e)]$. Note that if $\mathrm{MST}_G(\ell - r^*)$ denotes the total weight of a minimum spanning tree after an optimal reduction r^* then $\frac{1}{\gamma}\mathrm{MST}_G(\ell - r^*) \in \mathcal{I}$. For each $K \in \mathcal{I}$ we define *compound weights* h_K for the edges of G in the following way:

$$h_K(e) := \min_{t \in [0, \ell(e) - \ell_{\min}(e)]} \left(\ell(e) - t + \frac{K}{B} c(e) t\right) = \begin{cases} \ell(e) & \text{if } K \geq B/c(e), \\ \ell_{\min}(e) + K\frac{(\ell(e) - \ell_{\min}(e))c(e)}{B} & \text{if } K < B/c(e). \end{cases} \quad (9)$$

Thus, for each edge e, the compound weight $h_K(e)$ viewed as a function of K is a linear function with exactly one breakpoint at $B/c(e)$. For $K \leq B/c(e)$, the function has the constant value $\ell(e)$, while for $K \geq B/c(e)$ it has slope $\frac{(\ell(e) - \ell_{\min}(e))c(e)}{B}$. If we plot the compound weight $h_K(e)$ for each edge $e \in E$, for increasing K we get a linear function with exactly one breakpoint at $B/c(e)$. It is easy to see that, given two edges e and e', their *ordering* with respect to the compound weights h_K changes at most twice when K varies. Also, these at most two values of K, can be computed in constant time.
The proofs of the following two lemmas can be found in the appendix.

Lemma 8.1 If $\mathrm{MST}_G(h_{K'}) \leq (1+\gamma)K'$ for some $K' > 0$, then $\mathrm{MST}_G(h_K) \leq (1+\gamma)K$ for all $K \geq K'$. Let K^* be the minimum value $K \in \mathcal{I}$ such that $\mathrm{MST}_G(h_K) \leq (1+\gamma)K$. Then $K^* \leq \mathrm{OPT}/\gamma$.

Lemma 8.2 If the ordering of the edges with respect to their h_{K^*}-weights is known, we can construct a tree T and a reduction r in time $\mathcal{O}(n + m \log \beta(m, n))$ with the following properties:

(i) The cost $\sum_{e \in E} c(e)r(e)$ of the reduction r is at most $(1+\gamma)B$.

(ii) The weight $(\ell - r)(T)$ in the modified graph is no more than $(1 + 1/\gamma)\mathrm{OPT}$.

Lemma 8.2 suggests finding an ordering of the edges in the graph according to their compound weight at K^*. Basically we wish to sort the set $\{h_{K^*}(e_1), \ldots, h_{K^*}(e_m)\}$ where K^* is not known. However, for any K we can decide whether $K^* \leq K$ or $K^* > K$ by one MST computation: We compute an MST with respect to edge weights given by h_K and compare its weight to $(1+\gamma)K$. If the weight is bounded from above by $(1+\gamma)K$, then we know that $K \leq K^*$. Otherwise, we can conclude that $K^* > K$.

Using the idea from above in conjunction with a standard sequential sorting algorithm, we could find the ordering of the edges at K^* by $\mathcal{O}(m \log m)$ minimum spanning tree computations. However, using the elegant technique of Megiddo [30], we can speed up the algorithm substantially.

8.2 Speeding Up the Algorithm

The crucial trick is to use an adaption of a sequentialized parallel sorting algorithm such as Cole's scheme [7]. Recall that a comparison essentially consists of a MST computation, so comparisons are expensive. Using the parallel sorting scheme, we basically accept a greater total number of comparisons, but we can use the parallelism to group the independent comparisons made in one stage of the parallel machine and then answer all of them together efficiently.

Cole's algorithm uses m processors to sort an array of m elements in parallel time $\mathcal{O}(\log m)$. The algorithm is simulated serially, employing one "processor" at a time, according to some fixed permutation, letting each perform one step in each cycle. When two values $h_{K^*}(e)$ and $h_{K^*}(e')$ have to be compared, we compute the critical values where the ordering changes. The crucial observation is that the intersection

points can be computed independently, meaning that each of the "processors" does not need any knowledge about the critical points computed by the other ones.

After the first of the $\mathcal{O}(\log m)$ stages, we are given at most $2m$ critical values of K, say $K_1 \leq K_2 \leq \cdots \leq K_r$ with $r \leq 2P$. For convenience set $K_0 := -\infty$ and $K_{r+1} := +\infty$. Using binary search, we find an interval $[K_i, K_{i+1}]$, where K^* must be contained.

This is done in the following way: Start with low $:= -\infty$ and high $:= +\infty$. Then compute the median $M := K_{\lfloor (r+1)/2 \rfloor}$ of the K_j in $\mathcal{O}(r)$ time. We then decide whether $K^* \leq M$ by computing a MST T with edge weights given by h_M. If $h_M(T) \leq (1+\gamma)M$, then we know that $K^* \leq M$. Otherwise, $K^* > M$. In the first case, we set high $:= M$ and remove all values K_j with $K_j > M$ from our set of critical values. Similarly, in the second case we set low $:= M$ and remove the values smaller than the median M. Clearly, this can be done in $\mathcal{O}(r)$ time. Since M was the median of the K_j the number of critical values decreases by a factor of one half.

Then, the total time effort Time(r) for the binary search satisfies the recurrence:

$$\text{Time}(r) = \text{Time}(r/2) + \text{T}_{\text{MST}} + \mathcal{O}(r),$$

where T_{MST} is the time needed for one MST computation. The solution of the recurrence is Time$(r) = \mathcal{O}(r + \text{T}_{\text{MST}} \log r)$. Since $r \in \mathcal{O}(m)$, this shows that we obtain the interval $[K_i, K_{i+1}]$ containing K^* by $\mathcal{O}(\log m)$ MST computations plus an overhead of $\mathcal{O}(m)$ elementary operations.

Notice that by construction the interval $[K_i, K_{i+1}]$ does not contain any critical points in the interior. If $K_i = K_{i+1}$, then we know that $K^* = K_i = K_{i+1}$. This way we have determined K^*. In this case we can compute the order of all edges with respect to h_{K^*} in $\mathcal{O}(m \log m)$ time and stop the modified sorting algorithm. Lemma 8.2 then enables us to compute a reduction with the properties (i) and (ii) stated there.

Otherwise, the interior of $[K_i, K_{i+1}]$ is nonempty. We compute a minimum spanning tree T with respect to h_{K_i} and test whether $h_{K_i}(T) \leq (1+\gamma)K_i$. If this is the case, then $K^* \leq K_i$, which implies that $K^* = K_i$ since we know that $K^* \in [K_i, K_{i+1}]$. Again, the adopted sorting procedure can stop after having computed the ordering of all edges with respect to h_{K^*}.

It remains the case that $K_i < K^* \leq K_{i+1}$. In this case it is easy to see that the ordering of the edges from the first round with respect to their h_{K^*}-weights coincides with the ordering with respect the weights given by h_τ, where $\tau \in (K_i, K_{i+1})$ is any interior point of the interval $[K_i, K_{i+1}]$.

Thus, at the end of the the first round, our algorithm has either found K^* and thus the ordering of *all* edges in the graph with respect to their h_{K^*}-weights, or we can answer the comparisons from the first round using the ordering of the edges with respect to h_τ.

The above process is repeated $\mathcal{O}(\log m)$ times, once for each parallel step of the parallel sorting machine. Since in each of the $\mathcal{O}(\log m)$ rounds we answer all comparisons of the parallel sorting scheme, upon termination we have found the ordering of the edges with respect to the h_{K^*}-weights. We then use Lemma 8.2 to compute a reduction strategy r.

The time needed for the algorithm above can be estimated as follows: There are $\mathcal{O}(\log m)$ cycles altogether. In each round we evaluate $\mathcal{O}(m)$ intersection points. Also, we need $\mathcal{O}(\log m)$ minimum spanning tree computations plus the overhead of $\mathcal{O}(m)$. This results in an overall time of $\mathcal{O}(m \log m + T_{\text{MST}} \log^2 m)$, where $T_{\text{MST}} = \mathcal{O}(n + m \log \beta(m, n))$ is the time needed for computing a minimum spanning tree. This gives us the following theorem:

Theorem 8.3 For any fixed $\gamma > 0$ the algorithm presented above is a $(1+1/\gamma, 1+\gamma)$-approximation algorithm for (Edge-cost, Total-weight, Spanning Tree). The running time of the algorithm is $\mathcal{O}(n \log^2 n + m \log^2 n \log \beta(n, m))$. □

Acknowledgements: We thank Cindy Phillips and Emanuel Knill for several useful conversations regarding network improvement problems.

References

[1] S. Arnborg, J. Lagergren and D. Seese, "Easy Problems for Tree-Decomposable Graphs", J. Algorithms, Vol. 12, 1991, pp. 308–340.

[2] S. Arnborg, B. Courcelle, A. Proskurowski and D. Seese, "An Algebraic Theory of Graph Problems", J. ACM, Vol. 12, 1993, pp. 308–340.

[3] O. Berman, "Improving The Location of Minisum Facilities Through Network Modification," *Annals of Operations Research*, 40(1992), pp. 1–16.

[4] M.W. Bern, E.L. Lawler and A.L. Wong, "Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs", *Journal of Algorithms*, 8, 1987, pp. 216–235.

[5] H.L. Bodlaender, "Dynamic programming algorithms on graphs with bounded treewidth", Proceedings of the 15th International Colloquium on Automata, Languages and Programming, Springer-Verlag, LNCS 317, 1988, pp. 105–119

[6] J. P. Cohoon and L. J. Randall, "Critical Net Routing," *IEEE Intern. Conf. on Computer Design*, 1991, pp. 174-177.

[7] R. Cole, "Parallel Merge Sort", *SIAM J. Computing*, 17(4), August 1988, pp. 770-785.

[8] J. Cong, A. B. Kahng, G. Robins, M. Sarafzadeh and C. K. Wong, "Provably Good Performance Driven Global Routing," *IEEE Transactions on Computer Aided Design*, 11(6), 1992, pp. 739-752.

[9] P. Crescenzi and V. Kann, "A compendium of NP optimization problems," Manuscript, (1995).

[10] W. Cunningham, "Optimal Attack and Reinforcement of a Network," *J. ACM*, 32(3), 1985, pp. 549-561.

[11] U. Feige, "A threshold of ln n for approximating set cover," To appear in the *Proceedings of the 28th Annual ACM Symposium on the Theory of Computation* (1996).

[12] G.N. Frederickson and R. Solis-Oba, "Increasing the Weight of Minimum Spanning Trees", *Proceedings of the Sixth Annual ACM-SIAM SODA'96*, March 1996.

[13] H. N. Gabow, Z. Galil, T. H. Spencer and R. E. Tarjan, "Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs," *Combinatorica*, 6 (1986), pp. 109–122.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[15] M. X. Goemans and D. P. Williamson, "A general approximation technique for constrained forest problems", In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'92)* (January 1992), pp. 307–316.

[16] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. B. Shmoys, E. Tardos, and D .P. Williamson, "Improved approximation algorithms for network design problems", In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'94)* (January 1994), pp. 223–232.

[17] R. Hassin, "Approximation schemes for the restricted shortest path problem", *Mathematics of Operations Research 17*, 1 (1992), 36–42.

[18] Dorit S. Hochbaum and David B. Shmoys, "A unified approach to approximation algorithms for bottleneck problems", *Journal of the ACM*, 33(3):533–550, July 1986.

[19] D.S. Johnson, "Approximation algorithms for combinatorial problems", *J. Comput. System Sci.* 9, 1974, 256-278.

[20] B. Kadaba and J. Jaffe, "Routing to Multiple Destinations in Computer Networks," *IEEE Trans. on Communication*, Vol. COM-31, Mar. 1983, pp. 343-351.

[21] P. Klein, and R. Ravi, "A nearly best-possible approximation for node-weighted Steiner trees," *Proceedings of the third MPS conference on Integer Programming and Combinatorial Optimization* (1993), pp. 323-332.

[22] V. P. Kompella, J. C. Pasquale and G. C. Polyzos, "Multicasting for Multimedia Applications," *Proc. of IEEE INFOCOM '92*, May 1992.

[23] V. P. Kompella, J. C. Pasquale and G. C. Polyzos, "Two Distributed Algorithms for the Constrained Steiner Tree Problem," Technical Report CAL-1005-92, Computer Systems Laboratory, University of California, San Diego, Oct. 1992.

[24] V. P. Kompella, J. C. Pasquale and G. C. Polyzos, "Multicast Routing for Multimedia Communication," *IEEE/ACM Transactions on Networking*, 1993, pp. 286–292.

[25] S. O. Krumke, H. Noltemeier, M. V. Marathe, S. S. Ravi and K. U. Drangmeister, "Modifying Networks to Obtain Low Cost Trees," to appear *Proc. Workshop on Graph Theoretic Concepts in Computer Science (WG'96)* June 1996.

[26] D. Karger and S. Plotkin, "Adding Multiple Cost Constraints to Combinatorial Optimization Problems, with Applications to Multicommodity Flows," *Proc. 27th Annual ACM Symp. on Theory of Computing* (STOC'95), May 1995, pp. 18–25.

[27] F.T. Leighton and S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Application to Approximation Algorithms", *Proceedings of the 29th Annual IEEE Conference on Foundations of Computer Science*, 1998, pp. 422–431

[28] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," *Proc., 25th Annual ACM Symp. on Theory of Computing* (STOC'93), May 1993, pp. 288–293.

[29] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz and H. B. Hunt III, "Bicriteria network design problems", In *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)* (1995), vol. 944 of *Lecture Notes in Computer Science*, pp. 487–498.

[30] N. Megiddo "Applying parallel computation algorithms in the design of serial algorithms", *Journal of the ACM 30*, 4 (October 1983), 852–865.

[31] D. Paik and S. Sahni, "Network Upgrading Problems," *Networks*, Vol. 26, 1995, pp. 45–58.

[32] C. Phillips, "The Network Inhibition Problem," *Proc. 25th Annual ACM STOC'93*, May 1993, pp. 288–293.

[33] F. Preparata, "New parallel-sorting schemes", *IEEE Transactions on Computing C-27* (1978), 669–673.

[34] N. Robertson and P. Seymour, "Graph Minors IV, Treewidth and Well-Quasi-Ordering", *J. Combin. Theory Ser. B*, 48, 1990, pp. 227–254.

[35] J. Valdes, R.E. Tarjan and E.L. Lawler, "The Recognition of Series-Parallel Digraphs", *SIAM Journal on Computing*, 11, 1982, pp. 1–12

[36] A. Warburton, "Approximation of pareto optima in multiple-objective shortest path problems". *Operations Research 35* (1992), 70–79.

[37] Q. Zhu, M. Parsa and W. Dai, "An Iterative Approach for Delay Bounded Minimum Steiner Tree Construction," Technical Report UCSC-CRL-94-39, University of California, Santa Cruz, Oct 1994.

Appendix

A Proofs for the Edge Improvement

A.1 Proof of Lemma 8.1:

The proof uses the following two results:

Lemma A.1 ([25]) Define F on $\mathbb{R}_{>0}$ by $F(K) := \frac{\mathrm{MST}_G(h_K)}{K}$. Then F is monotonically nonincreasing on $\mathbb{R}_{>0}$.

Corollary A.2 ([25]) If $\mathrm{MST}_G(h_{K'}) \leq (1+\gamma)K'$ for some $K' > 0$, then $\mathrm{MST}_G(h_K) \leq (1+\gamma)K$ for all $K \geq K'$.

In view of Lemma A.1 and Corollary A.2 it suffices to show that $\mathrm{MST}_G(h_{\tilde{K}}) \leq (1+\tilde{K})$, where $\tilde{K} = \mathrm{OPT}/\gamma$. Let r^* be an optimal feasible reduction and let T^* be a minimum spanning tree in G with respect to the weight function $\ell - r^*$. Let $\mathrm{OPT} := (\ell - r^*)(T^*)$ be its total weight in the graph with the edge lengths resulting from the optimal reduction r^*.

For each edge $e \in T^*$ we can estimate the weight $h_{\tilde{K}}(e)$ in the following way

$$h_{\tilde{K}}(e) = \min_{t \in [0, \ell(e) - \ell_{\min}(e)]} \left(\ell(e) - t + \frac{\tilde{K}}{B} c(e)(t) \right) \leq \ell(e) - r^*(e) + \frac{\tilde{K}}{B} c(e) r^*(e). \tag{10}$$

Summing up the inequalities in (10) over all $e \in T^*$, we obtain:

$$h_{\tilde{K}}(T^*) \leq \mathrm{OPT} + \frac{\tilde{K}}{B} B = \mathrm{OPT} + \tilde{K}. \tag{11}$$

We have seen that the weight of T^* under $h_{\tilde{K}}$ is no more than $\mathrm{OPT} + \tilde{K}$. Consequently, the *minimum* spanning tree with respect to $h_{\tilde{K}}$ has $h_{\tilde{K}}$-weight at most $\mathrm{OPT} + \tilde{K} = \gamma\tilde{K} + \tilde{K} = (1+\gamma) \cdot \tilde{K}$. This completes the proof. □

A.2 Proof of Lemma 8.2:

Given the ordering of the edges according to their weights, we can use the minimum spanning tree algorithm of Gabow et. al. [13] to compute a minimum spanning tree with respect to the h_{K^*}-weights, without actually knowing these weights. The ordering suffices for this purpose. The algorithm given in [13] runs in time $\mathcal{O}(n + m \log \beta(m, n))$

Let T be a minimum spanning tree with respect to h_{K^*}. We define a reduction r on T in the following way:

$$r(e) := \begin{cases} 0 & \text{if } K^* \geq B/c(e), \\ \ell(e) - \ell_{\min}(e) & \text{if } K^* < B/c(e). \end{cases}$$

By construction of the reduction r it then follows that

$$h_{K^*}(T) = \sum_{e \in T} \ell(e) - r(e) + \frac{K^*}{B} c(e) r(e) \geq (\ell - r)(T).$$

Since $h_{K^*} \leq (1+\gamma)K^*$ and $K^* \leq \mathrm{OPT}/\gamma$ this yields:

$$(\ell - r)(T) \leq (1+\gamma)K^* \leq (1+\gamma)\mathrm{OPT}/\gamma = (1+1/\gamma)\mathrm{OPT}.$$

The cost $\sum_{e\in T} r(e)c(e)$ of the reduction r can be estimated as follows. We have that

$$\frac{K^*}{B}\sum_{e\in T} c(e)r(e) \leq \sum_{e\in T}(\underbrace{\ell(e)-r(e)}_{\geq 0}+\frac{K^*}{B}c(e)r(e)) = h_{K^*}(T) \leq (1+\gamma)K^*.$$

Dividing the last chain of inequalities by $\frac{K^*}{B}$ yields that $\sum_{e\in T} r(e)c(e) \leq (1+\gamma)B$.

We have just shown that there is a reduction r on the particular tree T which involves a budget of at most $(1+\gamma)B$ and which reduces the weight of T to at most $(1+1/\gamma)$OPT. In fact, if we knew K^* we could construct the reduction r from above in time $\mathcal{O}(n)$ once we know the tree T.

But by the assumption of the lemma, we only have knowledge only about the *ordering* of the edges and *not* about K^* or h_{K^*}. Nevertheless, this is not grave. It is easy to see [25] that, given a tree and a budget, we can construct an optimal reduction on this tree for that budget in $\mathcal{O}(n)$ time by a Greedy-type algorithm that repeatedly reduces the length of the cheapest edge until the budget is exhausted. Thus, if we compute such a reduction r' on our tree T with the budget set to $(1+\gamma)B$, the length of T under $\ell - r'$ will be at most $(\ell - r)(T)$, which we have shown to be bounded from above by $(1+1/\gamma)$OPT. □

B Fully Polynomial-Time Approximation Schemes

The basic technique underlying our algorithm for the diameter case is approximate binary search using rounding and scaling, a method similar to that used by Hassin [17] and Warburton [36].

As in the previous subsection, let G be a treewidth-bounded graph. Let B be a bound on the cost of the nodes to be upgraded. Let ε be an accuracy parameter. Without loss of generality we assume that $\frac{1}{\varepsilon}$ is an integer. Let Alg$(G, \ell, \text{cost}, C)$ be a pseudopolynomial time algorithm for (Total-weight, Node-cost, Spanning Tree) on treewidth-bounded graphs, i.e. Alg$(G, \ell, \text{cost}, C)$ outputs a tree of upgraded length no more than C and minimizes the cost of the upgrading set[6]. Let the running time of Alg be $p(n, C)$ for some polynomial p. For carrying out our approximate binary search we need a testing procedure Test which is shown in Figure 2.

Procedure Test(M):
Input: G - treewidth bounded graph, C - bound on length of the upgraded tree, M - testing parameter, Alg - a pseudopolynomial time algorithm for (Total-weight, Node-cost, Spanning Tree) on treewidth-bounded graphs, ε - an accuracy parameter.

1 Let $\lfloor \frac{\ell}{M\varepsilon/(n-1)} \rfloor$ denote the cost function obtained by setting the cost of edge e to $\lfloor \frac{\ell(e)}{M\varepsilon/(n-1)} \rfloor$.

2 If there exists a C in $[0, \frac{n-1}{\varepsilon}]$ such that Alg$(G, \lfloor \frac{\ell}{M\varepsilon/(n-1)} \rfloor, \text{cost}, C)$ produces a spanning tree with total upgraded length at most C then output LOW otherwise output HIGH.

Output: HIGH/LOW.

Figure 2: Test Procedure.

Claim B.1 summarizes the property of Procedure Test(M). Finally, Algorithm FPAS-Upgrade shown in Figure 3, which uses Test, describes the overall strategy to compute near optimal solution. In the sequel we denote by OPT the optimal upgraded length of a mininum spanning tree T^* after upgrading a vertex set of cost at most B.

Claim B.1 If OPT $\leq M$ then Procedure Test(M) outputs LOW. If OPT $> M(1+\varepsilon)$ then the result of Test(M) is HIGH.

[6]Note that (Total-weight, Node-cost, Spanning Tree) is symmetric to the problem (Node-cost, Total-weight, Spanning Tree).

Algorithm FPAS-Upgrade

Input: G - treewidth-bounded graph, B - bound on the node upgrading cost, Alg- a pseudopolynomial time algorithm for (Total-weight, Node-cost, Spanning Tree) on treewidth-bounded graphs, ε - an accuracy parameter.

1 Let C_{hi} be an upper bound on the total weight of an MST after upgrading a node set of cost no more than B. Let $LB = 0$ and $UB = C_{hi}$.

2 **while** $UB \geq 2LB$ **do**

3 Let $M = (LB + UB)/2$.

4 **if** Test(M) returns HIGH **then** set $LB = M$ **else** set $UB = M(1+\varepsilon)$.

5 Run Alg$(G, \lfloor \frac{\ell}{LB\varepsilon/(n-1)} \rfloor, \text{cost}, C)$ for all C in $[0, 2(\frac{n-1}{\varepsilon})]$ and among all the trees with total upgrading cost at most B the tree with the lowest total weight (in the upgraded network).

Output: A spanning tree with total upgrading cost at most B and with total weight at most $(1+\varepsilon)$ times that of the optimally upgraded minimum spanning tree.

Figure 3: Approximation scheme on treewidth-bounded graphs.

Lemma B.2 Algorithm FPAS-Upgrade outputs a spanning tree with total node weight at most B and with total length in the upgraded network of at $(1+\varepsilon)$OPT.

Proof: It follows easily from Claim B.1 that the loop in Step 2 of Algorithm FPAS-Upgrade executes $\mathcal{O}(\log C_{hi})$ times before exiting with $LB \leq \text{OPT} \leq UB < 2LB$.

Since

$$\sum_{e \in T^*} \lfloor \frac{\ell(e)}{LB\varepsilon/(n-1)} \rfloor \leq \sum_{e \in T^*} \frac{\ell(e)}{LB\varepsilon/(n-1)} \leq \frac{\text{OPT}}{LB\varepsilon/(n-1)} \leq 2(\frac{n-1}{\varepsilon})$$

we get that Step 5 of Algorithm FPAS-Upgrade definitely outputs a spanning tree. Let T be the tree output. Then we have that the length $\ell'(e)$ of T in the upgraded network is given by

$$\sum_{e \in T} \ell'(e) \leq LB\varepsilon/(n-1) \sum_{e \in T} \frac{\gamma_e \ell(e)}{LB\varepsilon/(n-1)} \leq LB\varepsilon/(n-1)(\sum_{e \in T} \lfloor \frac{\gamma_e \ell(e)}{LB\varepsilon/(n-1)} \rfloor + 1).$$

Here $\gamma_e, \kappa_e \in \{1, 1/\varrho, 1/\varrho^2\}$, and reflects the way each edge has been upgraded in the heuristic and an optimal solution respectively. But since Step 5 of Algorithm FPAS-Upgrade outputs the spanning tree with minimum length we have that

$$\sum_{e \in T} \lfloor \frac{\gamma_e \ell(e)}{LB\varepsilon/(n-1)} \rfloor \leq \sum_{e \in T^*} \lfloor \frac{\kappa_e \ell(e)}{LB\varepsilon/(n-1)} \rfloor.$$

Therefore ¡

$$\ell'(T) \leq LB\varepsilon/(n-1) \sum_{e \in T^*} \lfloor \frac{\kappa_e \ell(e)}{LB\varepsilon/(n-1)} \rfloor + \varepsilon LB \leq \sum_{e \in T^*} \kappa_e \ell(e) + \varepsilon\text{OPT} \leq (1+\varepsilon)\text{OPT}.$$

This proves the claim.

□

As mentioned before, similar theorems hold for the other problems in Table 1 and all these results extend directly to Steiner trees.

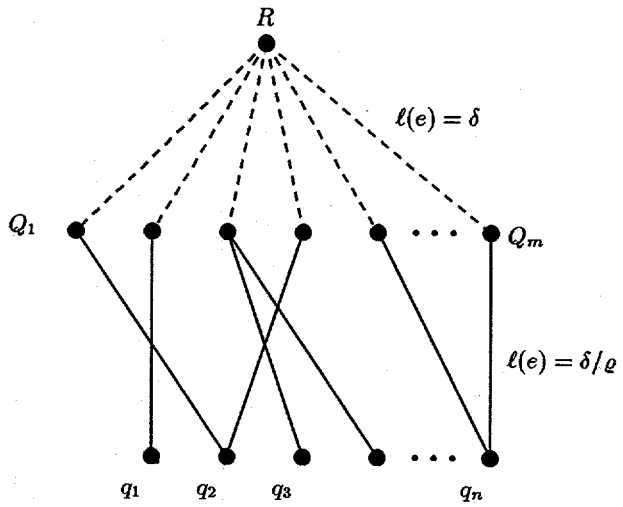

Figure 4: Reduction from Set Cover for the hardness of (Bottleneck, Node-cost, Spanning Tree).

C Hardness Results

C.1 Hardness of Bottleneck Tree Upgrading Problems

Theorem C.1 (Bottleneck, Node-cost, Spanning Tree) is NP-hard for any fixed speedup factor $0 < \varrho < 1$ and bound $\delta > 0$ on the bottleneck delay of the tree even for bipartite graphs and even if $\text{cost}(v) = 1$ for all $v \in V$.

Proof: We show that Set Cover ([SP5] in [14]) reduces to (Bottleneck, Node-cost, Spanning Tree) in polynomial time. An instance of Set Cover consists of a set Q of ground elements $\{q_1, \ldots, q_n\}$, a collection $Q_1, \ldots, Q_m$ of subsets of Q and an integer K. The question is whether one can pick at most K sets such that their union equals Q.

Given an instance of Set Cover, we first construct the natural bipartite graph, one side of the partition for set nodes Q_j, $j = 1, \ldots, m$, and the other for element nodes q_i, $i = 1, \ldots, n$. We insert an edge $\{Q_j, q_i\}$ iff $q_i \in Q_j$. We now add one more node R (the "root") and connect R to all the set nodes.

In the remainder of this proof, we do not distinguish between a node and the set or element that it represents. Note that the resulting graph is a bipartite (with R and the element nodes on one side and the set nodes on the other side). An example of the graph constructed is shown in Figure 4.

For each edge of the form (R, Q_i), the delay is δ, while each edge the form (Q_i, q_j) has delay δ/ϱ (in Figure 4 each of the dotted lines has delay δ and the solid lines have each delay δ/ϱ). We set the bound on the bottleneck delay parameter to δ.

We now claim that there is an upgrading set of size K resulting in a bottleneck spanning tree of bottleneck delay at most δ for the instance of (Bottleneck, Node-cost, Spanning Tree) just constructed if and only if the Set Cover instance has a cover of size at most K.

First assume that we can cover the elements in Q by at most K sets. Without loss of generality assume that the set cover consists of exactly K sets, which are $Q_1 \ldots, Q_K$. We then upgrade the corresponding set nodes. Consider the resulting graph. For each element node q_j there is now an edge of delay δ connecting q_j to some set node $S(q_j)$ from $Q_1, \ldots, Q_K$ (If q_j appears in two or more sets whose corresponding nodes are upgraded, then choose one such set node arbitrarily). Then the set $\{(r, Q_i) : i = 1, \ldots, m\} \cup \{(q_j, S(q_j)) : j = 1, \ldots, n\}$ is the edge set of a spanning tree of G, where none of the edges has weight more than δ.

Now assume conversely that there is an upgrading set of size at most K for the instance of (Bottleneck, Node-cost, Spanning Tree) constructed above. Let $V' \subseteq V$, $|V'| \leq K$ be a set of nodes that are upgraded and let $T = (V, E_T)$ be a spanning tree in the resulting graph of bottleneck cost at most δ.

We now transform the tree T into a tree T' of at most the same bottleneck cost such that each set node Q_j is adjacent to R in T'. To this end, do the following for each set node $Q_j, j = 1, \ldots, m$: If $(R, Q_j) \in T$, then continue with the next set node. Otherwise, since T is connected, there must be a path P from Q_j to R in T. By the bipartiteness of G, the first two edges in this path are of the from (Q_j, q_i) and $(q_i, Q_{j'})$ for some element node q_i and some other set node $Q_{j'}$. Adding the edge (R, Q_j) to T will induce a simple cycle in T containing the edges in P and (R, Q_j). Thus if we remove (Q_j, q_i) from T the resulting graph will again be a spanning tree of G.

Observe that the bottleneck cost of the tree does not increase during the procedure from above, since each edge (R, Q_j) inserted has already delay δ in the original graph.

Now, consider the set V' of upgraded nodes. If $R \in V'$, we can safely remove from V' without affecting the bottleneck cost of our spanning tree T'. For each element node $q_i \in V'$, we have at least one set node $S(Q_i)$ adjacent to q_i in T'. We replace q_i by $S(q_i)$ in V' and, continuing this, obtain a set V'' of at most K set nodes, which are upgraded.

We will now argue that the set nodes from V'' form a set cover. To this end, consider an arbitrary element node q_i. If $q_i \in V'$, i.e. q_i was one of the upgraded element nodes, then we have added some set node $S(q_i)$ to V' that is adjacent to q_i in G. Thus, $S(q_i) \in V''$ contains q_i and, consequently, q_i is covered by the set in V''. In the other case, $q_i \notin V'$. The tree T' contains at least one edge (Q_j, q_i) of delay at most δ. But, since q_i was not upgraded, the only possibility of (Q_j, q_i) having delay δ is that Q_j had been upgraded, i.e. $Q_j \in V'$. Since we have not removed any set node from V' in the transition to V'', it follows that $Q_j \in V''$ and thus, again, q_i is covered by the sets in V''. □

Note that the reduction in the proof of Theorem C.1 preserves approximations. Any set cover of size K becomes an upgrading set of size K and any upgrading set of size K becomes a cover of size at most K. Combining this approximation preserving transformation with the nonapproximability results of Feige [11] about the the optimization version of Set Cover we get the following theorem.

Theorem C.2 Unless NP $\subseteq$ DTIME($n^{\log\log n}$), there can be no polynomial time approximation algorithm for (Bottleneck, Node-cost, Spanning Tree) with a performance of (α, β) for any fixed $\alpha < \ln n$ and $\beta < 1/\varrho$, where $0 < \varrho < 1$ is the speedup factor occuring in the instance.

C.2 Hardness of Diameter Upgrading Problems

Theorem C.3 (Node-cost, Diameter, Spanning Tree) is NP-hard even for bipartite graphs and even if cost(v) $= 1$ for all $v \in V$.

Proof: Again, we use a reduction from Set Cover. Without loss of generality we will assume for the rest of the proof $K < n$ and that there is no single set Q_j covering all the elements.

Given any instance of Set Cover, we first construct the natural bipartite graph. We also add the "root" R and make it adjacent to all the set nodes. Now we add "enforcer" nodes f_i , one for each element node q_i, where f_i is adjacent exactly to q_i in the graph. An example of the graph constructed can be seen in Figure 5. The edges incident with R have delay 1, all the other edges have delay equal to 8. We now set $\delta := 13$ and the number of nodes to be upgraded to $K + n$, where n is the number of elements.

Assume that there is a set cover of size at most K. Then we upgrade the corresponding set nodes and, moreover, upgrade also *all* element nodes. The result is a feasible upgrading set. Since we have upgraded all the element nodes and the nodes corresponding to a set cover, for each element node q_i we have an edge $(q_i, S(q_i))$ in the resulting graph of delay equal to 2. We let the tree T consist of all these edges $(q_i, S(q_i))$ plus the edges from $\{(R, Q_j) : j = 1, \ldots, m\} \cup \{(q_i, f_i) : i = 1, \ldots, n\}$. It is easy to see that T has a diameter of at most 13.

Now, suppose conversely that there is a feasible upgrade set V' in G resulting in a diameter spanning tree T of diameter at most 13.

First, we claim that in this case for each element node q_i either this node or the "enforcer node" f_i must have been updated. Assume the contrary. In this case, each edge of the form $(q_i, Q) \in T$ has length at least

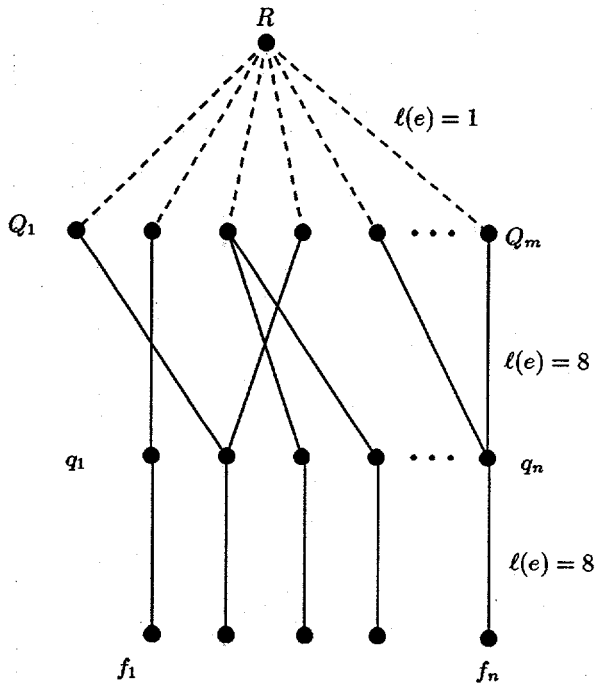

Figure 5: Reduction from Set Cover for the Hardness of (Node-cost, Diameter, Spanning Tree).

4. Any path from f_i to another enforcer node f_j must contain at least two edges (q_i, Q), (q_j, Q) and the edges (q_i, f_i) and (q_j, f_j). It is thus of length at least $4 + 2 + 8 + 2 = 16 > 13$, which is a contradiction.

We have seen that V' contains already at least n nodes from the set $\{q_i, f_i : 1 = 1, \ldots, n\}$. Thus, V' can have at most K set nodes that are upgraded. Consequently, if for each element node q_i, the tree T contains an edge $(q_i, S(q_i))$ of length 2, then the set nodes $s(q_i), i = 1, \ldots, n$ are a set cover of size at most K.

Now for $i = 1, \ldots, n$ we do the following. If T contains an edge (q_i, Q_j) of length 2, we continue with the next element node. Otherwise all edges in T connecting q_i to a set node have length at least 4.

If q_i has not been upgraded, we know that $f_i \in V'$. We remove f_i from V' and replace it by q_i. Clearly, the diameter of T does not increase during this process. If q_i has been upgraded, but f_i has not been upgraded, then we also step to the next element node. In the last case, both f_i and q_i belong to the set V'. Then, *none* of the set nodes adjacent to q_i in T has been upgraded. We pick an arbitrary set node Q_j such that $(Q_j, q_i) \in T$ and upgrade Q_j, removing f_i from V'. We claim that the diameter of the tree T does *not* increase in this step. In fact, assume $\mathcal{P}$ were a path of length greater than 13 in the tree with the new edge weights. Then it follows that q_i is adjacent to at least two set nodes in T (see Figure 6).

Moreover, the last edge of $\mathcal{P}$ must be (q_i, f_i), since otherwise $\mathcal{P}$ would already have been a path of length at least 13 in T *before* the operation. Now, if we consider the path $\mathcal{P}'$, where we replace (q_i, f_i) by (q_i, Q'), we see that $\mathcal{P}'$ is a path of the same length as $\mathcal{P}$ but this time in the tree T *before* the operation. This contradicts that the diameter of T is no more than 13.

Now consider the result of the process. After the iteration, *all* the element nodes belong to V'. Either now each element is connected to a set node via an edge of length at most 2 in T, in which case we have found a set cover. Otherwise, there is an element node q for which still each edge connecting it to a set node is of delay 4 (The edges can not have delay 8, since $q_i \in V'$). We now prove that this case can not happen, which then will complete the proof.

Assume that there is an element node q such that any edge connecting it to a set node in the tree T (which is of diameter at most 13 as we recall), has length 4.

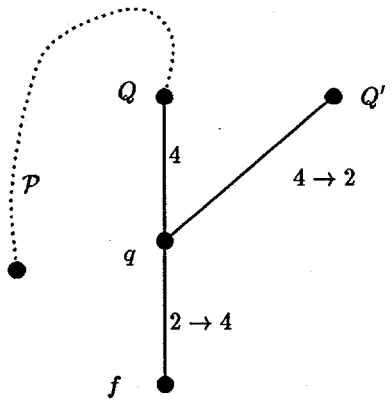

Figure 6: The diameter of T does not increase.

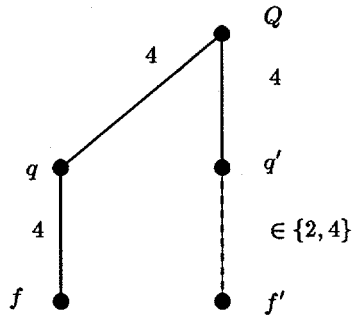

Figure 7: First case in the proof of Theorem C.3.

Choose an arbitrary element node $q' \neq q$. In the first case (see Figure 7), there is a set node Q adjacent to both q and q' in T. The length of the (unique) path in T from f to f' is then at least $4+4+4+2 = 14 > 13$, contradicting the assumption that T is of diameter at most 13.

In the second case, no set node is adjacent to both both q and q' in T. The path from f to f' in T then consists of the edge (f, q) an edge (q, Q) (of length 4), a path $\mathcal{P}$ from Q to another set node Q', and the edges (Q', q') and (q', f') (see Figure 8). Since Q is *not* upgraded (otherwise the edge (Q, q) would have

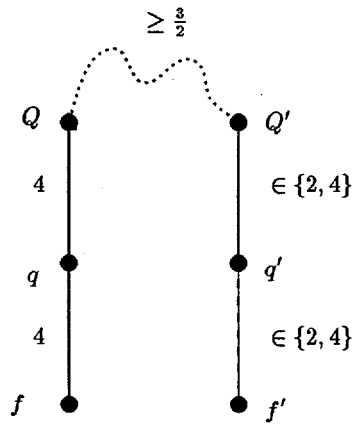

Figure 8: Second case in the proof of Theorem C.3.

length 2), the length of the path $\mathcal{P}$ is at least $1 + \frac{1}{2} = \frac{3}{2}$ (a shortest path in the graph G after the update always consists of the edges (Q, R) and (R, Q')). Summing up, we obtain that the distance between f and f' in T is at least $4 + 4 + \frac{3}{2} + 2 + 2 = \frac{27}{2} > 13$. This completes the proof. □

¿From the results from Theorem C.3 we can immediately conclude:

Theorem C.4 Unless P = NP, for any fixed $\varepsilon > 0$ there can be no polynomial time approximation algorithm for (Node-cost, Diameter, Spanning Tree) with a performance guarantee of $(1, \frac{27}{26} - \varepsilon)$.

C.3 Hardness Results for Treewidth Bounded Graphs

We first recall the definition of the Partition problem [14]. As an instance of the Partition problem we are given a multiset of (not necessarily distinct) positive integers $\{a_1, \ldots, a_n\}$ and the question is whether there exists a subset $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} a_i = S/2$, where $S := \sum_{i=1}^{n} a_i$ denotes the sum of all elements in A. Partition is well known to be NP-complete (cf. [14]).

Theorem C.5 (Node-cost, Diameter, Spanning Tree) and (Node-cost, Total-weight, Spanning Tree) are NP-hard even when restricted to simple trees with maximum node degree no more than 2.

Proof: We use a reduction from the Partition problem. Given an instance of Partition, we construct a tree with G with $3n$ vertices, x_i, y_i and z_i, $i = 1, \ldots, n$. and $3n - 1$ edges. The edges in the tree are $(x_i, y_i), (y_i, z_i), (z_i, x_{i+1})$ for $1 \leq i \leq n$ (excluding the edge (z_n, x_{n+1})). We now specify the necessary parameters. The upgrade factor is any number $0 < \varrho < 1$. The weights on the nodes y_i is a_i, while the nodes x_i and z_i have all weight $S := \sum_{i=1}^{n} a_i$. The cost of the edges (x_i, y_i) and (y_i, z_i) is is $a_i/2$. All edges (z_i, x_{i+1}) have cost ε. The simple tree constructed this way is shown in Figure 9.

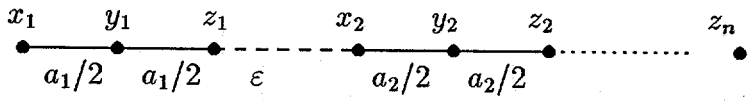

Figure 9: Chain used in the hardness proof on trees.

Now we claim that G has a spanning tree with diamater (total cost) no more than $\frac{S}{2}(1 + \varrho) + (n - 1)\varepsilon$ under the constraint that the total cost of the nodes in the improvement set is not more than $S/2$, if and only if A can be partitioned into two sets of equal weight. Observe that the diameter and total cost of the original tree equal $\sum_{i=1}^{n} a_i + (n - 1)\varepsilon = S + (n - 1)\varepsilon$

In fact, if I is an index set such that $\sum_{i \in I} a_i = S/2$, we the vertices y_i with $i \in I$ form an upgrading set of cost $S/2$. Upgrading these vertices will decrease the length of the corresponding edges (x_i, y_i) and (y_i, z_i) to $\varrho a_i/2$. Thus, after upgrading the vertices y_i with $i \in I$, this will result in a diameter (total cost) of $\sum_{i \notin I} a_i + \varrho \sum_{i \in I} a_i + = S/2 + \varrho S/2 + (n - 1)\varepsilon = \frac{S}{2}(1 + \varrho) + (n - 1)\varepsilon$.

Conversely, if we have an upgrading set of cost at most $S/2$, this set can contain only vertices y_i by the choice of the weights on the nodes x_i and z_i. Let I be the set of indices such that y_i belongs to the upgrading set. Again, upgrading a vertex y_i with $i \in I$ decreases the weight of the edges (x_i, y_i) and (y_i, z_i) to $xa_i/2$. Thus, after upgrading the vertices y_i the tree has diameter (total cost) $\sum_{i \notin I} a_i + \varrho \sum_{i \in I} a_i + (n - 1)\varepsilon = S + (\varrho - 1) \sum_{i \in I} a_i + (n - 1)\varepsilon$, which is at most $\frac{S}{2}(1 + \varrho) + (n - 1)\varepsilon$ by assumption. Simple algebra now shows that $\sum_{i \in I} a_i \geq S/2$. Since the total cost of the vertices y_i with $i \in I$ is exactly $\sum_{i \in I} a_i$ which is at most $S/2$, this shows that $\sum_{i \in I} a_i = S/2$. □