, , 1–35 ()

Approximation Algorithms for Certain Network Improvement Problems

SVEN O. KRUMKE krumke@zib.de
Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department of Optimization, Takustr. 7, 14195 Berlin-Dahlem, Germany.

MADHAV V. MARATHE madhav@lanl.gov
Madhav V. Marathe, Los Alamos National Laboratory, P.O.Box 1663, MS B265, Los Alamos NM 87545, USA.

HARTMUT NOLTEMEIER noltemei@informatik.uni-wuerzburg.de
Department of Computer Science, University of Würzburg, Am Hubland, 97074 Würzburg, Germany.

R. RAVI ravi+@cmu.edu
GSIA, Carnegie Mellon University, Pittsburgh, PA 15213.

S. S. RAVI ravi@cs.albany.edu
Department of Computer Science, University at Albany – SUNY, Albany, NY 12222, USA.

Editor:

Abstract. We study *budget constrained network upgrading problems.* Such problems aim at finding optimal strategies for improving a network under some cost measure subject to certain budget constraints. Given an edge weighted graph $G = (V, E)$, in the *edge based upgrading model*, it is assumed that each edge e of the given network also has an associated function $c_e(t)$ that specifies the cost of upgrading the edge by an amount t. 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 $c(v)$. Such an upgrade reduces the delay of each edge incident on v. 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.

After providing a brief overview of the models and definitions of the various problems considered, we present several new results on the complexity and approximability of network improvement problems.

Keywords: Complexity, NP-hardness, Approximation Algorithms

1. Introduction and Motivation

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 [7, 19, 21, 13, 14].

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 switching equipment at that node. Such an upgrade reduces the communication delay along each edge emanating from the node. Similarly, upgrading an edge corresponds to replacing an existing link with a new type of link. 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 given 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 finding optimal networks (e.g. spanning trees) in graphs, significantly less work has been reported on how to *modify* a graph so as to optimize the cost of a suitable subgraph of the resulting graph, when there is a budget constraint on the modification cost. Here, building on our recent work in [13, 14], we formulate and study such *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. In Section 5, we discuss the robustness and generality of our formulations. Section 4 briefly summarizes our results. Section 6 provides a comparison with related work.

In Section 7 we present our approximation algorithm for the bottleneck node upgrading problem on unweighted graphs and establish its performance guarantee. Section 8 investigates edge upgrading problems for constrained Steiner trees. Section 9 describes our results for the case when the whole graph needs to be upgraded.

2. Preliminaries and Upgrading Models

Throughout the presentation we assume that $G = (V, E)$ is a connected undirected graph. Let w be a nonnegative edge weight function defined on G. For a tree T in G, the *bottleneck-delay* of T under w is defined to be the weight of the heaviest edge in T. The *total weight* of T under the cost function w is the sum of the weights $w(e)$ of the edges $e \in T$. Finally, the *diameter* of T with respect to w, denoted by $\mathrm{dia}_w(T)$, is the length of a longest simple path in T.

Given a subset $K \subseteq V$ of distinguished nodes called *terminals*, any subgraph T of G which is a tree and which contains all the terminals from K is said to be a *Steiner tree* of G for K.

We now describe our node based and edge based upgrading models.

2.1. Node Based Upgrading Model

In the *node based upgrading model* we are given a connected undirected graph $G = (V, E)$. Further, for each edge $e = (u, v) \in E$, we are given three nonnegative numbers: $d(e)$ represents the *length* or *delay* of the link e. If exactly one of the endpoints u and v is upgraded, the delay of e decreases to $d_m(e)$, the "medium" delay. If both endpoints are upgraded, then the delay falls to $d_l(e)$, the "low" delay. It is assumed that $d_l(e) \le d_m(e) \le d(e)$.

Thus, the upgrade of a node v reduces the delay of each edge incident on v. For each node $v \in V$ the value $c(v)$ specifies how expensive it is to upgrade the node. For a subset W of V, the cost of upgrading all the nodes in W, denoted by $c(W)$, is equal to $\sum_{v \in W} c(v)$.

2.2. Edge Based Upgrading Model

In the *edge based upgrading model*, each edge $e \in E$ is associated with two nonnegative numbers as follows: $\ell(e)$ denotes the *length* or the *weight* 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 $\ell_{\min}(e) \le \ell(e)$. The nonnegative *cost function* $c_e(t)$ associated with edge e indicates how expensive it is to reduce the length of e by an amount t. We assume without loss of generality that $c_e(0) = 0$ for all edges $e \in E$.[1] We also make the natural assumption that each cost function c_e is nondecreasing. In this paper we will restrict ourselves to the case when the cost functions c_e are *linear*, i.e., $c_e(t) = c_e \cdot t$ for some constant $c_e \ge 0$.

A *reduction strategy* (or simply *reduction*) r on the edges of G specifies how to reduce the ℓ-length of each edge e to a value in the range $[\ell_{\min}(e), \ell(e)]$. Formally, we require that

$$\ell(e) - r(e) \ge \ell_{\min}(e) \qquad \text{for all } e \in E.$$

The *cost* of the reduction r is $\sum_{e \in E} c_e(r(e))$. If r is a reduction on G, then we can consider the graph G with edge weights given by the "reduced lengths", namely $(\ell - r)(e) := \ell(e) - r(e)$ $(e \in E)$.

3. Problem Formulations and Notion of Approximation

As already mentioned, the problems studied in this paper are formulated as multicriteria optimization problems.

Definition 1. [**k**-Criteria Optimization Problem] For any fixed $k \ge 2$, a **k**-*criteria minimization problem* Π on a weighted graph is defined by specifying k polynomial time computable minimization objectives, $f_1, \ldots, f_k$, and a membership requirement in a class of weighted subgraphs Γ (not necessarily weighted exactly the same way as the original graph).

The problem specifies budget values $F_2, \ldots, F_k$ as upper bounds on the objectives $f_2, \ldots, f_k$. The goal is to find a subgraph from the set

$$\{\, x \in \Gamma : f_i(x) \leq F_i, \quad i = 2, \ldots, k \,\}, \tag{1}$$

having minimum possible value for f_1.

Given an instance of a k-criteria problem, the *set of feasible solutions* for the instance is defined as the set given in (1).

As an example, consider the following node based bottleneck spanning tree upgrading problem, denoted by (Node-Upgrade, Bottleneck, Spanning Tree). (This problem is studied in Section 7.)

Definition 2. [Node Weighted Bottleneck Tree Upgrading Problem] Given an edge and node weighted graph $G = (V, E)$ as in Section 2.1 and a bound D, the problem (Node-Upgrade, Bottleneck, Spanning Tree) is to upgrade a set $S \subseteq V$ of nodes such that the resulting graph has a spanning tree of bottleneck delay at most D and $c(S)$ is minimized.

In the above problem, the class Γ of subgraphs consists of all the spanning trees. The first objective (which is to be minimized) is the upgrading cost, while the second objective is the bottleneck weight of the tree after the upgrade.

Similarly, the edge based Steiner tree upgrading problem, denoted by (Total Weight, Diameter, Edge-Upgrade, Steiner Tree), can be formulated as follows. (This problem is studied in Section 8.)

Definition 3. [Diameter Constrained Steiner Tree Problem] Let $G = (V, E)$ be an edge weighted graph as in Section 2.2 and let d be an additional edge weight function d which is independent of ℓ. Given a bound D on the d-diameter and a nonnegative budget value B, the *diameter and budget constrained minimum total cost Steiner tree problem*, (Total Weight, Diameter, Edge-Upgrade, Steiner Tree), is to find a subgraph from the set

$$\left\{ (T, r) : \begin{array}{l} T \text{ is a Steiner tree of } G \text{ with the terminals } K, \\ r \text{ is a reduction on } G, \\ \mathrm{dia}_d(T) \leq D \text{ and } \sum_{e \in T} c_e \cdot r(e) \leq B \end{array} \right\}, \tag{2}$$

such that the weight of the subgraph under the weight function $(\ell - r)$ is as small as possible.

We now introduce the notion of approximation that is used throughout the paper.

Definition 4. [**k**-Criteria Approximation Algorithm] For any fixed $k \geq 2$, a **k**-*criteria approximation algorithm* for a k-criteria minimization problem Π with performance $(\alpha_1, \ldots, \alpha_k)$ is a polynomial time algorithm with the following properties:

Given any instance of Π with a nonempty set of feasible solutions, the algorithm finds an "almost feasible solution", that is a solution y from the set

$$\{\, x \in \Gamma : f_i(x) \leq \alpha_i \cdot F_i, \quad i = 2, \ldots, k \,\} \tag{3}$$

such that $f_1(y) \leq \alpha_1 \cdot \mathrm{OPT}$. Here OPT denotes the minimum value of a solution satisfying all the constraints.

If there is no feasible solution for an instance, the algorithm has the choice of producing an "almost feasible solution" from the set given in (3) or providing the (correct) information that the set of feasible solutions is empty.

In general, each *performance factor* α_i depends on the input size and on the budget values $F_2, \ldots, F_k$. However, for several problems considered in this paper, the α_i values are constants. The performance of an approximation algorithm is measured in terms of both near-optimality and the extent to which the constraints are violated in the solution produced. Notice that any solution produced by the algorithm is contained in the set (3). Thus, the solution belongs to the subgraph-class Γ.

For example, an (α, β)-approximation algorithm for (Node-Upgrade, Bottleneck, Spanning Tree) finds for any instance of the problem, an upgrading set S of cost at most $\alpha \cdot \mathrm{OPT}$ such that the upgraded graph has a spanning tree of bottleneck delay at most β times the given threshold D. Here, OPT is the minimum upgrade cost needed to obtain a spanning tree of bottleneck delay at most D.

4. Summary of Results

We study the complexity and approximability of a number of node based and edge based upgrading problems. We consider three objectives to evaluate the cost of a (sub-)graph in the modified network: the bottleneck delay, the diameter and the total cost.

Tables 1 and 2 summarize our results. The node based upgrading problems use the model from Section 2.1 while the edge based upgrading problems utilize the model defined in Section 2.2. In Tables 1 and 2, the rows are indexed by the problem. The columns indicate the hardness and approximation results for each problem. Given a bicriteria problem (i.e., a problem involving two criteria), we can obtain a dual (or symmetric) version of the problem by interchanging the budgeted objective and the minimization objective. In Section 5, we establish a general result (Theorem 2) that allows us to obtain an approximation algorithm for the dual problem given an approximation algorithm for the original problem. For this reason, results for the dual problems are not explicitly indicated in the tables. The tables include the following results.

1. The problems (Edge-Upgrade, Bottleneck, Graph), (Edge-Upgrade, Total-Weight, Graph) and (Edge-Upgrade, Bottleneck, Spanning Tree) are solvable in polynomial time.
2. The problem (Node-Upgrade, Bottleneck, Graph) is MaxSNP-hard but has a polynomial time $(2, 1)$-approximation algorithm.
3. (U-Node-Upgrade, Bottleneck, Spanning Tree) has a polynomial time $(5 + 4 \ln \Delta, 1)$ approximation algorithm. Here, (U-Node-Upgrade, Bottleneck, Spanning Tree) is the restriction of the problem (Node-Upgrade,

Table 1. Results for node based upgrading problems. (The hardness results assume that $\mathsf{NP} \not\subseteq \mathsf{DTIME}(n^{\mathcal{O}(\log\log n)})$.)

Problem	Hardness Result	Performance Guarantee
(U-NODE-UPGRADE, BOTTLENECK, SPANNING TREE)	Not approximable within $(\alpha, f(n))$ for any $\alpha < 1/3 \cdot \ln n$ and polynomial time computable function f (see [14]).	$(5 + 4\ln\Delta, 1)$, where Δ is the maximum degree in the graph.
(NODE-UPGRADE, BOTTLENECK, GRAPH)	Not approximable within $(\alpha, f(n))$ for any $\alpha < 1.038$ and any polynomial time computable function f.	$(2, 1)$
(NODE-UPGRADE, TOTAL-WEIGHT, GRAPH)	Not approximable within $(\alpha, f(n))$ for any $\alpha < 1.038$ and any polynomial time computable function f.	open

Table 2. Results for edge based upgrading problems. (The hardness results assume that $\mathsf{P} \neq \mathsf{NP}$.)

Problem	Hardness Result	Performance Guarantee
(TOTAL WEIGHT, DIAMETER, EDGE-UPGRADE, STEINER TREE)	Not approximable within $(\alpha, \beta, 1)$ or $(1, \alpha, \beta)$ for any $\alpha, \beta \geq 1$.	$(\mathcal{O}(\log\|K\|), \mathcal{O}(\log\|K\|), \mathcal{O}(\log\|K\|))$, where K is the set of terminals.
(EDGE-UPGRADE, BOTTLENECK, GRAPH)	polynomial time solvable.	
(EDGE-UPGRADE, BOTTLENECK, SPANNING TREE)	polynomial time solvable.	

BOTTLENECK, SPANNING TREE) to those instances where the cost of upgrading each node is 1, and Δ denotes the maximum node degree in the input graph.

4. There is an approximation algorithm for the (TOTAL WEIGHT, DIAMETER, EDGE-UPGRADE, STEINER TREE) problem with a performance guarantee of $(\mathcal{O}(\log|K|), \mathcal{O}(\log|K|), \mathcal{O}(\log|K|))$.

In addition to the above results for specific network improvement problems, we also extend the results in [18] and show that our formalism for k-criteria approximation is both robust and general. It is more general because it subsumes the case considered in [18] where one wishes to minimize some functional combination of two criteria. It is more robust because the quality of approximation is independent of which of the two criteria is chosen as the budgeted objective.

5. Generality and Robustness of Formulations

5.1. A Note on the Infeasibility of Multicriteria Problems

A seeming drawback of our Definition 4 is that when an instance of a k-criteria optimization problem does not have a feasible solution, a k-criteria approximation algorithm might give us an "almost feasible solution" with a very bad objective function value (since in this case the algorithm is allowed to pick *just any* "almost feasible solution").

Of course, if we can check the feasibility of an instance in polynomial time, then there are no severe problems. We can perform this check *before* running the k-criteria approximation algorithm. If we detect infeasibility, we simply provide an indication of that fact and stop.

For the rest of this subsection we assume that, given an instance of the k-criteria minimization problem Π, determining whether the set of feasible solutions is empty is NP-complete. (This is true of all the NP-hard problems considered in this paper.) Under this assumption, the existence of any polynomial time algorithm that outputs a feasible solution if and only if a problem instance is feasible, will imply that $\mathsf{P} = \mathsf{NP}$. Thus, it seems necessary to allow a k-criteria approximation algorithm the "freedom" of violating most or all of the constraints by small factors while approximating the minimization objective.

Let A be an $(\alpha_1, \dots, \alpha_k)$-approximation algorithm for the problem Π, where for simplicity the α_i are assumed to be constants greater or equal to 1. Suppose that given an instance I of Π the algorithm A returns the solution $\tilde{x}$, and does not produce an indication of infeasibility. We denote by

$$\mathcal{S}_I = \{\, x \in \Gamma : f_i(x) \le F_i, i = 2, \dots, k \,\}$$

the set of feasible solutions for I. We now run A again on the instance I' obtained from I by setting the bounds on the objectives f_i to $F_i' := \alpha_i F_i$ for $i = 2, \dots, k$. Thus, for I' the problem consists of finding a subgraph from the set

$$\begin{aligned} \mathcal{S}_{I'} &= \{\, x \in \Gamma : f_i(x) \le F_i', i = 2, \dots, k \,\} \\ &= \{\, x \in \Gamma : f_i(x) \le \alpha_i F_i, i = 2, \dots, k \,\} \end{aligned} \tag{4}$$

minimizing f_1. Notice that the set of feasible solutions defined in (4) is nonempty, since $\tilde{x} \in \mathcal{S}_{I'}$. Let the corresponding solution delivered by A be x'. We then output $\tilde{x}$ or x', depending on which of the two solutions has a better objective function value under f_1.

The solution $\hat{x}$ generated this way will violate the constraints $f_i(x) \le F_i$ at most by the factors α_i^2, $i = 2, \dots, k$. Moreover, $\hat{x}$ is a good approximate solution for the "relaxed" instance I' corresponding to I and satisfies

$$f_1(\hat{x}) \le \alpha_1 \cdot \min\{\, f_1(x) : x \in \mathcal{S}_I \cup \mathcal{S}_{I'} \,\}.$$

Also, if the set of feasible solutions $\mathcal{S}_I$ of the original instance I is nonempty, then

$$f_1(\hat{x}) \leq \alpha_1 \cdot \min\{\, f_1(x) : x \in \mathcal{S}_I \,\}.$$

The above arguments show that we can augment a k-criteria approximation algorithm to produce acceptably good solutions even if our original problem instance is infeasible. Instead of just outputting *any* "almost feasible solution", we can find one that has a *good* objective function value f_1 for a relaxed problem instance. In view of the assumption that deciding feasibility is NP-complete, this appears to be the best that we can accomplish in polynomial time (assuming that $\mathsf{P} \neq \mathsf{NP}$).

5.2. Generality

By "generality" we mean that our results for k-criteria formulations can be easily modified to obtain results for classes of problems involving only one criterion. In this section we discuss how approximation algorithms for multicriteria problems can be used to obtain approximation algorithms for certain related unicriterion problems.

Let Π be a k-criteria minimization problem. For a fixed weight-vector $\sigma = (\sigma_1, \ldots, \sigma_k) \in \mathbb{N}^k$, we denote by Π_+^σ and $\Pi_\times^\sigma$ the unicriterion problems of finding a subgraph from Γ minimizing $f_+^\sigma := \sum_{i=1}^k \sigma_i f_i$ and $f_\times^\sigma := \prod_{i=1}^k f_i^{\sigma_i}$ respectively. Throughout this section we will assume that the objectives $f_1, \ldots, f_k$ are integer valued. In case of rational objectives, the results can be extended with a small loss in the performance guarantees. The restriction to integral cost functions will help us to focus on the essential connections between multicriteria approximation and unicriterion approximation. We will also make the following assumption about the *boundedness* of the objective function values of Π_+^σ and $\Pi_\times^\sigma$:

Assumption 1. Let $\Pi' \in \{\Pi_+^\sigma, \Pi_\times^\sigma\}$. There is a polynomial p with the following property: for any instance I of Π' with encoding length $|I|$ there exist (polynomial time computable) numbers $B_2, \ldots, B_k$ with $0 \leq B_i \leq 2^{p(|I|)}$ for $i = 2, \ldots, k$ such that the optimal solution x^* to the instance of Π' satisfies:

$$1 \leq f_i(x^*) \leq B_i, \qquad \text{for } i = 2, \ldots, k.$$

It should be noted here that Assumption 1 is not a severe restriction. If we assume a reasonable encoding scheme (such as the usual binary encoding) for numbers, then the assumption merely states that the encoding lengths of the single objective function values $f_i(x^*)$ of an optimal solution x^* remain polynomially bounded in the input length. If this were not the case, then the encoding lengths of $f_+^\sigma(x^*)$ and $f_\times^\sigma(x^*)$ would not be polynomially bounded (assuming again a reasonable encoding scheme). Hence, there can be no polynomial time algorithm, deterministic or nondeterministic, that finds an optimal solution *and* outputs the optimal function value, since the encoding length of the objective function value is exponential in the input size.

Theorem 1 *Assume that there is an $(\alpha_1,\ldots,\alpha_k)$-approximation algorithm for a k-criteria problem Π. Then for any $\varepsilon > 0$ and $\sigma = (\sigma_1,\ldots,\sigma_k) \in \mathbb{N}^k$ the following statements hold:*

(i) There is a polynomial time approximation algorithm for Π_+^σ with a performance guarantee of $(1+\varepsilon)\max\{\alpha_1,\ldots,\alpha_k\}$.

(ii) There is a polynomial time approximation algorithm for $\Pi_\times^\sigma$ with a performance guarantee of $(1+\varepsilon)^{\sum_{i=2}^k \sigma_i} \prod_{i=1}^k \alpha_i^{\sigma_i}$.
In particular, if $\sigma = (1,1,\ldots,1)$, then the performance of the approximation algorithm for $\Pi_\times^\sigma$ is $(1+\varepsilon)^{k-1}\prod_{i=1}^k \alpha_i$.

Proof: Let A be an $(\alpha_1,\ldots,\alpha_k)$-approximation algorithm for the problem Π. We first consider Π_+^σ. The idea behind our approximation algorithm for the unicriterion problem Π_+^σ is the following: Let x^* be an optimal solution of value $\sum_{i=1}^k \sigma_i f_i(x^*)$ for a given instance of Π_+^σ. Then, x^* is a feasible solution to the instance of the k-criteria problem Π of minimizing $f_1(x)$ subject to the constraints that $f_i(x) \le F_i := f_i(x^*)$. The feasibility of x^* allows us to argue that there is a "good" optimal solution to this instance of Π, meaning that its objective value under f_1 is at most that of x^*. Using the k-criteria algorithm A would enable us to find a "good" solution where also the linear combination of the objective values f_i, $i = 1\ldots,k$ via σ is reasonably close to the solution value for x^*. Unfortunately, we do not know the "right bounds" $f_i(x^*)$ to use as upper bounds. Thus, we construct an appropriate search space of bounds which is large enough to find good bounds and small enough to be searchable exhaustively in polynomial time.

We formalize the above idea in Algorithm Convert-Sum shown in Figure 1. Since $\varepsilon > 0$, k and σ_i are all constants, it follows easily that Algorithm Convert-Sum can be implemented to run in polynomial time.

Let x^* be an optimal solution to the unicriterion problem Π_+^σ. Consider the call in Algorithm Convert-Sum to the k-criteria algorithm A, when the parameter $\mathbf{z}$ satisfies:

$$f_i(x^*) \le z_i < (1+\varepsilon) f_i(x^*), \qquad i = 2,\ldots,k. \tag{5}$$

Observe that by construction of the algorithm, such a call will be issued. The algorithm A is hence applied to the k-criteria problem of finding a subgraph from

$$\{\, x \in \Gamma : f_i(x) \le z_i, i = 2,\ldots,k \,\}, \tag{6}$$

having minimum possible value for f_1. Because of (5), the optimal solution x^* to the unicriterion problem Π_+^σ is contained in the feasible set (6), thus proving that this set is nonempty. Therefore, by the performance guarantee of A, the k-criteria approximation algorithm will return a subgraph $\hat{x} := \mathsf{A}(\mathbf{z})$ from

$$\{\, x \in \Gamma : f_i(x) \le \alpha_i \cdot z_i, i = 2,\ldots,k \,\} \tag{7}$$

ALGORITHM CONVERT-SUM

- *Input:* A k-criteria approximation algorithm A for Π, an instance of Π_+^σ.

1. **for** $i \leftarrow 2, \ldots, k$
   (a) $r_i \leftarrow \lceil \log_{(1+\varepsilon)} B_i \rceil$.
   (b) $M_i \leftarrow \{ (1+\varepsilon)^j : j = 0, \ldots, r_i \}$.
2. **endfor**
3. $M \leftarrow M_2 \times \cdots \times M_k$
4. For a vector $\mathbf{z} = (z_2, \ldots, z_k)$, let $\mathsf{A}(\mathbf{z})$ denote the subgraph returned by the k-criteria algorithm A, when called with the bounds z_i on the values of the objectives f_i, $i = 2, \ldots, k$.
5. Find $\mathbf{z}^* \in M$ such that $\sum_{i=1}^k \sigma_i f_i(\mathsf{A}(\mathbf{z}))$ is a minimum among all $z \in M$.
6. **return** $\mathsf{A}(\mathbf{z}^*)$.

Figure 1. Converting a k-criteria algorithm into a unicriterion algorithm.

such that $f_1(\hat{x}) \leq \alpha_1 \cdot \mathrm{OPT}$. Here, OPT denotes the minimum objective value under f_1 of a solution from (6). Combining the fact that $\hat{x}$ is contained in (7) with Inequality (5) yields

$$f_i(\hat{x}) \leq (1+\varepsilon)\alpha_i \cdot f_i(x^*), \qquad i = 2, \ldots, k. \tag{8}$$

Moreover, since we have seen that the optimal solution x^* of the unicriterion problem is contained in the set (6) of feasible solutions of the instance of the k-criteria problem just constructed, we obtain that $\mathrm{OPT} \leq f_1(x^*)$. Thus, again by the performance of A this implies

$$f_1(\hat{x}) \leq \alpha_1 \cdot \mathrm{OPT} \leq \alpha_1 \cdot f_1(x^*). \tag{9}$$

Combining (8) and (9) gives us:

$$\begin{aligned} \sum_{i=1}^k \sigma_i f_i(\hat{x}) &\leq \alpha_1 \cdot \sigma_1 f_1(x^*) + (1+\varepsilon) \sum_{i=2}^k \alpha_i \cdot \sigma_i f_i(x^*) \\ &\leq (1+\varepsilon) \cdot \max\{\alpha_1, \ldots, \alpha_k\} \cdot \sum_{i=1}^k \sigma_i f_i(x^*). \end{aligned}$$

Since ALGORITHM CONVERT-SUM chooses the $\mathbf{z}^* \in M$ such that $\sum_{i=1}^k \sigma_i f_i(\mathsf{A}(\mathbf{z}))$ is minimum among all $\mathbf{z} \in M$, the claimed performance guarantee now follows in the case of Π_+^σ.

The proof for the problem $\Pi_\times^\sigma$ follows along the same lines. We just have to change Step 5 of ALGORITHM CONVERT-SUM to select $\mathbf{z}^*$ such that $\prod_{i=1}^k f_i^{\sigma_i}(\mathsf{A}(\mathbf{z}^*))$ is minimum among all $\mathbf{z} \in M$. Then, using the same arguments as above, we get from (8) and (9) that

$$\begin{aligned}\prod_{i=1}^{k} f_i^{\sigma_i}(\hat{x}) &\leq \alpha_1^{\sigma_1} \cdot f_1^{\sigma_1}(x^*) \cdot \prod_{i=2}^{k}(1+\varepsilon)^{\sigma_i} \alpha_i^{\sigma_i} \cdot f_i^{\sigma_i}(x^*) \\ &= \left((1+\varepsilon)^{\sum_{i=2}^{k}\sigma_i} \prod_{i=1}^{k} \alpha_i^{\sigma_i}\right) \cdot \prod_{i=1}^{k} f_i^{\sigma_i}(x^*).\end{aligned}$$

This completes the proof. □

Theorem 1 establishes a connection between the approximability of a multicriteria problem and certain unicriterion problems. In particular, if a bicriteria problem is "well approximable" with performance (α_1, α_2), where α_1 and α_2 are constants, then there is also a "good" constant factor approximation for the problem of minimizing the sum of the two objectives.

The converse is not true in general. To see this, we will provide an example of a problem Π_+ of minimizing the objective $f := f_1 + f_2$, which is well-approximable, in fact polynomial time solvable. In contrast, the bicriteria problem Π of minimizing f_1 subject to a constraint on the value of f_2 turns out to be very hard to approximate.

Consider the bicriteria problem Π of finding an independent dominating set U in a graph $G = (V, E)$ minimizing $f_1(U) := |U|$ subject to the constraint that $f_2(U) := |V - U| \leq F_2$. The problem Π_+ where f_1 and f_2 are functionally combined to $f := f_1 + f_2$ consists of finding an independent dominating set U minimizing $f_1(U) + f_2(U) \equiv |V|$. Clearly, Π_+ can be solved in polynomial time, e.g. by the trivial algorithm that simply outputs *any* independent dominating set in the graph G. Thus, Π_+ can be deemed "well-approximable".

In contrast, Π exhibits a different behavior. If we set the bound F_2 on the cardinality of $|V - U|$ to be $F_2 := |V|$, then Π becomes the problem of finding an independent dominating set of minimum cardinality. This problem is hard to approximate [3]: unless $\mathsf{P} = \mathsf{NP}$, for any $\varepsilon > 0$ there can be no polynomial time approximation algorithm with a performance of $|V|^{1/6-\varepsilon}$ for this problem. Thus, the bicriteria problem Π contains as a special case a problem which is hard to approximate, while the unicriterion problem Π_+ is well-approximable.

5.3. Robustness

In this section we will elaborate on the "robustness" of our formulations and the notion of approximation in the case of two criteria. By "robustness" we mean that the quality of approximation is independent of which of the two criteria is chosen as the budgeted objective. To see this, note that there are two natural ways to formulate a bicriteria problem:

1. (f_1, f_2, Γ) - find a subgraph in Γ whose f_2-objective value is at most B and which has minimum f_1-objective value,
2. (f_2, f_1, Γ) - find a subgraph in Γ whose f_1-objective value is at most B and which has minimum f_2-objective value.

Theorem 2 *If there is an (α, β)-approximation algorithm for (f_1, f_2, Γ), then for any $\varepsilon > 0$ there is a $((1+\varepsilon)\beta, \alpha)$-approximation algorithm for the dual problem (f_2, f_1, Γ).*

Moreover, if the objective f_1 is integer valued, then there is a (β, α)-approximation algorithm for the dual problem (f_2, f_1, Γ).

Proof: Let A be an (α, β)-approximation algorithm for (f_1, f_2, Γ). We will show how to use A to construct a $((1+\varepsilon)\beta, \alpha)$-approximation algorithm for the dual problem. The basic idea is to use A to search for the optimal objective function value under f_2 for the given instance of (f_2, f_1, Γ).

An instance of (f_2, f_1, Γ) is specified by a weighted graph $G = (V, E)$ and a bound F_2 on the objective f_1. Let OPT be the optimal function value under f_2 subject to the constraint on f_1. By Assumption 1, we can compute a number R in polynomial time such that $0 \leq \text{OPT} \leq R$. For a given number $0 \leq D \leq R$, let I_D be the instance of (f_1, f_2, Γ) obtained by specifying the bound D on objective f_2.

We search the set

$$\mathcal{M} := \{0, 1+\varepsilon, (1+\varepsilon)^2, \ldots, (1+\varepsilon)^k\}, \quad \text{where } k = \lceil \log_{1+\varepsilon} R \rceil \tag{10}$$

and find the minimum value $D \in \mathcal{M}$ such that A as applied to I_D outputs a subgraph x with objective function value f_1 at most αB. It is easy to see that this binary search indeed works and terminates with a value $D' \leq (1+\varepsilon)\text{OPT}$.

The corresponding subgraph $x' \in \Gamma$ has objective function value, under f_2, of at most $\beta D' \leq (1+\varepsilon)\beta \cdot \text{OPT}$ and violates the constraint on objective f_1 by a factor of at most α. This way we obtain a $((1+\varepsilon)\beta, \alpha)$-approximation for (f_2, f_1, Γ).

If f_1 is integer valued, then instead of performing a search on the set $\mathcal{M}$ defined in (10) we can perform the binary search over the integers in the interval $[0, R]$. This provides a (β, α)-approximation for (f_2, f_1, Γ). □

By the result of the last theorem, our approximation results for (f_1, f_2, Γ) problems in the following sections will also yield approximation algorithms for the symmetric problem (f_2, f_1, Γ).

6. Comparison with Related Work

Several researchers have worked on versions of network upgrading problems. Frederickson and Solis-Oba [7] considered the problem of increasing the weight of a 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 results presented here, they show that while the integral case is NP-hard, the rational case is solvable in polynomial time using tools from matroid theory. Berman [2] considers the problem of shortening the edges of a given tree to minimize the weight of its shortest path tree and shows that the problem can be solved in strongly polynomial time.

Plesnik [20] has shown that the budget-constrained minimum diameter problem (i.e., given a graph $G = (V, E)$ with a length $\ell(e)$ and cost $c(e)$ for each edge $e \in E$ and a cost budget B, select a subset E' of E so that the total cost of edges in E'

is at most B and the diameter of the graph formed by E' is a minimum among all subsets satisfying the budget constraint) is NP-hard. He also shows that, if the budget constraint cannot be violated, then even approximating the diameter to within a factor of less than 2 is NP-hard.

Hambrusch and Tu [12] consider budget constrained edge-based upgrading problems for directed graphs. In their work, the performance of the modified network is characterized by the length of a longest path. They present hardness results for general graphs and polynomial algorithms for special classes such as in-trees and series-parallel graphs. Phillips [19] studies the problem of finding an optimal strategy for reducing the capacity of a given network so that the residual capacity in the modified network is minimized. The problems studied here and in [19, 2] can be broadly classified as types of bicriteria problems. Recently, there has been substantial work on finding efficient approximation algorithms for a variety of bicriteria problems (see [15, 11, 18, 25, 24, 26] and the references cited therein).

Some node upgrading problems have been studied under a simpler model by Paik and Sahni [21]. In their model, upgrading a node causes the delay of each edge incident on that node to be reduced by a given constant factor $\delta < 1$. When both end points of an edge are upgraded, the delay of the edge is reduced by the factor δ^2. It is easy to see that this model is a special case of the model treated in our paper.

Under their model, Paik and Sahni studied the upgrading problem for several performance measures including the maximum delay on an edge and the diameter of the resulting network. They presented NP-hardness results for several problems. Their focus was on polynomial time algorithms for special classes of networks (e.g. trees, series-parallel graphs) rather than on approximation algorithms for NP-hard versions. Our constructions can be modified to show that all the problems considered here remain NP-hard even under the Paik-Sahni model. The approximation results presented here hold under our generalized model.

7. Approximation Algorithm for Unit Cost Node Upgrading

Recall that the problem (Node-Upgrade, Bottleneck, Spanning Tree) consists in finding an upgrading set of minimum cost such that such that after the upgrade the graph has a spanning tree of bottleneck delay at most the given bound D. In [14], we presented a $(2 \ln n, 1)$ approximation algorithm for the general case, when each node has a cost $c(v)$ associated with it and the goal is to minimize the upgrading cost. Here, n is the number of nodes in the input graph.

In this section, we will show how to obtain an improved approximation algorithm for the problem in the case of unit costs on the vertices. We present an algorithm for (U-Node-Upgrade, Bottleneck, Spanning Tree) with a performance guarantee of $(5 + 4 \ln \Delta, 1)$, where Δ denotes the maximum degree of the input graph. For $\Delta \in o(\sqrt{n})$, the result presented here for the unit cost case improves on the performance of our algorithm for the general case. In particular, for the class of bounded-degree graphs, we obtain a constant factor approximation.

We describe the algorithm in two stages. In the first stage, we only consider case when the edge weights can take on two possible values. We then extend this to handle the case when the edges take on three values depending on the subset of adjacent nodes upgraded. In the remainder of this paper, we use the following terminology. We say that an edge is *uncritical* if its delay is less than or equal to the given bound D. An edge is 1-*critical* if its delay is $> D$ and it can be made less than or equal to D by upgrading one of its endpoints. An edge is 2-*critical* if its delay is $> D$ and it can be made less than or equal to D only by upgrading both of its endpoints. Our algorithm also uses the following definition.

Definition 5. Given a graph $G = (V, E)$, an edge weight function d and a number D, the **bottleneck subgraph** of G, denoted by $\text{Bottleneck}(G, d, D)$, consists of all edges e such that $d(e) \leq D$.

7.1. The Case of no **2**-*Critical Edges*

Recall that in the case of (U-NODE-UPGRADE, BOTTLENECK, SPANNING TREE) the objective is to find an upgrading set of *minimum cardinality.* We first consider the case when all edges are either uncritical or 1-critical. Our ALGORITHM UNIT-COST NODE-UPGRADING shown in Figure 2 gives the steps of our heuristic.

The performance of this heuristic is summarized in the following theorem.

THEOREM 3 ALGORITHM UNIT-COST NODE-UPGRADING *is a polynomial time approximation algorithm for* (U-NODE-UPGRADE, BOTTLENECK, SPANNING TREE) *when all edges are either uncritical or* 1-*critical. Given any such instance of* (U-NODE-UPGRADE, BOTTLENECK, SPANNING TREE)*, the algorithm finds an upgrading set S satisfying the condition*

$$|S| \leq 2(1 + \ln \Delta) \cdot \mathrm{OPT} - 1,$$

where Δ is the maximum degree in the input graph G and OPT *is the size of an optimal upgrading set. In particular, the algorithm has a performance guarantee of* $(2 + 2\ln\Delta, 1)$.

The proof of this theorem relies on several lemmas. Step 3 of ALGORITHM UNIT-COST NODE-UPGRADING detects the case when no node needs to be upgraded. Thus, for the remainder of the analysis, we assume that every optimal solution contains at least one vertex. We use S^* to denote an optimal upgrading set and define $\mathrm{OPT} := |S^*|$. Also, let T^* be a corresponding bottleneck spanning tree, that is, a tree which has bottleneck delay at most D after upgrading the vertices in S^*.

LEMMA 1 *Let $\mathcal{C}^*$ be a minimum size set cover for the instance $(Q, \mathcal{F})$ of the* MIN SET COVER *problem constructed in Step 4 of* ALGORITHM UNIT-COST NODE-UPGRADING. *Then $|\mathcal{C}^*| \leq \mathrm{OPT}$.*

Proof: We show that the sets S_v with $v \in S^*$ form a set cover for the instance $(Q, \mathcal{F})$ of MIN SET COVER. This will prove the claim of the lemma.

ALGORITHM UNIT-COST NODE-UPGRADING

• *Input:* A graph $G = (V, E)$, three edge weight functions d, d_m, d_l, and a number D.

1. $G' \leftarrow \text{Bottleneck}(G, d, D)$
2. $C_1, \ldots, C_q \leftarrow$ connected components of G'
3. **if** $q = 1$ **then return** $\emptyset$
4. Construct an instance $(Q, \mathcal{F})$ of MIN SET COVER as follows: Let the ground elements be $Q := \{C_1, \ldots, C_q\}$. For each $v \in V$ define the set

$$S_v := \{\, C_j : v \in C_j \text{ or } v \text{ is adjacent to } C_j \text{ via a 1-critical edge} \,\}$$

   and let $\mathcal{F} := \{\, S_v : v \in V \,\}$.
5. Find a set cover $\mathcal{C} = \{S_{v_1}, \ldots, S_{v_k}\}$ of size at most $1 + \ln \max\{\, |S_v| : S_v \in \mathcal{F} \,\}$ times the minimum size set cover for the instance $(Q, \mathcal{F})$.
6. $S \leftarrow \{v_1, \ldots, v_k\}$
7. $S' \leftarrow \emptyset$
8. **for all** $v \in S$
   (a) **for all** $e \in E$ incident on v
      (i) $d(e) \leftarrow d_m(e)$ /* thus the delay of such an edge will be at most D */
   (b) **endfor**
9. **endfor**
10. $G' = (V, E') \leftarrow \text{Bottleneck}(G, d, D)$
11. $C'_1, \ldots, C'_s \leftarrow$ connected components of G'
12. Construct an auxiliary graph $\hat{G} = (\{C'_1, \ldots, C'_s\}, \hat{E})$ with $(C'_i, C'_j) \in \hat{G}$ if and only if there are vertices $v \in C'_i$ and $w \in C'_j$ such that $(v, w) \in E$.
13. $\hat{T} \leftarrow$ spanning tree of $\hat{G}$
14. **for all** edges $(C'_i, C'_j) \in \hat{T}$
   (a) Choose $v \in C'_i$ and $w \in C'_j$ such that $(v, w) \in E$.
   (b) $S' \leftarrow S' \cup \{v\}$
15. **endfor**
16. **return** $S \cup S'$

Figure 2. Approximation algorithm for unit costs without 2-critical edges.

Consider an arbitrary component $C_j \in Q$. We have to show that C_j is contained in the union of the sets $\bigcup_{v \in S^*} S_v$.

Since $G' = \text{Bottleneck}(G, d, D)$ contains more than one connected component, the tree T^* must contain a 1-critical edge (u, w) with $u \in C_j$ and $w \notin C_j$. Since after the upgrade of the vertices in S^* the tree T^* has bottleneck delay at most D it follows that either u or w must be contained in S^*.

We have $C_j \in S_w$ (because w is connected to C_j via a 1-critical edge) and $C_j \in S_u$ (since u is contained in C_j). Thus, we can conclude that C_j is covered by the sets S_v, $v \in S^*$. □

Lemma 2 *Let S be the set constructed in Step 6. Then the bottleneck graph G' computed in Step 10 has at most $|S|$ connected components.*

Proof: We show that for each node v there is a path in G' to at least one node from S. Assume for the sake of contradiction that this is not the case. Then there is a vertex v in a connected component C'_v of G' where the component G' does not contain any vertex of S. By construction, C'_v must contain one of the connected components, say C, computed in Step 2 *before* the upgrade of the vertices in S. Since the sets S_v corresponding to the vertices $v \in S$ formed a feasible set cover, it follows that C must have either contained a vertex from S or must have been adjacent to a vertex from S via a 1-critical edge. In both cases after the upgrade this vertex from S is contained in the same connected component as the vertices in C. But this means that C'_v must contain at least one vertex from S, and this is a contradiction. □

The proof of the following lemma is just as easy.

Lemma 3 *The set $S \cup S'$ output by* Algorithm Unit-Cost Node-Upgrading *is a valid upgrading set in G.* □

Using the results of Lemma 1, Lemma 2 and Lemma 3, we can now complete the proof of Theorem 3. By Lemma 1 we know that

$$|S^*| = \text{OPT} \geq |\mathcal{C}^*|, \tag{11}$$

where $\mathcal{C}^*$ is an optimal set cover for the instance $(Q, \mathcal{F})$ of Min Set Cover constructed in Step 4. Step 5 of the algorithm, that is, finding an approximation to the optimal set cover, can be done using the well known greedy algorithm [5]. This algorithm will produce a set cover $\mathcal{C}$ of size at most $(1 + \ln \max\{\, |S_v| : v \in V \,\}) \cdot |\mathcal{C}^*|$. Since $|S_v| \leq \Delta$, where Δ is the maximum degree in the graph G and $|S| = |\mathcal{C}|$, this yields

$$|S| = |\mathcal{C}| \leq (1 + \ln \Delta) \cdot |\mathcal{C}^*| \overset{(11)}{\leq} (1 + \ln \Delta) \cdot \text{OPT}. \tag{12}$$

We now address the cardinality of the set S'. Since by Lemma 2 the bottleneck graph G' computed in Step 10 has at most $|S|$ connected components, it follows that the tree $\hat{T}$ computed in Step 13 contains no more than $|S| - 1$ edges. Since in the loop in Steps 14 to 15 for each edge in T_H we add one vertex to S', we have that $|S'| \leq |S| - 1$. Combining this result with (12) we obtain:

$$|S \cup S'| \leq |S| + |S'| \leq 2|S| - 1 \leq 2(1 + \ln \Delta) \cdot \mathrm{OPT} - 1.$$

This completes the proof of Theorem 3. $\square$

The running time of Algorithm Unit-Cost Node-Upgrading can be analyzed as follows. As usual, let $G = (V, E)$ with $n := |V|$ and $m := |E|$. Computing the connected components of G' needs $\mathcal{O}(n + m)$ time. The instance of Min Set Cover can also be constructed in linear time. As shown in [5] the greedy algorithm for Min Set Cover can be implemented to run in time $\mathcal{O}(\sum_{v \in V} |S_v|)$, which is $\mathcal{O}(m)$ in our case. Computing the auxiliary graph in Step 12 can be accomplished in $\mathcal{O}(n+m)$ time. It is straightforward to verify that all of the remaining steps can also be carried out in linear time. We therefore conclude:

Theorem 4 Algorithm Unit-Cost Node-Upgrading *is an approximation algorithm for* (U-Node-Upgrade, Bottleneck, Spanning Tree) *with a performance guarantee of* $(2 + 2 \ln \Delta, 1)$*, when all edges are either uncritical or* 1*-critical. Here,* Δ *is the maximum degree in the input graph* G*. Further, the algorithm can be implemented to run in time* $\mathcal{O}(n + m)$*.* $\square$

7.2. Extension to the General Unit Cost Case

We now extend the result of Theorem 4 to the case where there are also 2-critical edges, that is, edges where *both* endpoints need to be upgraded in order to make the delay of the edge fall below the threshold D.

The basic idea for the extended algorithm is the following. We first compute the edge subgraph G' of G consisting only of the 1-critical and uncritical edges. Notice that all the edges between the different connected components of G' are 2-critical. For each connected component C_i of G' we run Algorithm Unit-Cost Node-Upgrading to obtain an upgrading set which makes C_i contain a spanning tree of delay at most D after the upgrade. In the final step, we find a good upgrading set which reduces the delay of some (2-critical) edges between the different components so that the whole graph becomes connected by edges of delay at most D. The extended heuristic Algorithm Unit-Cost-General Node-Upgrading is displayed in Figure 3.

In the sequel we again use OPT to denote the cardinality of an optimal solution S^*.

Lemma 4 *Let* r *be the number of components of the graph* G' *computed in Step 3. Further, let* U *be the set of vertices constructed in Steps 10 to 11. Then*

$$\mathrm{OPT} \geq r \qquad \text{and} \qquad |U| \leq 2(r-1), \tag{13}$$

Proof: Let T^* be a bottleneck spanning tree in the graph resulting from the upgrade of the vertices in the optimal set S^*. Since there are r connected components

ALGORITHM UNIT-COST-GENERAL NODE-UPGRADING

• *Input:* A graph $G = (V, E)$, three edge weight functions d, d_m, d_l, and a number D.

1. $S \leftarrow \emptyset$
2. $E' \leftarrow \{ e \in E : e \text{ is either uncritical or 1-critical} \}$
3. $G' \leftarrow (V, E')$
4. $C_1, \ldots, C_r \leftarrow$ connected components of G'
5. **for** $i \leftarrow 1, \ldots, r$
   (a) Run ALGORITHM UNIT-COST NODE-UPGRADING on the connected component C_i and add the upgrading set produced by the algorithm to S
6. **endfor**
7. Construct an auxiliary graph $H = (\{C_1, \ldots, C_r\}, E_H)$ which contains an edge $(C_i, C_j) \in E_H$ if and only if there exist vertices $v \in C_i$ and $u \in C_j$ such that $(v, u) \in E$.
   /* Notice that such an edge (v, u) is 2-critical */
8. $T \leftarrow$ spanning tree of H
9. $U \leftarrow \emptyset$
10. **for all** edges $(C_i, C_j) \in T$
    (a) Choose $v \in C_i$ and $u \in C_j$ with $(v, u) \in E$
    (b) $U \leftarrow U \cup \{v, u\}$
11. **endfor**
12. **return** $S \cup U$

Figure 3. Approximation algorithm for node weighted upgrading with unit costs.

$C_1, \ldots, C_r$ in the graph G', T^* contains at least $r-1$ 2-critical edges having endpoints in different components. For each of these edges, both endpoints must belong to OPT, which implies that $|\mathrm{OPT}| \geq r$. This proves the first inequality in (13).

On the other hand, ALGORITHM UNIT-COST-GENERAL NODE-UPGRADING computes a spanning tree T in Step 8 which has $r-1$ edges. For each of the $r-1$ edges of T the algorithm adds two vertices to U. Thus, $|U| \leq 2(r-1)$. □

7.3. Performance Guarantee

We are now ready to establish the performance of the extended approximation algorithm.

LEMMA 5 *The performance of* ALGORITHM UNIT-COST-GENERAL NODE-UPGRADING *as applied to* (U-NODE-UPGRADE, BOTTLENECK, SPANNING TREE) *is* $(5{+}4\ln\Delta, 1)$, *where* Δ *denotes the maximum degree in the graph given in the input.*

Proof: Again, let T^* be a bottleneck spanning tree in the graph resulting from the upgrade of the vertices in the optimal upgrading set S^*. Let OPT_i be the minimum number of nodes which must be upgraded in component C_i to make it contain a spanning tree of bottleneck delay at most D, that is, OPT_i is the optimal solution value of the instance of (U-NODE-UPGRADE, BOTTLENECK, SPANNING TREE) given by the graph $G[C_i]$, the restriction of d, d_m, d_l to the edges of $G[C_i]$ and the bound D. Consider the intersection of S^* with the cluster C_i. We now show that the inequality

$$|S^* \cap C_i| \geq \frac{1}{2}\mathrm{OPT}_i \tag{14}$$

holds. In fact, let $S_i := S^* \cap C_i$. We first prove that, after upgrading the nodes in S_i, each node in $C_i \setminus S_i$ is connected to at least one node from S_i via paths in C_i containing edges of delay no more than D.

Assume that this is not the case. Then there is a node $v \in C_i \setminus S_i$, whose unique path in T^* to any node in S_i contains at least one 2-critical edge. Fix $w \in S_i$ and consider the path $(v = u_0, u_1, \ldots, u_k = w)$ from v to w in T^*. Without loss of generality, we can assume that this path does not contain any node from S_i different from w. Let ℓ be the smallest number such that $(u_\ell, u_{\ell+1})$ is 2-critical. As $v \notin S_i$, we have $\ell \geq 1$. Since after the upgrade T^* contains only edges of delay at most D, we see that both nodes u_ℓ and $u_{\ell+1}$ belong to S_i, contradicting the fact that the path did not contain any other node from S_i other than w.

We have seen that after upgrading the nodes in S_i the cluster C_i restricted to edges of delay at most D contains at most $|S_i| = |S^* \cap C_i|$ connected components. It is now easy to see that upgrading at most one more node from each of these components will make C_i connected by edges of delay at most D. Thus, there exists an upgrading set in C_i of size at most twice the size of S_i. Consequently, the minimum cardinality upgrading set in C_i has also size at most $2|S_i|$, which proves (14).

Since the clusters are disjoint, we can conclude that the size of the optimal upgrading set S^* is bounded from below as follows:

$$\mathrm{OPT} = |S^*| = \sum_{i=1}^{r} |S^* \cap C_i| \overset{(14)}{\geq} \frac{1}{2} \sum_{i=1}^{r} \mathrm{OPT}_i. \tag{15}$$

We know by Theorem 3 that Algorithm Unit-Cost Node-Upgrading finds a solution for each cluster whose value is bounded from above by $2(1+\ln \Delta_i)\cdot\mathrm{OPT}_i - 1$, where Δ_i is the maximum degree in the subgraph of G induced by C_i. Using this fact in conjunction with Lemma 4, we can estimate the cardinality of the solution set S generated by our algorithm:

$$\begin{aligned}
|S| &\leq 2(r-1) + \sum_{i=1}^{r} (2(1+\ln \Delta_i) \cdot \mathrm{OPT}_i - 1) \\
&\leq r - 2 + 2(1+\ln \Delta) \sum_{i=1}^{r} \mathrm{OPT}_i \\
&\overset{(15)}{\leq} r - 2 + 4(1+\ln \Delta) \cdot \mathrm{OPT} \\
&\overset{\text{Lemma 4}}{\leq} (5 + 4\ln \Delta) \cdot \mathrm{OPT} - 2.
\end{aligned}$$

□

7.4. Running Time

The running time of Algorithm Unit-Cost-General Node-Upgrading can be estimated in the following manner. Computing the graph G' and its connected components needs time $\mathcal{O}(n+m)$. Let component C_i have n_i nodes and m_i edges. Then, since Algorithm Unit-Cost Node-Upgrading as applied to component C_i needs $\mathcal{O}(n_i + m_i)$ time and we have $\sum_{i=1}^{r} n_i = n$ and $\sum_{i=1}^{r} m_i \leq m$, we see that the time needed by the **for**-loop in Steps 5 to 6 is $\mathcal{O}(n+m)$. The auxiliary graph H in Step 7 as well as a spanning tree of H can be constructed in linear time. Also, the final **for**-loop can be implemented to use only $\mathcal{O}(n+m)$ time. Hence, the whole Algorithm Unit-Cost-General Node-Upgrading runs in linear time. We summarize the results of this section in the following theorem.

Theorem 5 Algorithm Unit-Cost-General Node-Upgrading *is an approximation algorithm for* (U-Node-Upgrade, Bottleneck, Spanning Tree) *with a performance of* $(5 + 4\ln \Delta, 1)$. *It can be implemented to run in time* $\mathcal{O}(n+m)$. □

8. Upgrading for Constrained Steiner Trees

In this section we investigate network improvement problems where the goal is to find a reduction strategy such that the weight of a constrained Steiner tree in the modified network is as small as possible. The problem (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) studied here is based on the edge upgrading model introduced in Section 2.2. Recall that in this problem we are given additionally d-weights on the edges of the graph. These weights are independent of the ℓ-lengths in the network and cannot be changed by a reduction. The goal becomes to find a reduction and a Steiner tree obeying the diameter constraint which is as light as possible after applying the reduction.

In our notation the problem (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) is a *tricriteria problem*. Consequently, we will be concerned with *tricriteria approximation algorithms* in this section. The objective function of the problem (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) is the length $(\ell - r)(T)$ of the Steiner tree in the modified network. We consider the diameter constraint to be the first constraint, the budget constraint as the second one. Let A be a tricriteria approximation algorithm with performance (α, β, γ) for the problem. If the set (2) of feasible solutions is nonempty, then A must find a reduction of cost at most γB and a tree T of diameter at most βD whose modified length $(\ell - r)(T)$ is at most α times the optimum. If there is no feasible solution, A has the choice of either providing the information that the set (2) is empty or returning a reduction r of cost at most γB and a tree of diameter at most βD.

In addition to graphs, we will also deal with *multigraphs*. Multigraphs are a variation of graphs where more than one edge can join two vertices. We refer to edges in a multigraph as *multiedges*.

We close this subsection by commenting on the hardness of the problem (Total Weight, Diameter, Edge-Upgrade, Steiner Tree). It follows easily from the NP-hardness of the classical Steiner tree problem [8, Problem ND12] that the problem (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) is NP-hard. Moreover, even if one is already given an optimal reduction r^* for one of the problems, it remains hard to find a corresponding optimal tree in the modified network.

The problem (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) contains the problem (Total-Weight, Edge-Upgrade, Graph) studied in [13] as a special case ($K = V$, $d \equiv 0$, $D = 0$). Since this problem has been shown to be NP-hard even on series-parallel graphs, we obtain the following theorem.

Theorem 6 *The problem* (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) *is NP-hard, even when restricted to series-parallel graphs. Unless $\mathsf{P} = \mathsf{NP}$, for any $\alpha, \beta \geq 1$, there is no polynomial time $(\alpha, \beta, 1)$-approximation for the* (Total Weight, Diameter, Edge-Upgrade, Steiner Tree) *problem even when restricted to bipartite graphs. Further, unless $\mathsf{P} = \mathsf{NP}$, for any $\beta, \gamma \geq 1$, there is no polynomial time $(1, \beta, \gamma)$-approximation algorithm for* (Total Weight, Diameter, Edge-Upgrade, Steiner Tree). *All of the above hardness results continue to hold, if the d-weights are all equal to zero.* □

8.1. *Algorithm Outline*

We now discuss our approximation algorithm for (TOTAL WEIGHT, DIAMETER, EDGE-UPGRADE, STEINER TREE). We first present an informal description of the algorithm in order to illustrate the main ideas behind the steps. We then proceed with a more detailed presentation and show the pseudo code for our approximation algorithm.

Our algorithm works roughly as follows: We first modify the graph by replacing each edge with a number of parallel multiedges. In order to establish the performance guarantee, we must show that the modification of the graph preserves the structure of the problem.

The next main step of the algorithm consists of $\mathcal{O}(\log |K|)$ *phases*, where K is the set of terminals. Initially, the solution consists of the empty set. During each phase of the algorithm we choose a set of edges to add to the solution. The set of edges chosen in each iteration is required to possess three desirable properties:

1. The solution cost with respect to the shortened ℓ-cost must be no more than OPT, where OPT denotes the optimally (under a budget B on the cost of improvement) improved total length of a Steiner tree with d-diameter at most D.
2. The diameter value with respect to d must not increase by more than D.
3. The reduction cost spent in each iteration is no more than B.

Since the number of iterations of the algorithm is $\mathcal{O}(\log |K|)$, this will lead to an approximation algorithm which violates the budget and the diameter constraint by a factor of at most $\mathcal{O}(\log |K|)$ and which is within a factor of $\mathcal{O}(\log |K|)$ of the optimal solution. We use a *solution based decomposition method* in the analysis of our algorithm. Its basic idea is to take advantage of the existence of an optimal solution to prove that, in each phase, it is possible to choose a good set of edges. We now refine the specification of our algorithm.

Main Step 1: Discretize the problem by using the transformation procedure shown in ALGORITHM TRANSFORM (Figure 4) This transformation builds a multigraph G' from the original graph G in the following way: Each edge $e = (u, v)$ with length $\ell(e)$ and minimum length $\ell_{\min}(e)$ is replaced by $b_e + 2$ parallel multiedges e^k, $k = -1, 0, \ldots, b_e$. These edges reflect discrete steps in improvement.

The edge e^{-1} corresponds to an "untouched" version of e with length $\ell'(e^{-1}) := \ell(e)$ and cost $c'(e^{-1}) := 0$. For $k \geq 0$, e^k, represents edge e shortened to $\ell(e) - (1+\varepsilon)^k$. Thus e^k has length $\ell'(e^k) := \ell(e) - (1+\varepsilon)^k$ and cost $c'(e^k) := (1+\varepsilon)^k c_e$. The d-weights for all the edges e^k coincide with that of e. To avoid additional notation, we also use d to denote these weights. This way, each multiedge e' in G' is assigned three numbers: $\ell'(e')$, $c'(e')$ and $d(e')$.

Main Step 2: The algorithm maintains a set of connected subgraphs or *clusters*, each with its own distinguished vertex or *center*. Initially, each terminal is in a cluster by itself. In each phase of the second main step, we set up an auxiliary graph G_i using the constrained shortest path algorithm of [11]. We then solve a variant of a constrained matching problem in G_i and, finally, merge the clusters in

ALGORITHM TRANSFORM
- *Input:* A graph $G = (V, E)$, four edge weight functions ℓ, $\ell_{\min}$, c, d, and a number B; a *fixed* accuracy parameter $\varepsilon > 0$

1. $V' \leftarrow V$ /* The vertex set of the graph to be constructed */
2. $E' \leftarrow \emptyset$ /* The edge set of the graph to be constructed */
3. **for all** edges $e = (u, v) \in G$
   (a) $b_e \leftarrow \lceil \log_{(1+\varepsilon)}(\ell(e) - \ell_{\min}(e)) \rceil$
   (b) Add $b_e + 2$ parallel edges e^k, $k = -1, 0, \ldots, b_e$ between u and v to E'
   (c) $\ell'(e^{-1}) \leftarrow \ell(e)$
   (d) $d(e^{-1}) \leftarrow d(e)$
   (e) $c'(e^{-1}) \leftarrow 0$
   (e) **for** $k = 0, \ldots, b_e$
      (i) $\ell'(e^k) \leftarrow \ell(e) - (1+\varepsilon)^k$
      (ii) $d(e^k) \leftarrow d(e)$
      (iii) $c'(e^k) \leftarrow (1+\varepsilon)^k c_e$
   (f) **endfor**
4. **endfor**
5. **return** $G' = (V', E')$, ℓ', d, c'

Figure 4. Procedure to transform G into G' in the first step.

pairs by adding paths between their centers. Since the number of clusters decreases by a factor of 2 due to the merging in each phase, the algorithm terminates in $\lceil \log_2 |K| \rceil$ phases with one cluster.
Main Step 3: We take a tree T' rooted at the center of the single cluster left at the end of the second main step with small diameter under the d-costs.
Main Step 4: In the final step, we use the information from the three values associated with each edge in T' to construct a reduction strategy r and a Steiner tree T for the original graph G. We output r and T.
A complete specification of our Algorithm is shown in Figure 5.

8.2. *Correctness and Performance Guarantee*

To facilitate the presentation, we will assume in the following that r^* is an optimal reduction strategy for the graph G and that T^* is a corresponding optimal Steiner tree of total weight $\mathrm{OPT} := (\ell - r^*)(T^*)$ and diameter $\mathrm{dia}_d(T^*) \leq D$. We will show on page 30 how to modify our algorithm to handle the case when the set of feasible solutions is empty.

8.2.1. Some Basic Lemmas Our algorithm needs as subroutines two approximation algorithms for certain bicriteria problems on graphs. The first is a constrained

ALGORITHM IMPROVE DIAMETER-STEINER-TREE

• *Input:* A graph $G = (V, E)$, a vertex subset K of terminals, four edge weight functions ℓ, $\ell_{\min}$, c, d, and two numbers B and D; a *fixed* accuracy parameter $\varepsilon > 0$.

1. Call ALGORITHM TRANSFORM(ε) to obtain a new (multi-) graph $G' = (V, E')$ with weights $\ell'(e)$, $c'(e)$, $d(e)$ on the edges $e \in E'$.
2. $i \leftarrow 1$ /* Initialize the phase count */
3. $\mathcal{C}_1 \leftarrow \{\, \{v\} : v \in K \,\}$
   /* Initialize the set of clusters $\mathcal{C}_1$ to contain $|K|$ singleton sets, one for each terminal in K */
4. **for all** $v \in K$
   (a) *center*$(\{v\}) \leftarrow v$
5. **endfor** /* For each cluster in $\mathcal{C}_1$, define the single node in the cluster to be its center. */
6. **while** there is more than one cluster in $\mathcal{C}_i$
   (a) Call ALGORITHM MERGE to merge clusters.
   (b) $i \leftarrow i + 1$ /* End of phase i */
7. **endwhile**
8. Let $\hat{C}$, with *center*$(\hat{C}) = \hat{v}$ be the single cluster left.
9. Compute a shortest path tree $\hat{T}$ of $\hat{C}$ rooted at $\hat{v}$ with respect to the d-weights (using only edges in $\hat{C}$).
10. For each multiedge $e^k = (u, v) \in \hat{T}$ define the reduction r on the corresponding edge $e = (u, v)$ in the original graph by

$$r(e) := \begin{cases} 0 & \text{if } k = -1, \\ (1+\varepsilon)^k & \text{if } k \geq 0. \end{cases}$$

    Define a tree T in G by including the corresponding edges e.
11. **return** r and T

Figure 5. Algorithm for diameter bounded Steiner tree improvement.

Algorithm Merge

• *Input:* A graph $G' = (V, E')$, a vertex subset K of terminals, three edge weight functions ℓ', c', d, and two numbers B and D, a set $\mathcal{C}_i = \{C_1 \ldots, C_{k_i}\}$ of clusters; a *fixed* accuracy parameter $\varepsilon > 0$.

1. Let the set of clusters at the beginning of the ith phase be $\mathcal{C}_i = \{C_1 \ldots, C_{k_i}\}$.
2. $s \leftarrow \lceil \log_{(1+\varepsilon)} B \rceil + 1$.
3. $\mathcal{M} \leftarrow \{0, (1+\varepsilon)^0, (1+\varepsilon)^1, \ldots, (1+\varepsilon)^s\}$.
4. Construct a (complete) multigraph $G_i = (V_i, E_i)$ as follows
5. $V_i \leftarrow \{\, v_x : \, v_x$ is the center of cluster $C_x \in \mathcal{C}_i.\}$
6. **for all** pairs (C_x, C_y) of clusters in $\mathcal{C}_i$
   (a) **for all** $B' \in \mathcal{M}$
      (i) Let v_x and v_y be the centers of C_x and C_y, respectively.
      (ii) Let the path $P_{xy}(B')$ be a $(1 + 1/\varepsilon, 1, 1)$-approximation to the restricted shortest path problem in G' between the centers v_x and v_y, with edge weights ℓ', c', d and the bounds B' on the c'-cost and D on the d-length respectively.
      (iii) Include a multiedge (v_x, v_y) in G_i of weight $\ell'(v_x, v_y)$ equal to the ℓ'-cost of $P_{xy}(B')$. The weight $c'(v_x, v_y)$ of the edge is set to that of the c'-cost of $P_{xy}(B')$.
   (b) **endfor**
7. **endfor**
8. Find a $(2, 2)$-approximation to the (Total Weight, Total Weight, Maximum Matching)problem with edge weights ℓ', c' and the bound $(1+\varepsilon)^2 B$ on the c'-weight of the matching.
9. For each edge $e = (v_x, v_y)$ in the matching, merge the clusters C_x and C_y, for which v_x and v_y were centers respectively, by adding the corresponding path P_{xy} to form a new cluster C_{xy}. The node (edge) set of the cluster C_{xy} is defined to be the union of the node (edge) sets of C_x, C_y and the nodes (edges) in P_{xy}. One of v_x and v_y is (arbitrarily) chosen to be the center v_{xy} of the cluster C_{xy}. C_{xy} is added to the cluster set $\mathcal{C}_{i+1}$ for the next phase, while C_x and C_y are removed.

Figure 6. Subroutine to merge clusters.

variant of the shortest path problem and is hence referred to as the *restricted shortest path problem.* The second one is a *constrained minimum weight matching problem.*

Definition 6. [Restricted Shortest Path Problem] The input for the **restricted shortest path problem**, (TOTAL WEIGHT, TOTAL WEIGHT, TOTAL WEIGHT, x-y-PATH), consists of a (multi-) graph $G = (V, E)$, two distinguished vertices $x, y \in V$, three nonnegative edge weight functions d_1, d_2 and d_3 and two numbers D_2 and D_3. The problem is to find a subgraph from

$$\left\{\, P : P \text{ is a path between } x \text{ and } y \text{ in } G,\ \sum_{e\in P} d_2(e) \le D_2 \text{ and } \sum_{e\in P} d_3(e) \le D_3 \,\right\}$$

having minimum value $d_1(P) := \sum_{e\in P} d_1(e)$.

The result of the following lemma is due to Warburton [26], but can also be proved by an extension of Hassin's work [11].

LEMMA 6 ([26, 11]) *For any fixed* $\varepsilon > 0$*, there is an approximation algorithm for* (TOTAL WEIGHT, TOTAL WEIGHT, TOTAL WEIGHT, x-y-PATH) *with a performance of* $(1 + \varepsilon, 1, 1)$. □

As mentioned, we also need results about finding *minimum weight matchings* in graphs. Recall that a matching in a graph $G = (V, E)$ is a subset $M \subseteq E$ of the edges such none of the edges in M are incident. A maximum matching is a matching of maximum cardinality.

Definition 7. [Minimum Weight Matching Problem] An instance of the **minimum weight matching problem**, MIN MATCHING, is given by a graph $G = (V, E)$ and an edge weight function d. The problem is to find a maximum matching $M \subseteq E$ in G having minimum weight $d(M) := \sum_{e\in M} d(e)$.

The minimum weight matching problem can be solved in polynomial time, in fact in time $\mathcal{O}(n^3)$ on a graph with n nodes, by classical algorithms, see e.g. [16, 17]. What we will need for the construction of our approximation algorithm in this section is an approximation algorithm for a bicriteria version of MIN MATCHING. In this problem, which we will denote by (TOTAL WEIGHT, TOTAL WEIGHT, MAXIMUM MATCHING), we are given *two* edge weight functions d_1 and d_2 on the edges of G and the goal is to find a maximum matching M of weight $d_2(M)$ at most a given bound D_2, having minimum weight $d_1(M)$.

An easy reduction from PARTITION shows that (TOTAL WEIGHT, TOTAL WEIGHT, MAXIMUM MATCHING) is NP-hard. On the other hand, the results of Marathe et al. in [18] show that the problem (TOTAL WEIGHT, TOTAL WEIGHT, MAXIMUM MATCHING) has an efficient bicriteria approximation algorithm:

LEMMA 7 ([18]) *There is a* $(2, 2)$*-approximation algorithm*[2] *for* (TOTAL WEIGHT, TOTAL WEIGHT, MAXIMUM MATCHING). □

We now show that the transformation procedure does not destroy important properties with respect to the problem.

LEMMA 8 *For any $\varepsilon > 0$, the graph G' obtained as a result of applying* ALGORITHM TRANSFORM *to G satisfies the following properties:*

1. *There exists a Steiner tree T' in G' of total ℓ'-length at most* OPT, *c'-cost no more than $(1+\varepsilon)B$ and of d-diameter at most D.*
2. *Let $u, v \in K$ be two terminals. Let P be the unique path between u and v in the optimally improved tree T^*. Let $(\ell - r^*)(P)$ denote the length with respect to $\ell - r^*$ and $C_P := \sum_{e \in P} r(e) \cdot c_e$ denote the money spent by r^* on the edges of P respectively. Then, in G' there exists a path between u and v of d-diameter at most D, ℓ'-length at most $(\ell - r^*)(P)$ and c'-cost at most $(1+\varepsilon)C_P$.*

Proof: We only show the first part of the lemma. The second part can be proved similarly. Consider the optimal reduction r^* and a corresponding optimal constrained Steiner tree T^* in G of minimum weight $\mathrm{OPT} := (\ell - r^*)(T^*)$.

We now define a tree T' in G' as follows. For each edge $e = (u, v) \in T^*$ which is reduced by $r^*(e)$ we select a multiedge e^k in G' with the following properties:

$$\begin{aligned} \ell(e) - (1+\varepsilon)r^*(e) &\leq \ell'(e^k) \leq \ell(e) - r^*(e) \\ c_e r^*(e) &\leq c'(e^k) \leq (1+\varepsilon)r^*(e) \\ d(e^k) &= d(e) \end{aligned}$$

The tree T' defined this way has a total ℓ'-length $\ell'(T')$ which is bounded from above by $(\ell - r)(T^*) = \mathrm{OPT}$. Also, the c'-cost of T' is at most $1+\varepsilon$ times the cost of the reduction r and thus does not exceed $(1+\varepsilon)B$. Clearly, the d-diameters of T' and T^* coincide. $\square$

8.2.2. A Bound on the Diameter of the Tree

OBSERVATION 1 *The set of cluster centers maintained by* ALGORITHM IMPROVE DIAMETER-STEINER-TREE *is always a subset of the terminals K of the original graph G.* $\square$

LEMMA 9 ALGORITHM IMPROVE DIAMETER-STEINER-TREE *terminates in $\lceil \log_2 |K| \rceil$ phases.*

Proof: Let k_i denote the number of clusters in phase i. Note that $k_{i+1} = \lceil \frac{k_i}{2} \rceil$ since we pair up the clusters (using a matching in Step 9 of the merging procedure in ALGORITHM MERGE). Hence we are left with one cluster after phase $\lceil \log_2 |K| \rceil$ and the algorithm terminates. $\square$

LEMMA 10 ([25]) *Suppose T is a tree with an even number of marked nodes. Then there is a pairing (v_1, w_1), …, (v_k, w_k) of the marked nodes such that the $v_i - w_i$ paths in T are edge-disjoint.* $\square$

LEMMA 11 *Let $C \in \mathcal{C}_i$ be any cluster in phase i of* ALGORITHM IMPROVE DIAMETER-STEINER-TREE *and let v be its center. Then any node u in C is reachable from v by a path in C of d-weight at most iD.*

Proof: By Observation 1, the set of cluster centers in phase i is a subset of the terminals K. By Lemma 8, there is a Steiner tree in G' of d-diameter at most D and of total c'-cost no more than $(1+\varepsilon)B$. This implies that all the paths $P_{xy}(B')$ which are constructed in Step 6 of ALGORITHM MERGE with the help of Hassin's restricted shortest path algorithm have d-weight at most D.

We now establish the claim of the lemma by induction on i. For $i = 1$ the claim is trivial. Assume now that all the clusters in phase i have the property stated in the claim. Consider the merging of C_x and C_y with centers v_x and v_y to a cluster C_{xy} that then appears in phase $i+1$. Without loss of generality assume that v_x is chosen to be the center of the cluster C_{xy}. By induction hypothesis, for any node u in C_{xy} that was in C_x there is a path from u to v_x of d-weight at most iD. This path is also present in C_{xy}. Consider a node $u \in C_y$. Then, again by induction hypothesis, there is a path of length at most iD from u to v_y in C_y (and thus also in C_{xy}). Since we have seen that the path added in phase i between v_x and v_y is of d-weight at most D, we can connect u to v_x by a path of d-weight at most $(i+1)D$. □

Using the result of the last lemma in conjunction with Lemma 9 we can establish the following lemma.

LEMMA 12 ALGORITHM IMPROVE DIAMETER-STEINER-TREE *outputs a Steiner tree T with d-diameter at most $2\lceil \log_2 |K| \rceil \cdot D$.*

Proof: It follows from Lemmas 11 and 9 that the tree $\hat{T}$ found in Step 9 of ALGORITHM IMPROVE DIAMETER-STEINER-TREE has d-diameter at most $2\lceil \log_2 |K| \rceil \cdot D$. By construction of r and T in Step 10, the d-diameter of T in G is no more than the d-diameter of T' in G'. □

This completes the proof of the performance guarantee with respect to the d-cost. We now proceed to prove the performance guarantee with respect to the modified ℓ-lengths.

8.2.3. The Total Modified Length

LEMMA 13 *Let T' be any minimum ℓ'-cost Steiner tree in G' subject to the constraints that its d-diameter is bounded by D and its total c'-cost does not exceed $(1+\varepsilon)B$. The ℓ'-weight of the constrained minimum weight matching found in Step 9 of* ALGORITHM MERGE *in any phase of* ALGORITHM IMPROVE DIAMETER-STEINER-TREE *is at most $2(1+1/\varepsilon)\ell'(T')$. Also, the c'-cost of the matching is at most $2(1+\varepsilon)^2 \cdot B$.*

Proof: Notice that by Lemma 8, a tree satisfying both the constraints as stated in the claim does exist.

Let i be fixed and consider the ith phase of ALGORITHM IMPROVE DIAMETER-STEINER-TREE. By Observation 1 all the centers $v_1, v_2, \ldots, v_{k_i}$ in the ith stage are terminals. Thus, all these vertices are contained in the Steiner tree T'. We mark the nodes $v_1, v_2, \ldots, v_{2\lfloor \frac{k_i}{2} \rfloor}$ in T' and apply Lemma 10 to T' with these vertices marked.

The lemma states that there is a pairing of the centers such that the paths between the paired vertices in T' are edge disjoint. Without loss of generality assume that this pairing is $(v_1, v_2), \ldots, (v_{2\lfloor \frac{k_i}{2} \rfloor - 1}, v_{2\lfloor \frac{k_i}{2} \rfloor})$.

By the disjointness of these paths, their total ℓ'-cost is at most $\ell'(T')$. Similarly, the total c'-cost of the edges in these paths does not exceed $(1+\varepsilon)B$, and none of the paths has length more than D under the d-weight function. By construction of the paths $P_{xy}(B')$ in Step 6 of Algorithm Merge, for each of the paths P between the centers from above we can find a path P' among the paths $P_{xy}(B')$ in G' with the following properties:

1. The d-length of P' is at most that of P. Thus, the d-length of P' is bounded by D.
2. The total c'-cost of P' is at most $1+\varepsilon$ times the c'-cost of P. The factor $1+\varepsilon$ stems from the discrete budget values, which are used for the constrained paths.
3. The ℓ'-length of P' is at most $1 + 1/\varepsilon$ times that of P. Here, the factor $1 + 1/\varepsilon$ comes from the approximation algorithm for the restricted shortest path problem.

Using the edges in G_i corresponding to this selection from the paths $P_{xy}(B')$ yields a matching of the vertices $(v_{2\lfloor \frac{k_i}{2} \rfloor - 1}, v_{2\lfloor \frac{k_i}{2} \rfloor})$ in G_i of ℓ'-length at most $(1+1/\varepsilon)\ell'(T')$ and of c'-cost at most $(1+\varepsilon)c'(T') \leq (1+\varepsilon)^2 \cdot B$.

In Step 8 of Algorithm Merge, a $(2,2)$-approximation for the (Total Weight, Total Weight, Maximum Matching)problem is found. This matching M must have c'-cost at most $2(1+\varepsilon)^2 \cdot B$ and ℓ'-weight at most $2(1+1/\varepsilon)\ell'(T')$. This completes the proof. □

We are now ready to estimate the ℓ'-length of the Steiner tree T.

Lemma 14 Algorithm Improve Diameter-Steiner-Tree *outputs a Steiner tree T with total length $(\ell - r)(T)$ under the modified length function $\ell - r$ of at most* $2(1+1/\varepsilon)\lceil \log_2 |K| \rceil \cdot \mathrm{OPT}$.

Proof: By the results of Lemma 9 and Lemma 13, the total ℓ'-length of the edges in the single cluster $\hat{C}$ is at most $2(1+1/\varepsilon)\lceil \log_2 |K| \rceil \cdot \ell'(T')$, where T' is an optimal constrained Steiner tree in G'. Thus, the total ℓ'-length of the shortest path tree $\hat{T}$, which is computed in Step 9 of Algorithm Improve Diameter-Steiner-Tree is also no more than $2(1+1/\varepsilon)\lceil \log_2 |K| \rceil \cdot \ell'(T')$.

By Lemma 8, $\ell'(T') \leq \mathrm{OPT}$ and hence

$$\ell'(\hat{T}) \leq 2(1+1/\varepsilon)\lceil \log_2 |K| \rceil \cdot \mathrm{OPT}. \tag{16}$$

By the construction of the reduction r and the tree T in the final Step 10 of Algorithm Improve Diameter-Steiner-Tree, the length of T under the modified length function $\ell - r$ is bounded from above by $\ell'(\hat{T})$. Now using (16) establishes the lemma. □

8.2.4. Bounding the Cost of the Reduction Strategy We now address the c'-cost of the edges that are added during an arbitrary phase of the algorithm. From Lemma 13 and Lemma 9 we get:

LEMMA 15 *The total c'-cost of the tree $\hat{T}$ found in Step 9 of* ALGORITHM IMPROVE DIAMETER-STEINER-TREE *is at most* $2(1+\varepsilon)^2\lceil\log_2|K|\rceil \cdot B$*. Moreover, the cost of the reduction r constructed in Step 10 is also bounded by* $2(1+\varepsilon)^2\lceil\log_2|K|\rceil \cdot B$. □

8.2.5. Handling Infeasibility of an Instance We have already all the main ingredients to establish the performance of our ALGORITHM IMPROVE DIAMETER-STEINER-TREE. As long as there exists a feasible solution, the algorithm will find a reduction r of cost $\mathcal{O}(\log|K|) \cdot B$ and a tree T of d-diameter $\mathcal{O}(\log|K|) \cdot D$ such that $(\ell - r)(T) \leq \mathcal{O}(\log|K|) \cdot \mathrm{OPT}$.

To make our algorithm have indeed a $(\mathcal{O}(\log|K|), \mathcal{O}(\log|K|), \mathcal{O}(\log|K|))$ performance, we must ensure that, if the algorithm outputs a solution, this solution will violate the constraints on the d-diameter and on the cost of the reduction by at most logarithmic factors. We have proved that if there is a feasible solution, then the multicriteria approximation algorithms for the restricted shortest path problems and the bicriteria matching problem will not stop and inform us about the infeasibility of the instances of these problems constructed during the run of our algorithm. Thus, if one of these two subroutines detects infeasibility of a subproblem, we can also stop our algorithm and inform about the infeasibility of the instance of (TOTAL WEIGHT, DEGREE, EDGE-UPGRADE, STEINER TREE)to be handled. Finally, by testing the d-diameter of the tree and the cost of the reduction produced by our algorithm and outputting these only if the constraints are violated by at most the logarithmic factors we make sure that we will always output an "almost feasible" solution as stated in Definition 4.

8.2.6. Combining all Ingredients One can verify that ALGORITHM IMPROVE DIAMETER-STEINER-TREE can be implemented to run in polynomial time. Thus, combining the results of Lemmas 12, 14 and 15, we obtain the following theorem.

THEOREM 7 *For any fixed $\varepsilon > 0$,* ALGORITHM IMPROVE DIAMETER-STEINER-TREE *is an approximation algorithm for* (TOTAL WEIGHT, DIAMETER, EDGE-UPGRADE, STEINER TREE)*with a performance of $(\alpha_\varepsilon, \beta_\varepsilon, \gamma_\varepsilon)$, where*

$$\begin{aligned} \alpha_\varepsilon &= 2(1+1/\varepsilon)\lceil\log_2|K|\rceil, \\ \beta_\varepsilon &= 2\lceil\log_2|K|\rceil \quad \text{and} \\ \gamma_\varepsilon &= 2(1+\varepsilon)^2\lceil\log_2|K|\rceil. \end{aligned}$$

Here, K is the set of terminals given in the input. □

9. Upgrading the Whole Graph

In the previous sections, we were searching for minimum cost node- or edge-upgrading strategies such that in the upgraded network the cost of a subgraph is minimized. It is natural to consider upgrading strategies so as to minimize the cost of the whole graph under some measure. For instance, it is meaningful to ask for a minimum cost upgrading set such that after the upgrade, *all* the edges in the network are of delay at most D. This problem has been called the *link delay problem* [21]. Following our terminology we call this the (Node-Upgrade, Bottleneck, Graph). In this section, we will discuss several such upgrade problems.

9.1. Upgrading under the Edge Based Model

Consider the case of rational edge upgrade functions for the (Edge-Upgrade, Total-Weight, Graph), where the cost functions c_e are all linear. For any graph G, it can be verified that the greedy-strategy that successively reduces a cheapest available edge until the weight of the graph has fallen below the given threshold on the total weight is an optimal reduction strategy. Thus this problem is solvable in polynomial time.

Now consider the bottleneck version of the problem, namely problem (Edge-Upgrade, Bottleneck, Graph), where we specify a bound on the bottleneck cost of the graph. The problem then is quite easy to solve. We first isolate all edges that have a cost greater than the specified bound. We then simply have to reduce these edges to get their values below the specified bound. Thus the bottleneck version is solvable in linear time for general graphs.

It should be noted that the version where we need to upgrade the graph so that the modified graph has a spanning tree of bottleneck cost at most a given bound D, that is, problem (Edge-Upgrade, Bottleneck, Spanning Tree), can also be solved efficiently. To do this, we set up a minimum cost spanning tree problem in the following manner. For each edge whose weight is at most D, the cost is set to zero; for each edge e whose weight is greater than D, the cost of the edge is set to $c_e(D - \ell(e))$. An edge set of minimum upgrading cost can be identified from a minimum cost spanning tree for the resulting graph.

9.2. Upgrading under the Node Based Model

Paik and Sahni showed in [21] that (Node-Upgrade, Bottleneck, Graph) is NP-hard by providing a reduction from Vertex Cover, which is the decision version of Min Vertex Cover. The reduction was done in their node upgrading model mentioned in Section 6. Since this model is a special case of our model, we immediately obtain the hardness of (Node-Upgrade, Bottleneck, Graph) in our setting. We now present a simple polynomial time approximation algorithm for (Node-Upgrade, Bottleneck, Graph) with a performance of $(2, 1)$. Its basic outline is as follows. (This outline uses some terminology introduced in Section 7.) The upgrading set S is initialized to be the empty set. If there are *infeasible*

Algorithm Approximate Linkdelay

- *Input:* A graph $G = (V, E)$, three edge weight functions d, d_m, d_l, a node weight function c, and a number D

1. **for all** $e \in E$
   (a) **if** e is a "useless edge", i.e., $d_l(e) > D$, **then return** "The link delay cannot be reduced to be at most D".
2. **endfor**
3. $E_2 \leftarrow \{\, e \in E : \ e \text{ is 2-critical} \,\}$
4. $S \leftarrow \{\, u, v : (u, v) \in E_2 \,\}$
5. **for all** $e \in E - E_2$ such that e is incident on some node in S
   (a) $d(e) \leftarrow d_m(e)$ /* one endpoint of e is already upgraded */
6. **endfor**
7. $E' := \{\, e \in E - E_2 : d(e) > D \,\}$
8. Compute a 2-approximation C to the problem of finding a minimum total cost vertex cover in the graph $G' = (V, E')$.
9. **return** $S \cup C$

Figure 7. Approximation algorithm for the Link Delay Problem.

edges (i.e., edges where upgrading cannot reduce the delay to be at most D), the algorithm terminates after reporting that the instance does not have a solution. So, we may assume that there are no infeasible edges in the instance. Then, for each 2-critical edge both endpoints are added to the upgrading set S. It follows that up to this point S is a subset of the optimal solution S^*. We update the delays of the edges incident on the nodes in S. Observe that for each of the edges which still has delay strictly greater than the bound D it suffices to upgrade *at most one* of its endpoints. Our algorithm finds an approximation C to the problem of finding a minimum cost vertex cover in the edge subgraph of G consisting of those edges whose delay still exceeds D. We output $S \cup C$ as an upgrading set.

The heuristic is shown in Algorithm 7. Step 8, that is, computing an approximate minimum cost vertex cover can be done using known approximation algorithms, see e.g. [8]. The following theorem states our approximation result:

Theorem 8 *Algorithm 7 is an approximation algorithm for* (Node-Upgrade, Bottleneck, Graph) *with a performance guarantee of* $(2, 1)$.

Proof: Let S^* be an upgrading set of minimum cost. Clearly, the set S in Step 4 of Algorithm 7 must be contained in S^*. It is also easy to see that $S^* \setminus V'$ must be a vertex cover in the graph G' as defined in Step 8 of the algorithm. Thus, the cost of the nodes in $S^* \setminus V'$ is at least that of the optimum vertex cover in G'. The proof can now be completed in a straightforward manner. $\square$

The proof of Theorem 8 shows also that, if we can compute a minimum cost vertex cover in G' efficiently, i.e., in polynomial time, then this result immediately carries over to (Node-Upgrade, Bottleneck, Graph). Bern et al. have shown [4] that the minimum cost vertex cover problem can be solved in polynomial time on graphs of bounded treewidth. It is easy to see that the treewidth of graph G' obtained in Algorithm 7 is no more than the treewidth of the original graph G. Thus, we can conclude:

Corollary 1 *The restriction of* (Node-Upgrade, Bottleneck, Graph) *to graphs of bounded treewidth can be solved in polynomial time.* □

We now address the hardness of (Node-Upgrade, Bottleneck, Graph). As already mentioned, the NP-hardness of this problem was established in [21]. We briefly argue that in fact one can find an L-reduction (see [22]) from Min Vertex Cover to (Bottleneck, Node-Upgrade, Graph) to show that the latter problem is MaxSNP-hard. Given an instance I of Min Vertex Cover given by a graph $G = (V, E)$ with $|V| =: n$ we construct an instance I' of (Node-Upgrade, Bottleneck, Graph) as follows: We use the same graph $G = (V, E)$, set $c(v) = 1$ for all $v \in V$, $d(e) = d_m(e) = f(n) + 1$ and $d_l(e) = 1$ for all $e \in E$, where f is any polynomial time computable function. Finally, let the bound D be 1.

Any vertex cover in G will give us a feasible upgrading set for the instance I' of (Node-Upgrade, Bottleneck, Graph). Thus, an optimum upgrading set has cardinality at most that of the minimum vertex cover: $\mathrm{OPT}(I') \leq \mathrm{OPT}(I)$. It is also straightforward to see that any feasible upgrading set for I' will form a vertex cover in G. This way, we have found a L-reduction from Min Vertex Cover to (Bottleneck, Node-Upgrade, Graph). Moreover, this L-reduction shows us that the existence of an $(\alpha, f(n))$-approximation algorithm for (Node-Upgrade, Bottleneck, Graph) implies that there is an α-approximation algorithm for Min Vertex Cover. Here, f is an arbitrary polynomial time computable function.

As shown in [22] the Min Vertex Cover problem is MaxSNP-hard.[3] Thus, the above arguments show that (Node-Upgrade, Bottleneck, Graph) is also MaxSNP-hard and by the results from [1] there can be no Ptas for (Node-Upgrade, Bottleneck, Graph), unless P = NP.

In [3] it is shown that, unless P = NP, Min Vertex Cover cannot be approximated to within a factor of $\alpha < 1.038$. Combining this result with our reduction above, it follows that a similar non-approximability result holds for the (Node-Upgrade, Bottleneck, Graph) problem. Further, it can be verified that the above reduction also applies to the (Node-Upgrade, Total-Weight, Graph) problem and so we obtain a non-approximability result for that problem as well. Thus, we conclude:

Theorem 9 *Unless* P = NP, *for any* $\alpha < 1.038$ *and any polynomial time computable function* f*, there are no* $(f(n), \alpha)$*-approximation algorithms for* (Node-Upgrade, Bottleneck, Graph) *and* (Node-Upgrade, Total-Weight, Graph) *problems.* □

Acknowledgments: We thank Ravi Sundaram (Delta Trading Inc), Cynthia Philips (Sandia National Laboratories), Hans-Christoph Wirth and Kai-Uwe Drangmeister (University of Würzburg) for several interesting discussions and suggestions related to the results presented in this paper. We also thank Jan Plesnik, Greg Frederickson and Roberto Solis-Oba for making available copies of their papers.

Madhav Marathe's research was supported by the Department of Energy under Contract W-7405-ENG-36. R. Ravi's research was supported by NSF CAREER Award CCR-9625297.

A preliminary version of the paper was presented at the *DIMACS workshop on Network Design: Connectivity and Facilities Location*, May 1997.

Notes

1. Any reduction strategy will incur a minimum cost of $\sum_{e\in E} c_e(0)$ and we can subtract this sum from the budget in advance.
2. Actually, reference [18] contains the stronger result that for any $\varepsilon > 0$ there is a $(1+1/\varepsilon, 1+\varepsilon)$-approximation algorithm. For our purposes, it suffices to use the slightly weaker result.
3. The problem is MaxSNP-complete for graphs of bounded degree.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science (FOCS'92), October 1992, pp. 13–23.
2. O. Berman, *Improving the location of minisum facilities through network modification*, Annals of Operations Research **40** (1992), pp. 1–16.
3. M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs and non-approximability — towards tight results*, Proceedings of the 36th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'95), 1995, pp. 422–431.
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. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Co., Cambridge, MA, 1990.
6. J. Edmonds and E. L. Johnson, *Matching, Euler tours and the Chinese postman*, Mathematical Programming **5** (1973), 88–124.
7. G. N. Frederickson and R. Solis-Oba, *Increasing the weight of minimum spanning tree*, Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96), January 1996, pp. 539–546.
8. M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.
9. M. W. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM Journal on Computing, **24** (1995), pp. 296–317.
10. M. M. Halldórsson, *Approximating the minimum maximal independence number*, Information Processing Letters **46** (1993), 169–172.
11. R. Hassin, *Approximation schemes for the restricted shortest path problem*, Mathematics of Operations Research, **17** (1992), pp. 36–42.
12. S. E. Hambrusch and H. Y. Tu, *Edge Weight Reduction Problems in Acyclic graphs*, J. Algorithms, **24** (1997), pp. 66–93.
13. S. O. Krumke, H. Noltemeier, S. S. Ravi, M. V. Marathe, and K. U. Drangmeister, *Modifying Edges of a Network to Obtain Short Subgraphs*, Accepted for publication in Theoretical Computer Science. (An extended abstract of this paper titled *Modifying networks to obtain low cost trees* appeared in Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science, Cadenabbia, Italy, Lecture Notes in Computer Science, June 1996, pp. 293–307.)

14. S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, S. S. Ravi, R. Sundaram, and H. C. Wirth, *Improving Spanning Trees by Upgrading Nodes*, Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97), Bologna, Italy, July 1997, pp. 281–291.
15. D. Karger and S. Plotkin, *Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows*, Proceedings of the 27th Annual ACM Symposium on the Theory of Computing (STOC'95), May 1995, pp. 18–25.
16. E. L. Lawler, *Combinatorial optimization: Networks and matroids*, Holt, Rinehart and Winston, 1976.
17. L. Lovász and M. D. Plummer, *Matching theory*, Akadémiai Kiadó, Budapest (1986), 1986.
18. M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, *Bicriteria network design problems*, Proceedings of the 22nd International Colloquium on Automata, Languages and Programming (ICALP'95), July 1995, pp. 487–498.
19. C. Phillips, *The network inhibition problem*, Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC'93), May 1993, pp. 288–293.
20. J. Plesnik, *The complexity of designing a network with minimum diameter*, Networks **11** (1981), pp. 77–85.
21. D. Paik and S. Sahni, *Network upgrading problems*, Networks **26** (1995), pp. 45–58.
22. C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, Journal of computer and system sciences **43** (1991), pp. 425–440.
23. R. Ravi, *Steiner trees and beyond*, Ph.D. thesis, Brown University, Providence, Rhode Island 02912, USA, 1993.
24. R. Ravi, *Rapid rumor ramification*, Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science (FOCS'94), November 1994, pp. 202–213.
25. R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, *Many birds with one stone: Multi-objective approximation algorithms*, Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC'93), May 1993, pp. 438–447.
26. A. Warburton, *Approximation of pareto optima in multiple-objective shortest path problems*, Operations Research **35** (1992), pp. 70–79.