Node Weighted Network Upgrade Problems
(Extended Abstract)

S.O. Krumke [1] H. Noltemeier [1] M.V. Marathe [2] S.S. Ravi [3]

January 22, 1997

Abstract

Several problems arising in the areas of communication networks and VLSI design can be expressed in the following general form: Enhance the performance of an underlying network by upgrading some of its nodes. We investigate one such problem.

Consider a network where nodes represent processors and edges represent bidirectional communication links. The processor at a node v can be upgraded at an expense of $\mathrm{cost}(v)$. Such an upgrade reduces the delay of each link emanating from v by a fixed factor x, where $0 < x < 1$. The goal is to find a minimum cost set of nodes to be upgraded so that the resulting network has a spanning tree in which each edge is of delay at most a given value δ.

We provide both hardness and approximation results for the problem. We show that the problem is NP-hard and cannot be approximated within any factor $\beta < \ln n$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\log\log n})$, where n is the number of nodes in the network. This result holds even when the cost of upgrading each node is 1. We then present the first polynomial time approximation algorithms for the problem. For the general case, our approximation algorithm comes within a factor of $2\ln n$ of the minimum upgrading cost. When the cost of upgrading each node is 1, we present an approximation algorithm with a performance guarantee of $4(2 + \ln\Delta)$, where Δ is the maximum node degree. For $\Delta = o(\sqrt{n})$, this algorithm performs better than the general algorithm. In particular, for graphs of bounded node degree, we obtain a constant factor approximation. Finally, we present a polynomial time algorithm for the class of treewidth-bounded graphs.

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

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

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

1 Introduction

Several problems arising in areas such as communication networks and VLSI design can be expressed in the following general form: Enhance the performance of an underlying network by carrying out upgrades at some nodes of the network [24]. In communication networks, upgrading a node corresponds to installing faster communication equipment at that node. Such an upgrade reduces the communication delay along each edge emanating from the node. In signal flow networks used in VLSI design, upgrading a node corresponds to replacing a circuit module at the node by a functionally equivalent module containing suitable drivers. Such an upgrade decreases the signal transmission delay along the wires connected to the module. Paik and Sahni [24] investigate several node upgrading problems. They present complexity results for general networks and efficient algorithms for special classes of networks.

In this paper we investigate the complexity and approximability of the following upgrading problem: We are given a network where nodes represent processors and edges represent bidirectional communication links. The processor at a node v can be upgraded at an expense of $\mathrm{cost}(v)$. Such an upgrade reduces the delay of each edge incident on v by a fixed factor x, where $0 < x < 1$. The goal is to find a minimum cost set of nodes to be upgraded so that the resulting network has a spanning tree in which every edge is of delay at most a given value δ.

We observe that the above problem is NP-hard even when the cost of upgrading each node is 1. We present the first polynomial time approximation algorithm for the general version of the problem. We also present an improved approximation algorithm for the case where the cost of upgrading each vertex is 1. Further, we establish lower bound results that match the performance guarantees provided by our approximation algorithms to within constant factors. Finally, we develop an efficient algorithm for the general version of the problem for the class of treewidth-bounded graphs.

The paper is organized as follows. Section 2 formally defines the problems under study and briefly summarizes our results. In Section 3 we present our approximation algorithm for the general case and establish its performance guarantee. In Section 4 we show the polynomial time solvability of the problems when restricted to the class of treewidth-bounded graphs. Section 5 contains the hardness results.

2 Our Contributions

2.1 Problem Formulation

Let $G = (V, E)$ be a connected undirected graph. With each edge $e \in E$, we associate a nonnegative number $d(e)$, which represents the *delay* of the link e. When the processor at a node v is replaced by a faster processor, the delay of each edge incident on v decreases by a fixed factor x, where $0 < x < 1$. Thus, if $e = (v, u)$ is an edge, its delay after upgrading exactly one of v and u is $xd(e)$; the delay of e falls to $x^2 d(e)$, if both v and u are upgraded. The cost of upgrading a node v is denoted by $\mathrm{cost}(v)$. For a subset V' of V, the cost of upgrading all the nodes in V', denoted by $\mathrm{cost}(V')$, is equal to $\sum_{v \in V'} \mathrm{cost}(v)$.

Let $T = (V, E_T)$ be a spanning tree of G. We denote by $\max(T)$ its *bottleneck-delay*, that is $\max(T) := \max\{d(e) : e \in E_T\}$.

Definition 2.1 Given a node and edge weighted graph G as above and a bound δ, the upgrading Bottleneck Spanning Tree Problem (BSTP) is to upgrade a set $V' \subseteq V$ of nodes such that the resulting graph has a spanning tree of bottleneck delay at most δ and $\mathrm{cost}(V')$ is minimized.

Problem	Hardness Result	Approximation Result
BSTP	Not approximable within $\beta < \ln n$ unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\log\log n})$.	Approximable within $2\ln n$.
UBSTP	Same as BSTP above.	Approximable within $4(2+\ln\Delta)$, where Δ is the maximum node degree.

Both problems are solvable in polynomial time on treewidth-bounded graphs.

Table 1: Summary of the results obtained in this paper.

By UBSTP we denote the subset of instances of BSTP, where the cost of upgrading each node is 1; i.e. $\mathrm{cost}(v) = 1$ for all $v \in V$. Thus, the goal is to find an upgrading set of minimum cardinality.

2.2 Summary of Results

For the first time in the literature, we study the complexity and approximability of the problems UBSTP and BSTP. We show that UBSTP (and thus also BSTP) is NP-hard for any fixed $0 < x < 1$ and $\delta > 0$, even for bipartite graphs. Given the hardness of finding optimal solutions, we concentrate on devising approximation algorithms with good performance guarantees. Recall that an approximation algorithm for a minimization problem Π provides a *performance guarantee* of β if for every instance I of Π, the solution value returned by the approximation algorithm is within a factor $\beta \geq 1$ of the optimal value for I. The main result of this paper is the following:

Theorem 2.2 There is a polynomial time approximation algorithm for BSTP with a performance guarantee of $2\ln n$.

We counterbalance this approximation result with the following lower bound result:

Theorem 2.3 Unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\log\log n})$, there can be no polynomial time approximation algorithm for UBSTP with a performance guarantee of $\beta < \ln n$.

Thus, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{\log\log n})$, the performance guarantees provided by our algorithm is within a factor of two of the best possible performance guarantee for the problem. For the case when all upgrading costs are 1, we give an algorithm with a performance guarantee of $4(2+\ln\Delta)$, where Δ is the maximum node degree in the graph. For $\Delta = o(\sqrt{n})$, this algorithm performs better than the general algorithm. In particular, for the class of bounded-degree graphs, we obtain a constant factor approximation. The algorithm and the corresponding proofs can be found in the appendix. Finally, we present a polynomial time algorithm for the class of treewidth-bounded graphs. The results obtained in this paper are summarized in Table 1.

2.3 Related Work

To the best of our knowledge, the problems considered in this paper have not been previously studied. As mentioned earlier, the node upgrading model used in this paper was introduced in a recent paper by Paik and Sahni [24].

Frederickson and Solis-Oba [11] considered the problem of increasing the weight of the minimum spanning tree in a graph subject to a budget constraint where the cost functions are assumed to be linear in the weight increase. In contrast to the work presented here, they showed that the problem is solvable in strongly polynomial time. Berman [3] considers the problem of shortening edges in a given tree to minimize its shortest path tree weight and shows that the problem can be solved in polynomial time by a greedy algorithm. Phillips [25] studies the problem of finding an optimal strategy for reducing the capacity of the network so that the residual capacity in the modified network is minimized. Reference [20] considers network improvement problems under a different model where there are cost functions associated with improving edge weights.

3 An Approximation Algorithm for BSTP

In this section, we present our approximation algorithm for BSTP. This algorithm provides a performance guarantee of $2 \ln n$. We can assume without loss of generality that all the delays on the edges of the given network are taken from the three element set $\{\delta/x^2, \delta/x, \delta\}$. If the delay of an edge is greater than δ/x^2, then vertex upgrading cannot reduce its delay value to δ. Thus, in the sequel we will assume that the delay of each edge is one of the three above values.

Overview

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

The algorithm iteratively merges clusters until only one cluster remains. To this end, in each iteration it determines a node v of minimum *quotient cost*. The quotient cost of a node v is the ratio whose numerator is the cost of v plus the costs of some nodes adjacent to v in different clusters via edges of delay δ/x^2, and whose denominator is the number of clusters which have nodes adjacent to v. (A precise definition of the quotient cost appears in Figure 1 on the next page.) This quotient cost measures the average upgrading cost of v and the vertices that are adjacent to v through edges of delay δ/x^2. The algorithm then adds v and the nodes mentioned above to the solution set S and merges the corresponding clusters.

The algorithm is shown in Figure 1 on the following page. Step 6 can be implemented in polynomial time as shown by Klein and Ravi in [16]. For each node v, define the *quotient cost* of v to be the minimum value of the ratio in Step 6 achieved by this node. To find the quotient cost of v, we can order the components in $\mathcal{C}$ as $C_1, C_2, C_3, \ldots$ in nondecreasing order of $c(v, C_j)$. In computing the quotient cost of v, it is sufficient to consider subsets of $\mathcal{C}$ of the form $\{C_1, C_2, \ldots, C_j\}$ where $2 \leq j \leq q$. Thus the quotient cost for a given vertex can be computed in polynomial time; by computing the quotient cost for each vertex, we can determine the minimum quotient cost, and thus carry out Step 6 in polynomial time.

It is easy to see that the set S output by algorithm Heuristic-BSTP is indeed a valid upgrading set, since all the edges added to F in Step 9 will be of delay at most δ after upgrading the nodes in S.

In the sequel, we use V^* to denote an optimal upgrading set; i.e. an upgrading set of minimal cost $\text{OPT} := \text{cost}(V^*)$. We now proceed to prove the following theorem which indicates the performance guarantee provided by the algorithm.

Theorem 3.1 Let Heu denote the cost of the nodes chosen by Heuristic-BSTP for upgrading. Then $\text{Heu}/\text{OPT} \leq 2 \ln n$.

1 Heuristic-BSTP

2 Let $G' := \text{bottleneck}(G, d, \delta)$ and let $C_1, \ldots, C_\ell$ be the connected components of G'.

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

4 Repeat while we have more than one component

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

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

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

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

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

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

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

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

10 Output S as the solution.

Figure 1: The approximation algorithm for BSTP.

Our proof of Theorem 3.1 relies on several lemmas, which are presented below. We estimate the cost of the nodes added by the heuristic in each iteration by first establishing an averaging lemma and then using a potential function argument. The notion of a *claw decomposition* which is introduced in the next subsection will be a crucial tool in the analysis.

Claw Decompositions

We employ the notion of *claw decompositions* in showing that the quotient cost of each node chosen in Step 6 is small compared to the optimal solution.

Definition 3.2 A *claw* is either a single node or a $K_{1,r}$ graph for some $r \geq 1$. If there are at most two nodes in the claw then we can choose any of the nodes as its *center*. Otherwise, the node with degree greater than 1 is the unique *center*. The vertices in the claw different from the center are said to be the *fingers* of the claw. A claw with at least two nodes is called a *non-trivial* claw.

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

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

Theorem 3.3 Let G be a connected graph with node set V, where $|V| \geq 2$. Then there is a claw decomposition of V in G.

An Averaging Lemma

Lemma 3.4 Let v be a node chosen in Step 6 and let C denote the total cost of the nodes added to the solution set S in this iteration. Let there be q clusters before v is chosen and assume that in this iteration r clusters are merged. Then

$$\frac{C}{r} \leq \frac{\mathrm{OPT}}{q}.$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

This proves the claim. □

A Potential Function Argument

We are now ready to complete the proof of Theorem 3.1. Assume that the algorithm uses f iterations of the loop and denote by $v_1, \dots v_f$ the vertices chosen in Step 6 of the algorithm.

Let ϕ_j denote the number of clusters *after* choosing vertex v_j in this iteration. Thus, for instance, $\phi_0 = q$, the number of components at the beginning of this iteration in (S, F). Let the number of clusters merged using vertex v_j be r_j and the total cost of the vertices added in that iteration be c_j. Then we have

$$\phi_j = \phi_{j-1} - (r_j - 1). \tag{6}$$

Notice that since $r_j \geq 2$, we have $r_j - 1 \geq \frac{1}{2} r_j$. Using this inequality in (6) we obtain

$$\phi_j \leq \phi_{j-1} - \frac{1}{2} r_j. \tag{7}$$

Observe that $\phi_j \geq 2$ for $j = 0, \dots, f-1$, since the algorithm does not stop before the f-th iteration. Notice also that $\phi_f \geq 1$. Then by Lemma 3.4, we have

$$r_j \geq \frac{c_j \phi_{j-1}}{\mathrm{OPT}} \tag{8}$$

for all $0 \leq j \leq f$. We now use an analysis technique due to Leighton and Rao [22] to complete the proof as in [16]. Substituting equation (8) into (7) yields

$$\phi_j \leq \phi_{j-1} - \frac{1}{2} \cdot \frac{c_j \phi_{j-1}}{\mathrm{OPT}} = \phi_{j-1} \cdot \left(1 - \frac{c_j}{2 \cdot \mathrm{OPT}}\right). \tag{9}$$

Using the recurrence (9), we obtain

$$\phi_f \leq \phi_0 \prod_{j=1}^{f} \left(1 - \frac{c_j}{2 \cdot \mathrm{OPT}}\right). \tag{10}$$

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

$$2 \cdot \mathrm{OPT} \cdot \ln(\frac{\phi_0}{\phi_f}) \geq \sum_{j=1}^{f} c_j. \tag{11}$$

Notice that by Lemma 3.4 we have

$$c_j \leq \mathrm{OPT} \cdot \frac{r_j}{\phi_{j-1}} \leq \mathrm{OPT} < 2 \cdot \mathrm{OPT},$$

and so the logarithms of all the terms in the product of (10) are well defined.

Note also that $\phi_0 \leq n$ and $\phi_f = 1$ and hence from (11) we get

$$\sum_{j=1}^{f} c_j \leq 2 \cdot \mathrm{OPT} \cdot \ln(n). \tag{12}$$

Notice that the total cost of the nodes chosen by the algorithm is exactly the sum $\sum_{j=1}^{f} c_j$. This completes the proof of Theorem 3.1. □

4 Treewidth-Bounded Graphs

In this section we will show that BSTP can be solved in polynomial time, if the underlying graph belongs to the class of treewidth-bounded graphs.

Treewidth-bounded graphs were introduced by Robertson and Seymour [26]. Independently, Bern, Lawler and Wong [4] introduced the notion of decomposable graphs. Later, it was shown [2] that the class of decomposable graphs and the class of treewidth-bounded graphs coincide. A class of treewidth-bounded graphs can be specified using a finite number of primitive graphs and a finite collection of binary composition rules. We use this characterization for proving our results. A class of treewidth-bounded graphs Γ is inductively defined as follows [4]:

1. The number of primitive graphs in Γ is finite.
2. Each graph in Γ has an ordered set of special nodes called *terminals*. The number of terminals in each graph is bounded by a constant, say k.
3. There is a finite collection of binary composition rules that operate only at terminals, either by identifying two terminals or adding an edge between terminals. A composition rule also determines the terminals of the resulting graph, which must be a subset of the terminals of the two graphs being composed.

Let Γ be any class of decomposable graphs. Let the maximum number of terminals associated with any graph G in Γ be k. Following [4], it is assumed that a given graph G is accompanied by a parse tree specifying how G is constructed using the rules and that the size of the parse tree is linear in the number of nodes. Moreover, we may assume without loss of generality that the parse tree is a binary tree.

Due to lack of space and for the sake of an easier presentation, we only show how the algorithm works for the class of series-parallel graphs. Extensions to graphs of treewidth k, where k is a fixed integer, are straightforward and omitted.

Let G be a series-parallel graph with the two terminals s and t. We call an acyclic edge subgraph G' of G consisting of exactly two connected components and with s and t in different components, an *almost-tree*. Define $D[s \wedge t]$ to be the least cost of an upgrading set in G containing *both* s and t such that after upgrading the nodes in the set, G contains a spanning tree of bottleneck delay at most δ. Similarly the values $D[s \wedge \bar{t}], D[\bar{s} \wedge \bar{t}], D[k, \bar{s} \wedge \bar{t}]$ are defined to be the minimum costs of upgrading sets that include s but not t, not s but t and neither s nor t respectively. For the sake of a more concise notation, we will use $S \in \{s, \bar{s}\}$ and $T \in \{t, \bar{t}\}$.

In the same manner as we defined $D[]$, we define $A[s \wedge t]$ to be the minimum cost of an upgrading set to obtain an almost-tree of bottleneck delay at most δ.

Clearly, if we know the array $D[]$ for G, we can easily find the optimum value OPT. Further, if we also store the corresponding upgrading sets, an optimum upgrading set can be found easily.

We will now show that for a series-parallel graph G, we can compute $D[]$ and $A[]$ arrays using the *decomposition tree* of G in $\mathcal{O}(n^2)$ time.

First, we will take care of the case where G is obtained by the series composition of G_1 and G_2. Assume that we have already computed the $D[]$ and $A[]$ arrays for G_1 and G_2. Denote them by $D_1[]$, $A_1[]$ and $D_2[]$, $A_2[]$ respectively.

It is easy to see that the restriction of any tree T to G_1 and G_2 respectively is also a tree. Thus, we can compute $D[]$ with the help of $D_1[]$ and $D_2[]$ in the following way:

$$D[S \wedge T] := \min\{D_1[S \wedge \bar{t}] + D_2[\bar{s} \wedge T], D_1[S \wedge t] + D_2[s \wedge T] - \text{cost}(t)\}$$

Similarly, we now compute $A[]$ for G. An almost-tree in G must either be an almost-tree in G_1 and a tree in G_2 or vice versa. No other possibilities exist. Thus, $A[S \wedge T]$ can be computed as follows.

$$\begin{aligned} A[S \wedge T] = \min\{&D_1[S \wedge \bar{t}] + A_2[\bar{s} \wedge T], D_1[S \wedge t] + A_2[s \wedge T] - \text{cost}(t), \\ &A_1[S \wedge \bar{t}] + D_2[\bar{s} \wedge T], A_1[S \wedge t] + D_2[s \wedge T] - \text{cost}(t)\} \end{aligned}$$

We now consider the case where G is obtained by the parallel composition of G_1 and G_2. Again, we assume that the two arrays $D[]$ and $A[]$ are already available for G_1 and G_2. We start with the computation of $D[]$ array for G. A tree T in G must be a tree in exactly one of the graphs G_1 and G_2 and an almost-tree in the other. We just need to distinguish between the cases covering the upgrade of the terminals of G_1 and G_2. We must make sure that s_1 is upgraded if and only if s_2 is. The same applies to t_1 and t_2. This way $D[]$ can be computed.

For the array $A[]$, observe that if G' is an almost-tree of G, its restriction to *both* graphs G_1 and G_2 must also be an almost-tree of that particular graph. Making sure that we always include s_1 (t_1) in an upgrading set if and only if we include s_2 (t_2), we can compute the array $A[]$.

Finally, observe that for a series-parallel graph consisting of two terminals s and t and the single edge (s, t), we can trivially compute the arrays $D[]$ and $A[]$. Using the above observations, the array $A[]$ can be computed in polynomial time for a series-parallel graph G, provided a decomposition tree for G is given. Since such a decomposition tree with $\mathcal{O}(n)$ nodes can be computed in $\mathcal{O}(n)$ time, we can conclude that the dynamic programming algorithm presented above runs in total time $\mathcal{O}(n^2)$.

Using the same basic idea, namely keeping track of the terminals of the subgraphs in a decomposition, one can devise a polynomial time dynamic programming algorithm for the class of treewidth-bounded graphs. This approach leads to the following theorem.

Theorem 4.1 BSTP can be solved in $\mathcal{O}(n^k)$-time for treewidth-bounded graphs with no more than k terminals.

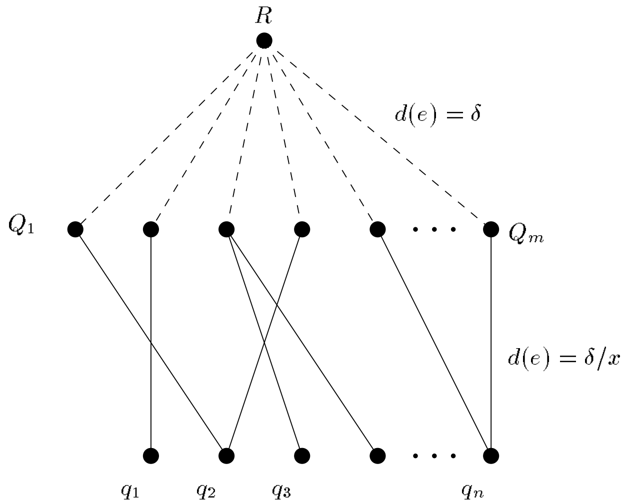

Figure 2: Graph used in the reduction from Set Cover.

5 Hardness Results

Theorem 5.1 UBSTP is NP-hard for any fixed $0 < x < 1$ and $\delta > 0$ even for bipartite graphs.

Proof: We show that Set Cover ([SP5] in [12]) reduces to UBSTP in polynomial time. An instance of Set Cover consists of a set Q of ground elements $\{q_1, \dots, q_n\}$, a collection $Q_1, \dots, Q_m$ of subsets of Q and an integer K. The question is whether one can pick at most K sets such that their union equals Q.

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

In the remainder of this proof, we do not distinguish between a node and the set or element that it represents. Note that the resulting graph is a bipartite (with R and the element nodes on one side and the set nodes on the other side). An example of the graph constructed is shown in Figure 2.

For each edge of the form (R, Q_i), the delay is δ, while each edge the form (Q_i, q_j) has delay δ/x (in Figure 2 each of the dotted lines has delay δ and the solid lines have each delay δ/x). We set the bound on the bottleneck delay parameter to δ.

We now claim that there is a solution to the UBSTP instance consisting of at most K nodes if and only if the Set Cover instance has a cover of size at most K.

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

Now assume conversely that there is an upgrading set of size at most K for the instance of UBSTP constructed above. Let $V' \subseteq V$, $|V'| \leq K$ be a set of nodes that are upgraded and let

$T = (V, E_T)$ be a spanning tree in the resulting graph of bottleneck cost at most δ.

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

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

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

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

Note that the reduction in the proof of Theorem 5.1 preserves approximations. Any set cover of size K becomes an upgrading set of size K and any upgrading set of size K becomes a cover of size at most K. Now, consider the optimization version Min Set Cover, where the objective is to find a minimum cardinality collection of the sets $Q_1, \dots, Q_m$ whose union is Q. Recently, Feige [10] has shown the following non-approximability result:

Theorem 5.2 Unless NP $\subseteq$ DTIME$(n^{\log\log n})$, the Min Set Cover problem, with a universe Q of size $|Q|$, cannot be approximated within a factor $\beta < \ln |Q|$.

Combining this hardness result with the approximation preserving reduction used above then establishes Theorem 2.3:

Theorem 2.3 Unless NP $\subseteq$ DTIME$(n^{\log\log n})$, there can be no polynomial time approximation algorithm for UBSTP with a performance guarantee of $\beta < \ln n$.

Using a different construction in the reduction from Set Cover, we can also show the following result, whose proof is omitted in this abstract.

Theorem 5.3 The hardness result of Theorem 2.3 continue to hold, if UBSTP is restricted to those instances, where *all* the edge weights are δ/x, but the graph G is not necessarily bipartite.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[20] S. O. Krumke, H. Noltemeier, M. V. Marathe, S. S. Ravi and K. U. Drangmeister, "Modifying Networks to Obtain Low Cost Trees," submitted for publication.

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

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

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

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

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

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

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

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

Heuristic-UBSTP-2del

Let $G' :=$ bottleneck(G, d, δ).
Let G' have the connected components $C_1, \ldots, C_r$.
If $r = 1$, then output $\emptyset$.
Construct an instance $\mathcal{SC}$ of Set Cover as follows:
Let the ground elements be $C_1, \ldots, C_r$ and for each $v \in V$ define the set $S_v := \{ C_j : v \in C_j$ or v is adjacent to C_j via an edge of weight $\delta/x \}$.
Find an approximate set cover $\mathcal{S} = \{S_{v_1}, \ldots, S_{v_k}\}$ for the instance $\mathcal{SC}$ of Set Cover.
Initialize $V' := \{v_1, \ldots, v_k\}$ and $V'' := \emptyset$.
Construct an auxiliary graph $\hat{G} := (V' \cup \{C_1, \ldots, C_r\}, E_1 \cup E_2)$ with edges $(v_i, C_j) \in E_1$ iff $v_i \in C_j$ or v_i is adjacent to the component C_j via an edge of delay δ/x and $(C_i, C_j) \in E_2$ iff there is an edge between $v \in C_i$ and $u \in C_j$ in the graph G.
Let H be the subgraph of $(\hat{G})^3$ induced by the vertices $v_1, \ldots, v_k$.
Find a spanning tree T_H of H.
For each edge $e = (v_i, v_j)$ in T_H do the following:
  If e was induced by a path $(v_i, C_l, C_{l'}, v_j)$ of length exactly three then
    Choose an edge (u, w) such that $u \in C_l$ and $w \in C_{l'}$ and $\{u, w\} \cap V'$ has maximum cardinality. If $\{u, w\} \cap V' = \emptyset$ then $V'' := V'' \cup \{u\}$.
Output $V' \cup V''$.

Figure 3: Improved algorithm for UBSTP, when the delays are from the two element set $\{\delta, \delta/x\}$.

Appendix

A An Improved Approximation Algorithm for Unit Costs

In this section, we will show how to obtain an approximation algorithm for the case of unit node costs with a performance guarantee of $4(2 + \ln \Delta)$. For $\Delta = o(\sqrt{n})$, this 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.*

Recall that in the case of UBSTP the objective is to find an upgrading set of *minimum cardinality.* We first consider the case, when all the delays on the edges are taken from the two element set $\{\delta/x, \delta\}$. Observe that this is equivalent to saying that all the delays are from the interval $[0, \delta/x]$, since the goal is to upgrade vertices such that there is a bottleneck spanning tree of delay no more than δ in the graph with the delays resulting from the upgrade.

The improved algorithm is depicted in Figure 3. Recall that the t^{th} *power* $G^t = (V, E^t)$ of a graph G contains an edge from u to v if and only if there is a path of length at most t edges in G connecting u and v.

We now establish the performance of the algorithm, which is summarized in the following theorem.

Theorem A.1 Heuristic-UBSTP-2del is a polynomial time approximation algorithm for UBSTP with performance guarantee of $2(1 + \ln \Delta)$, when all the delays are in the range $(0, \delta/x]$, where Δ is the maximum degree in the input graph G.

The proof relies on several lemmas presented below. Heuristic-UBSTP-2del clearly detects the case, when no node needs to be upgraded (Step 3). Thus, for the rest of the analysis we will restrict ourselves to the case, when the optimal solution contains at least one node.

Lemma A.2 Let C^* be a minimum size set cover for the instance I' of Set Cover constructed in step 4 of Heuristic-UBSTP-2del. Then $|C^*| \leq \mathrm{OPT}$.

Proof: Let V^* be an upgrading set in the original graph and let T^* be a corresponding bottleneck spanning tree. We show that the sets S_v with $v \in V^*$ form a set cover for the instance I'. This will prove the claim of the lemma.

Consider an arbitrary cluster node C_j. Since $G' = \mathrm{bottleneck}(G, d, \delta)$ contains more than one connected component, it follows that in the tree T^* there must be an edge (u, w) of delay at most δ with $u \in C_j$ and $w \notin C_j$. Since $w \notin C_j$ this edge must have had delay δ/x before the upgrade. It follows that either u or w must have been upgraded. We have $C_j \in S_w$ (because w is connected to C_j via an edge of delay δ/x) 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, $v \in V^*$. □

Observation A.3 The graph $\hat{G}$ and the subgraph H of $(\hat{G})^3$ which is computed in Step 8 of the algorithm are connected.

Proof: We first show that $(\hat{G})^3$ is connected. Note that the clusters $C_1, \ldots, C_r$ are connected since they merely partition the original vertices of G'. Each vertex $v_i \in V'$ is connected to a cluster C_j by an edge since either $v_i \in C_j$ or v_i is adjacent to a vertex $v_k \in C_j$. Thus $(\hat{G})^3$ is connected.

Now consider H. Since V' forms a feasible dominating set in the graph $(\hat{G})$, and by the fact that $(\hat{G})$ is connected it is easy to observe that H is connected. □

Lemma A.4 The set $V' \cup V''$ output by Heuristic-UBSTP-2del is a valid upgrading set in G.

Proof: We show that after upgrading the vertices in $V' \cup V''$ there is a spanning tree T in G (with the resulting delay function) of delay no more than δ. To this end, define an edge-subgraph G_T of G in the following way: G_T contains all the edges of G that have delay at most δ (thus each connected component C_j of the bottleneck graph G' will again be connected in G_T). Moreover, for each edge $e \in T_H$, where T_H is the spanning tree of H computed in Step 9 of the algorithm, add the corresponding edges of the path of length at most three in $\hat{G}$, which induced e. Observe that by construction of the sets V' and V'' each of the edges in G_T has delay at most δ. Using the fact that H and $\hat{G}$ were connected and that T_H was a spanning tree of H, it is now easy to see that G_T must also be connected. Consequently, there is a spanning tree of G_T (and thus also of the upgraded graph) that contains no edge of delay larger than δ. □

Using the results of Lemma A.2, Observation A.3 and Lemma A.4, we can now complete the proof of Theorem A.1 in the following way. By Lemma A.2, we know that

$$|V^*| = \mathrm{OPT} \geq |C^*|, \tag{13}$$

where C^* is an optimal set cover for the instance $\mathcal{SC}$. Using known approximation techniques for Set Cover (cf. [14]), we can find a set cover $\mathcal{S}$ in Step 5 of the algorithm Heuristic-UBSTP of size at most

$$|V'| = |\mathcal{S}| \leq (1 + \max_{v \in V} \ln |S_v|) \cdot |C^*| \leq (1 + \ln \Delta) \cdot |C^*|. \tag{14}$$

We now address the cardinality of the set V''. The tree T_H which is computed in Step 9 contains $|V'|-1$ edges. Since in the loop in Steps 10 to 14 for each edge in T_H we add at most one vertex to V'', we have that $|V''| \leq |V'|$. Combining this result with (14) and (13), we obtain:

$$|V' \cup V''| \leq |V'| + |V''| \leq 2|V'| \leq 2(1+\ln\Delta) \cdot |C^*| \leq 2(1+\ln\Delta) \cdot \mathrm{OPT}$$

□

We now extend the result of Theorem A.1 to the case where the edge delays are in the range $(0, \delta/x^2]$. Again, we can assume without loss of generality that the delays of the edges are taken from the three element set $\{\delta, \delta/x, \delta/x^2\}$. We use OPT to denote the cardinality of an optimal solution V^* to I. We also use Heu to denote the solution obtained by the extended heuristic, which will be called Heuristic-UBSTP in the sequel.

We give an informal description of the algorithm Heuristic-UBSTP. First, we construct the bottleneck graph G'' which contains all the edges of delay δ and δ/x. Let G'' have the connected components $C_1, \ldots, C_r$. We find an upgrading set and a spanning tree for each component using algorithm Heuristic-UBSTP-2del. We then join these spanning trees using $r-1$ edges each of delay δ/x^2. We upgrade both endpoints of each of these $r-1$ edges. Denote the set of nodes which are added to the solution in this final step by U_{δ/x^2}.

Lemma A.5 For the size of an optimal solution and the number of vertices added in the final step to join the clusters the following inequalities hold:

$$\mathrm{OPT} \ \geq \ r \qquad \text{and} \qquad |U_{\delta/x^2}| \ \leq \ 2(r-1). \tag{15}$$

Proof: Let T^* be a bottleneck spanning tree in the graph resulting from the upgrade of the vertices in the optimum set V^*. Since there are r connected components $C_1, \ldots, C_r$ in the bottleneck graph bottleneck$(G, d, \delta/x)$, T^* contains at least $r-1$ edges of original delay δ/x^2 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 (15).

On the other hand, the heuristic sketched above joins the r connected components by $r-1$ edges and chooses both the end points of each edge. Thus, $|U_{\delta/x^2}| \leq 2(r-1)$. □

We are now ready to establish the performance of the extended algorithm Heuristic-UBSTP.

Theorem A.6 Heuristic-UBSTP is a polynomial time approximation algorithm for UBSTP with performance guarantee of $4(2+\ln\Delta)$.

Proof: Again, let T^* be a bottleneck spanning tree in the graph resulting from the upgrade of the vertices in the optimal upgrading set V^*. Let OPT_i be the minimum number of nodes which must be upgraded in cluster C_i to make it contain a spanning tree of delay at most δ. Consider the intersection of OPT with the cluster C_i. We have the following claim.

$$|V^* \cap C_i| \geq \frac{1}{2}\mathrm{OPT}_i. \tag{16}$$

In fact, let $V^* \cap C_i = V_i'$. We first prove that, after upgrading the nodes in V_i', each node in $C_i - V_i'$ is connected to at least one node from V_i' via paths in C_i containing edges of delay no more than δ. Assume that this is not the case. Then there is a node $v \in C_i - V_i'$, whose unique path in T^* to any node in V_i' contains at least one edge of original delay δ/x^2. Fix $w \in V_i'$ and consider the path $(v = u_0, u_1, \ldots, u_k = w)$ from v to w in in T^*. Without loss of generality, we can assume that this path does not contain any node from V_i' different from w. Let ℓ be the smallest number such that $(u_\ell, u_{\ell+1})$ has original delay δ/x^2. As $v \notin V_i'$, we have $\ell \geq 1$. Since after the upgrade T^* contains only edges of delay at most δ, we see that both nodes u_ℓ and $u_{\ell+1}$ belong to V_i' contradicting the fact that the path did not contain any other node from V_i' other than w.

We have seen that after upgrading the nodes in V_i' the cluster C_i restricted to the edges of delay at most δ contains at most $|V_i'|$ 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 δ. Thus, there is an upgrading set in C_i of size at most twice the size of V_i', which proves (16).

We know by Theorem A.1 that the algorithm Heuristic-UBSTP-2del finds a solution for each cluster whose value is bounded from above by $2(1+\ln\Delta_i)\mathrm{OPT}_i$, where Δ_i is the maximum degree in the subgraph of G induced by C_i. Since the clusters are disjoint we have that

$$\mathrm{OPT} = |V^*| = \sum_{i=1}^{r} |V^* \cap C_i| \overset{(16)}{\geq} \frac{1}{2}\sum_{i=1}^{r} \mathrm{OPT}_i. \tag{17}$$

Thus, using additionally Lemma A.5, the cardinality of the solution set generated by our algorithm can be estimated as follows:

$$\begin{aligned}
\mathrm{Heu} &\leq 2(r-1) + \sum_{i=1}^{r} 2(1+\ln\Delta_i)\mathrm{OPT}_i \\
&\leq 2(r-1) + 2(1+\ln\Delta)\sum_{i=1}^{r}\mathrm{OPT}_i \\
&\overset{(17)}{\leq} 4(r-1) + 4(1+\ln\Delta)\cdot\mathrm{OPT} \\
&\overset{\text{Lemma A.5}}{\leq} 4(2+\ln\Delta)\cdot\mathrm{OPT} - 4.
\end{aligned}$$

□

B The Link Delay Problem

In this section we briefly treat a related problem to BSTP which has been defined by Paik and Sahni in [24] and which is called the *Link Delay Problem*. Given a weighted graph as for BSTP, one searches again for a minimum cost upgrading set, but this time subject to the constraint that upgrading those vertices will reduce the delay of *all* links in the network to δ instead of just those in a minimum bottleneck spanning tree.

Paik and Sahni showed in [24] that LinkDelay is NP-hard by providing a reduction from Vertex Cover ([GT1] in [12]). In this section we give a simple polynomial time approximation algorithm for LinkDelay with a performance of 2. The heuristic is shown in Figure 4 on the following page. The following theorem states the approximation result:

Theorem B.1 For any instance of LinkDelay the algorithm Heuristic-LinkDelay either provides a set V' such that upgrading the vertices in V' will result in a link delay of at most δ and the cost $\mathrm{cost}(V')$ of the solution produces is at most twice the optimum upgrading cost or it correctly informs that there is no upgrading set yielding a link delay of at most δ.

Heuristic-LinkDelay

1 If $d(e) > \frac{\delta}{x^2}$ for some $e \in E$ then output "The link delay cannot be reduced to be at most δ" and stop.
2 Let $E_1 := \{e \in E : d(e) > \frac{\delta}{x}\}$.
3 Set $V' := \{u, v : (u, v) \in E_1\}$.
4 For all $e \in E - E_1$ such that e is incident on some node in V', set $d(e) := xd(e)$.
5 Let $E' := \{e \in E - E_1 : d(e) > \delta\}$.
6 Compute a 2-approximation C to the problem of finding a minimum total cost vertex cover in the graph $G' = (V, E')$ (see [12]).
7 Output $V' \cup C$.

Figure 4: Approximation algorithm for the Link Delay Problem.

Proof: Let V^* be an upgrading set of minimum cost. Clearly, the set V' computed in Step 3 of the algorithm must be contained in V^*. It is also easy to see that $V^* - V'$ must be a vertex cover in the graph G' as defined in Step 6 of the algorithm. Thus, the cost of the nodes in $V^* - V'$ is at least that of the optimum vertex cover in G'. The proof can now be completed in a straightforward manner. □

Moreover, the proof of Theorem B.1 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 LinkDelay. Thus, by the results in [4], we can conclude:

Corollary B.2 The restriction of LinkDelay to graphs of bounded treewidth can be solved in polynomial time.