Increasing the Weight of Minimum Spanning Trees *

Greg N. Frederickson † Roberto Solis-Oba ‡

Department of Computer Sciences, Purdue University, West Lafayette IN, 47907-1398

Abstract

The problems of computing the maximum increase in the weight of the minimum spanning trees of a graph caused by the removal of a given number of edges, or by finite increases in the weights of the edges, are investigated. For the case of edge removals, the problem is shown to be NP-hard and an $\Omega(1/\log k)$-approximation algorithm is presented for it, where k is the number of edges to be removed. The second problem is studied assuming that the increase in the weight of an edge has an associated cost proportional to the magnitude of the change. An $O(n^3m^2\log(n^2/m))$ time algorithm is presented to solve it.

1 Introduction

Consider a communication network in which information is broadcast over a minimum spanning tree. There are applications for which it is important to determine the maximum degradation in the performance of the broadcasting protocol that can be expected as a result of traffic fluctuations and link failures [25]. Also, there are several combinatorial optimization problems whose solutions (or approximate solutions) require finding a minimum spanning tree of a graph and, thus, for which a change in the weight of the minimum spanning trees means a change in the value of the solution [14, 17, 20, 28]. In this paper we study the effect that edge removals and increments in the weights of the edges of a graph have on the weight of its minimum spanning trees.

Consider an undirected connected graph $G = (V, E, w)$, where w is a function that assigns non-negative weights to the edges. We assume that the weight of an edge is not a static quantity, but it can be increased at a certain cost. The *robustness problem for minimum spanning trees* consists in finding the maximum increase in the weight of the minimum spanning trees of G that

*A preliminary version of this paper appeared in the Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, pages 539-546, 1996.

†Research of this author partially supported by the National Science Foundation under grant CCR-9322501.

‡Research of this author partially supported by NSF grant CCR-9322501, by CONACyT-Fulbright, and by Universidad Autonoma Metropolitana. Contact author: solis@cs.purdue.edu.

can be obtained by spending a given budget to augment the weights of the edges. In this paper we study the robustness problem for minimum spanning trees under two particular cost models for increasing edge weights. In the first one a unit cost is assigned for removing an edge from the graph (or, in other words, for increasing its weight to ∞). In this *discrete* version of the problem the objective is to find, for a given value k, the set of k edges whose removal from a graph causes the largest increase in the weight of its minimum spanning trees. The second cost model that we consider assigns a cost of $c(e) * \delta$ for increasing the weight of an edge e by δ units, where $c(e)$ is a positive real value. For this *continuous* version of the problem the goal is to know, for a given value B, the maximum possible increase in the weight of the minimum spanning trees that can be produced by edge weight increases of total cost B.

The discrete version of our problem has appeared in the literature under the name of the most vital edges in the minimum spanning tree problem [15, 16, 18, 21]. In [21] it is shown that when the edges of a graph have arbitrary destruction costs, the problem of computing the maximum increase in the weight of the minimum spanning trees achievable by removing edges of a certain total destruction cost is NP-hard. In [15, 16, 18, 27] algorithms are given for finding the single most vital edge for a minimum spanning tree. The continuous version of the robustness problem for minimum spanning trees can be seen as a generalization of the sensitivity analysis for minimum spanning trees [27], since we need to consider simultaneous changes in the weights of several edges. Other properties of a network that have been studied in the context of link failures or degeneration of the performance of links are maximum flow [23], minimum cost flow [26] and shortest distance between two distinguished vertices [2, 9, 22].

In this paper we prove that the discrete version of our robustness problem is NP-complete even if the edges have weight either 0 or 1. We present the first approximation algorithm for this problem in the case when the edge weights are arbitrary non-negative real numbers. The algorithm runs in $O(k \log n(m + kn \log n))$ time and finds a solution whose value is $\Omega(1/\log k)$ times the optimal. The approximate solution is obtained by solving a series of $O(\log n)$ instances of the *maximum components problem*, which consists in selecting a set of k edges whose removal from a graph maximizes the number of its connected components. We give a (1/2)-approximation algorithm for this latter problem, which can be shown to be NP-complete by a trivial reduction from the *minimum k-cut problem*.

For the continuous version of the robustness problem we show that, given a budget value B, any optimal solution can be constructed incrementally in such a way that every partial solution is optimal for some budget value $B' \leq B$. This allows us to design an efficient greedy algorithm for the problem. The core of the algorithm is a procedure for selecting a subset S of edges that maximizes the ratio of the increase in the weight of the minimum spanning trees of G to the cost of modifying the weights of the edges in S. This selection can be made efficiently by a careful combination of edge contractions and *graph strength* [3, 11] computations. We show that the maximum weight of the minimum spanning trees of G attainable by increasing edge weights is a piecewise linear concave function f of the budget value; we also prove that f has at most mn breakpoints. Our algorithm can compute f for all budget values $B \geq 0$, in $O(n^3 m^2 \log(n^2/m))$ time.

Since solving the continuous version of the robustness problem may raise the weights of an exponential number of minimum spanning trees, it is somewhat surprising that this version of the problem is not NP-hard as its discrete counterpart. There are a variety of problems in graph theory [2, 9, 22], scheduling [1] and transportation [6] with a similar property of having a discrete version that is NP-hard and a continuous version that can be solved efficiently.

Interestingly, the approach presented in this paper to solve the continuous version of the robustness problem for minimum spanning trees can be extended to arbitrary matroids. We have designed an algorithm for computing the maximum increase in the weight of the minimum weight bases of a matroid attainable by increases of a given total cost on the weights of its elements. The algorithm runs in strongly polynomial time on matroids for which independence can be tested in strongly polynomial time. Moreover, we have more efficient algorithms to solve the problem for transversal, scheduling and partition matroids. These results will appear in a forthcoming paper [8].

The rest of the paper is organized in the following way. In Section 2.1 we present a (1/2)-approximation algorithm for the maximum components problem. In Section 2.2 we present an $\Omega(1/\log k)$-approximation algorithm for the discrete version of the robustness problem for minimum spanning trees. In Section 2.3 we add a preprocessing step to our approximation algorithm that reduces its overall time complexity. In Section 2.4 we prove that the robustness problem for minimum spanning trees is NP-hard even when each edge weight is either 0 or 1. Finally, in Section 3 we present the algorithm for solving the continuous version of the robustness problem.

2 Discrete Version

Let $G = (V, E, w)$ be an undirected connected graph. Given an edge $e \in E$, let G' be the subgraph of G including only the edges of weight at most $w(e)$. If we remove from G any subset of edges that partitions G' into $i+1$ connected components, then every minimum spanning tree of G would have at least i edges of weight larger than $w(e)$. We use this idea to build an approximation algorithm for the discrete version of the robustness problem for minimum spanning trees. However, we need to know for a given value k how to select a set of k edges whose removal from a graph decomposes it into the maximum possible number of connected components.

2.1 The Maximum Components Problem

A *k-cut* of a graph $G = (V, E, w)$ is a set of edges that when removed from G partitions it into k connected components. The minimum k-cut problem [13] consists in finding a k-cut of minimum cardinality. This problem is similar to the maximum components problem, except that in the latter problem we want to maximize the number of components created by the removal of a given number of edges.

It is easy to see that an algorithm to solve the maximum components problem could be used to solve the minimum k-cut problem. Simply run the algorithm over the values $i = 1, 2, \ldots, m$

until we find the value i for which the number of components created by removing i edges from the graph is equal to k. Since the minimum k-cut problem is NP-hard [13], the above reduction implies the NP-hardness of the maximum components problem.

We modify an algorithm by Saran and Vazirani [24] to obtain a (1/2)-approximation algorithm for the maximum components problem. Our algorithm uses a greedy approach to chop up a graph into as many pieces as possible by removing k edges from it.

Algorithm *dice* (G, k)

repeat

Remove from G the edges in a smallest cut that divides one of the connected components of G.

until $k' \leq k$ edges have been removed and no additional cut can be made with the remaining $k - k'$ edges

Output the set of edges that were removed from G.

THEOREM 2.1. *Given an undirected graph G, algorithm dice finds a set S of at most k edges whose removal from G partitions it into $d > a/2$ components, where a is the maximum number of components that can be formed by removing any set of k edges from G.*

We first prove the following Lemma for connected graphs. Assume that we run *dice* on some connected graph G. Let $D = \{D_1, D_2, ..., D_{d-1}\}$ be the set of cuts selected by *dice* indexed in the order in which they were chosen. We assume that $a > d$, because otherwise D would be an optimal solution and hence Theorem 2.1 would follow trivially. For convenience, let D_d denote the d-th cut that *dice* would choose. Let A be a set of k edges that divides G into the maximum possible number of connected components, and let $V_1, V_2, \ldots, V_a$ be such components. For all $1 \leq i \leq a$, let A_i be the cut that separates V_i from $V - V_i$. We assume that the cuts A_i are indexed in non-decreasing order of size.

LEMMA 2.1. $|D_i| \leq |A_i|$ *for all* $1 \leq i \leq d$.

Proof. The proof is by induction on i.

Basis. Clearly $|D_1| \leq |A_1|$ because D_1 is a minimum cut of G.

Induction step. Assume as induction hypothesis that $|D_i| \leq |A_i|$ for all $1 \leq i < d$. Note that since $a > d$, the cuts $A_1, A_2, \ldots, A_d$ divide G into at least $d+1$ components. This means that $\cup_{i=1}^{d} A_i \not\subset \cup_{i=1}^{d-1} D_i$ because the cuts $D_1, D_2, \ldots, D_{d-1}$ divide G into only d components, say $V_1', V_2', \ldots, V_d'$. Hence, there must be some cut A_j, $1 \leq j \leq d$, that partitions at least one of the components $V_1', V_2', \ldots, V_d'$. Such a cut has size $|A_j| \geq |D_d|$ because algorithm *dice* would choose in the d-th iteration the smallest cut that splits any of the existing components. Since the cuts A_i are indexed in non-decreasing order of size, then $|A_j| \leq |A_d|$ and, therefore, $|D_d| \leq |A_d|$. □

Proof of Theorem 2.1. To simplify the proof we assume first that the graph G is connected. Then we show that the Theorem also holds for disconnected graphs.

Let D and A be as above. The cuts in D are pairwise disjoint, and hence $\sum_{i=1}^{d-1} |D_i| = k' \leq k$. Combining this inequality with Lemma 2.1 we get $\sum_{i=1}^{d} |A_i| \geq \sum_{i=1}^{d} |D_i| = k' + |D_d| > k$. Every edge $e \in A$ appears in exactly two of the cuts A_i and, therefore, $\sum_{i=1}^{a} |A_i| \leq 2k$. From these last two inequalities it follows that $\sum_{i=d+1}^{a} |A_i| \leq 2k - \sum_{i=1}^{d} |A_i| < k < \sum_{i=1}^{d} |A_i|$. Since the cuts A_i are in non-decreasing order of size, then the number of cuts in the set $\{A_{d+1}, A_{d+2}, \ldots, A_a\}$ is smaller than the number of cuts in the set $\{A_1, A_2, \ldots, A_d\}$. This means that $a - d < d$ and thus that $d > a/2$.

Now we consider the case when G is disconnected. Suppose that G consists of $j+1$ connected components. Add a set J of j edges to connect these components in a tree-like fashion and call the resulting graph G'. It is clear that if we use algorithm *dice* to remove k edges from G and to remove $k+j$ edges from G', the number of components created in both cases will be equal to d. Similarly, the maximum number of components that can be formed by removing k edges from G and $k+j$ edges from G' is a. Since the graph G' is connected, by the above argument it follows that $d > a/2$. □

Theorem 2.2. *Algorithm dice runs in* $O(km + k^2 n \log n)$ *time.*

Proof. We can implement algorithm *dice* so that in each iteration, except the first one, it only

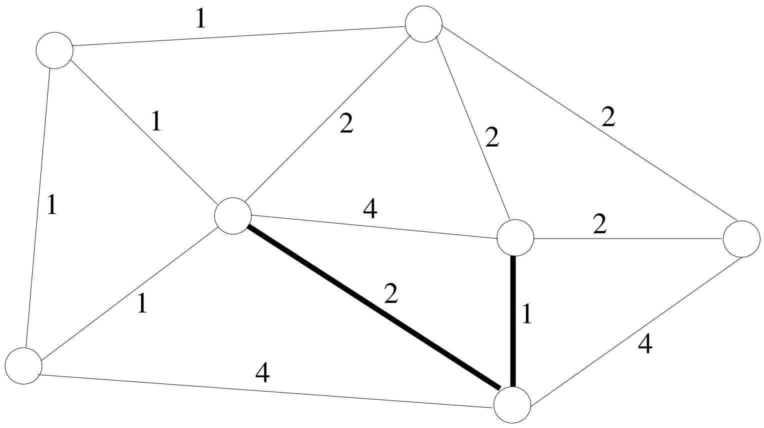

Figure 1: An instance of the discrete robustness problem, with the two edges in bold being those whose removal maximizes the weight of the minimum spanning trees.

computes connectivity cuts for the two components that were created in the preceding iteration. Gabow's algorithm [10] finds the edge connectivity λ and a connectivity cut for a given graph in $O(m + \lambda^2 n \log(n/\lambda))$ time. Furthermore, if $k < \lambda$ this algorithm only needs $O(m + k^2 n \log(n/k))$ time to check that the graph does not have a cut of size k. A simple calculation shows that algorithm *dice* runs in $O(km + k^2 n \log n)$ time if implemented with Gabow's algorithm. □

2.2 The Robustness Problem for Minimum Spanning Trees

Given an undirected graph $G = (V, E, w)$ and an integer k, we wish to find the maximum increase in the weight of the minimum spanning trees of G attainable by the removal of k edges from it. Figure 1 shows a simple example. For $k = 2$, the maximum increase in the weight of the minimum spanning trees of the graph shown is achieved by removing the edges drawn in bold. By doing this we replace, in every minimum spanning tree of the graph, the edge in bold of weight 1 by an edge of weight 4. Furthermore, in some minimum spanning trees the edge in bold of weight 2 is also replaced by some other edge of weight 2. Therefore, the weight of the minimum spanning trees increases from 8 to 11.

The approach that we follow to approximate the solution of the problem consists in forcing into every minimum spanning tree of G as many edges of "large" weight as possible. It is easy to see that an edge $e \in E$ belongs to a minimum spanning tree of G if and only if e is the smallest weight edge in some cut of G. This means that if some edge e' does not belong to any minimum spanning tree of G, we can delete the edges of weight smaller than $w(e')$ in some cut that separates the endpoints of e' to get a new graph in which e' belongs to at least one minimum spanning tree. Since all minimum spanning trees of a graph have the same number of edges of a given weight, the process just described can be used to add an edge of weight $w(e')$ into every minimum spanning tree of G.

The above observation suggests the following heuristic for the robustness problem. Let $G_x = (V, E_x, w)$ denote the subgraph of G that includes all the edges of weight at most x. We form the subgraphs G_{w_i} for every edge weight w_i, and find for each one of them a set S_i of k edges that partitions the subgraph into approximately the maximum possible number of connected components. The set S_i whose removal from G maximizes the increase in the weight of the minimum spanning trees is selected as our solution.

Algorithm *slice_n_dice* $(G = (V, E, w), k)$
    max_mst_wgt $\leftarrow 0$
    for each distinct edge weight w_i in G **do**
        $S \leftarrow$ *dice* (G_{w_i}, k)
        mst_wgt $\leftarrow$ weight of a minimum spanning tree of $(V, E - S, w)$
        if *mst_wgt* $>$ *max_mst_wgt* **then**
            max_mst_wgt $\leftarrow$ *mst_wgt*
            $S_{s\&d} \leftarrow S$
        end if
    end for
    Output $S_{s\&d}$ and *max_mst_wgt*.

The algorithm outputs the selected set $S_{s\&d}$ and also the weight of the minimum spanning trees that can be achieved by removing those edges from the graph. We note that the sequence of graphs G_{w_i} is not well-behaved in the sense that we could not perform binary search on the weights w_i to find the set $S_{s\&d}$. The algorithm performs at most m iterations and the time complexity of *dice* dominates each one of them, hence *slice_n_dice* runs in $O(km^2 + k^2 mn \log n)$ time.

To simplify the performance analysis of *slice_n_dice* we assume first that all of the edges in G have weights of the form 2^i, where i is an integer. Let S^* be a set of k edges whose removal from G causes the largest possible increase in the weight of its minimum spanning trees. Let T, T^*, and $T_{s\&d}$ denote minimum spanning trees of G, $(V, E - S^*, w)$, and $(V, E - S_{s\&d}, w)$ respectively. Given a subset of edges $S \subseteq E$, we denote by $w(S)$ the sum of the weights of its edges.

LEMMA 2.2. *If the edges of G have weights of the form 2^i, for some integer i, then the performance ratio of slice_n_dice is $r = w(T_{s\&d})/w(T^*) \geq 1/(1 + 2\log(k+1))$.*

Proof. Let the smallest and largest edge weights in G be 2^s and 2^u respectively. We can write the weight of the optimal tree T^* as

$$w(T^*) = w(T) + \sum_{i=s+1}^{u} a_i 2^i - \sum_{i=s}^{u-1} b_i 2^i$$

where a_i (b_i) is the number of edges of weight 2^i that are added to (taken off) every minimum spanning tree of G after the removal of the edges in S^*. It is easy to see that, for every $s < i \leq u$, a_i is no larger than the maximum number of new components that can be formed in $G_{2^{i-1}}$ by removing k edges from it. Let $a_\ell 2^\ell = \max \{ a_i 2^i \mid s < i \leq u \}$. From the description of *slice_n_dice* it follows that $w(T_{s\&d}) \geq w(T)$ and that $w(T_{s\&d}) \geq a^\ell 2^\ell / 2$ (the factor $1/2$ comes from the fact that the algorithm *dice* gives a $(1/2)$-approximation to the solution of the maximum components problem). Combining these inequalities we get

$$\begin{aligned} \frac{1}{r} = \frac{w(T^*)}{w(T_{s\&d})} &\leq \frac{w(T) + \sum_{i=s+1}^{u} a_i 2^i}{w(T_{s\&d})} \\ &\leq 1 + \frac{2\sum_{i=s+1}^{u} a_i 2^i}{a_\ell 2^\ell} \end{aligned}$$

Since at most k edges are removed from G, then at most k new edges can be included in any minimum spanning tree. This means that at most k of the coefficients a_i in the above summation can be non-zero. Let such non-zero coefficients be $a_{\pi_0}, a_{\pi_1}, \ldots, a_{\pi_t}$, $t < k$. Since $a_{\pi_i} 2^{\pi_i} \leq a_\ell 2^\ell$ for all $0 \leq i \leq t$, the performance ratio r achieves its minimum possible value when $a_{\pi_i} 2^{\pi_i} = a_\ell 2^\ell$ for all $0 \leq i \leq t$. Without loss of generality assume that $\pi_0 > \pi_1 > \ldots > \pi_t$, then

$$k \geq \sum_{i=0}^{t} a_{\pi_i} = a_{\pi_0} \sum_{i=0}^{t} 2^{\pi_0 - \pi_i} \geq a_{\pi_0} \sum_{i=0}^{t} 2^i \geq 2^{t+1} - 1$$

Taking logarithms we get $t < \log(k+1)$. Thus at most $\log(k+1)$ of the coefficients a_i can be different from zero and therefore $r \geq 1/(1 + 2\log(k+1))$. □

Lemma 2.3. *The maximum weight of the minimum spanning trees of a graph $G = (V, E, w)$ that can be achieved by the removal of k edges from G is no larger than 2 times the corresponding optimal value for the graph $G' = (V, E, w')$ formed by rounding down the weight of every edge in E to its nearest power of 2.*

Proof. Let T^* and T' be minimum spanning trees having the maximum possible weight that can be achieved by removing k edges from G and G' respectively. It is easy to see that for any $S \subseteq E$, $w(S)/2 < w'(S) \leq w(S)$.

By the definition of T' it follows that $w'(T^*) \leq w'(T')$ and, therefore, that $w(T^*) < 2w'(T^*) \leq 2w'(T') \leq 2w(T')$. □

Lemmas 2.2 and 2.3 allow us to state the following result.

Theorem 2.3. *The performance ratio of slice_n_dice when the input graph has edges with arbitrary non-negative weights is $r = w(T_a)/w(T^*) \geq 1/(2 + 4\log(k+1))$.* □

The bound stated in Lemma 2.2 for the performance ratio of *slice_n_dice* is tight to within a constant factor. To prove this, construct the following graph G. Let $\hat{k}$ be the largest power of 2 no larger than k. Let N_i, for $i = 0, 1, \ldots, \hat{k}/2$ be cliques of size $k+2$ each. Every edge of the cliques N_i has weight $\varepsilon = 2^{-L}$ for some $L \gg 1$. For all $0 \leq i < \hat{k}/2$, add an edge of weight ε between one of the vertices of N_i and one of the vertices of N_{i+1}. For all $1 \leq j \leq \log \hat{k}$, add an edge of weight 2^j between one vertex of N_i and one of N_{i+1} for all $0 \leq i < \hat{k}/2^j$. To increase the edge connectivity of G above k, we add k edges of weight $2^{\log \hat{k} - \lfloor \log i \rfloor - 1}$ between N_i and N_{i+1} for each $1 \leq i < \hat{k}/2$, and we also add k edges of weight $\hat{k}$ between N_0 and N_1 (see Figure 2).

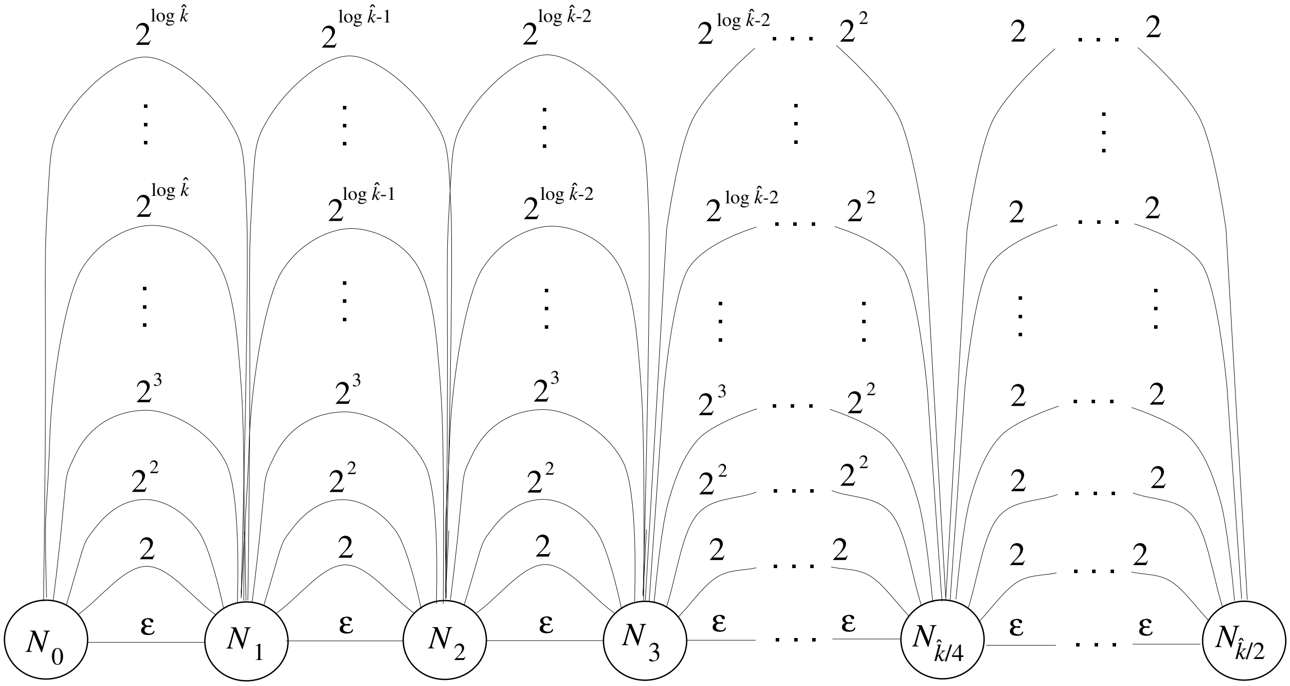

Figure 2: Graph used to show the tightness of the analysis.

The set of k edges whose removal from G maximizes the weight of its minimum spanning trees includes all the edges of weight ε and the edges of weight smaller than $2^{\log \hat{k} - \lfloor \log i \rfloor - 1}$ between N_i and N_{i+1}, for all $1 \le i < \hat{k}/2$. If these edges are removed from G, a minimum spanning tree of the resulting graph has weight $w(T^*) = 2\hat{k}/2^2 + 2^2\hat{k}/2^3 + 2^3\hat{k}/2^4 + \ldots + 2^{\log \hat{k}-1}\hat{k}/2^{\log \hat{k}} + 2^{\log \hat{k}} = \hat{k}(\log \hat{k} + 1)/2$.

Algorithm *slice_n_dice* chooses the edges of weight smaller than $2^{\log \hat{k}}$ placed between N_0 and N_1. The largest weight minimum spanning tree that can be obtained by removing the set of edges chosen by the algorithm has weight $w(T_{s\&d}) = \hat{k} + (\hat{k}/2 - 1)\varepsilon$. Hence $w(T^*)/w(T_{s\&d}) > \log k/2$.

We consider now the following more general version of the problem. Assume that the edges in a graph G have arbitrary destruction costs and that we are given a positive budget B to remove edges from G so as to maximize the weight of its minimum spanning trees. We can modify algorithm *dice so that in each iteration it finds a cut with smallest destruction cost. This allows us to use* slice_n_dice *to approximate the solution to this problem.*

LEMMA 2.4. *If the edges of a graph $G = (V, E, w)$ have arbitrary destruction costs, the per-*

formance ratio of slice_n_dice is $r \geq 1/(1 + 4\log m)$.

Proof. We can modify the proof of Theorem 2.1 to show that the performance ratio stated in it for algorithm *dice still holds in this case. The only change that we need to make in the proof is to replace, in each inequality involving cuts, the cardinality of a cut by the sum of the destruction costs of its edges.*

To find the performance ratio of slice_n_dice, *assume first that the edges of* G *have weights of the form* 2^i, *where* i *is integer. Proceeding similarly as in the proof of Lemma 2.2 we get*

$$\frac{1}{r} \leq 1 + \frac{2\sum\limits_{i=s+1}^{u} a_i 2^i}{a_\ell 2^\ell}.$$

Up to $m - n + 1$ *edges can be removed from* G, *and thus at most* $m - n + 1$ *of the coefficients* a_i *in the above summation can be non-zero. This implies that* $r \geq 1/(1 + 2\log(m - n + 1))$. *From this and Lemma 2.3 we get the desired bound for the performance ratio of the algorithm on graphs with arbitrary edge weights.* □

2.3 A Faster Algorithm

In this section we show how to improve the time complexity of slice_n_dice *by reducing the number of iterations that it has to perform to at most* $\lceil \log k \rceil$. *The idea is to change the algorithm so that it does not build all the subgraphs* G_{w_i}, *but only those for "large" weights* w_i. *The intuition behind this approach is the following. The cuts made on* G_{w_i} *can be guaranteed to force edges of large weight into a minimum spanning tree only if* w_i *is large. If* w_i *is small and a large weight edge is added to the minimum spanning trees, then the same edge can be included in the minimum spanning trees by making cuts in a subgraph* G_{w_j} *for some larger weight* w_j. *We need to be careful when selecting these large weights so that the approximation ratio of* slice_n_dice *is only slightly affected.*

While we wish to build subgraphs only for large weights, we need to be aware that some edges of large weight cannot possibly be part of a solution. Given an undirected graph G *and a value* k, *an edge e of G is said to be* redundant *if it is not possible to remove k edges from G in such a way that e belongs to some minimum spanning tree of the resulting graph. Clearly* slice_n_dice *can ignore the redundant edges of a graph without affecting the quality of the solution that it computes. This observation is essential to design a faster version of* slice_n_dice.

To simplify the following discussion, we assume that the weights of the edges in a graph G are integer powers of 2. Let a largest non-redundant edge of G have weight 2^L. *Let* $T_{s\&d}$ *be the minimum spanning tree found by* slice_n_dice *for the input* (G, k). *Let us modify* slice_n_dice *so that it forms only the graphs* G_{w_i} *for all* $2^{L-\lceil \log k \rceil} \leq w_i \leq 2^L$. *Let* $T'_{s\&d}$ *be the minimum spanning tree found by this modified algorithm.*

LEMMA 2.5. $w(T_{s\&d})/w(T'_{s\&d}) < 2$.

Proof. Let T be a minimum spanning tree of G. We assume that $L > \lceil \log k \rceil$ and that the solution chosen by the original algorithm *slice_n_dice* comes from removing k edges from some graph G_{w_i} with $w_i < 2^{L-\lceil \log k \rceil}$, because otherwise $T_{s\&d}$ and $T'_{s\&d}$ would be the same.

From these assumptions it is clear that $w(T_{s\&d}) \leq w(T) + k_0 2^{L-\lceil \log k \rceil} + \Delta - k_1$, where $k_0 \leq k$ is the number of edges of weight at most $2^{L-\lceil \log k \rceil}$ added to the minimum spanning trees of G, Δ is the increase in the weight of the minimum spanning trees caused by the inclusion of edges of weight larger than $2^{L-\lceil \log k \rceil}$ into them, and $k_1 \leq k$ is the number of edges that were replaced in the minimum spanning trees. It is easy to see that $w(T'_{s\&d}) \geq w(T) + \Delta - k_1$. Note also that $w(T'_{s\&d}) \geq 2^L$ because at least one non-redundant edge has weight 2^L. From these inequalities we get

$$\begin{aligned} \frac{w(T_{s\&d})}{w(T'_{s\&d})} &\leq \frac{w(T) + \Delta - k_1 + k_0 2^{L-\lceil \log k \rceil}}{w(T'_{s\&d})} \\ &\leq 1 + \frac{k_0 2^{L-\lceil \log k \rceil}}{2^L} \leq 1 + \frac{k_0}{2^{\lceil \log k \rceil}} \leq 2. \end{aligned}$$

□

For a graph $G = (V, E, w)$ *with arbitrary non-negative weights, our modified algorithm* slice_n_dice *first rounds down the weight of every edge to its nearest power of 2. If any edge has weight smaller* *than 1, then it multiplies all the weights by 2^s, where 2^{-s} is the smallest edge weight resulting after the rounding. This scaling of the edge weights does not affect the performance ratio of the algorithm and leaves edge weights that are integer powers of 2.*

THEOREM 2.4. *The performance ratio of the modified algorithm slice_n_dice described above is $r \geq 1/(4 + 8\log(k+1))$.*

Proof. The Theorem follows form Lemmas 2.2, 2.3, and 2.5. □

We turn our attention now to the problem of computing the weight w_i of a largest non-redundant edge in a graph G. An edge e is non-redundant if there is a cut separating its endpoints that has at most k edges of weight smaller than $w(e)$. Hence, the desired weight w_i can be found by sorting the edges non-decreasingly by weight and then performing a binary search over the weights to find the smallest weight w_i for which G_{w_i} has a minimum cut of size larger than k.

THEOREM 2.5. *The time complexity of the modified slice_n_dice algorithm is $O(k \log n(m + kn\log n))$.*

Proof. We use Gabow's algorithm [10] to implement the procedure for finding the weight of a largest non-redundant edge in a graph. □

2.4 NP-completeness

In this section we prove that the problem of selecting k edges that when removed from a graph maximize the weight of its minimum spanning trees is NP-hard even if the weights of the edges are either 0 or 1. The reduction is from the minimum k-cut problem.

THEOREM 2.6. *Let $G = (V, E, w)$ be a graph whose edges have associated a weight of value either 0 or 1, and let M be a non-negative integer value. The problem of deciding if there is a set S of at most k edges such that any minimum spanning tree of the graph $(V, E - S, w)$ has weight at least M, is NP-complete.*

Proof. The problem is clearly in NP. To show that it is NP-hard we give a reduction from the minimum k-cut problem. Given an undirected graph $G_H = (V_H, E_H, w_H)$, build a graph $G =$

(V, E, w) by combining G_H and a clique $G_Z = (V_Z, E_Z, w_Z)$ of size $k+2$ as follows: $V = V_H \cup V_Z$ and $E = E_H \cup E_Z \cup \{(z_i, v_j) \mid z_i \in V_Z \text{ and } v_j \in V_H\}$. Assign weight 0 to the edges in E_H and weight 1 to the other edges in E. Observe that any cut of G has at least $k+1$ edges.

We now prove that G_H can be partitioned into j connected components by deleting k edges from it if and only if it is possible to remove k edges from G in such a way that any minimum spanning tree of the resulting graph has weight $j+k+1$.

Suppose first that there is a set S of k edges that partitions G_H into j components. If the edges in S are removed from G, then any minimum spanning tree of the resulting graph has to use j edges of weight 1 to connect the j components of G_H to V_Z and $k+1$ edges of weight 1 to connect the vertices of V_Z among themselves.

Suppose now that there is a set S of k edges that when removed from G causes any minimum spanning tree T of the resulting graph to have weight $j+k+1$. Observe that at least j of the edges of weight 1 in T have to be incident to vertices in G_H and, thus, that the set S must split G_H into j components. □

3 Continuous Version

In this section we study the problem of determining the maximum increase in the weight of the minimum spanning trees of a graph caused by finite increases in the weights of its edges. Every edge e has associated a rate $c(e)$ and we assume a cost function that charges $c(e) * \delta$ for increasing the weight of e by δ. The total cost of the changes in the weights of the edges must not exceed some given budget value B. In Figure 3 we show how to spend optimally a budget of value 4 increasing edge weights. Each edge e in the graph shown is labeled by an ordered pair consisting of its weight $w(e)$ and its rate $c(e)$. The maximum increase in the weight of the minimum spanning trees achievable by the expenditure of a budget $B = 4$ is obtained by raising from 1 to 2 the weights of the edges drawn in bold. Note that as long as the weights of the edges in bold are smaller than 2 at least two of them must be in every minimum spanning tree. Thus, the weight of the minimum

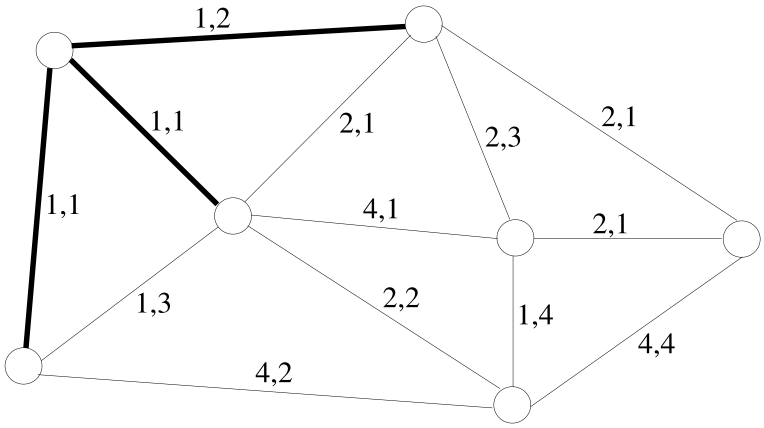

Figure 3: An instance of the continuous robustness problem, with a best set of edges for raising weights shown in bold.

spanning trees increases from 8 to 10.

Let $\mathcal{T}$ be the set of minimum spanning trees of weighted graph $G = (V, E, w)$. The set $\mathcal{T}$ may change when the weights of some of the edges in G are increased, because some trees may have a large weight increase and leave $\mathcal{T}$, while other trees not in $\mathcal{T}$ may have less or no change in their weights and become members.

Given a subset $S \subseteq E$ of the edges, we define coverage (S, G) *as the minimum number of edges that any minimum spanning tree of G must share with S. Our approach for the continuous robustness problem consists in raising the weights of edges in some appropriately chosen set S, up to the point where* coverage (S, G) *decreases. Then we choose a new set S and repeat this process until the total budget spent has value B.*

We say that the weights of the edges in a set S are lifted *when the weight of every edge in S is increased by the same amount. Let* $\text{tolerance}(S, G)$ *be the maximum amount by which the weights of the edges in S can be lifted until* coverage (S, G) *decreases. Let $c(S)$ denote the sum of the costs of the edges in S. Define the* cost per unit increment *of S, denoted as* $\text{inc\_cost}(S, G)$, *as* $c(S)/\text{coverage}\ (S, G)$. *Clearly, for any value $\varepsilon \leq \text{tolerance}(S, G)$, the cost of increasing the weight of any minimum spanning tree of G by at least ε when only the weights of the edges in S are raised is* $\varepsilon * \text{inc\_cost}(S, G)$.

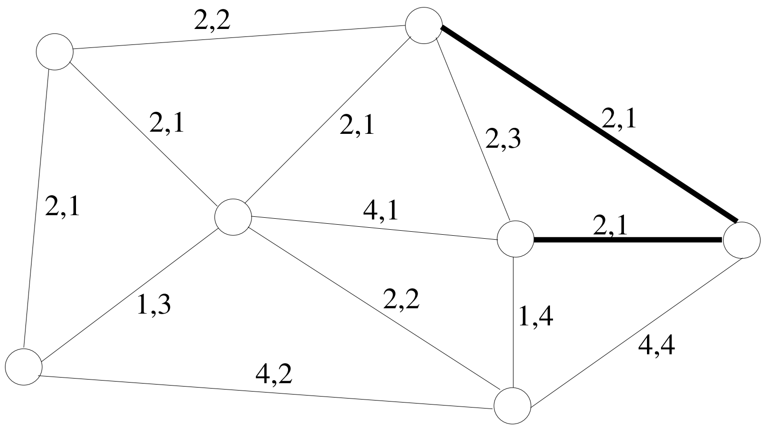

Figure 4: Second iteration of *raise_mst*, with a best choice of edges to be raised shown in bold.

The following greedy algorithm solves the continuous robustness problem for minimum spanning trees.

Algorithm raise_mst $(G = (V, E, w), B)$

balance $\leftarrow B$

wmst $\leftarrow$ *weight of a minimum spanning tree of* G

while balance > 0 **do**

Find a set $S \subseteq E$ *that minimizes* inc_cost(S, G).

$\Delta \leftarrow \min \{\, \text{tolerance}(S, G), \text{balance}/c(S)\, \}$

Lift the weights of the edges in S *by* Δ.

balance $\leftarrow$ balance $- \Delta * c(S)$

end while

increase $\leftarrow$ *(weight of a minimum spanning tree of* G*)* $-$ wmst

Output increase.

Figure 4 shows the second iteration of raise_mst *for the same graph of Figure 3. In the first iteration the set of smallest cost per unit increment includes the edges in bold shown in Figure 3. This set has a* coverage *of value 2 and cost per unit increment of value* $(1 + 1 + 2)/2 = 2$. *Our algorithm* raise_mst *increases the weights of these edges from 1 to 2. At this point, the* coverage *of the set decreases to 1, and its cost per unit increment goes up to 4. Thus, in the second iteration* raise_mst *selects the edges in bold shown in Figure 4 because they have a* coverage *of value 1 and cost per unit increment of value only* $(1+1)/1 = 2$. *Note as long as the weights of these edges are smaller than 4 at least one of them belongs to every minimum spanning tree of the graph.*

We now establish correctness for raise_mst. *Given a graph G and a budget value B, consider an optimal solution w^* for the robustness problem. Let $w^*(e)$ denote the weight of any edge e in this optimal solution. Assume that $\mathcal{A}$ is an algorithm that finds this optimal solution and that $\mathcal{A}$ builds it in several stages: in each stage a set of edges is selected and the weights of its edges are lifted by some amount, in such a way that at the end of all stages the weight of every edge $e \in E$ is $w^*(e)$.*

We show that the increase in the weight of the minimum spanning trees of G achieved by raise_mst *is the same as that produced by $\mathcal{A}$. For this purpose only, it is convenient to imagine that whenever $\mathcal{A}$ or* raise_mst *increase the weight of an edge e by some amount Δ, they do not just add Δ to $w(e)$, but gradually rise the weight from $w(e)$ to $w(e)+\Delta$. This allows us to consider the partial solutions built by the algorithms when any fraction b of the budget B has been spent by them increasing edge weights. For any value b, $0 \leq b \leq B$, let S_b be the set of edges whose weights are being lifted by* raise_mst *when the budget spent by it reaches value b and $G(b) = (V, E, w_b)$ describe the graph at that precise moment. Similarly, let $S^*_b, 0 \leq b \leq B$, be the set of edges whose weights are being lifted by $\mathcal{A}$ when the budget spent by it reaches value b and $G^*(b) = (V, E, w^*_b)$ be the graph at that moment.*

To show that raise_mst *finds an optimal solution, we only have to prove that* $\mathrm{inc\_cost}(S_b, G(b)) = \mathrm{inc\_cost}(S^*_b, G^*(b))$, *for all $0 \leq b \leq B$. Let* $\mathrm{sm\_eq}(e, G)$ *be the subset of edges in G of weight no larger than the weight of the edge e.*

Lemma 3.1. *Let $G = (V, E, w), G' = (V, E, w')$ be undirected graphs and $S \subseteq E$. If for every edge $e \in S$, $sm\_eq(e, G') \subseteq sm\_eq(e, G)$ then coverage $(S, G) \leq$ coverage (S, G').*

Proof. Let T' be a minimum spanning tree of G' for which $|T' \cap S| =$ *coverage* (S, G'). If T' is not a minimum spanning tree of G, it can be transformed into one by taking every edge

$e \in E - T'$, one at a time, including e in T' and removing the element of largest weight in the unique cycle that is formed.

Note that by including in T' any edge $e \in S - T'$, we create a cycle C in which all edges $f \in C$ have weights $w'(f) \leq w'(e)$ because T' is a minimum spanning tree of G'. Moreover, since $f \in \textit{sm\_eq}(e, G') \subseteq \textit{sm\_eq}(e, G)$ then $w(f) \leq w(e)$ and thus it is possible to construct a minimum spanning tree of G that does not include any element from $S - T'$. This means that *coverage* $(S, G) \leq$ *coverage* (S, G'). $\square$

Lemma 3.1 suggests a way in which $\mathcal{A}$ can select the sets S_b^* that makes it easy to compare their costs per unit increment with those of the sets S_b. We would like that during the construction of the optimal solution, $\mathcal{A}$ always chooses S_b^* to include only edges e for which $\mathrm{sm\_eq}(e, G(b)) \subseteq \mathrm{sm\_eq}(e, G^*(b))$. *If this is possible, then by Lemma 3.1,* $\mathrm{inc\_cost}(S_b^*, G(b)) \leq \mathrm{inc\_cost}(S_b^*, G^*(b))$. *Since* raise_mst *always chooses the set S_b with smallest cost per unit increment in $G(b)$, then it would follow that* $\mathrm{inc\_cost}(S_b, G(b)) \leq \mathrm{inc\_cost}(S_b^*, G^*(b))$.

Let us specify how $\mathcal{A}$ can select the sets S_b^*. Define $w_b^*(e) = min\{w^*(e), w_b(e) + \Delta_b\}$ for all $e \in E$, where Δ_b is a constant selected so that the total cost of bringing the weight of every edge e from its initial value $w(e)$ up to $w_b^*(e)$ is exactly b. Note that $w_b^*(e) \leq w^*(e)$ and so $w_B^*(e) = w^*(e)$ for all $e \in E$. Select S_b^* to include all of edges e for which $w_b^*(e) = w_b(e) + \Delta_b \leq w^*(e)$ and $w_{b-\epsilon}^*(e) < w^*(e)$ for all $b \geq \epsilon > 0$.

Theorem 3.1. *Algorithm raise_mst finds the maximum increase in the weights of the minimum spanning trees of a graph attainable by spending a finite budget B augmenting edge weights.*

Proof. Consider any value $b, 0 \leq b \leq B$. Let $e_1 \in S_b^*$ and $e_2 \in E$. By definition of S_b^* and $w_b^*(e)$ it follows that $w_b^*(e_1) = w_b(e_1) + \Delta_b$ and $w_b^*(e_2) \leq w_b(e_2) + \Delta_b$. Hence if $w_b(e_2) \leq w_b(e_1)$ then $w_b^*(e_2) \leq w_b^*(e_1)$. Therefore, $\textit{sm\_eq}(e_1, G(b)) \subseteq \textit{sm\_eq}(e_1, G^*(b))$, and by the above discussion $\textit{inc\_cost}(S_b, G(b)) \leq \textit{inc\_cost}\ (S_b^*, G^*(b))$. $\square$

3.1 Finding a Set of Edges of Smallest Cost Per Unit Increment

Consider first the problem of finding a set of edges of smallest cost per unit increment in a graph $G = (V, E, w)$ *in which all edges have the same weight. Since in such a graph every spanning tree has minimum weight, then any set* $S \subseteq E$ *with* $\text{coverage}(S, G) > 0$ *must partition the graph into* $\text{coverage}(S, G) + 1$ *components. This means that the smallest cost per unit increment of any set* $S \subseteq E$ *is equal to the* strength of G *[4], defined as* $\text{strength}(G) = \min\{c(S)/\text{comps}(S, G) \mid \emptyset \neq S \subseteq E\}$, *where* $\text{comps}(S, G)$ *is the number of connected components of* $(V, E - S, w)$ *minus one.*

In the case of a graph G *with an arbitrary edge weight function we can still use the efficient graph strength algorithms [3, 11] to compute a set of smallest cost per unit increment. To see this, consider the following easily verifiable property of the minimum spanning trees of* G*. Let* $w_1, w_3, \ldots, w_p$ *be the different edge weights in* G *and* $\tilde{G}_{w_i} = (\tilde{V}_{w_i}, \tilde{E}_{w_i}, \tilde{w})$ *be the graph obtained by contracting from* G_{w_i} *all edges of weight smaller than* w_i.

Property 3.1. *T is a minimum spanning tree of G if and only if it contains a spanning tree for every non-trivial component of each* $\tilde{G}_{w_i}$.

From Property 3.1 it follows that for any $S \subseteq E$, $\text{coverage}(S, G) = \sum_{i=1}^{p} \text{coverage}(S, \tilde{G}_{w_i})$. *This means that if* S *is a set of smallest cost per unit increment in* G *then* $\text{inc\_cost}(S, G) = \sum_{i=1}^{p} c(S \cap \tilde{E}_{w_i}) / \sum_{i=1}^{p} \text{coverage}(S, \tilde{G}_{w_i}) = c(S \cap \tilde{E}_{w_j})/\text{coverage}(S, \tilde{G}_{w_j})$ *for any* w_j *such that* $S \cap \tilde{E}_{w_j} \neq \emptyset$*. Therefore, every graph has a subset of edges of smallest cost per unit increment in which all of the edges have the same weight.*

Suppose that $S \subseteq E$ *is a set of smallest cost per unit increment in* G *and that every edge of* S *has weight* w_i*. By the above discussion we know that* $\text{inc\_cost}(S, G) = c(S)/\text{comps}(S, \tilde{G}_{w_i}) = \text{strength}(\tilde{G}_{w_i})$*. Hence we can find a set of smallest cost per unit increment in* S *by computing the strength of each graph* $\tilde{G}_{w_i}$ *and selecting the set corresponding to the minimum strength value. The algorithm is the following.*

Algorithm optimal_subset $(G = (V, E, w))$

$s_{min} \leftarrow \infty$

$G' \leftarrow G$

while *there is at least one edge in* G' **do**

Let w_i *be the weight of a smallest weight edge in* G'*.*

if $\text{strength}(G'_{w_i}) < s_{min}$ **then**

$s_{min} \leftarrow \text{strength}(G'_{w_i})$

$S \leftarrow$ *any set of edges for which* $\text{inc\_cost}\,(S, G'_{w_i}) = \text{strength}\,(G'_{w_i})$

end if

Contract every edge of weight w_i *in* G' *and remove self loops.*

Let G' *be the resulting graph.*

end while

Output S*.*

LEMMA 3.2. *Algorithm optimal_subset runs in* $O(n^2 m \log(n^2/m))$ *time.*

Proof. Algorithm *optimal_subset* solves at most m graph strength subproblems. In each one of these subproblems a subgraph of m_i edges and n_i vertices is considered, where $m_i \leq |E_{w_i}|$ and n_i is the number of vertices of non-zero degree in G'_{w_i}. Let p be the number of different edge weights in G. In the contraction step of the i-th iteration of the while-loop at least $n_i/2$ vertices are removed from G', hence $\sum_{i=1}^{p} n_i \leq 2n$. The $O(m)$ graph strength subproblems created by *optimal_subset* can be solved in at most the same time required to find the strength of the graph $\tilde{G}$ formed by the union of the graphs G'_{w_i} considered in *optimal_subset.* Since $\tilde{G}$ has at most $2n$ vertices and m edges, we can use the graph strength algorithm in [3] or in [11] to implement *optimal_subset* in $O(n^2 m \log(n^2/m))$ time. □

In order to compute the time complexity of raise_mst*, we need to know the complexity of computing* $\text{tolerance}\,(S, G)$ *for any set* S *of smallest cot per unit increment. If all of the edges in a set* S *have the same weight, say* w_i*, then* $\text{coverage}\ (S, G)$ *is equal to* $\text{comps}\,(S, G_{w_i})$*. This fact makes it possible to compute* $\text{tolerance}(S, G)$ *efficiently for a set* S *of edges of the same weight.*

Simply find the smallest edge weight $w_j > w_i$ *for which* $\text{comps}(S, G_{w_j}) < \text{comps}(S, G_{w_i})$. *If such a weight exists then* $\text{tolerance}\ (S, G) = w_j - w_i$, *otherwise* $\text{tolerance}(S, G) = \infty$. *To find* w_j *we proceed as follows. Build the graph* G'_{w_i} *obtained by removing the edges in* S *from* G_{w_i}. *Take, one by one, the edges of weight larger than* w_i *until we find one that connects two of the components of* G'_{w_i}. *If the edges of* G *are sorted non-decreasingly by weight, then we can compute* $\text{tolerance}(S, G)$ *in* $O(m)$ *time.*

LEMMA 3.3. *Algorithm raise_mst performs at most* $(m-1)(n-1)+1$ *iterations.*

Proof. Let w_i be the weight of some edge in G. Recall that all minimum spanning trees of G have the same number of edges of weight w_i. Therefore, the number of iterations of *raise_mst* in which at least one edge of weight w_i is added to every minimum spanning tree of G is no larger than the number of iterations in which at least one edge of weight smaller than w_i is removed from every minimum spanning tree of G.

Since there are at most $n-1$ edges of weight smaller than w_i in a tree, then in at most $n-1$ iterations an edge of weight w_i is added to the minimum spanning trees. Let w_s and w_ℓ be the smallest and largest edge weights respectively in the graph G. It is not possible to add edges of weight w_s into the minimum spanning trees of G. Moreover, in at most one iteration *raise_mst* adds edges of weight larger than w_ℓ to every minimum spanning tree of G. Therefore, the total number of iterations of *raise_mst* is at most $(n-1)(m-1)+1$. □

Let $f(b)$ *be the maximum weight of the minimum spanning trees of a graph attainable by spending a budget* b *increasing the weights of its edges.*

LEMMA 3.4. *The function* f *is concave, non-decreasing and piecewise linear with* $O(mn)$ *breakpoints.*

Proof. The proof of Theorem 3.1 shows that algorithm *raise_mst* "marches" along the curve f. Hence f is a piecewise linear function since its slope throughout an iteration of the while-loop of *raise_mst* is constant and equal to *coverage* $(S, G)/c(S)$, where S is the set of edges selected in the iteration. Lemma 3.3 gives a bound on the number of breakpoints of f.

To show that f is concave and non-decreasing, let us consider any two consecutive iterations of the while-loop of *raise_mst*. Suppose that at the beginning of the first iteration the edge weights are as described by the function w_1. In that iteration *raise_mst* selects a set S_1 and lifts the weights of its edges by some amount. Let w_2 describe the resulting weights. Suppose now that in the next iteration a set S_2 is selected and that $\mathit{coverage}(S_1,(V,E,w_1))/c(S_1) < \mathit{coverage}(S_2,(V,E,w_2))/c(S_2)$. This situation can only happen if $\mathit{coverage}(S_2,(V,E,w_2)) > \mathit{coverage}(S_1,(V,E,w_1))$ or, in other words, if lifting the weights of the edges in S_1 increases the coverage of S_2. But, this would mean that $\mathit{coverage}(S_1 \cup S_2,(V,E,w_1))/c(S_1 \cup S_2) > \mathit{coverage}(S_2,(V,E,w_1))/c(S_2)$ and thus that S_1 was not the set of smallest cost per unit increment in the first iteration. $\square$

THEOREM 3.2. *The time complexity of algorithm raise_mst is $O(n^3 m^2 \log(n^2/m))$.* $\square$

4 Conclusion

We have formulated the robustness problem for minimum spanning trees, which consists in measuring the sensitivity of their weight to edge removals and increases in the weights of the edges. The discrete version of the problem, which deals with edge removals, has been shown to be NP-complete even if the weights of the edges are 0 or 1. We have presented an $\Omega(1/\log k)$-approximation algorithm for this problem that uses as main subroutine a $(1/2)$-approximation algorithm for the maximum components problem. Interestingly, the continuous version of the problem, which deals with edge weight increases, is solvable in strongly polynomial time despite the fact that a possibly exponential number of spanning trees have to be considered. We have presented a greedy algorithm that solves this problem in $O(m^2 n^3 \log(n^2/m))$ time.

Some of the techniques presented in this paper for the continuous version of the problem are powerful enough to be used in the solution of a more general robustness problem for minimum weight bases of matroids. We will present our results for this latter problem in a forthcoming paper.

5 Acknowledgments

The authors thank Samir Khuller for his useful comments on earlier versions of this paper and David Eppstein for suggesting the tighter bound on the number of iterations of raise_mst *stated in Lemma 3.3.*

References

[1] K.R. Baker, "Introduction to Sequencing and Scheduling," John Wiley, New York, 1974.

[2] M.O. Ball, B.L. Golden and R.V. Vohra, Finding the most vital arcs in a network, *Operations Research Letters* **8** (1989), 73–76.

[3] E. Cheng and W.H. Cunningham, A faster algorithm for computing the strength of a network, *Information Processing Letters* **49** (1994), 209–212.

[4] W.H. Cunningham, Optimal attack and reinforcement of a network, *Journal of the ACM* **32** (1985), 549–561.

[5] E. Dalhaus, D. S. Johnson, C. H. Papadimitriou, P. Seymour and M. Yannakakis, The complexity of the multiway cuts, *SIAM Journal on Computing* **23** (1994), 864–894.

[6] G.N. Frederickson and D. J. Guan, Preemptive ensemble motion planning on a tree, *SIAM Journal on Computing* **21** (1992), 1130–1152.

[7] G.N. Frederickson and R. Solis-Oba, Increasing the weight of minimum spanning trees, *in* "Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 1996," pp. 539–546.

[8] G.N. Frederickson and R. Solis-Oba, Efficient algorithms for robustness in matroid optimization, Manuscript, 1996.

[9] D.R. Fulkerson and G.C. Harding, Maximizing the minimum source-sink path subject to a budget constraint, *Mathematical Programming* **13** (1977), 116–118.

[10] H.N. Gabow, A matroid approach to finding edge connectivity and packing arborescences, *Journal of Computer and System Sciences* **50** (1995), 259–273.

[11] H.N. Gabow, Algorithms for graphic polymatroids and parametric s-sets, *in* "Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 1995," pp. 88–97.

[12] M.R. Garey and D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W.H. Freeman, New York, 1979.

[13] O. Goldschmidt and D.S. Hochbaum, Polynomial algorithm for the k-cut problem, *Mathematics of Operations Research* **19** (1994), 24–37.

[14] R.L. Graham and P. Hell, On the history of the minimum spanning tree problem, *Annals of the History of Computing* **7** (1985), 43–57.

[15] L. Hsu, R. Jan, Y. Lee and C. Hung, Finding the most vital edge with respect to minimum spanning tree in weighted graphs, *Information Processing Letters* **39** (1991), 277–281.

[16] L. Hsu, P. Wang and C. Wu, Parallel algorithms for finding the most vital edge with respect to minimum spanning tree, *Parallel Computing* **18** (1992), 1143–1155.

[17] F.K. Hwang, On Steiner minimal trees with rectilinear distance, *SIAM J. Appl. Math.* **30** (1976), 104–114.

[18] K. Iwano and N. Katoh, Efficient algorithms for finding the most vital edge of a minimum spanning tree, *Information Processing Letters* **48** (1993), 211–213.

[19] S. Khuller and U. Vishkin, Biconnectivity approximations and graph carvings, *Journal of the ACM* **41** (1994), 214–235.

[20] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, "The Traveling Salesman Problem," John Wiley & Sons, 1985.

[21] K. Lin and M. Chern, The most vital edges in the minimum spanning tree problem, *Information Processing Letters* **45** (1993), 25–31.

[22] K. Malik, A.K. Mittal and S.K. Gupta, The k most vital arcs in the shortest path problem, *Operations Research Letters* **8** (1989), 223–227.

[23] C.A. Phillips, The network inhibition problem, *in* "Proceedings of the 25th Annual ACM Symposium on Theory of Computing, 1993," pp. 776–785.

[24] H. Saran and V.V. Vazirani, Finding k-cuts within twice the optimal, *SIAM Journal on Computing* **24** (1995), 101–108.

[25] M. Schwartz and T.E. Stern, Routing techniques used in computer communication networks, *IEEE Transactions on Communications* **28** (1980), 539–552.

[26] D.R. Shier, Arc tolerances in shortest path and network flow problems, *Networks* **10** (1980), 277–291.

[27] R.E. Tarjan, Sensitivity analysis of minimum spanning trees and shortest path trees, *Information Processing Letters* **14** (1982), 30–33.

[28] H. Takahashi and A. Matsuyama, An approximate solution for the Steiner problem in graphs, *Math. Japonica* **24** (1980), 573–577.