Robustness Analysis in Combinatorial Optimization

Greg N. Frederickson * Roberto Solis-Oba †

Abstract

The *robustness function* of an optimization problem measures the maximum change in the value of its optimal solution that can be produced by changes of a given total magnitude on the values of the elements in its input. The problem of computing the robustness function of matroid optimization problems is studied under two cost models: the *discrete model*, which allows the removal of elements from the input, and the *continuous model*, which permits finite changes on the values of the elements in the input.

For the discrete model, an $O(\log k)$-approximation algorithm is presented for computing the robustness function of minimum spanning trees, where k is the number of edges to be removed. The algorithm uses as key subroutine a 2-approximation algorithm for the problem of dividing a graph into the maximum number of components by removing k edges from it.

For the continuous model, a number of results are presented. First, a general algorithm is given for computing the robustness function of any matroid. The algorithm runs in strongly polynomial time on matroids with a strongly polynomial time independence test. Faster algorithms are also presented for some particular classes of matroids: (1) an $O(n^3m^2\log(n^2/m))$-time algorithm for graphic matroids, where m is the number of elements in the matroid and n is its rank, (2) an $O(mn(m+n^2)|E|\log(m^2/|E|+2))$-time algorithm for transversal matroids, where $|E|$ is a parameter of the matroid, (3) an $O(m^2n^2)$-time algorithm for scheduling matroids, and (4) an $O(m\log m)$-time algorithm for partition matroids. For this last class of matroids an optimal algorithm is also presented for evaluating the robustness function at a single point.

1 INTRODUCTION

The robustness function of a (minimization) optimization problem measures the maximum increase in the value of the optimal solution that can be produced by changes of a given total "magnitude" on the values of the elements in its input. The robustness function of a maximization problem can be defined in a similar way. In this paper we are interested in computing the robustness functions of combinatorial optimization problems. Combinatorial optimization is an

*Department of Computer Science, Purdue University, West Lafayette IN 47907. Research of this author partially supported by the National Science Foundation under grant CCR-9322501. Email: gnf@cs.purdue.edu.

†Max Planck Institut für Informatik. Im Stadtwald, 66123 Saarbrücken. Most of this research was done while this author was a graduate student at the Department of Computer Sciences at Purdue University. Research of this author partially supported by NSF grant CCR-9322501, by CONACyT-Fulbright and by Universidad Autonoma Metropolitana. Email: solis@mpi-sb.mpg.de.

exciting branch of mathematics, whose importance derives from its large number of applications and from the elegant algorithmic techniques that have been developed to solve combinatorial optimization problems [21, 55, 83, 99]. A combinatorial optimization problem requires finding the maximum or minimum of a certain function, called the objective function. The input of the problem consists of a finite set of discrete "elements" each one having some associated value.

Consider a combinatorial optimization problem defined on a dynamic environment. The dynamic nature of the input might cause frequent changes on the value of the solution of the problem. Recomputing the solution could be expensive, and thus, one might desire to do it infrequently. The robustness function for the problem can help decide when the updates need to be performed. One might decide to compute the solution for the problem, and recompute it only after the robustness function indicates that the changes in the input have had a large effect on the value of the actual solution for the problem.

Similar sensitivity issues arise in the solution of a problem for which the exact values of the input data are not known, and hence estimates must be used. A measure of the quality of the solution computed with the estimates is given by determining how sensitive the solution is to changes in the values of the estimates.

Sensitivity analysis provides only a partial answer to the above questions. The sensitivity analysis of an optimization problem determines for each element of an optimal solution, the magnitude of the largest perturbation in the value of the element that would not affect the optimality of the solution [56, 88, 106, 111]. If simultaneous changes in the values of all the elements in the input are considered, like required in the above two situations, then sensitivity analysis does not suffice [36, 38, 42, 61, 100]. Our concept of robustness function of an optimization problem generalizes the notion of sensitivity analysis by considering changes in the values of all the elements in the input of the problem.

1.1 Matroid Optimization Problems

In this paper we focus our study of robustness functions on the large class of matroid optimization problems. Matroids have an elegant and strikingly simple structure that captures the essence of many problems that can be solved using greedy algorithms [9, 83, 95, 98, 115]. Several fundamental problems in graph theory [1, 22, 54, 83, 90], scheduling [43, 83], mathematical programming [85, 98, 115], electrical networks [85, 103], mechanics [44, 85, 103], and operations research [8, 69, 85] have an inherent matroid structure, and can thus be solved using tools from matroid theory.

A matroid $M = (E, \mathcal{I})$ consists of a finite set of elements E and a family $\mathcal{I}$ of *independent* subsets of E satisfying well known axioms (see e.g. [115]). A fundamental property of a matroid is that all maximal independent sets, called *bases*, have the same cardinality, called the *rank* of the matroid. A base of a matroid (or a minimum or maximum weight base, if the elements have associated weights) can be found using a greedy algorithm (see e.g. [115]).

There is an important practical consequence for a problem to have a matroid structure: results from matroid theory can be used to discover structural properties of the set of possible solutions of a problem that might be difficult to show otherwise. Many combinatorial optimization problems have been solved only after their connection with matroid theory was established [25, 103, 86], and efficient solutions to some other problems heavily rely on their inherent matroid structure [43, 46, 47, 85].

Some fundamental problems of diverse areas of research are matroid optimization problems that can be solved using the greedy algorithm. In graph theory two such problems are finding a minimum spanning tree and finding a maximum cardinality matching. In scheduling, a problem in this class is to find a largest set of unit-time jobs, with integer release times and deadlines, that can be scheduled for execution in a single processor. In mathematical programming, the problem of finding a pivot rule for the simplex method that avoids cycling. In mechanics, the problem of deciding if a structure formed by rods and joins is rigid. In electrical networks, the problem of selecting the smallest set of electrical equations that uniquely determine the values of the electrical currents in all branches of a circuit.

Other important prototypical matroid optimization problems are the assignment problem, the maximum weight matching problem, the problem of finding a minimum weight branching in a directed graph, finding a spanning tree with degree constraint on some vertices, and the problem of finding the maximum number of edge disjoint spanning trees in a graph. A fundamental problem in graph theory whose fastest known solution uses tools from matroid theory is the problem of computing the edge connectivity of a graph [47].

In the rest of this section we describe some specific matroids for which we have designed algorithms for computing their robustness functions. Throughout, we will let m be the number of elements in the ground set of a matroid and n be its rank. When describing graph algorithms, we denote by m the number of edges in the graph and by n the number of vertices. This use of m is consistent since in a graphic matroid the ground set is formed by all the edges in the graph. However, our use of n is inconsistent because the rank of a graphic matroid is one less than the number of vertices. The context will make it clear when n refers to the rank of a matroid or to the number of vertices in a graph.

1.1.1 Graphic Matroids

Given an undirected graph $G = (V, E)$, a graphic matroid $M = (E, \mathcal{I})$ has ground set equal to the set of edges in G, and its independent sets are the subsets of edges that do not form cycles. If the elements in E have non-negative weights, then a minimum weight base of M is a minimum spanning tree (forest) of G.

Finding a minimum spanning tree is a fundamental problem in graph theory. This seminal problem has a long and rich history that goes back to the beginning of the century [11]. Graham and Hell have written an excellent survey paper on the history of the problem up to 1985 [54]. In the last 20 years many algorithms have been designed to compute efficiently a minimum spanning tree. The fastest sequential algorithm for the problem is due to Gabow et al. [45] and it runs in $O(m \log \beta(m,n))$ time, where $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}$. Karger et al. [72] have designed a randomized linear time algorithm for finding a minimum spanning tree.

Fredman and Willard [40] also discovered a linear time algorithm for the problem, but they consider a model of computation that allows bit manipulations on the binary representation of edge weights.

Minimum spanning trees find applications in areas as diverse as network design [1], numerical methods [12], image processing [54], data bases [54], biology [76], and archaeology [59]. The importance of minimum spanning trees comes not only from its wealth of applications, but from their structure, typical of matroid optimization problems [83]. Graphic matroids are among the first matroids that were ever studied [116, 113], and much of the early development in matroid theory came from the study of graphic matroids. As we mentioned above a fundamental property of matroids is that a minimum weight base can be found using a greedy strategy. This property was first discovered for minimum spanning trees [11, 81], and then extended by Rado to arbitrary matroids [102].

Many of the algorithmic techniques initially created to solve the minimum spanning tree problem have found applications in other problems as well. The desire for finding faster ways for computing minimum spanning trees has also led to the development of sophisticated data structures, some of which have been used to speed up the solutions to many other combinatorial optimization problems.

1.1.2 Transversal Matroids

A *transversal matroid* can be defined in terms of matchings in bipartite graphs. A *bipartite graph* $G = (D \cup D', A)$ has its vertices divided in two disjoint sets D and D', and every edge $e \in A$ has one endpoint in D and the other in D'. A *matching* of G is a set of edges T such that no two edges in T share a common endpoint. Given a set of vertices $S \subseteq D$ and a matching T, we say that the set S is covered by matching T if T has one edge incident to every vertex in S.

Given a bipartite graph $G = (D \cup D', A)$, a transversal matroid $M = (D, \mathcal{I})$ is defined as having ground set equal to the set of vertices on one side of G. A subset S of E is independent in M if and only if it can be covered by a matching. A matching of maximum cardinality in G defines a base of M.

Finding a maximum cardinality matching in a graph is a classical problem in graph theory [112]. There is an interesting relationship between maximum cardinality matchings in bipartite graphs and maximum flows, that was first noticed by Ford and Fulkerson [33]. They showed that an integer maximum flow of a bipartite graph with unit capacity edges defines a maximum cardinality matching. This observation led to the first algorithm for finding maximum matchings in bipartite graphs. The algorithm was later improved by Hopcroft and Karp who gave an $O(\sqrt{n}m)$ time algorithm [62] for the problem.

If the edges of the bipartite graph have non-negative weights, the minimum weight matching problem consists in finding a maximum cardinality matching in which the sum of the weights of the edges is minimum. This problem is equivalent to the assignment problem and it has applications in resource allocation problems.

Matchings in bipartite graphs are related to the so called *systems of distinct representatives*, an important topic in combinatorial analysis. Typical of the viewpoint of combinatorial analysis is a classic theorem of P. Hall which states necessary and sufficient conditions for the existence of a system of distinct representatives, or equivalently of a matching of maximum cardinality in a bipartite graph. Given a bipartite graph $G = (D \cup D', A)$, and a set $S \subseteq D$, let $\mathcal{N}(S) = \{v \in D' \mid v \text{ is adjacent to some vertex in } S\}$ be the set of neighbors of vertices in S. Hall's Theorem states that G has a matching covering the vertices of S if and only if $|\mathcal{N}(S')| \geq |S'|$ for all $S' \subseteq S$. We use this result in our algorithm for computing the robustness function of a transversal matroid.

1.1.3 Scheduling Matroids

Consider the following scheduling problem. Let $J = \{j_1, j_2, \ldots, j_m\}$ be a set of jobs. Each job j_i requires one unit of processing time and it cannot be preempted, i.e., the execution of the job cannot be interrupted. Job j_i has integer release time r_i and deadline d_i, with $d_i > r_i$. The problem is to select the largest set of jobs that can be executed on a single processor so that no job is started before its release time and all jobs are completed by their deadlines.

A *convex bipartite graph* $G = (D \cup D', A)$ is a bipartite graph for which there is an indexing for the vertices in D' such that every vertex $v \in D$ is adjacent to consecutively indexed vertices from D'. Interestingly, the above scheduling problem can be reduce to a matching problem on a convex bipartite graph $G_J = (J \cup J', A)$. We think of the ith vertex of J as job j_i, and the ith vertex of J' as the unit time interval from $i-1$ to i. For each job j_i, graph G_J has an edge from j_i to every unit time interval between r_i and d_i. It can be proved that a maximum cardinality matching in G_J corresponds to a valid scheduling of a largest subset of jobs from the set of jobs J [43].

A *scheduling matroid* $M = (J, \mathcal{I})$ has a set of jobs J as its ground set, and a subset of jobs is independent in M if and only if there is a valid non-preemptive scheduling for them on a single processor. By the previous discussion on convex bipartite graphs, it is clear that a scheduling matroid is a special kind of transversal matroid.

The above scheduling problem is perhaps the most fundamental problem in scheduling theory [18, 41, 94]. It can be solved with Jackson's earliest deadline first rule [71]. This rule schedules at each unit time interval one job that has not been scheduled yet, has release time at least equal to the current time, and has smallest deadline. The fastest known algorithm to solve the problem is due to Frederickson [34] who showed how to solve it in $O(m)$ time and using $O(L+m)$ space, where L is the size of the largest deadline.

This problem is a special case of the problem in which arbitrary values are allowed for the release times and deadlines. This more complex problem can be solved in $O(m \log n)$ time [50]. Another generalization of the problem considers different processing times for the jobs, instead of unit times. This version of the problem is NP-hard [49]. Many other scheduling problems can be defined by allowing the jobs to be preempted, i.e. the processing of a job can be interrupted and resumed later, or by having more than one processor for performing the jobs. Still more modalities arise when the processors have different speeds, or when some of the jobs can be executed only by certain processors. Furthermore, the solution of a scheduling problem does not necessarily need to find a maximum set of jobs that can be scheduled for execution, but it might be desired to schedule jobs so as to minimize the maximum completion time of a job, or to minimize some penalty function that is activated when a jobs fails to be completed by its deadline [18, 73, 94].

1.1.4 Partition Matroids

A *partition matroid* $M = (E, \mathcal{I})$ is defined over a finite set of elements E partitioned into ℓ disjoint *blocks* $E_1, E_2, \ldots, E_\ell$. Given a set of ℓ integer values $\{n_1, n_2, \ldots, n_\ell\}$, where $n_i \leq |E_i|$ for all $i = 1, \ldots, \ell$, a set $S \subseteq E$ is independent in M if an only if $|S \cap E_i| \leq n_i$ for all $i = 1, \ldots, \ell$. A *uniform matroid* is a partition matroid in which $\ell = 1$.

Despite their simplicity, partition matroids have interesting applications in graph theory [43] and resource allocation problems [83]. As an example of the use of the robustness function of a partition matroid consider the following situation. Suppose that you are considering investing your lifelong savings in the stock market. Your broker gives you the names of some very promising companies and, in order to minimize risk, he advises you to buy equal amounts of stock in a certain number of them. To help you make the best choice, the broker based on a financial analysis of the companies gives you the following information for each company: (a) an estimate of the profit that you would make if you buy stock in the company, hold the stock for one year, and sell it at the end of the year; and (b) a "coefficient of faith" which reflects how confident the broker feels about his prediction. The smaller the coefficient of faith is, the less confident the broker feels about his estimate, and the larger the variation on the real profit can be.

With this information you build the following model. An adversary is allowed to use a finite amount of resources to decrease the estimated profits of the companies. The cost of each unit decrease in the profit of a company is proportional to the value of the coefficient of faith for that company. The resources given to the adversary reflect your degree of incredulity with respect to the clairvoyant abilities of the broker.

Plotting the total profit that you hope to get versus the amount of resources given to the adversary, yields the robustness function of a partition matroid. The robustness function provides information that might help you decide whether it is worth risking your money in the stock market.

The importance of partition matroids is probably best appreciated when considering the large number of applications of matroid intersection algorithms. Some problems that can be modeled with the intersection of two matroids, one of which is a partition matroid are: the assignment problem, the problem of finding, in a graph with red and green edges, a spanning tree of minimum weight and having at most k red edges, and the problem of finding a *branching* of minimum weight, where a branching of a digraph is a set of edges forming a spanning tree in the underlying undirected graph.

1.2 Continuous and Discrete Models

In this paper we study robustness functions for minimization matroid optimization problems only. It is easy to see how to extend our concepts and algorithms for maximization optimization problems.

Let $M = (E, \mathcal{I})$ be a matroid in which every element $e \in E$ has an associated weight $w(e)$. We study robustness functions of matroid optimization problems under two different models. The first model, that we call the *continuous model*, assigns to each element $e \in E$ a non-negative coefficient $c(e)$ that indicates the cost of each unit-increase in the weight of the element. If the weight of element e is increased by some amount δ, then the total cost of the increase is $c(e) * \delta$. The robustness function in this model, that we call the *continuous robustness function*, measures the maximum increase in the weight of the minimum weight bases of M caused by changes of a given total cost on the weights of its elements.

Consider, for example, a communications network in which information is broadcast through a minimum spanning tree. Traffic congestion might increase the time needed to send a message between the two endpoints of a link. This, in turn, might affect the weight of a minimum spanning tree of the network, and hence the quality of the broadcasting algorithm. The continuous robustness function can be used to quantify this decrease in performance.

The second model that we consider is called the *discrete model*. It assigns to each element $e \in E$ a cost $c(e)$ for removing the element from the input. In this case the robustness function, called the *discrete robustness function*, gives the maximum increase in the weight of the minimum weight bases of M that can be obtained by removing elements of a given total cost from E. Consider the same communications network as above. Link failures might affect the performance of the broadcasting algorithm, and the discrete robustness function can be used to model this situation.

The problem of computing the robustness function of a matroid optimization problem in the continuous model is called the *continuous robustness problem*, and the problem of computing the robustness function in the discrete model is the *discrete robustness problem*.

1.3 Overview of Results

1.3.1 Discrete Robustness Problems

Let $G = (V, E)$ be an undirected graph with non-negative weights on the edges. Given an integer $k > 0$, the discrete robustness problem for minimum spanning trees is to select a set of k edges that when removed from G maximizes the weight of any minimum spanning tree in the resulting graph.

We prove that the discrete robustness problem for minimum spanning trees is NP-hard even if the weights of the edges are 0 or 1. This fact strongly suggests that there is no algorithm that can solve exactly the problem in polynomial time [49, 74]. One natural approach to deal with NP-hard problems is to compromise on the quality of the solution for the sake of efficiently computing a sub-optimal solution. Algorithms that compute sub-optimal solutions which can be proved to be of value within some multiplicative factor of the value of the optimal solution are called *approximation* algorithms. Such multiplicative factor is called the *performance guarantee*, or *performance ratio* of the algorithm.

More formally, the performance ratio of an approximation algorithm for a minimization problem is the maximum value, over all instances of the input, of the ratio of the value of the solution computed by the algorithm to the value of the optimum solution for the instance. The performance ratio of an approximation algorithm for a maximization problem is defined as the reciprocal of this ratio. If an algorithm has a performance ratio of α, it sometimes will be referred to as an α-approximation algorithm.

We present the first approximation algorithm for the discrete robustness problem for minimum spanning trees when the edges have arbitrary non-negative weights. 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.

If the weights of the edges can only be 0 or 1, the problem is equivalent to the *maximum components problem.* This latter problem consists in selecting a set of k edges, for some given $k > 0$, that when removed from an undirected graph divides it into the maximum possible number of connected components. We give a simple 2-approximation algorithm for this problem, that runs in $O(km + k^2 n \log n)$ time.

The maximum components problem is in some sense the dual of the *minimum k-cut* problem, defined as follows [53]. Given an undirected graph G, and an integer $k > 0$ the problem is to select the smallest subset of edges whose removal from G divides it into k components. We show that the maximum components problem is NP-hard by a simple reduction from the minimum k-cut problem. Despite the seemingly equivalence between these two problems, an approximation algorithm for one does not translate into an approximation algorithm with a similar performance ratio for the other.

We have also studied the more general version of the discrete robustness problem for minimum spanning trees in which the edges have a non-negative destruction cost. Given a positive budget $b > 0$ the problem is to find a set of edges of total destruction cost at most b whose removal from the graph maximizes the weight of any minimum spanning tree in the resulting graph. We have designed an $O(\log n)$-approximation algorithm for this problem by extending our algorithms for the restricted version of the problem in which the edges have unit destruction cost.

The discrete robustness problem for partition matroids can be formulated as a discrete version of the *optimal distribution of effort problem* [31, 75]. If the elements of the matroid have arbitrary destruction costs, then the problem is NP-hard even for uniform matroids. To see this, note that any instance (E, p, v, b) of the knapsack problem (the instance consists of a set E, weight and value functions w and v, and bound b on the total value of the solution) corresponds to an instance of the discrete robustness problem for a uniform matroid $M_U = (E_U, \mathcal{I}_U)$. Set E_U has $n + k$ elements, k of them have very large weight L and cost L. Each one of the other elements e corresponds to an element e' of the knapsack problem; the cost of e is $c(e) = v(e')$ and its weight is $w(e) = L - p(e')$.

If the elements have unit destruction cost, a simple dynamic programming algorithm solves the problem in $O(mk)$ time, where k is the number of elements to be removed from the matroid.

1.3.2 Continuous Robustness Problems

Given a matroid $M = (E, \mathcal{I})$, we show that its robustness function F_M is piecewise linear and non-decreasing. Moreover, we prove that F_M has at most mn breakpoints. We present a general algorithm for computing all the breakpoints in the robustness function of an arbitrary matroid. The algorithm incrementally modifies the weights of the elements in the ground set in such a way that it actually "marches" along the curve F_M.

The algorithm runs in strongly polynomial time for any matroid with a strongly polynomial time independence test, i.e., a matroid for which it can be decided whether a set is independent in strongly polynomial time. The algorithm requires time $O(m^5n^2 + m^4n^4\tau)$, where τ is the time needed by an oracle to test independence for a set of at most n elements.

There are two key ideas in this algorithm. The first is a reduction from the problem of computing the robustness function of a weighted matroid to that of computing the robustness function of some family $\mathcal{F}$ of minors of the matroid (for the definition of minor of a matroid see Chapter 3. This family $\mathcal{F}$ has the nice property that each minor in it is formed by elements of the same weight. The second key idea is a transformation from the latter problem to the membership problem on a matroid polyhedron (see Chapter 3), for which there are several known algorithms that solve it in strongly polynomial time [23, 96].

We also consider particular classes of matroids that have interesting applications in graph theory, scheduling, and operations research. Specifically we have studied the problem of computing the continuous robustness function for graphic, transversal, scheduling, and partition matroids. All of this algorithms have the same general structure as our general robustness algorithm. However, we take advantage of the special structure of these kinds of matroids to design very efficient algorithms for the key subroutines.

For the case of graphic matroids, we present an algorithm that computes the set of breakpoints of the robustness function in $O(n^3m^2 \log(n^2/m))$ time. 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* [16, 46] computations.

Since computing the robustness function of a minimum spanning tree might require increasing 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.

For transversal matroids, we give an algorithm that computes the robustness function in $O(mn(m + n^2)|E| \log(m^2/|E| + 2))$ time, where E is the set of edges in the bipartite graph that defines the transversal matroid. We prove an extension of Hall's Theorem for the class of minors that constitute the family $\mathcal{F}$. This extended theorem allows us to solve the membership problem for the matroid polyhedron associated with each submatroid in $\mathcal{F}$ by performing a single minimum-cut computation over a bipartite graph.

Our algorithm for scheduling matroids finds all the breakpoints of the robustness function in $O(m^2n^2)$ time. Each breakpoint is computed by solving a series of scheduling-with-preemption subproblems in which some subset of elements of the matroid must be scheduled to completion. We present a formulation for these scheduling subproblems that corresponds to a generalization of the *off-line min* problem defined in [1]. This reformulation allows us to solve optimally each scheduling subproblem.

For partition matroids, we have designed an algorithm that computes all the breakpoints of the robustness function in $O(m \log m)$ time. As the algorithm computes the breakpoints, it identifies increasingly larger clusters of elements that must undergo the same weight increases in the computation of the remaining breakpoints. These clusters have a certain "convexity" property that we exploit to compute each breakpoint in $O(\log m)$ time. The increasing size of the clusters allows us to prove that the robustness function of a partition matroid has only $O(m)$ breakpoints.

If we do not want to compute all the breakpoints of the robustness function of a matroid M, but wish only to evaluate it at a certain given point then it might be possible to design faster algorithms. Stated in a different way, suppose that for a given fixed value b we only want to know the maximum increase that can be expected in the weight of the minimum weight bases of a matroid when changes of a total fixed cost b are allowed. The above algorithms compute one by one the breakpoints of the robustness function, and so if b is large they might require a long time to give the desired answer. In this paper we have also studied the problem of designing faster algorithms that evaluate the robustness function at only a required point.

For the case of a partition matroid M, the problem of evaluating F_M at a fixed point b can be formulated as an optimal distribution of effort problem (see e.g. [75]). Under this formulation, the problem is to split the budget b among the blocks E_i so that when the partial budget b_i assigned to block E_i is optimally used, the weight of a minimum weight base of M is maximized. Methods of solution for the optimal distribution of effort problem [31, 75] assume that it is possible to compute efficiently the optimal way of spending each partial budget b_i increasing the weights of the elements in block E_i. But we do not know how to do this. Hence, instead of using those methods we present a new approach that optimally solves the problem by a sophisticated application of a linear-time selection algorithm [10], that interleaves searches on weights with searches on costs.

1.4 Related Work

1.4.1 Discrete Robustness Problems

The discrete robustness problem for minimum spanning trees has appeared in the literature under the name of "the most vital edges for a minimum spanning tree" [63, 64, 70, 89]. In [89] 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 [63, 64, 70, 110] algorithms are given for finding the single edge that when removed from a graph causes the largest possible increase in the weight of the minimum spanning trees of the resulting graph.

Several related problems have also been studied. The problem of finding the k most vital edges in the shortest path problem consists in selecting the k edges that when removed from an undirected graph G maximize the shortest distance between two distinguished vertices s and t [4, 91]. In [77] it is proved that this problem is NP-hard even if all edges in G have the same weight. In [77, 91] exponential time branch and bound algorithms are given for solving the problem.

Given a capacitated network with destruction costs on the edges, the problem of spending a given budget destroying edges so as to minimize the maximum flow that the network can transport from a vertex s to a vertex t, is shown to be NP-hard in [100]. In [100] it is also given a fully polynomial time approximation scheme for the problem on planar networks. This algorithm uses the properties of the dual of a planar graph to cast the problem in terms of shortest paths instead of maximum flows. This formulation of the problem allows an approximate solution to be obtained with a dynamic programming approach.

1.4.2 Continuous Robustness Problems

The concept of continuous robustness function in combinatorial optimization can be seen as a generalization of that of sensitivity analysis [56, 106, 110], since it deals with simultaneous changes in the weights of all the elements in the input. Although the name "continuous robustness function" has not been used before, the concept has been previously studied. In [42, 52] algorithms are given for computing the continuous robustness function for the problem of finding the shortest distance between two points. These algorithms are based on a linear programming formulation for the problem which shows that it is equivalent to a minimum cost flow problem.

In [100] a fully polynomial time approximation scheme is given for the continuous robustness function for maximum flows on planar graphs. Like the algorithm for the discrete robustness function for maximum flows, this algorithm also relies on the properties of the dual of a planar graph.

Some other related problems that have been considered are the following. In [28] a constant-approximation algorithm is given for the problem of spending a fixed budget reducing the weights of the edges in a given graph to minimize the weight of its minimum spanning trees. In this problem every edge e has a lower bound $\ell(e)$ below which the weight of the edge cannot be reduced. Interestingly, this minimization version of the continuous robustness problem for minimum spanning trees is NP-hard, as opposed to our version that is solvable in strongly polynomial time. The algorithm in [28] assigns to every edge e in the graph a modified weight $w'(e)$ that depends on its original weight $w(e)$ and cost $c(e)$. Specifically, for a parameter $\alpha \geq 0$, $w'(e) = \min\{w(e), w(e) + (w(e) - \ell(e))(\alpha c(e) - 1)\}$. The core of the algorithm is a result showing that for some value of α a minimum spanning tree T computed with the modified weights is such that if the weights of the edges in T are reduced to their lower bounds, then the weight of a minimum spanning tree computed with these new weights is close to the value of the optimal solution.

In [28] constant-approximation algorithms are also given for the problems of reducing edge weights so as to minimize the diameter of a graph, and for minimizing the weight of an optimal Steiner tree. The algorithms are similar to that for minimum spanning trees. In [6] a simple algorithm is given for optimally shortening the lengths of edges in a given rooted tree to minimize the sum of the distances from the root to all of the other vertices in the tree. This algorithm heavily relies on the fact that the underlying graph is a tree.

1.5 Presentation Overview

In Section 2 we prove that the maximum components problem is NP-hard and present a 2-approximation algorithm for it. This algorithm serves as the key subroutine for our $O(\log k)$-approximation algorithm for the discrete robustness problem for minimum spanning trees. The approximation algorithm runs in $O(km(m + kn \log n))$ time, but we show that by preprocessing the graph our algorithm can be made to run in only $O(k \log n(m + kn \log n))$ time. Finally, we show that the discrete robustness problem for minimum spanning trees is NP-hard even if the weights of the edges are 0 or 1.

In Section 3 we present some definitions and results from matroid theory needed to understand our algorithms for computing the continuous robustness functions of matroid optimization problems.

In Section 4 we present our general algorithm for computing the continuous robustness function of an arbitrary matroid. We prove that the robustness function is piecewise linear and that it has at most mn breakpoints. We also specialize the algorithm for the case of graphic matroids and design a faster algorithm that runs in $O(n^3m^2 \log(n^2/m))$ time. As we show our algorithms can compute all the breakpoints of the robustness function.

In Section 5 we give the algorithm for computing the continuous robustness function for transversal matroids. We formulate the problem first in terms of matroid and polymatroid theory, and then bring it to the realm of parametric maximum flows. We also present an algorithm for computing the robustness function for scheduling matroids. For this class of matroids, the problem is formulated as a series of scheduling with-preemption subproblems. We introduce an interesting generalization of the *off-line min* problem defined in [1] which allows us to design an $O(m^2n^2)$ time algorithm for computing the breakpoints in the robustness function of a scheduling matroid.

In Section 6 we give an $O(m \log m)$ time algorithm for computing the continuous robustness function for partition matroids. An algorithm is presented first for uniform matroids, and then we show how to generalize it for arbitrary partition matroids. In the second part of the section we describe an algorithm for evaluating the robustness function at only a given point in $O(m)$ time.

In Section 7 we present our conclusions and directions for future research.

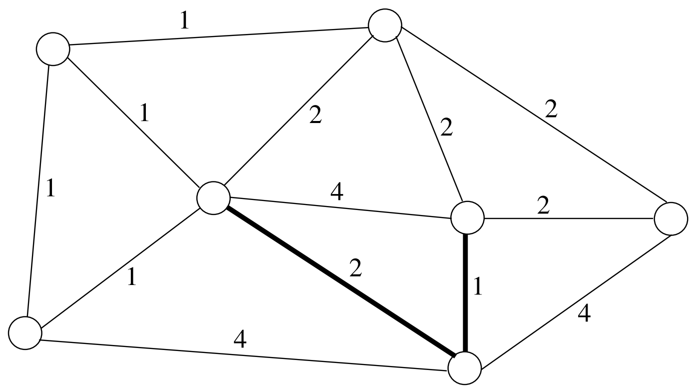

Figure 1: An instance of the discrete robustness problem. The two edges in bold are those whose removal maximize the weight of the minimum spanning trees.

2 Discrete Robustness Problem for Minimum Spanning Trees

In this section we study the effect that the removal of a given number of edges from a graph has over the weight of its minimum spanning trees. We show that the discrete robustness problem for minimum spanning trees is NP-hard and present an $O(\log k)$-approximation algorithm for it, where k is the number of edges to be removed. This algorithm uses as key-subroutine a 2-approximation algorithm for the problem of removing a given number of edges from a graph so as to maximize the number of its connected components.

The discrete robustness problem for minimum spanning trees consists in selecting a set of k edges, for a given value $k > 0$, that when removed from a graph G maximizes the weight of its minimum spanning trees. Consider for example the graph shown in Figure 1. For $k = 2$, the maximum increase in the weight of the minimum spanning trees of this graph is achieved by removing the edges shown in bold. By doing this, the edge of weight 1 shown in bold in Figure 1 is replaced in every minimum spanning tree 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. The weight of the minimum spanning trees increases from 8 to 11.

To motivate this problem, consider a packet-switching network in which periodic information has to be sent from a distinguished vertex to the rest of the network. Suppose that this network is used in a real time application, and that the broadcast information has to be delivered in a timely manner. The solution chosen is to use a virtual circuit formed with the edges of a minimum spanning tree [7, 5]. To find a minimum spanning tree, every edge of the network is assigned a weight that indicates the time needed to sent a packet between its two endpoints. In the event that some of the links in the network fail, the virtual circuit has to be changed and this might affect the performance of the broadcasting algorithm. The discrete robustness problem for minimum spanning trees can be used to model this situation.

Minimum spanning trees are used to approximate minimum Steiner trees[66, 67, 117] and minimum weight Hamiltonian circuits [84, 17]. The solution to the discrete robustness problem for minimum spanning trees can be used to approximate the maximum increase on the weight of a minimum Steiner tree or on the weight of a minimum Hamiltonian cycle that can be produced by the removal of a certain number of edges from a graph.

In this section we present an $O(\log k)$-approximation for the discrete robustness problem for minimum spanning trees. The algorithm is based in the following property of minimum spanning trees.

Property 2.1 *Let $G = (V, E)$ be an undirected graph, and let $w_1 < w_2 < \ldots < w_p$ be its different edge weights. Let $G_{\leq w_i} = (V, E_{\leq w_i})$ be the subgraph of G formed by all edges of weight at most w_i, for $w_1 \leq w_i \leq w_p$. A minimum spanning tree of G induces a spanning tree in every connected component of $G_{\leq w_i}$. Therefore, the maximum number of edges of weight larger than w_i in any minimum spanning tree of G is equal to the number of connected components in $G_{\leq w_i}$ minus one.*

Proof. Let T be a minimum spanning tree of G. Suppose that in some connected component of $G_{\leq w_i}$, T does not induce a spanning tree but a forest $\{F_1, F_2, \ldots, F_q\}, q > 1$. Let $e \in E_{\leq w_i}$ be an edge that connects a vertex from F_i with a vertex from F_j, $1 \leq i, j \leq q$. Include e in T to form a unique cycle, and then remove the edge of largest weight in that cycle to get a new tree T'. Clearly, this tree T' has smaller weight than T, which is a contradiction. □

We use the following approach to find an approximate solution for the discrete robustness problem for minimum spanning trees. For every edge weight w_i, remove from $G_{\leq w_i}$ a subset S_i of k edges that partitions $G_{\leq w_i}$ into the maximum possible number of connected components, n_i. By Property 2.1, if the edges in S_i are removed from G, then every minimum spanning tree of G will have at least $n_i - 1$ edges of weight larger than w_i. We choose as our approximation the set S_i that causes the largest increase in the weight of the minimum spanning trees of G.

An essential component of this algorithm is a procedure for selecting a set of k edges that splits a graph into the maximum number of connected components. We study this problem in the following section.

2.1 The Maximum Components Problem

A *k-cut* of a graph $G = (V, E)$ is a set of edges that when removed from G partitions it into k connected components. The minimum k-cut problem [53] 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 smallest value i for which the number of components created by removing i edges from the graph is at least k. Since the minimum k-cut problem is NP-hard [53], the above reduction implies the NP-hardness of the maximum components problem.

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.

Figure 2: Algorithm *dice*.

We modify an algorithm by Saran and Vazirani [104] to obtain a 2-approximation algorithm for the maximum components problem. Our algorithm, described in Figure 2, uses a greedy approach to chop up a graph into as many pieces as possible by removing k edges from it.

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.*

Before proving the theorem, we prove the following Lemma for connected graphs. Assume that we run algorithm *dice* on a 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 one of the existing components. Since the cuts A_i are indexed in non-decreasing order of size, then $|A_j| \leq |A_d|$ and, so, $|D_d| \leq |A_d|$. $\square$

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 indexed 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 the value of d is larger than $a/2$.

Now we consider the case when G is disconnected. Suppose that G consists of $j+1$ connected components. Add 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 is 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 computes connectivity cuts for the two components that were created in the preceding iteration. Gabow's algorithm [47] 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 when implemented with Gabow's algorithm. □

2.2 The Discrete Robustness Problem

As we mentioned earlier, the approach that we follow to approximate the solution of the discrete robustness problem for minimum spanning trees consists in forcing into every minimum spanning tree of G as many edges of "large" weight as possible. By Property 2.1, 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, by deleting the edges of weight smaller than $w(e')$ in some cut that separates the endpoints of e' we 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.

Our approximation algorithm for the discrete robustness problem for minimum spanning trees is described in Figure 3. The algorithm outputs both, the selected set $S_{s\&d}$ and 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 $G_{\leq w_i}$ 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.

Algorithm *slice_n_dice* $(G = (V, E), k)$
    $max\_mst\_wgt \leftarrow 0$
    for each distinct edge weight w_i in G **do**
        $S \leftarrow dice\ (G_{\leq w_i}, k)$
        $mst\_wgt \leftarrow$ weight of a minimum spanning tree of $(V, E - S)$
        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.

Figure 3: Algorithm *slice_n_dice*.

Figure 4 shows the solution found by the algorithm for the graph of Figure 1 when $k = 2$. For graph $G_{\leq 1}$, the algorithm removes the two edges shown in bold in Figure 4(a). This increases the weight of the minimum spanning trees from 8 to 10. For graph $G_{\leq 2}$, the algorithm might choose to discard the edges shown in bold in Figure 4(b). The removal of these edges increases the weight of the minimum spanning trees to 10. Graph $G_{\leq 4}$ has connectivity 3 and so no cut can be made by deleting two edges. Algorithm *slice_n_dice* might either choose the edges discarded in $G_{\leq 1}$ or those removed from $G_{\leq 2}$, increasing the weight of the minimum spanning trees by 2.

2.2.1 Performance Ratio of the Algorithm

To simplify the analysis of the performance ratio of *slice_n_dice* we assume first that all 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, $(V, E - S^*)$, and $(V, E - S_{s\&d})$ 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, where i is an integer, 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 $G_{\leq 2^{i-1}}$

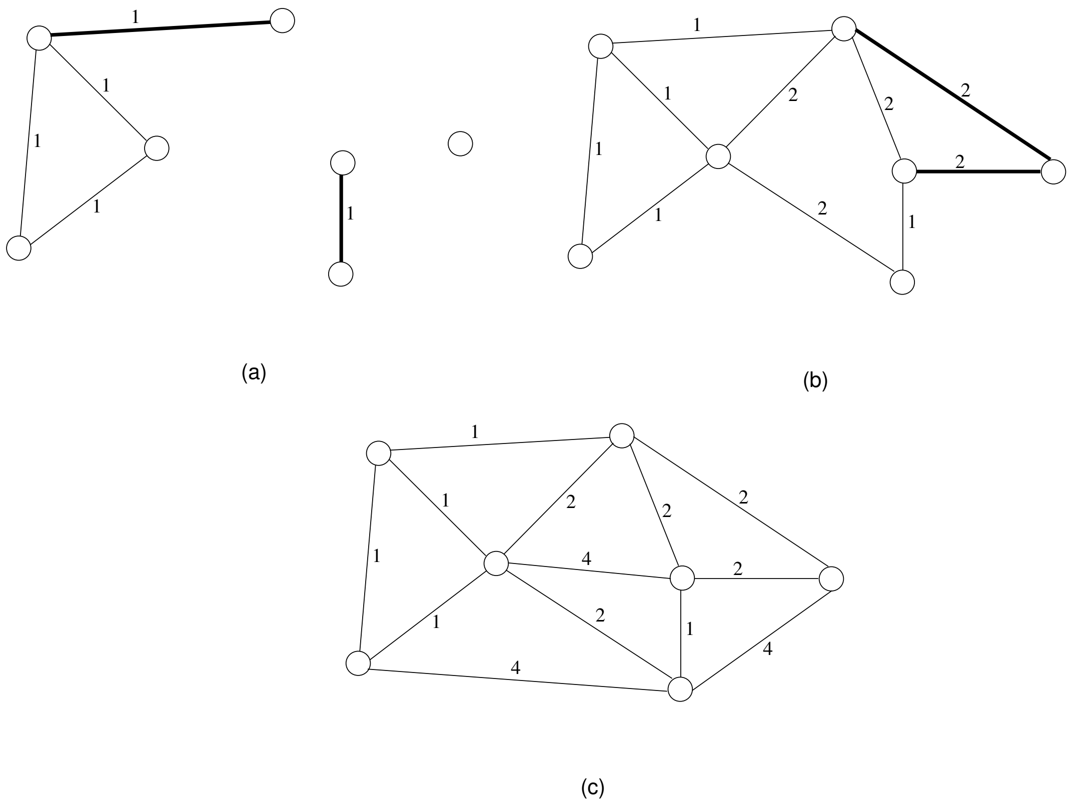

Figure 4: Figure showing an execution of algorithm *slice_n_dice*.

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 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 no more than 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 $r \geq 1/(1 + 2\log(k+1))$. $\square$

Lemma 2.3 *The maximum weight of the minimum spanning trees of a graph $G = (V, E)$ 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 $G' = (V, E)$ formed by rounding down the weight of every edge in E to its nearest integer power of 2.*

Proof. For every $e \in E$, let $w'(e)$ be equal to the value of $w(e)$ rounded down to its nearest integer power of 2. 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 $w(T^*) < 2w'(T^*) \leq 2w'(T') \leq 2w(T')$. $\square$

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 $r = w(T_a)/w(T^*) \geq 1/(2 + 4\log(k+1))$.* $\square$

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 a clique of size $k+2$. Every edge of clique 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 5).

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 $1 \leq i < \hat{k}/2$. 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 weight of a minimum spanning tree in the graph obtained after removing the set of edges chosen by the algorithm has weight $w(T_{s\&d}) = \hat{k} + (\hat{k}/2 - 1)\varepsilon$. Hence $w(T^*)/w(T_{s\&d}) > \log k/2$.

2.2.2 Graphs with Arbitrary Destruction Costs on the Edges

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 modify algorithm *dice* so that in each iteration of the repeat-loop it removes from G the edges in a cut with smallest destruction cost. The repeat-loop is performed until all the budget is used

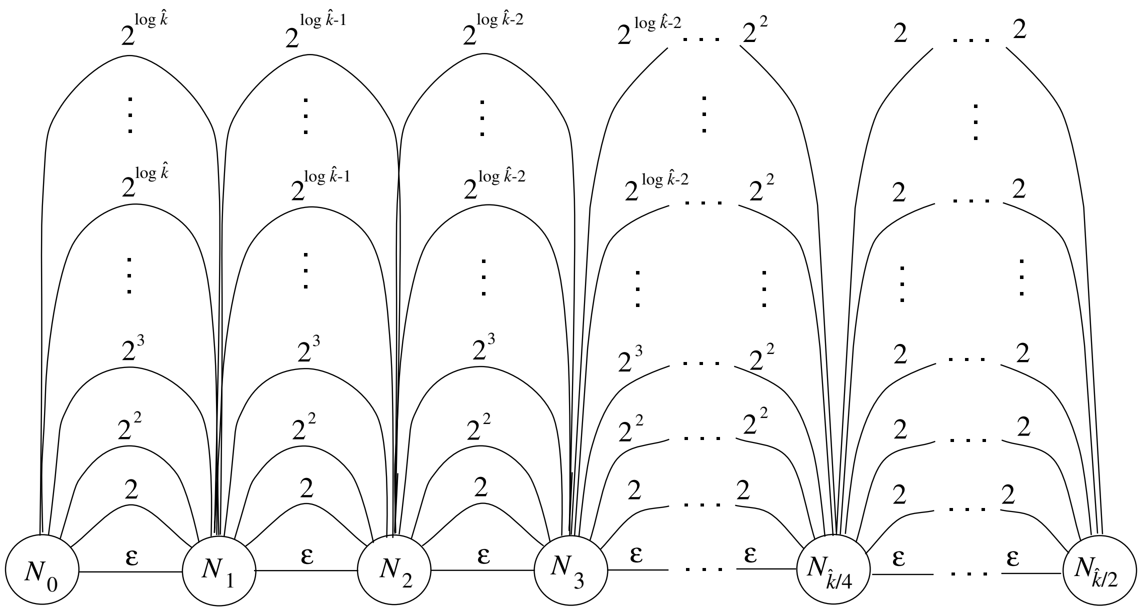

Figure 5: Graph used to show the tightness of the analysis.

up, or no cut can be made with the remaining budget. With these changes on *dice*, algorithm *slice_n_dice* can be used to approximate the solution to this version of the problem.

Lemma 2.4 *The performance ratio of slice_n_dice is* $r \geq 1/(1 + 4\log m)$ *when the edges of the input graph have arbitrary destruction costs and edges of a total given destruction cost are to be removed.*

Proof. We can modify the proof of Theorem 2.1 to show that the performance ratio of algorithm *dice* is still 2 when the edges have arbitrary destruction costs. 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

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

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 destruction costs on the edges. $\square$

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 modify the algorithm so that it does not build all the subgraphs $G_{\leq w_i}$, but only those for "large" weights w_i. The intuition behind this approach is that the cuts made on $G_{\leq w_i}$ can be guaranteed to force edges of large weight into a minimum spanning tree only if w_i is large. Note that it might happen that for a small weight w_i, the cuts made by *slice_n_dice* on $G_{\leq w_i}$ add a large weight edge to the minimum spanning trees of G. This situation causes no trouble since the same large weight edge can be included in the minimum spanning trees by making cuts in a subgraph $G_{\leq w_j}$ for some larger weight w_j.

While we wish to build subgraphs only for large weights, we need to be aware that some edges of large weight might not possibly be part of a solution. Given an undirected graph G and a value k, an edge e 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 graph G when it is desired to remove k edges. We modify *slice_n_dice* so that it forms only the graphs $G_{\leq w_i}$ 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 $G_{\leq w_i}$ 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 $G = (V, E)$ 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 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 from 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 $G_{\leq w_i}$ has a minimum cut of size larger than k. If we use Gabow's algorithm [47] to perform the connectivity computations in this binary search, then the weight of a largest non-redundant edge can be found in $O(m \log m + k^2 n \log(n/k) \log m)$ time.

Theorem 2.5 *The modified slice_n_dice algorithm runs in* $O(km \log n + k^2 n \log^2 n)$ *time.* □

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* $G = (V, E)$ *be a graph whose edges have a weight of value either 0 or 1, and let* ℓ *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* $(V, E - S)$ *has weight at least* ℓ*, 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 $G_H = (V_H, E_H)$, build a graph $G = (V, E)$ by combining G_H and a clique $G_Z = (V_Z, E_Z)$ 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 (see Figure 6). 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 it has to use $k+1$ edges of weight 1 to connect the vertices of V_Z among themselves.

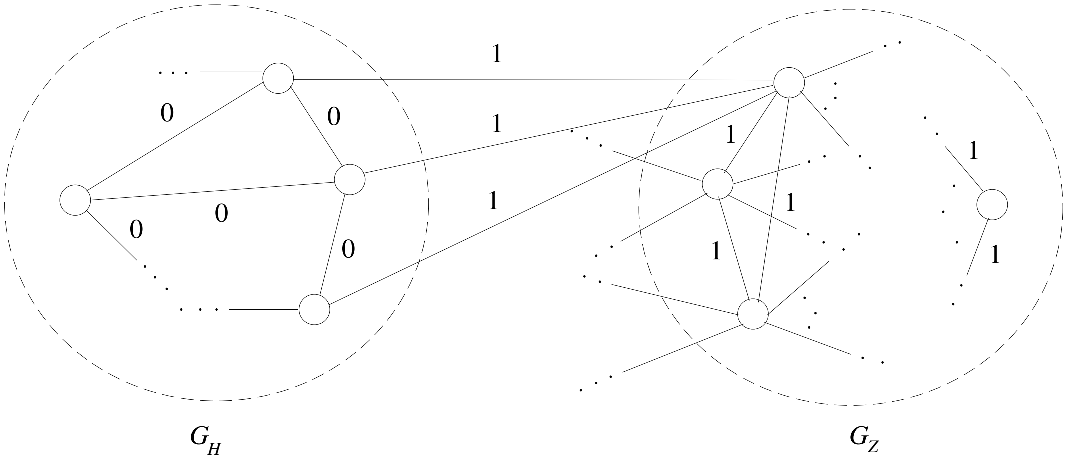

Figure 6: Construction of the graph G.

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. $\square$

3 Matroid Preliminaries

3.1 Definitions

A matroid $M = (E, \mathcal{I})$ consists of a finite set E of *elements* and a collection $\mathcal{I}$ of subsets of E that satisfy the following three axioms:

1. $\emptyset \in \mathcal{I}$,
2. if $S \in \mathcal{I}$ and $T \subseteq S$, then $T \in \mathcal{I}$,
3. if $S, T \in \mathcal{I}$ and $|S| = |T| + 1$ then there exists an element $x \in S - T$ such that $T \cup \{x\} \in \mathcal{I}$.

Set E is called the ground set of the matroid, and $\mathcal{I}$ is called the family of independent sets of M. Consider for example a graphic matroid $M = (E, \mathcal{I})$. It is easy to see that, the independent sets of M (subsets of edges that do not form any cycles) satisfy these properties,

1. the empty set does not have any cycles,
2. any subset of edges taken from a forest cannot form cycles, and
3. given two forests S and T with $|S| = |T| + 1$, some edge of G must be incident to a vertex of zero degree in T. Adding such an edge to T does not create any cycles.

A maximal independent set in a matroid is called a *base*. All bases of a matroid have the same cardinality, and the number of elements in a base is called the *rank* of the matroid. A *cycle* or *dependent set* is a subset of E that is not independent. For the case of a graphic matroid, a base is a spanning tree of G if G is connected, or a spanning forest of G if G is not connected.

Algorithm *greedy* (M)
Let $e_1, e_2, \ldots, e_m$ be the elements of M indexed non-decreasingly by weight.
$B \leftarrow \emptyset$
for $i = 1$ **to** m **do**
  if $B \cup \{e_i\}$ does not have any cycles **then** $B \leftarrow B \cup \{e\}$ **end if**
end for
Output B

Figure 7: Algorithm *greedy*.

There are two functions defined on the elements of a matroid. Function w assigns a non-negative weight to each element in E, and function c gives the cost for a unit decrease in the weight of each element. We denote a matroid M as $M = (E, \mathcal{I}, w, c)$ when we want to emphasize that the elements of the matroid have weight and cost. The most well-known algorithmic property of matroids is that a base of minimum weight can be found using the greedy algorithm described in Figure 7. or the case of a graphic matroid, the above algorithm is just Kruskal's algorithm [81] for finding a minimum spanning tree.

3.2 Matroid Operations

In this section we describe two operations that can be used to form "smaller" matroids from a given matroid M. These "smaller" matroids are called *submatroids* of M or *minors* of M.

Let $M = (E, \mathcal{I})$ be a matroid. For any set $T \subseteq E$, the *restriction* of M to T, denoted as $M|T$, is the matroid whose ground set is T and whose independent sets are $\{ S \mid S \in \mathcal{I}$ and $S \subseteq T \}$. Consider for example the graph $G = (V, E)$ shown in Figure 8 (a). The graphic matroid $M_G(E, \mathcal{I})$ for this graph has as independent sets all forests of G. Let $T = \{a, c, d\}$. Matroid $M|T$ has as independent sets $\{\emptyset, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{c, d\}\}$.

The *contraction* of T from M, denoted as M/T, is the matroid with ground set $E - T$ and

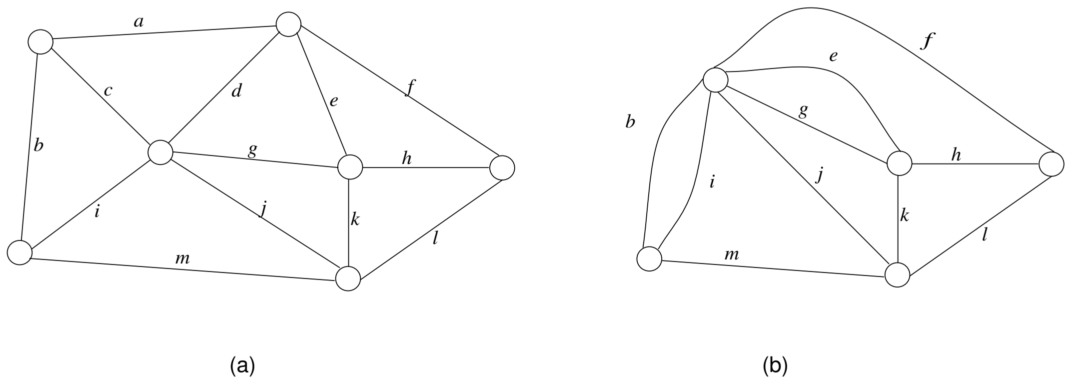

Figure 8: Figure to illustrate the matroid operations of restriction and contraction.

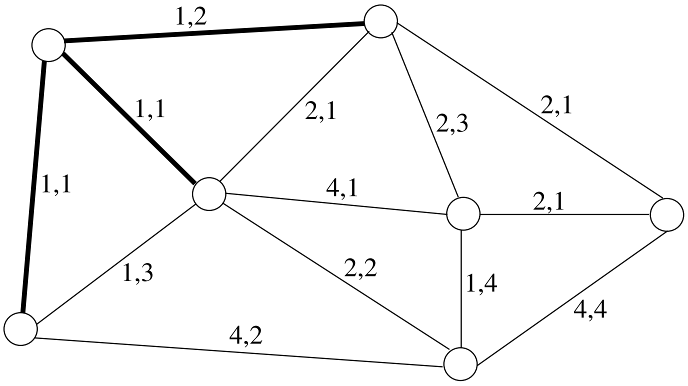

Figure 9: An instance of the continuous robustness problem, with a best set of edges for raising their weights shown in bold.

independent sets $\{ S \mid S \subseteq E - T \text{ and } S \cup Y \in \mathcal{I} \}$, where Y is any base of $M|T$. For the same example as above, the contraction of T from M is the graphic matroid corresponding to the graph G' obtained by contracting in G the endpoints of edges $\{a, c, d\}$ to a single vertex. We show graph G' in Figure 8 (b). The independent sets of submatroid M/T are the forests of the multi-graph G'.

3.3 Continuous Robustness Function

The continuous robustness function F_M of a matroid M gives the maximum increase in the weight of the minimum weight bases of M that can be caused by increases of a given total cost on the weights of its elements. Consider for example the graphic matroid M_G corresponding to the graph G shown in Figure 9. Every edge e in the graph is labeled by an ordered pair consisting of its weight $w(e)$ and its cost $c(e)$. We want to evaluate $F_{M_G}(4)$, i.e. we desire to find the maximum increase in the weight of the minimum spanning trees of G that can be caused by increases of total cost 4 on the weights of its edges. The largest increase in the weight of the minimum spanning trees is obtained by raising from 1 to 2 the weights of the edges shown in bold in Figure 9. Note that as long as the weights of the edges shown in bold are smaller than 2, at least two of them must be in every minimum spanning tree. Thus, the weight of the minimum spanning trees increases from 8 to 10, and so $F_{M_G}(4) = 2$.

Consider a matroid $M = (E, \mathcal{I}, w, c)$. Fix a subset $S \subseteq E$. The weight of S, denoted as $w(S)$, is equal to the sum of the weights of the elements in S. The cost of S, $c(S)$, is defined in a similar way. Let $\mathit{rank}(S, M)$ denote the size of the largest independent subset of S. We define *coverage* (S, M) as the minimum number of edges that any minimum weight base of M shares with S. Let S, for example, be the set of edges shown in bold in Figure 9. This set of edges does not form any cycles, so $\mathit{rank}\,(S, M_G) = 3$. Note that every minimum spanning tree of the graph G shown in Figure 9 includes at least 2 edges from S since these are the smallest weight edges incident to the two vertices drawn at the top of the figure. Therefore, *coverage* $(S, M_G) = 2$.

We say that the weights of the edges in set S are *lifted* when the weight of every edge in S is increased by the same amount. We define *tolerance*(S, M) as the maximum amount by which the weights of the edges in S can be lifted until *coverage* (S, M) decreases. For the same example as before, the value of *tolerance* (S, M_G) is 1 because if the weights of the edges in S are lifted above 2, then only the leftmost vertex at the top of Figure 9 must use the edges from S to be connected to a minimum spanning tree.

Define the *rate* of S in M, denoted as *rate* (S, M), as $c(S)/$*coverage* (S, M). For any value $\varepsilon \leq$ *tolerance*$(S, M) *$ *coverage* (S, M), the cost of increasing the weight of all minimum weight bases of M by at least ε when only the weights of the edges in S are raised is $\varepsilon *$ *rate* (S, M). For the same set S of edges shown in bold in Figure 9, *rate* $(S, M_G) = (1 + 1 + 2)/2 = 2$. If the weight of the edges in S are lifted by 1, the weight of every minimum spanning tree is increased by 2. The total cost of these increases is $2 * 2 = 4$.

3.4 Polymatroids

Given a finite set E, let $f : 2^E \to I\!R$ be a function that assigns a real value to each subset of E. Function f is said to be *submodular* if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq E$. It is well know that the function *rank* (S, M) giving the size of the largest independent set in any subset S of elements of a matroid M is a submodular function.

Let $I\!R_+$ denote the set of nonnegative real numbers. Given a set function f on E, we let $\mathcal{P}(f)$ denote the polyhedron $\{x \in I\!R_+^E \mid x(A) \leq f(A) \text{ for all } A \subseteq E\}$, where $x(A) = \sum_{e \in A} x(e)$. A *polymatroid* is a polyhedron of the form $\mathcal{P}(f)$ where f is submodular and non-negative. If f is the rank function of a matroid, then $\mathcal{P}(f)$ is called a *matroid polyhedron*.

Consider a matroid $M = (E, \mathcal{I})$. Let $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Let S be an independent set of M and x_S be an $|E|$ dimensional vector with components $x_S(e_i) = 1$ for all $e_i \in S$, and $x_S(e_j) = 0$ for all $e_j \notin S$. Vector x_S is called the *incidence vector* or the *characteristic vector* of the independent set S.

It is well known that the convex hull of incidence vectors of the independent subsets of a matroid M form the matroid polyhedron $\mathcal{P}(M)$. Thus, an explicit representation for the matroid polyhedron of a matroid M is $\mathcal{P}(M) = \{z \in I\!R_+^E \mid z(S) \leq$ *rank* (S, M) for all $S \subseteq E\}$. Given a vector $y \in I\!R_+^E$, a *base* of y is a maximal vector $x \in \mathcal{P}(M)$ such that $x(e) \leq y(e)$ for all $e \in E$.

We need the following result [9] in our algorithm for computing the continuous robustness function of a matroid.

Theorem 3.1 *Let $y \in I\!R_+^E$ be an $|E|$-dimensional vector, and x be a base of y in $\mathcal{P}(M)$. Then $x(E) = \min\{\text{rank}\,(S, M) + y(E - S) \mid S \subseteq E\}$.*

Proof. Observe that for any $S \subseteq E$, $x(E) = x(S) + x(E - S) \leq$ *rank* $(S, M) + y(E - S)$. Therefore it is enough to prove that there is a set S^* for which equality holds. Let e be an element of E. If $x(e) \neq y(e)$ then by maximality of x there is a set $S_i \subseteq E$ such that $x(S_i) =$ *rank* (S_i, M).

Algorithm *poly_greedy* $(M = (E, \mathcal{I}), y)$
    $x(e_i) \leftarrow 0$ for all $e_i \in E$
    for each $e_i \in E$ **do**
        $\delta \leftarrow \min\{ \mathit{rank}\,(T, M) - x(T) \mid e_i \in T \subseteq E\}$
        $x(e_i) \leftarrow \min\{\delta, y(e_i)\}$
    end for
    Output x

Figure 10: Algorithm *poly_greedy*.

Set S_i is called a *tight set*. Let A, B be tight sets, then

$$\begin{aligned} x(A \cup B) + x(A \cap B) &\leq \mathit{rank}\,(A \cup B, M) + \mathit{rank}\,(A \cap B, M) \;\text{ since } x \in \mathcal{P}(M) \\ &\leq \mathit{rank}\,(A, M) + \mathit{rank}\,(B, M), \;\text{ by submodularity} \\ &= x(A) + x(B), \;\text{ because } A \text{ and } B \text{ are tight} \\ &= x(A \cup B) + x(A \cap B) \end{aligned}$$

Since $x(A \cup B) \leq \mathit{rank}\,(A \cup B, M)$ and $x(A \cap B) \leq \mathit{rank}\,(A \cap B, M)$, it follows that $x(A \cup B) = \mathit{rank}\,(A \cup B, M)$ and $x(A \cap B) = \mathit{rank}\,(A \cap B, M)$. Thus the union and intersection of tight sets are tight. Let S^* be the union of all tight sets. Then $x(S^*) = \mathit{rank}\,(S^*, M)$, $x(E - S^*) = y(E - S^*)$, and so $x(E) = \mathit{rank}\,(S^*, M) + y(E - S^*)$. □

A corollary of Theorem 3.1 is that all bases of a vector $y \in \mathbb{R}_+^E$ have the same component-sum. Another consequence of the theorem is that a base x of a vector y can be found using the "polygreedy" algorithm. This algorithm begins with $x = 0$, and for each element $e \in E$ increases the component $x(e)$ as much as possible but maintaining x in $\mathcal{P}(M)$. The algorithm is described in Figure 10.

4 Continuous Robustness Function of a Matroid

In Section 2 we studied the discrete robustness problem for minimum spanning trees. Now we study robustness functions in the continuous model, i.e. we are interested in how the optimal solution of a matroid optimization problem changes when the weights of the elements are increased by finite amounts.

We present a general algorithm for computing the continuous robustness function of an arbitrary matroid. This algorithm is mainly of theoretical interest due to its high time complexity. Then, we show how to exploit the special structure of graphic matroids to design a faster algorithm for computing the robustness function for this class of matroids.

4.1 The General Algorithm

Our approach for computing the robustness function F_M of a matroid $M = (E, \mathcal{I}, w, c)$ consists in raising the weights of the elements in a set S of smallest *rate* in M up to the point where *coverage* (S, M) decreases. Then a new set S of smallest *rate* is chosen and the process is repeated until the total cost of the weight increases reaches a desired value, b. The algorithm is shown in Figure 11.

Consider the execution of algorithm *uplift* on the graphic matroid M_G corresponding to the graph shown in Figure 9 when the budget is $b = 10$. In the first iteration of the while-loop the set S of smallest *rate* includes the edges shown in bold in Figure 9. This set has a *coverage* of value 2 and *rate* of 2. Algorithm *uplift* increases the weights of these edges from 1 to 2. At this point, the *coverage* of S decreases to 1, and its *rate* goes up to 4. In the second iteration, *uplift* selects the edges shown in bold in Figure 12, since they have a *coverage* of value 1 and *rate* of only $(1+1)/1 = 2$. Note that as long as the weight of each one of these edges is smaller than 4, one of them must belong to every minimum spanning tree of the graph. By lifting the weights of these edges to 4 the total cost of the weight increases is 10, and the weight of the minimum spanning trees goes up from 8 to 12.

4.1.1 Correctness of the General Algorithm

Given a matroid $M = (E, \mathcal{I}, w, c)$ and a budget value b^*, consider an optimal solution w^* for this instance of the continuous robustness problem for M. Let $w^*(e)$ denote the weight of element e in the 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 $\mathcal{A}$ selects some set of elements and lifts their weights by a certain amount, in such a way that at the end of all stages the weight of every element $e \in E$ is $w^*(e)$.

We show that the increase in the weight of the minimum weight bases of M produced by *uplift* is the same as that produced by $\mathcal{A}$. For this purpose only, it is convenient to imagine that whenever $\mathcal{A}$ or *uplift* increases the weight of an element e by some amount Δ, it does not

---

Algorithm *uplift* (M, b)
    balance $\leftarrow b$
    orig_wgt $\leftarrow$ weight of a minimum weight base of M
    while *balance* > 0 **do**
        Find a set $S \subseteq E$ of smallest *rate* in M.
        Lift the weights of the elements in S by $\Delta = \min\{\, tolerance\,(S, M),\, balance/c(S)\,\}$.
        balance $\leftarrow$ *balance* $- \Delta * c(S)$
    end while
    new_wgt $\leftarrow$ weight of a minimum weight base of M
    Output (*new_wgt* $-$ *orig_wgt*)

Figure 11: Algorithm *uplift*.

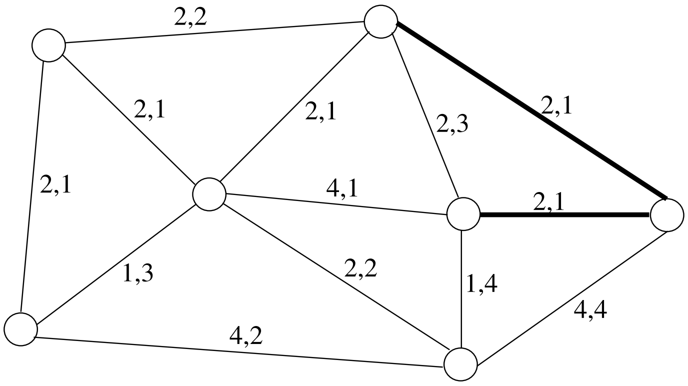

Figure 12: Second iteration of *uplift*, with best choice of edges to be raised shown in bold.

just add Δ to $w(e)$, but gradually raises 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 element weights. For any value b, $0 \leq b \leq b^*$, let S_b be the set of elements whose weights are being lifted by *uplift* when the budget spent by it reaches value b and $M(b) = (E, \mathcal{I}, w_b, c)$ describe the matroid at that precise moment. Similarly, let $S^*_b, 0 \leq b \leq b^*$, be the set of elements whose weights are being lifted by $\mathcal{A}$ when the budget spent by it reaches value b and $M^*(b) = (E, \mathcal{I}, w^*_b, c)$ be the matroid at that moment.

To show that *uplift* finds an optimal solution, we only have to prove that *rate* $(S_b, M(b)) =$ *rate* $(S^*_b, M^*(b))$, for all $0 \leq b \leq b^*$. Let *sm_eq*(x, M) be the subset of elements in M of weight no larger than x.

Lemma 4.1 *Let* $M = (E, \mathcal{I}, w, c), M' = (E, \mathcal{I}, w', c)$ *be matroids and* S *be a subset of* E*. If for every element* $e \in S$*, sm_eq* $(w'(e), M') \subseteq$ *sm_eq* $(w(e), M)$ *then coverage* $(S, M) \leq$ *coverage* (S, M')*.*

Proof. Let T' be a minimum weight base of M' for which $|T' \cap S| =$ *coverage* (S, M'). If T' is not a minimum weight base of M, it can be transformed into one by taking every element $e \in E - T'$, 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 element $e \in S - T'$, we create a cycle C in which all elements $a \in C$ have weights $w'(a) \leq w'(e)$ because T' is a minimum weight base of M'. Moreover, since $a \in$ *sm_eq*$(w'(e), M') \subseteq$ *sm_eq*$(w(e), M)$ then $w(a) \leq w(e)$, and thus it is possible to construct a minimum weight base of M that does not include any element from $S - T'$. This implies that *coverage* $(S, M) \leq$ *coverage* (S, M'). □

Lemma 4.1 suggests a way in which $\mathcal{A}$ can select the sets S^*_b that makes it easy to compare their rates 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 elements e for which *sm_eq*$(w_b(e), M(b)) \subseteq$ *sm_eq*$(w^*_b(e), M^*(b))$. If this is possible, then by Lemma 3.1, *rate* $(S^*_b, M(b)) \leq$ *rate* $(S^*_b, M^*(b))$. Since *uplift* always chooses the set S_b with smallest *rate* in $M(b)$, then it would follow that *rate* $(S_b, M(b)) \leq$ *rate* $(S_b^*, M^*(b))$.

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 element 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 elements 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 4.1 *Algorithm uplift finds the maximum increase in the weight of the minimum weight bases of a matroid M that can be attained by spending a finite budget b^* augmenting element 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, *sm_eq*$(w_b(e_1), M(b)) \subseteq$ *sm_eq*$(w_b^*(e_1), M^*(b))$, and by the above discussion *rate* $(S_b, M(b)) \leq$ *rate* $(S_b^*, M^*(b))$. □

Lemma 4.2 *Algorithm uplift performs at most mn iterations.*

Proof. Consider any iteration of the while-loop of *uplift*. Suppose that in that iteration the algorithm selects set S, and that it increases the weights of its elements up to w_i. Note that S intersects every minimum weight base of M, and so the above weight increases augment by at least one the number of elements of weight w_i in every minimum weight base of the matroid. Hence, in at most n iterations the weights of the elements in the sets selected by *uplift* can be increased up to w_i. There are at most m different element weights, and thus the algorithm performs at most mn iterations. □

Lemma 4.3 *The robustness function F_M is piece-wise linear, concave and nondecreasing.*

Proof. The proof of Theorem 4.1 shows that any partial solution constructed by *uplift* is optimal for some budget value. Hence, *uplift* "marches" along the whole curve F_M when given as input an infinite budget value. In each iteration of the while-loop, algorithm *uplift* "traverses" a linear segment of F_M of slope equal to the inverse of the *rate* of the set S selected during that iteration. Since the slope of F_M can only change when algorithm *uplift* selects a different set, then F_M is piecewise linear.

To show that F_M is concave and nondecreasing, suppose that in some iteration *uplift* selects a set S_1 with *rate* $(S_1, M) = r$ and that in the next iteration it chooses a set S_2 with *rate* $(S_2, M) < r$. This would mean that S_1 was not the set of smallest *rate* in the previous iteration since $S_1 \cup S_2$ would have smaller *rate*. Therefore, the slope of F_M is non-decreasing, and so F_M is concave and non-decreasing. □

4.1.2 Finding a Set of Smallest Rate in M

In this section we describe an algorithm for finding a set of smallest *rate* in a matroid M. Our approach consists of two stages. In the first stage we form a family of submatroids of M, each formed by elements of the same weight. We show that at least one of these submatroids M_j is such that any subset of elements with smallest *rate* in M_j also has smallest *rate* in M. In the second stage we exploit the fact that all elements in M_j have the same weight to find a set of smallest *rate* in it.

Fix a matroid $M = (E, \mathcal{I}, w, c)$. Let $w_1, w_2, \ldots, w_p$ be the different element weights in M. For a given subset $S \subseteq E$, let $S_{<w_i}$, $S_{=w_i}$, $S_{\leq w_i}$, and $S_{>w_i}$ denote, respectively, the sets formed by all elements in S of weight smaller than w_i, equal to w_i, at most w_i, and larger than w_i.

Lemma 4.4 *Let B be a minimum weight base of M, and $B_{=w_i} \neq \emptyset$ for some element weight w_i. Set $B_{=w_i}$ is a minimum weight base of the submatroid $M_i = (M|E_{\leq w_i})/E_{<w_i}$.*

Proof. It is clear that $B_{<w_i}$ is a minimum weight base of $(M|E_{\leq w_i})|E_{<w_i}$. Thus, by definition of the contraction operation, $B_{=w_i}$ is a base of $(M|E_{\leq w_i})/E_{<w_i}$. Moreover, $B_{=w_i}$ has minimum weight since all elements in M_i have weight w_i. □

Lemma 4.5 *Let B be a minimum weight base of M, and $B_{=w_i} \neq \emptyset$ for some element weight w_i. If A is a minimum weight base of the submatroid $M_i = (M|E_{\leq w_i})/E_{<w_i}$, then $B_{<w_i} \cup A \cup B_{>w_i}$ is a minimum weight base of M.*

Proof. Let A' be a base of $M|E_{\leq w_i}$ such that $A \subseteq A'$. For every $y \in B_{=w_i} - A$ there is an element $x \in A' - B_{\leq w_i}$ such that $S = B_{\leq w_i} - \{y\} \cup \{x\}$ is a base of $M|E_{\leq w_i}$. Since $w(y) = w_i$ is the largest weight of any element in $M|E_{\leq w_i}$, then $w(x) = w(y)$ because $B_{\leq w_i}$ is a minimum weight base of $M|E_{\leq w_i}$. It follows that $x \in A$ and that S is a minimum weight base of $M|E_{\leq w_i}$. This process can be repeated for all other elements $y \in B_{=w_i} - A$ to show that $B_{<w_i} \cup A \cup B_{>w_i}$ is a minimum weight base of M. □

The above two Lemmas allow us to prove that a set of smallest *rate* in M can be found by considering only the submatroids M_i.

Lemma 4.6 *If $S \subseteq E$ is a set of smallest rate in matroid M, and w_i is the weight of some element in S, then*

1. *$S_{=w_i}$ is a set of smallest rate in M, and*
2. *any set of smallest rate in $M_i = (M|E_{\leq w_i})/E_{<w_i}$ has also smallest rate in M.*

Proof. We prove (1) first. Let B^j, for all $j = 1, 2, \ldots, p$, be minimum weight bases of M such that *coverage* $(S_{=w_j}, M) = |B^j \cap S_{=w_j}|$. By Lemma 4.5, $B = \cup_{j=1}^{p} B^j_{=w_j}$ is a minimum weight base of M. Thus *coverage* $(S, M) = |S \cap B| = \sum_{j=1}^{p}$ *coverage* $(S_{=w_j}, M)$. Since S has smallest *rate* in M, then *rate* $(S, M) = \sum_{j=1}^{p} c(S_{=w_j}) / \sum_{j=1}^{p}$ *coverage* $(S_{=w_j}, M) =$ $c(S_{=w_i})/$*coverage* $(S_{=w_i}, M) =$ *rate* $(S_{=w_i}, M)$ for any $S_{=w_i} \neq \emptyset$.

We now prove (2). By Lemmas 4.4 and 4.5, the set of minimum weight bases of M_i is $\{\, A_{=w_i} \mid A$ is a minimum weight base of $M \,\}$. Hence, for any set $T \subseteq E_{=w_i}$, *coverage* $(T, M_i) =$ *coverage* (T, M), and *rate* $(T, M_i) =$ *rate* (T, M). It follows that any set of smallest rate in M_i has also smallest *rate* in M. □

Lemma 4.6 states that we can find a set of smallest *rate* in a matroid M with arbitrary element weights by finding a set S_i of smallest *rate* in each submatroid $M_i = (M|E_{\leq w_i})/E_{<w_i}$, and selecting the set S_j for which the *rate* is minimum. All elements in the submatroid M_i have the same weight, and thus, every base of M_i has minimum weight. This means that for the purpose of computing a set of smallest *rate* in M_i we can consider that M_i is an unweighted matroid. This observation greatly simplifies the problem of computing a set of smallest *rate* in M_i.

Consider an unweighted matroid $M_i = (E_i, \mathcal{I}_i)$ with rank n_i and $m_i = |E_i|$. Let σ_i be the smallest *rate* of any subset of E_i. Define the function $g_i(\lambda) = \min\{\, c(T) - \lambda * \mathit{coverage}\,(T, M_i) \mid T \subseteq E_i \,\}$, for every $\lambda \geq 0$. Then, σ_i is equal to the largest value of λ for which $g_i(\lambda) = 0$. Note that the slope of the curve g_i at any given point λ is equal to $\mathit{coverage}\,(S, M_i)$ for some set $S \subseteq E_i$, and thus g_i is piecewise linear and has at most n_i breakpoints. Furthermore, function g_i is non-increasing and its leftmost linear piece has slope 0. The value of σ_i is, then, equal to the value λ of the first breakpoint of g_i.

Since $\sigma_i \leq c(E_i)/n_i$, we compute the value of σ_i by starting at the point $\lambda = c(E_i)/n_i$ and moving backwards along the curve g_i using Newton's method. This procedure for computing the value of σ_i needs to solve at most n_i parametric problems of the form: $\min\{\, c(T)/\lambda - \mathit{coverage}\,(T, M_i) \mid T \subseteq E_i \,\}$. Since M_i is an unweighted matroid, $\mathit{coverage}\,(S, M_i) = \mathit{rank}\,(E_i, M_i) - \mathit{rank}(E_i - S, M_i)$ for any set $S \subseteq E_i$. Hence, the above parametric problem is equivalent to

$$\min\{c(T)/\lambda + \mathit{rank}\,(E_i - T, M_i) \mid T \subseteq E_i \,\}. \tag{1}$$

Cunningham [23] (see Theorem 3.1), shows that problem (1) is equivalent to the problem of finding a base for the vector c/λ in the matroid polyhedron $\mathcal{P}\,(M_i)$. The fastest known algorithm for solving this latter problem is due to Narayanan [96].

Lemma 4.7 *A set of smallest rate in an unweighted matroid $M_i = (E_i, \mathcal{I}_i)$ can be computed in $O(m_i^4 n_i + m_i^3 n_i^3 \tau)$ time, where τ is the time required to test whether a set of at most n_i elements is independent in M_i.*

Proof. We use Narayanan's algorithm [96] to solve problem (1) in $O(m_i^4 + m_i^3 n_i^2 \tau)$ time. Since at most n_i problems of the form (1) have to be solved to find a set of smallest *rate* in M_i, the total time needed to solve the latter problem is $O(m_i^4 n_i + m_i^3 n_i^3 \tau)$. □

The process that we have described finds a set S of smallest *rate* in M such that all the elements in S have the same weight, say w_i. To compute $\mathit{tolerance}\,(S, M)$ we just find the smallest weight w_j to which the weights of the elements in S must be lifted in order to decrease $\mathit{coverage}\,(S, M)$. If such a weight w_j exists, then $\mathit{tolerance}\,(S, M) = w_j - w_i$, otherwise $\mathit{tolerance}\,(S, M) = \infty$. Note that since all elements in S have weight w_i, then $\mathit{coverage}\,(S, M) = \mathit{coverage}\,(S, (M|E_{\leq w_i})/E_{<w_i})$. Also, since $M_i = (M|E_{\leq w_i})/E_{<w_i}$ is an unweighted matroid, then $\mathit{coverage}\,(S, M_i) = \mathit{rank}\,(E_{=w_i}, M_i) - \mathit{rank}\,(E_{=w_i} - S, M_i)$. Hence, $\mathit{coverage}\,(S, M_i)$ can be computed by performing at most $|E_{=w_i}|$ independence tests in M_i.

To test if a set $T \subseteq E_{=w_i}$ is independent in M_i, find a minimum weight base B of M, and test if $(B - B_{=w_i}) \cup T$ is independent in M. Since independence in M_i can be tested within the same time needed to test independence in M, then *coverage* (S, M_i) can be computed in $O(|E|\tau)$ time, and *tolerance* (S, M) in $O(|E|^2\tau)$ time.

Theorem 4.2 *The robustness function of a matroid $M = (E, \mathcal{I}, w, c)$ can be computed in $O(m^5n^2 + m^4n^4\tau)$ time.*

Proof. A set of smallest *rate* in M is computed by finding sets of smallest *rate* in the submatroids $(M|E_{\leq w_i})/E_{<w_i}$, for all $1 \leq i \leq p$. By Lemma 4.7, the total time required to solve these subproblems is $\sum_{i=1}^{p} O(|E_{=w_i}|^4 n + |E_{=w_i}|^3 n^3 \tau) = O(m^4 n + m^3 n^3 \tau)$. This time dominates each iteration of *uplift*, and since *uplift* performs at most mn iterations the total time needed to compute the robustness function is $O(m^5n^2 + m^4n^4\tau)$. □

Corollary 4.1 *The breakpoints of the robustness function of a matroid M can be computed in $O(m^5n^2 + m^4n^4\tau)$ time.* □

4.2 Continuous Robustness Problem for Minimum Spanning Trees

We use algorithm *uplift* to compute the robustness function for the minimum spanning trees of a graph. However, we take advantage of the special structure of graphic matroids to design a faster algorithm for computing a set of smallest *rate*.

4.2.1 Finding a Set of Edges of Smallest Rate

We consider first the problem of finding a set of edges of smallest *rate* in a connected graph $G = (V, E)$ assuming that all the edges have the same weight. Since in this graph every spanning tree has minimum weight, then any set $S \subseteq E$ with *coverage* $(S, G) > 0$ must partition the graph into *coverage* $(S, G) + 1$ components. So, what we want to do is to find a set S that minimizes the ratio $c(S)/(\textit{comps}\,(S) - 1)$, where *comps* (S, G) is the number of connected components of the graph $(V, E - S)$. This ratio is called the *strength* of the graph [25] and is denoted as *strength*(G).

The concept of strength of a graph was introduced by Cunningham [25], who gave an $O(mnM_F)$ time algorithm for computing the strength of a graph G, where M_F is the time for computing a maximum flow in G. Gusfield [57] later reduced this time to $O(mM_F)$, and recently Cunningham and Cheng [16] and Gabow [46] independently improved the time to $O(nM_F)$.

For the case when the graph G has arbitrary edge weights we can still use the above graph strength algorithms to compute a set of smallest *rate* in G. But first, we need the following property of the minimum spanning trees of G. Let $w_1, w_3, \ldots, w_p$ be the different edge weights in G. Let $\tilde{G}_{=w_i} = (\tilde{V}_{=w_i}, \tilde{E}_{=w_i})$ be the graph obtained by contracting from $G_{\leq w_i}$ all edges of weight smaller than w_i. To contract an edge (u, v), its two endpoints are replaced by a single vertex uv. All edges incident to u or v are made incident to uv. Self-loops are removed, and if there are multiple edges between uv and some vertex w, then only the edge of smallest weight

Algorithm *optimal_subset* (G)
    $s_{min} \leftarrow \infty$
    while there is at least one edge in G **do**
        Let w_i be the weight of a smallest weight edge in G.
        if $strength(G_{\leq w_i}) < s_{min}$ **then**
            $s_{min} \leftarrow strength(G_{\leq w_i})$
            $S \leftarrow$ any set of edges for which $rate\,(S, G_{\leq w_i}) = strength\,(G_{\leq w_i})$
        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.

Figure 13: Algorithm *optimal_subset*.

is retained. If several edges have smallest weight, they are replaced by a single edge with the same weight and cost equal to the sum of their costs.

Property 4.1 *Let T be a spanning tree of graph G. Tree T is a minimum spanning tree of G if and only if it contains a spanning tree for every non-trivial component of each $\tilde{G}_{=w_i}$.*

Proof. Follows from Property 2.1 □

By Property 4.1, for any subset of edges $S \subseteq E$, the value of $coverage\,(S, G)$ is equal to $\sum_{i=1}^{p} coverage\,(S, \tilde{G}_{=w_i})$. This means that if set S has smallest *rate* in G then $rate\,(S, G) = \sum_{i=1}^{p} c(S \cap \tilde{E}_{=w_i}) / \sum_{i=1}^{p} coverage\,(S, \tilde{G}_{=w_i}) = c(S \cap \tilde{E}_{=w_j}) / coverage\,(S, \tilde{G}_{=w_j})$ for any w_j such that $S \cap \tilde{E}_{=w_j} \neq \emptyset$. Therefore, every graph has a subset of edges of smallest *rate* in which all of the edges have the same weight.

Suppose that $S \subseteq E$ is a set of smallest *rate* in G and that every edge of S has weight w_i. By the above discussion we know that $rate\,(S, G) = c(S)/comps\,(S, \tilde{G}_{=w_i}) = strength\,(\tilde{G}_{=w_i})$. Hence we can find a set of smallest *rate* in S by computing the strength of each graph $\tilde{G}_{=w_i}$ and selecting the set corresponding to the minimum overall strength value. The algorithm is described in Figure 13.

Lemma 4.8 *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 $m_i \leq |E_{\leq w_i}|$ and n_i is the number of vertices of non-zero degree in $G_{\leq w_i}$. 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 the same time required to find the strength of the graph $\bar{G}$ formed by the union of the graphs $G_{\leq w_i}$ considered by *optimal_subset*. Since $\bar{G}$ has at most $2n$ vertices and m edges, we can use the graph strength algorithm in [16] or in [46] to implement *optimal_subset* in $O(n^2 m \log(n^2/m))$ time. □

The only thing that remains to be done to have a faster algorithm for computing the robustness function of a graphic matroid is to show how to compute efficiently *tolerance* (S, G) for any set S of smallest *rate* in G. If all of the edges in set S have the same weight, say w_i, then *coverage* (S, G) is equal to *comps* $(S, G_{\leq w_i}) - 1$. This fact makes it possible to compute *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 *comps* $(S, G_{\leq w_j})$ < *comps* $(S, G_{\leq w_i})$. If such a weight exists then *tolerance* $(S, G) = w_j - w_i$, otherwise *tolerance*$(S, G) = \infty$. To find w_j we proceed as follows. Build a graph G' by removing the edges in S from $G_{\leq w_i}$. Take, one by one, the edges of weight larger than w_i until we find an edge, of weight w_j, that connects two components of G'. If the edges of G are sorted non-decreasingly by weight, then we can compute *tolerance*(S, G) in $O(m)$ time.

Theorem 4.3 *Algorithm uplift computes the robustness function of a graphic matroid in* $O(n^3 m^2 \log(n^2/m))$ *time.* □

5 Transversal and Scheduling Matroids

In this section we study the problem of computing the robustness function of transversal and scheduling matroids. We use again algorithm *uplift* to compute the robustness function for these classes of matroids. However, we take advantage of the special structure of transversal and scheduling matroids to design efficient algorithms for finding a set of smallest *rate*.

5.1 Transversal Matroids

Given a bipartite graph $G = (D \cup D', E)$, a *transversal matroid* can be defined in terms of matchings in G. A transversal matroid $M = (D, \mathcal{I}, w, c)$ has as ground set, D, the set of vertices in one side of the bipartite graph G, and a subset of D is independent in M if and only if it can be covered by a matching of G. Let $w_1, w_2, \ldots, w_p$ be the different element weights in M.

Fix a transversal matroid $M = (D, \mathcal{I}, w, c)$ and an element weight w_i. Let $M_i = (D_i, \mathcal{I}_i) = (M|D_{\leq w_i})/D_{<w_i}$ be the submatroid of M whose independent sets are the sets of elements of weight w_i that belong to some minimum weight base of M. For any set $S \subseteq D_i$ and vector $x \in \mathbb{R}^{D_i}$ we let $x(S) = \sum_{e \in S} x(e)$. Let $\mathcal{P}(M_i)$ be the matroid polyhedron for submatroid M_i. It is well known [23, 29] (also see Section 3.4) that for any vector $x \in \mathbb{R}^{D_i}$,

$$\min\{\, x(S) + \textit{rank}\,(D_i - S, M_i) \mid S \subseteq D_i \,\} = \max\{y(D_i) \mid y \in \mathcal{P}(M_i) \text{ and } y \leq x\,\}. \qquad (2)$$

We exploit this relationship between the rank function of a matroid and the base of a vector in its matroid polyhedron to design an algorithm for finding a set of smallest *rate* that is more efficient than the general algorithm presented in Chapter 4. Following the same approach of the previous section, we reduce the problem of computing a set of smallest *rate* in a transversal matroid to a sequence of n parametric problems of the form (1). Using identity (2) problem (1) can be formulated as

$$\max\{y(D_i) \mid y \in \mathcal{P}(M_i) \text{ and } y \leq c/\lambda\}. \tag{3}$$

We derive below an expression for the polyhedron $\mathcal{P}(M_i)$ that allows us to solve efficiently problem (3). Consider again the bipartite graph $G = (D \cup D', A)$ that defines transversal matroid M. For any set $S \subseteq D$, let $\mathcal{N}(S) = \{v \in D' \mid v \text{ is adjacent to some vertex of } S\}$ be the set of neighbors of vertices in S. By Hall's Theorem (see e.g. [115]), graph G has a matching covering some set of vertices $S \subseteq D$ if and only if $|\mathcal{N}(S')| \geq |S'|$ for all $S' \subseteq S$. Our expression for the matroid polyhedron $\mathcal{P}(M_i)$ will come from a description of the independent sets of the submatroids M_i of the same flavor as that provided by Hall's Theorem.

Let B be a maximum weight base of M, and let $\bar{B}_{=w_i} = B - B_{=w_i}$. For any set $T \subseteq D_i$, we define the function

$$f(T) = \min\{\, |\mathcal{N}(T \cup S)| - |S| : S \subseteq \bar{B}_{=w_i} \,\}. \tag{4}$$

This function can be used to characterize the independent sets of the submatroid M_i.

Lemma 5.1 *A set $S \subseteq D_i$ is independent in M_i if and only if $|S \cap T| \leq f(T)$ for all $T \subseteq D_i$.*

Proof. Suppose that set $S \subseteq D_i$ is independent in M_i. Then, $S \cup \bar{B}_{=w_i}$ is an independent set of M, and by Hall's Theorem, $|N(S')| \geq |S'|$ for all $S' \subseteq (S \cup \bar{B}_{=w_i})$. Let $T \subseteq D_i$ and $F \subseteq \bar{B}_{=w_i}$. Clearly $(S \cap T) \cup F \subseteq (S \cup \bar{B}_{=w_i})$, and thus $|\mathcal{N}((S \cap T) \cup F)| \geq |(S \cap T) \cup F| = |S \cap T| + |F|$. Since $|\mathcal{N}(T \cup F)| \geq |\mathcal{N}((S \cap T) \cup F)|$, then $|S \cap T| \leq |\mathcal{N}(T \cup F)| - |F|$, and therefore, $|S \cap T| \leq f(T)$ for all $T \subseteq D_i$.

Suppose now that for some set $S \subseteq D_i$, it holds that $|S \cap T| \leq f(T)$ for all sets $T \subseteq D_i$. This means that $|S \cap T| \leq |\mathcal{N}(T \cup F)| - |F|$ for all $T \subseteq D_i$ and $F \subseteq \bar{B}_{=w_i}$. In particular, by choosing T to be a subset of S we get $|\mathcal{N}(T \cup F)| \geq |T| + |F| = |T \cup F|$. Thus, $|\mathcal{N}(T \cup F)| \geq |T \cup F|$ for all $(T \cup F) \subseteq (S \cup \bar{B}_{=w_i})$, and hence, by Hall's Theorem, $S \cup \bar{B}_{=w_i}$ is independent in M. □

This result and the following property of function f will provide the desired formulation for the polyhedron $\mathcal{P}(M_i)$. A set function $h : 2^H \mapsto I\!R$ is said to be *submodular* if $h(S) + h(T) \geq h(S \cup T) + h(S \cap T)$ holds for all $S, T \subseteq H$.

Lemma 5.2 *Function f is submodular.*

Proof. Let T and T' be subsets of D_i. Let $S \subseteq \bar{B}_{=w_i}$ and $S' \subseteq \bar{B}_{=w_i}$ be such that $f(T) = |\mathcal{N}(T \cup S)| - |S|$ and $f(T') = |\mathcal{N}(T' \cup S')| - |S'|$. Then (see Figure 14),

$$\begin{aligned} f(T) + f(T') &= |\mathcal{N}(T \cup S)| + |\mathcal{N}(T' \cup S')| - |S| - |S'| \\ &= |\mathcal{N}(T \cup S)| + |\mathcal{N}(T' \cup S') - \mathcal{N}(T \cup S)| + |\mathcal{N}(T' \cup S') \cap \mathcal{N}(T \cup S)| \\ &\quad - |S \cup S'| - |S \cap S'| \end{aligned}$$

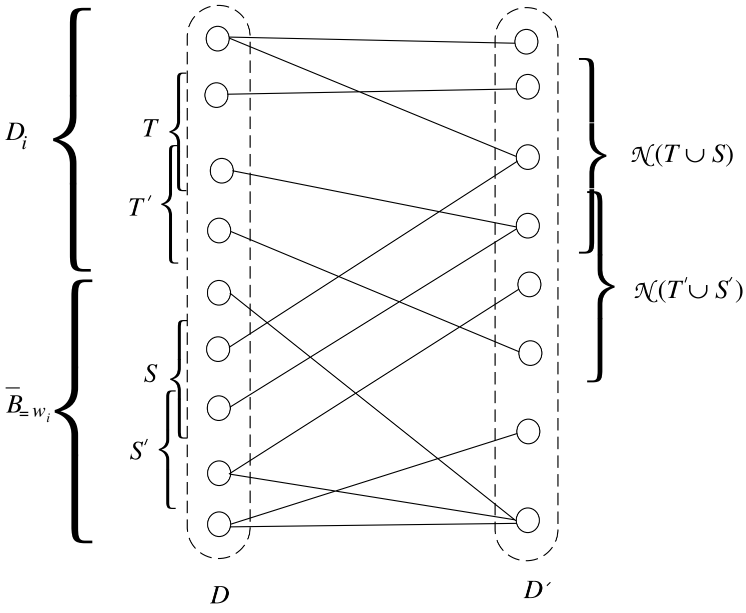

Figure 14: Auxiliary diagram to show that function f is submodular.

$$
\begin{aligned}
&\geq \; |\mathcal{N}(T \cup S \cup T' \cup S')| + |\mathcal{N}(T \cap T') \cup \mathcal{N}(S \cap S')| - |S \cup S'| - |S \cap S'| \\
&\geq \; f(T \cup T') + f(T \cap T') \qquad \square
\end{aligned}
$$

We now use the following result by Edmonds [29, 24] to give explicit representations for the independent sets of M_i and for its matroid polyhedron $\mathcal{P}(M_i)$.

Theorem 5.1 (Edmonds) *Let $h : 2^H \mapsto \mathbb{R}$ be a submodular function and $\mathcal{J} = \{ S \mid S \subseteq H$ such that $|S \cap F'| \leq h(F')$ for all $F' \subseteq H \}$. Then $\mathcal{J}$ is the family of independent sets of a matroid on H, and the convex hull of incidence vectors of members of $\mathcal{J}$ is $\mathcal{C} = \{ y = (y_1, y_2, \ldots, y_{|H|}) \mid 0 \leq y_i \leq 1$ for all $1 \leq i \leq |H|$ and $y(F') \leq h(F')$ for all $F' \subseteq H \}$.* $\square$

From this Theorem, Lemma 5.2, and Lemma 5.1 we derive the following expression for $\mathcal{P}(M_i)$.

Theorem 5.2 *The family of independent sets of $M_i = (D_i, \mathcal{I}_i) = (M|D_{\leq w_i})/D_{<w_i}$ is $\mathcal{I}_i = \{ S \mid S \subseteq D_i$ and $|S \cap F| \leq f(F)$ for all $F \subseteq D_i \}$ and its matroid polyhedron is $\mathcal{P}(M_i) = \{ z = (z_1, z_2, \ldots, z_{|D_i|}) \mid 0 \leq z_i \leq 1$ for all $1 \leq i \leq |D_i|$ and $z(F) \leq f(F)$ for all $F \subseteq D_i \}$.*

By Theorem 5.2, we can solve problem (3) using a slight modification of the polymatroid greedy-algorithm (see e.g. Chapter 3) since function f can be used as the rank function for submatroid M_i. The algorithm is described in Figure 15.

Algorithm *poly_greedy* can be implemented with a single maximum flow computation on a bipartite graph. To see how, consider an iteration of the for-loop of algorithm *poly_greedy*. Using the definition of f, the value of δ_ℓ can be rewritten as $\min \{ |\mathcal{N}(T \cup F)| - |F| - y(T) \mid e_\ell \in T \subseteq D_i$ and $F \subseteq \bar{B}_{=w_i} \}$. This minimization problem can be solved by computing a minimum cut in

Algorithm *poly_greedy* (M_i, f, c, λ)
    $y(e_\ell) \leftarrow 0$ for all $e_\ell \in D_i$
    for each $e_\ell \in D_i$ **do**
        $\delta_\ell \leftarrow \min\{\, f(T) - y(T) \mid e_\ell \in T \subseteq D_i \,\}$
        $y(e_\ell) \leftarrow \min\{1, \delta_\ell, c(e_\ell)/\lambda\}$
    end for
    Output y

Figure 15: Modified polymatroid greedy-algorithm.

an auxiliary graph $\hat{G}_\ell = (\{s,t\} \cup D_i \cup \bar{B}_{=w_i} \cup D', E_\ell)$ (see Figure 16). Graph $\hat{G}_\ell$ has an edge of infinite capacity from s to e_ℓ, and an edge of capacity $y(e_i)$ from s to each $e_i \in D_i - \{e_\ell\}$. It also has an edge of capacity 1 from s to every vertex in $\bar{B}_{=w_i}$, and an edge of infinite capacity from $v \in D_i \cup \bar{B}_{=w_i}$ to $w \in D'$ whenever the edge (v,w) belongs to the bipartite graph G. Finally, $\hat{G}_\ell$ has an edge of capacity 1 from each vertex in D' to t.

Lemma 5.3 *Let $(R, \bar{R})$ be a minimum cut of $\hat{G}_\ell$ separating s from t, and $c(R,\bar{R})$ be its capacity. Then,* $c(R,\bar{R}) - y(D_i) - |\bar{B}_{=w_i}| = \min\{\, |\mathcal{N}(T \cup F)| - |F| - y(T) \mid e_\ell \in T \subseteq D_i$ and $F \subseteq \bar{B}_{=w_i} \,\}$.

Proof. Without loss of generality assume that $s \in R$. Clearly vertex e_ℓ has to be in R. Observe that for any $v \in R$, $\mathcal{N}(v) \subseteq R$ because otherwise the minimum cut would have infinite capacity, and from Figure 16 it is apparent that a minimum cut of $\hat{G}_\ell$ has finite capacity. Let $R_{D_i} = R \cap D_i$ and $R_{\bar{B}_{=w_i}} = R \cap \bar{B}_{=w_i}$. From Figure 16 we can see that $c(R,\bar{R}) = |\mathcal{N}(R_{D_i} \cup R_{\bar{B}_{=w_i}})| + \sum_{e \in (D_i - R_{D_i})} y(e) + |\bar{B}_{=w_i} - R_{\bar{B}_{=w_i}}|$. Therefore,

$$\begin{aligned}
& c(R,\bar{R}) - \sum_{e \in D_i} y(e) - |\bar{B}_{=w_i}| \\
= \;& |\mathcal{N}(R_{D_i} \cup R_{\bar{B}_{=w_i}}) - \sum_{e \in R_{D_i}} y(e) - |R_{\bar{B}_{=w_i}}| \\
= \;& \min\{\, |\mathcal{N}(T \cup F)| - \sum_{e \in T} y(e) - |F| : e_\ell \in T \subseteq D_i \text{ and } F \subseteq \bar{B}_{=w_i} \,\}
\end{aligned}$$ □

By Lemma 5.3, we can implement *poly_greedy* by performing $|D_i|$ maximum flow computations on auxiliary graphs $\hat{G}_\ell$. However, we can do better than that. Note that in each iteration of the for-loop of *poly_greedy* the value of δ_ℓ can be computed by finding a maximum flow of $\hat{G}_\ell$ that saturates every edge leaving s, except the edge (s, e_ℓ) of infinite capacity. By Lemma 5.3, δ_ℓ is equal to the value of the flow going through edge (s, e_ℓ) in this maximum flow. If the value of the flow on edge (s, e_ℓ) is larger than $\min\{1, c(e_\ell)/\lambda\}$, then *poly_greedy* reduces it to $\min\{1, c(e_\ell)/\lambda\}$. Thus, since in each iteration of the for-loop *poly_greedy* essentially computes the maximum flow that can be sent through edge (s, e_ℓ), we can modify the auxiliary graphs so that the all values δ_ℓ can be determined with a single maximum flow computation on a new auxiliary graph G_i.

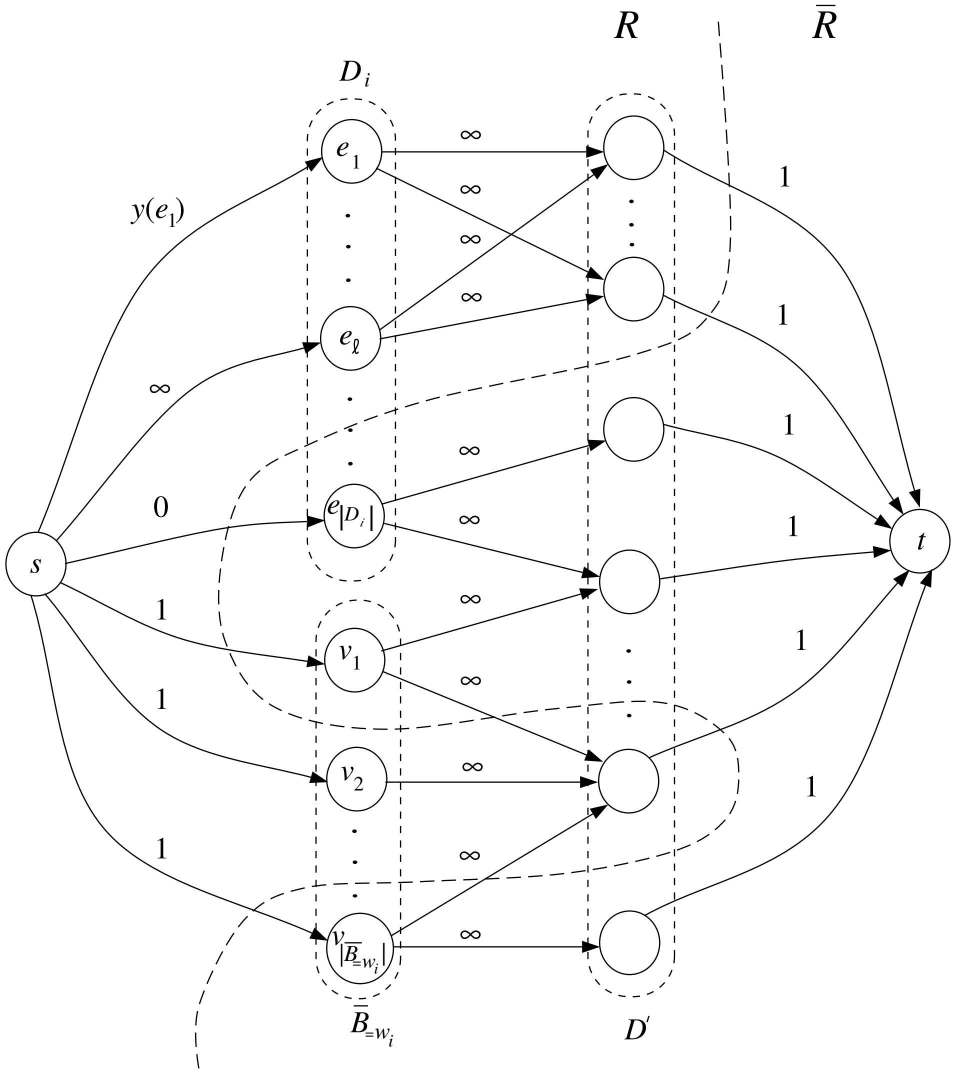

Figure 16: Auxiliary graph $\hat{G}_\ell$ showing a minimum cut $(R, \bar{R})$.

This new auxiliary graph graph $G_i = (\{s,t\} \cup D_i \cup \bar{B}_{=w_i} \cup D', E_i)$ is built with the same topology and edge capacities as any of the graphs $\hat{G}_\ell$, except that each edge from s to $e_j \in D_i$ has capacity $\min\{1, c(e_j)/\lambda\}$ (there is no edge of infinite capacity incident to s). Let z be a maximum flow for G_i that saturates every edge from s to $\bar{B}_{=w_i}$. (Such a flow must exist since the elements in $\bar{B}_{=w_i}$ form an independent set of the transversal matroid M, and hence, there has to be a matching in G_i covering them.) Let z' be the vector giving the value of the flow z through the edges from s to vertices in D_i.

Lemma 5.4 *The vector z' is a maximizer for problem* (3).

Proof. It is clear that $z'(s, e_j) \leq c(e_j)/\lambda$ for every $e_j \in D_i$. We need only prove that z' is a maximal vector in $\mathcal{P}(M_i)$ with this property. Let F and S be, respectively, subsets of D_i and $\bar{B}_{=w_i}$. Since z is a valid flow function for G_i, then it follows that $z(F \cup S) \leq \mathcal{N}(F \cup S)$. Also, $z(F \cup S) = z(F) + |S|$ because z saturates every edge from s to $\bar{B}_{=w_i}$. Combining these two inequalities we get that $z'(F) \leq \mathcal{N}(F \cup S) - |S|$ for any $F \subseteq D_i$ and $S \subseteq \bar{B}_{=w_i}$. Therefore, by Theorem 5.2, $z' \in \mathcal{P}(M_i)$.

Vector z' is maximal because the vector y computed by *poly_greedy* forms a flow function for the edges from s to D_i of value no larger than the flow z'. □

Lemma 5.5 *A set of smallest rate in the submatroid* $M_i = (D_i, \mathcal{I}_i)$ *can be computed in* $O(|D_i \cup \bar{B}_{=w_i}| + |E_i| \log(|D_i \cup \bar{B}_{=w_i}|^2/|E_i| + 2))$ *time.*

Proof. We compute a set of smallest *rate* in M_i using Newton's method to find the largest value σ_i for which $g_i(\sigma_i) = 0$, as described in the previous section. Note that each iteration of Newton's method decreases the value of the parameter λ of problem (1) and increases the value of vector c/λ that defines the capacities of the edges from s to D_i in G_i. Hence in two successive iterations of Newton's method, the only change in G_i is an increase in the capacities of some edges leaving the source vertex. Using parametric flow techniques, all iterations of Newton's method can be performed in $O(|D_i \cup \bar{B}_{=w_i}| * |E_i| \log(|D_i \cup \bar{B}_{=w_i}|^2/|E_i| + 2))$ time using the algorithm FIFO with dynamic trees described in [2].

This algorithm gives only the value of the *rate* of a set of smallest *rate*, we show now how to find such a set. Let z' be the vector giving the value of the final flow through the edges from s to vertices in D_i. Let λ_i be the *rate* of a set of smallest *rate* in M_i. We show that the set $S \subseteq D_i$ formed by all elements $e \in D_i$ such that $z'(s, e) = c(e)/\lambda_i$ is a set of smallest *rate* in M_i. To see this note that $\min\{\, c(T)/\lambda_i + \mathit{rank}\,(D_i - T, M_i) \mid T \subseteq D_i \,\} = \max\{y(D_i) \mid y \in \mathcal{P}(M_i) \text{ and } y \leq c/\lambda_i \,\} = z'(D_i)$, hence $z'(D_i) = c(S)/\lambda_i + \mathit{rank}(D_i - S, M_i)$, and therefore S is a set of smallest *rate* in M_i. □

Theorem 5.3 *The breakpoints of the robustness function of a transversal matroid* M *can be computed in* $O(mn(m + n^2)|E| \log(m^2/|E| + 2))$ *time, where* E *is the set of edges in its bipartite graph.*

Proof. By the previous lemma, the total time required to find a set of smallest *rate* in M is $O(\sum_{i=1}^{p} |D_i \cup \bar{B}_{=w_i}| * |E_i| \log(|D_i \cup \bar{B}_{=w_i}|^2/|E_i| + 2)) = O(|E| \log(m^2/|E| + 2) \sum_{i=1}^{p} |D_i \cup \bar{B}_{=w_i}|)$. Observe that the sets D_i and $\bar{B}_{=w_i}$ are disjoint and that $\sum_{i=1}^{p} |D_i| = |D| = m$. Furthermore, note that at most n of the matroids M_i have to be considered when computing a set of smallest *rate* in M. To see why, consider a minimum weight base B of M. Let w_i be the weight of some element in M such that no element of B has weight w_i. Then, the only independent set in matroid $M_i = (M|E_{\leq w_i})/E_{<w_i}$ is the empty set. Therefore, $\sum_{i=1}^{p} |\bar{B}_{=w_i}| \leq n^2$, and the total time needed to compute a set of smallest rate in matroid M is $O((m + n^2)|E| \log(m^2/|E| + 2)$. Algorithm *uplift* adds an additional factor mn to this time. □

5.2 Scheduling Matroids

Scheduling matroids arise in the following class of scheduling problems. Let $J = \{j_1, j_2, \ldots, j_m\}$, be a set of unit-time jobs. Each job j_i has a weight $w(j_i)$, and integer release time r_i and deadline

d_i. The problem is to select a largest subset of jobs of minimum total weight that can be executed on a single processor. Without loss of generality we assume that the largest deadline has value at most m.

This problem is equivalent to a weighted matching problem on a *convex* bipartite graph $G = (J \cup T, A)$ (see e.g. [43]). To build G we think of the ith vertex of J as job j_i, and the ith vertex of T as the time interval from $i-1$ to i. For each job j_i, graph G has edges of weight $w(j_i)$ from j_i to each time interval between r_i and d_i. A maximum cardinality matching of minimum weight in G defines a scheduling of a largest subset of J of minimum total weight.

A scheduling matroid $M = (J, \mathcal{I}, w, c)$ has a set J of jobs as its ground set and a subset of jobs is independent if and only if there is a valid scheduling for them. By the above discussion, the class of scheduling matroids is a proper subset of the class of transversal matroids. Let $w_1, w_2, \ldots, w_p$ be the different weights of elements in M.

In the previous section we showed how to find a set of smallest *rate* in a transversal matroid by performing a series of minimum-cut computations over auxiliary bipartite graphs G_k. We use this same approach here, but now we can design a faster algorithm by taking advantage of the fact that the auxiliary graphs G_k are convex bipartite.

Fix an auxiliary graph G_k. Let $M_k = (J_{=w_k}, \mathcal{I}_k) = (M|J_{\leq w_k})/J_{<w_k}$. For reasons that will be apparent later, we explicitly perform all the maximum flow computations on auxiliary graph G_k needed to find the largest value σ_k for which $g_k(\sigma_k) = 0$, instead of using parametric flow techniques as we did for transversal matroids. We show that at most $|B_{=w_k}|$ maximum flow computations have to be performed on G_k, where B is a minimum weight base of M. For convenience we show graph G_k in Figure 17.

In the sequence of maximum flow computations that have to be performed over G_k the value of the parameter λ is monotonically decreased, and so the capacities of the edges form s to $J_{=w_k}$ are monotonically increased. Let $f_k(1/\lambda)$ denote the value of a maximum flow in G_k as a function of $1/\lambda$. It is well known [30, 107] that in an s-t graph in which the capacities of the edges leaving the source are linear functions of a parameter $1/\lambda$, the value of a maximum flow is a piecewise-linear concave function of $1/\lambda$.

By definition of B, there is a matching of maximum cardinality in G that covers the vertices in B. Therefore, for any value of the parameter $1/\lambda$ there is a maximum flow of G_k that saturates at least $|\bar{B}_{=w_k}|$ edges incident to vertex t. Let us consider this maximum flow. Note that if λ is decreased, the value of the flow increases linearly with $1/\lambda$ until one more edge incident to t is saturated. Since a maximum matching of G has size $|B|$, then for any value of $1/\lambda$ at most $|B|$ of the edges incident to t can be saturated by a flow. This implies that $f_k(1/\lambda)$ has at most $|B_{=w_k}|$ breakpoints.

It is easy to see that a set of smallest *rate* in a scheduling matroid $M = (J, \mathcal{I}, w, c)$ has *rate* $\lambda^* \leq c(J)/n$. To compute the value of λ^*, we use Newton's method on each function $f_k(1/\lambda)$ to find the largest value λ_k for which $f_k(1/\lambda_k) = |B|$. By the above discussion, the value of λ_k can be found by performing at most $|B_{=w_k}|$ maximum flow computations on G_k. The value of λ^* is equal to the smallest λ_k, for $k = 1, 2, \ldots, p$.

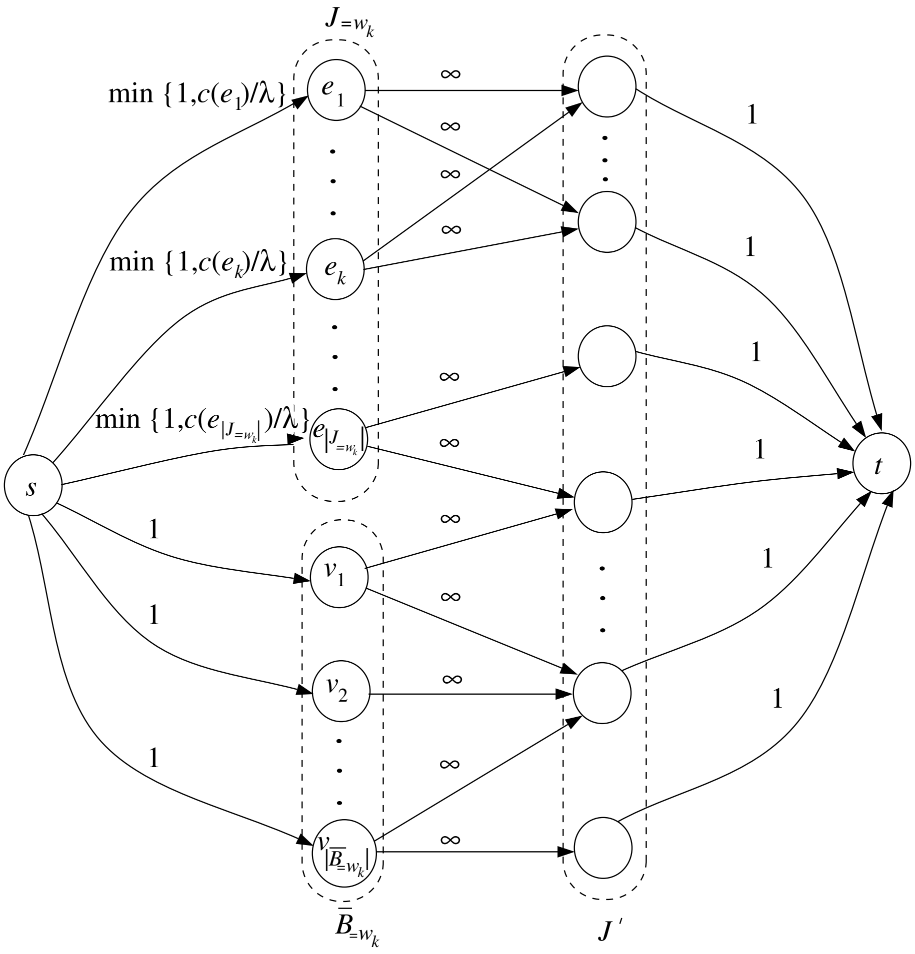

Figure 17: Auxiliary graph G_k.

We show that a maximum flow of G_k that saturates all edges from s_k to $\bar{B}_{=w_k}$ can be computed in $O(|J_{=w_k} \cup \bar{B}_{=w_k}|)$ time, whereas the previous best algorithm for finding a maximum flow in a bipartite graph G_k needs $O(|J_{=w_k} \cup \bar{B}_{=w_k}||E_k|\log(|J_{=w_k} \cup \bar{B}_{=w_k}|^2/|E_k| + 2)$ time [2]. We describe first how to find in linear time a maximum flow for G_k, and then we show how to modify it so that it saturates all edges from s_k to $\bar{B}_{=w_k}$ as required by Lemma 5.4.

Consider one of the auxiliary graphs $G_k = (\{s, t\} \cup J_{=w_k} \cup \bar{B}_{=w_k} \cup J', E_k)$. The problem of computing a maximum flow of G_k can be interpreted as a scheduling-with-preemption problem on a single processor. Let the vertices $\tau_k \in J'$ be time slots and the vertices $j_i \in J_{=w_k} \cup \bar{B}_{=w_k}$ be jobs. The capacity of edge (s, j_i) is the processing time required by job j_i, and each infinite capacity edge (j_i, τ_k) identifies a time slot τ_k where job j_i can be scheduled. Under this interpretation, any flow function for G_k defines a feasible preemptive schedule for various portions of the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$: the value of the flow on edge (s, j_i) determines the total time that job j_i is scheduled for execution, and the flow on edge (j_i, τ_k) defines the portion of job j_i that is scheduled in time interval τ_k. The converse of this statement is also true, i.e., a preemptive schedule that specifies the execution order of portions of the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$

on a single processor defines a valid flow for G_k.

By the previous discussion, any algorithm that finds a preemptive schedule for the jobs in $\bar{B}_{=w_k} \cup J_{=w_k}$ that maximizes the amount of time that the jobs are executed on a single processor, also computes a maximum flow for G_k. We present below an efficient implementation of the *preemptive earliest deadline first rule* [71], which finds in linear time an optimal preemptive schedule for the jobs $\bar{B}_{=w_k} \cup J_{=w_k}$, and thus, a maximum flow for G_k.

We assume that the largest deadline among the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$ is at most $|J_{=w_k} \cup \bar{B}_{=w_k}|$. If this condition does not hold, we use the techniques in [34] to modify the deadlines so that this condition is satisfied. This preprocessing step requires $O(|J_{=w_k} \cup \bar{B}_{=w_k}|)$ time. The extra time required by this step does not affect the overall time complexity of our algorithm.

A straightforward implementation of the preemptive earliest deadline first rule uses an initially empty set T to store all those jobs that can be scheduled at a given time. Starting at the earliest release time, and moving forward in time, a schedule is generated as follows. A job is added into T whenever it becomes available for scheduling, and in each time interval (a fraction of) the job with smallest deadline in T is scheduled.

Our scheduling problem can be seen as an extension of the *off-line min problem* defined in [1]. In this latter problem we are given a set of integers $\{1, 2, \ldots, i\}$, an initially empty set T, and two operations: *insert*, that adds an integer to T, and *extract_min*, that removes the smallest integer from T. It is desired to maintain T under a given sequence of *insert* and *extract_min* operations, assuming that each integer is inserted in T only once.

As in an off-line min problem, we want to maintain in our scheduling problem a set of jobs T under some sequence $\mathcal{S}$ of two operations: insertion of all the jobs with a particular release time, and extraction of a job with smallest deadline. However, contrary to the off-line min problem, we might have several jobs with the same deadline, and we do not know in advance how many jobs will be extracted between two consecutive insertion operations. We define the *generalized off-line min problem* over a set of elements $\{e_1, e_2, \ldots, e_m\}$ as follows. Each element has two attributes, an integral *value* from the domain $\{1, 2, \ldots, m\}$, and a positive real *magnitude* no larger than 1. There is an initially empty set T over which two operations are defined, *insert* and *retrieve.* Operation *insert* adds an element to T. Operation *retrieve* extracts from T elements of smallest value until either T is empty or total magnitude 1 is achieved, splitting the magnitude of the last element before extraction, as necessary. Given a sequence $\mathcal{S}$ of *insert* and *retrieve* operations, the generalized off-line min problem consists in finding which portions of what elements are removed by each *retrieve* operation.

Consider the following instance of the generalized off-line min problem defined over the set $\{a, b, c, d, e\}$. The values of the elements are $1, 1, 3, 4$, and 2, respectively, and their magnitudes are $0.3, 0.5, 0.4, 0.8$, and 1.0. The sequence $\mathcal{S}$ is: *insert* a, *insert* b, *insert* c, *retrieve*, *insert* d, *insert* e, *retrieve*, *retrieve.* The first *retrieve* operation extracts from T elements a and b, and half of element c (so it leaves in T the rest of element c with magnitude 0.2). The second *retrieve* operation extracts element e, while the third *retrieve* extracts the remaining of c plus element d.

```
Algorithm generalized_off-line(S, {e_1, e_2, ..., e_m})
    Initialize each Q(i) to ∅, and set j ← 1.
    while j < m do
        i ← FIND(e_j).
        Add e_j to the end of list Q(i).
        Find S_m, the sum of magnitudes of the elements in Q(i).
        if S_m ≥ 1 then
            UNION(i, succ(i), succ(i))
            magnitude (e_j) ← S_m − 1
        end if
        if S_m ≤ 1 then j ← j + 1 end if
    end while
    Output Q
```

Figure 18: Algorithm *generalized_off_line*

To solve the generalized off-line min problem, it is convenient to write the sequence $\mathcal{S}$ as $I_1, R, I_2, R, \ldots, I_q, R$, where R is a *retrieve* operation and each I_i is a sequence of *insert* operations. Let i be the set of elements inserted by I_i. As in [1], we use disjoint-set data structures to represent each set i; these sets are stored in a doubly linked list with $succ(i)$ the successor of i. We use an array Q, and store in $Q(i)$ a list with the elements extracted by the ith *retrieve* operation. Let the elements be indexed non-decreasingly by value. The algorithm to solve the generalized off-line min problem is described in Figure 18. Function FIND(x) returns the set to which element x belongs and function UNION(x, y, y) on sets x and y, makes $y \leftarrow x \cup y$.

Lemma 5.6 *Algorithm generalized_off-line runs in $O(m)$ time.*

Proof. We can use *counting sort* (see e.g. [22]) to index the elements non-decreasingly by value, as required by *generalized_off-line*, in $O(m)$ time. Note that the only UNION operations that the algorithm performs are of the form UNION (i, *succ* (i), *succ* (i)). This special order of the UNION operations allows us to use the algorithm of Gabow and Tarjan [44] to perform all the UNION and FIND operations in $O(m)$ time. $\square$

We use algorithm *generalized_off-line* to find an optimal scheduling for the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$ as follows. We initialize the data structures of *generalized_off-line* by storing in set i all jobs with the ith smallest release time. The attribute magnitude of each job is set equal to the processing time for the job and the attribute value is equal to the deadline of the job. The jobs are indexed non-decreasingly by deadline. Run algorithm *generalized_off-line* with the following slight change: in each iteration of the while-loop, after finding the set i containing job e_j, *generalized_off-line* inserts e_j in $Q(i)$ only if its deadline is at least i (otherwise job e_j cannot be scheduled in the ith time slot). The schedule can easily be obtained from the information that *generalized_off-line* stores in Q.

Lemma 5.7 *An optimal scheduling for the jobs $J_{=w_k} \cup \bar{B}_{=w_k}$ can be computed in $O(|J_{=w_k} \cup \bar{B}_{=w_k}|)$ time and $O(|J_{=w_k} \cup \bar{B}_{=w_k}|)$ space.* □

The maximum flow of the auxiliary graph G_k corresponding to the schedule generated by *generalized_off-line* might not saturate all edges from s_k to $\bar{B}_{=w_k}$. We show below how to find a flow that complies with this condition.

First, modify the deadlines of the jobs in $\bar{B}_{=w_k}$ so that no two of them have the same deadline. To do this, find an optimal schedule for the jobs in $\bar{B}_{=w_k}$ using a time-reversed version of the preemptive earliest deadline first rule. This new rule schedules jobs by starting at the largest deadline and moving backwards in time towards the smallest release time. The rule schedules in each interval the job with latest release time that is available. The correctness of this rule can be easily established. Let S' be the schedule for the jobs in $\bar{B}_{=w_k}$ obtained with this rule. Modify the deadline of each job $j_i \in \bar{B}_{=w_k}$ by making it equal to the completion time of job j_i in S'. Let d'_i be the modified deadline for job j_i.

Then, modify algorithm *generalized_off-line* so that whenever two or more jobs have the same modified deadline, it chooses to schedule first the jobs from $\bar{B}_{=w_k}$. (The only change that we actually need to make is to index the jobs so that if several jobs have the same modified deadline, the jobs from $\bar{B}_{=w_k}$ are indexed first.) Use this modified *generalized_off-line* algorithm to find a schedule S^* for the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$.

Lemma 5.8 *The schedule S^* maximizes the amount of time that the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$ are executed on a single processor, and it schedules to completion all jobs from $\bar{B}_{=w_k}$.*

Proof. We need only prove that there is an optimal schedule $\hat{S}$ for the jobs in $J_{=w_k} \cup \bar{B}_{=w_k}$ in which all jobs from $\bar{B}_{=w_k}$ are scheduled to completion, and such that no job $j_i \in \bar{B}_{=w_k}$ is scheduled after its modified deadline d'_i. The proof is by contradiction.

Let S' be the schedule used to determine the modified deadlines d'_i. Suppose there is no optimal schedule $\hat{S}$ for the modified deadlines, but there is one for the original deadlines. Choose S to be an optimal schedule for the original deadlines that schedules to completion all jobs from $\bar{B}_{=w_k}$, such that the first job $j_i \in \bar{B}_{=w_k}$ that is not completed by its modified deadline has d'_i as large as possible. Since jobs are scheduled in S' as late as possible, there must be at least one job $j_k \in \bar{B}_{=w_k}$ with $d'_k > d'_i$ and $r_k \geq r_i$ that is scheduled in S to be completed at a time $\ell_k < d'_i$. Clearly, we can modify S so that j_i is scheduled within the portions of time alloted to j_k, and j_k is scheduled in the time intervals assigned to j_i. This modification to S produces a new optimal scheduling in which the completion time for j_i does not exceed d'_i. This is a contradiction. □

Theorem 5.4 *The robustness function of a scheduling matroid $M = (J, \mathcal{I}, w, c)$ can be computed in $O(m^2n^2)$ time, where $m = |J|$ and $n = |B|$ is the size of a maximum independent set in $\mathcal{I}$.*

Proof. In each auxiliary graph G_k, the largest value λ_k for which a maximum flow has value $|B|$ can be computed in $O(|B_{=w_k}||J_{=w_k} \cup \bar{B}_{=w_k}|)$ time. Hence, the total time used to find a set of smallest *rate* in the scheduling matroid M is $O(\sum_{i=1}^{p} |B_{=w_k}||J_{=w_k} \cup \bar{B}_{=w_k}|) =$

$O|B||D| + |B|^2) = O(mn)$. Since algorithm *uplift* computes $O(mn)$ sets of smallest *rate*, the total time needed to compute all the breakpoints in the robustness function in $O(m^2n^2)$. $\square$

6 Partition Matroids

In this section we study the problem of computing the robustness function of a partition matroid. As for transversal and scheduling matroids, we use algorithm *uplift* to compute the robustness function. But, we show how to exploit the simple structure of partition matroids in the design of a very efficient algorithm for finding a set of smallest *rate*. Furthermore, we show that for partition matroids, algorithm *uplift* needs to perform at most $O(m)$ iterations.

We also study the problem of evaluating the robustness function at a single point, instead of generating all the breakpoints. For this version of the problem we design a sophisticated prune-and-search algorithm that optimally solves it.

6.1 Robustness Function for Partition Matroids

A *partition matroid* $P = (E, \mathcal{I}, w, c)$ is defined over a finite set E of m elements, partitioned into ℓ disjoint *blocks* $E_1, E_2, \ldots, E_\ell$. Given a set of ℓ positive integers $n_i \leq |E_i|, i = 1, \ldots, \ell$, a set $S \subseteq E$ is independent in P if and only if $|S \cap E_i| \leq n_i$, for all $1 \leq i \leq \ell$. A minimum weight base of P is a minimum weight subset B of E such that $|B \cap E_i| = n_i$, for all $1 \leq i \leq \ell$.

Consider a partition matroid $P = (E, \mathcal{I}, w, c)$. The blocks E_i of E are independent in the sense that any change in the weights of the elements in some block E_i does not affect the set of elements of another block E_j that might belong to a minimum weight base. Hence, there must be at least one block E_i that contains a subset of elements having the smallest *rate* in P. This observation simplifies the problem of finding a set of smallest *rate* since we need only show how to compute a set of smallest *rate* in one of the blocks E_i, or equivalently, how to find a set of smallest *rate* in a uniform matroid. A uniform matroid is a partition matroid in which $\ell = 1$. In a uniform matroid $P = (E, \mathcal{I}, w, c)$ of rank n a set $S \subseteq E$ is independent if and only if $|S| \leq n$.

Fix a uniform matroid $U = (E, \mathcal{I}, w, c)$ with rank n. Let w_n be the weight of the nth smallest element in E. Note that the only elements of E that can belong to a minimum weight base of U are those elements of weight at most w_n. Let $E_{<w_n}$, $E_{=w_n}$, and $E_{>w_n}$ denote the subsets of E formed by all elements of weight smaller than w_n, equal to w_n, and larger than w_n, respectively. Let $\Delta = |E_{<w_n}| + |E_{=w_n}| - n$.

Lemma 6.1 *If $S \subseteq E$ is a set of smallest rate in U then*

$$\begin{aligned} \textit{rate}\,(S, U) = \min \{ \quad & \min \{ c(e) \mid e \in E_{<w_n} \}, \\ & \min \{ c(T)/(|T| - \Delta) \mid T \subseteq E_{=w_n} \text{ and } |T| > \Delta \} \} \end{aligned}$$

Proof. For any set $T \subseteq E_{<w_n}$, *coverage* $(T, U) = |T|$ since all elements in $E_{<w_n}$ belong to every minimum weight base of U. Also, for any set $T \subseteq E_{=w_n}$, *coverage* $(T, U) = \max\{0, |T| - \Delta\}$

Algorithm *descend* $(T_{=w_n}, U)$
  Let x be the root of $T_{=w_n}$.
  while x is not a leaf **do**
    if $|L(x)| \leq \Delta$, or $rate\,(L(x), U) > rate\,(L(next\,(x)), U)$ **then**
      $x \leftarrow$ right child of x
    else $x \leftarrow$ left child of x **end if**
  end while
  if (cost of the element stored in node x) $= rate\,(L(x), U)$ **then**
    Find the rightmost leaf y that stores an element of cost equal to $rate\,(L(x), U)$.
    Output the set of elements in $L(y)$
  else Output the set of elements in $L(x)$ **end if**

Figure 19: Algorithm to find the largest set of smallest *rate* in $E_{=w_n}$.

because there is a maximum weight base of U that contains $\min\{n - |E_{<w_n}|, |E_{=w_n}| - |T|\}$ elements from $E_{=w_n} - T$. □

By Lemma 6.1, there is a set of smallest *rate* in U that either consists of a single element of smallest cost in $E_{<w_n}$, or that is formed by the k elements of smallest cost in $E_{=w_n}$, for some $\Delta < k \leq |E_{=w_n}|$. The following observation allows us to compute efficiently the value of k. Let $\{e_1, e_2, \ldots, e_{|E_{=w_n}|}\}$ be the elements of $E_{=w_n}$ indexed in non-decreasing order of cost.

Lemma 6.2 *The discrete function $f(i) = \sum_{j=1}^{i} c(e_j)/(i - \Delta)$ with integer argument i is strictly decreasing in the interval $\Delta + 1 \leq i \leq k'$, and non-decreasing in the interval $k' \leq i \leq |E_{=w_n}|$, where k' is the smallest integer where f reaches its minimum value.*

Proof. Is it not difficult to show that if $f(i) \geq f(i-1)$ for any $\Delta + 2 \leq i \leq |E_{=w_n}| - 1$, then $f(i+1) \geq f(i)$. □

We use a balanced binary search tree $T_{=w_n}$ to compute efficiently the value of k . The elements of $E_{=w_n}$ are stored in the leaves of $T_{=w_n}$, maintaining a dictionary order on the costs. Given a node x of $T_{=w_n}$, let $L(x)$ be the set of all elements stored in the leaves of $T_{=w_n}$ that appear before x in an in-order traversal of the tree. Let $next(x)$ denote the first leaf of $T_{=w_n}$ that appears after x in an in-order traversal of the tree. The largest set of smallest *rate* in $E_{=w_n}$ can be found by a search in $T_{=w_n}$ from the root to the leaves as described in Figure 19. The correctness of this algorithm follows from Lemma 6.2.

Lemma 6.3 *Algorithm descend runs in $O(\log |T_{=w_n}|)$ time.*

Proof. Since $rate\,(L(x), U) = c(L(x))/(|L(x)| - \Delta)$, then to compute $rate\,(L(x), U)$ we need to know in each iteration of *descend* the number of elements in $L(x)$ and their total cost. This can easily be computed as the algorithm traverses the tree if each internal node x of $T_{=w_n}$ stores the number and total cost of all elements in the subtree rooted at x, and the smallest cost of any element in that subtree. This information can also be used by the algorithm to compute *rate* $(L(next(x)), U)$. Since $T_{=w_n}$ is a balanced tree, the algorithm runs in $O(\log |T_{=w_n}|)$ time. $\square$

Let $H_{<w_n}$ be a min-heap that stores the elements of $E_{<w_n}$ maintaining heap order on their costs. With this heap and algorithm *descend*, a set of smallest *rate* in U can be found in $O(\log |T_{=w_n}|)$ time.

We describe now how our data structures need to be updated so that a set of smallest *rate* in U can be computed efficiently in each iteration of algorithm *uplift*. We use an additional data structure, a list $L_{>w_n}$ containing the elements of $E_{>w_n}$ in non-decreasing order of weight. There are two cases that have to be considered: (1) if a singleton $\{e_s\} \subseteq E_{<w_n}$ is chosen as the subset of smallest *rate* in U, then *uplift* increases the weight of e_s up to w_n and moves e_s from $H_{<w_n}$ into $T_{=w_n}$; (2) if a set $S \subseteq E_{=w_n}$ is the set of smallest *rate* in U, then algorithm *uplift* increases the weights of the elements in S to w_g, the smallest element weight larger than w_n. Since then w_g becomes the weight of the nth smallest element in E, all elements in $E_{=w_n} - S$ are moved from $T_{=w_n}$ to $H_{<w_n}$. Also each element of weight w_g in $L_{>w_n}$ is removed form $L_{>w_n}$ and inserted in $T_{=w_n}$.

Lemma 6.4 *The robustness function of a uniform matroid* $U = (E, \mathcal{I}, w, c)$ *can be computed in* $O(|E| \log |E|)$ *time and using* $O(|E|)$ *space.*

Proof. We first show that *uplift* performs at most $2|E| - n$ iterations. In each iteration in which *uplift* selects some set $S \subseteq E_{=w_n}$ as the set of smallest *rate* in U, at least one element is removed from $L_{>w_n}$. This can happen at most $|E| - n$ times, since no element is ever inserted into $L_{>w_n}$ while updating the data structures. In each iteration in which *uplift* chooses a singleton $\{e_s\}$ as the set of smallest *rate*, the element e_s is moved from $H_{<w_n}$ to $T_{=w_n}$, and it remains in $T_{=w_n}$ until the algorithm ends. To show this, suppose that in a later iteration j, element e_s is moved back to $H_{<w_n}$. This means that in iteration j a set $T \subseteq E_{=w_n}$, with $e_s \notin T$, has the smallest *rate* in U. Since *descend* finds the largest set of smallest *rate* in $E_{=w_n}$, it follows that $c(e_s) > rate\,(T, U)$. But, this is not possible since the robustness function is non-decreasing. Therefore, the number of iterations in which a singleton is selected as a set of smallest *rate* is at most $|E|$, and hence the total number of iterations that *uplift* performs is at most $2|E| - n$.

A set S of smallest *rate* in U can be found in $O(\log |E|)$ time, and *tolerance* (S, U) can be computed in constant time assuming that the weights of the elements are initially sorted non-decreasingly by weight.

It only remains to bound the time required to update the data structures. Removing the smallest element from $H_{<w_n}$ takes $O(\log |H_{<w_n}|)$ time, and removing the first element from $L_{>w_n}$ can be done in constant time. Inserting an element into $T_{=w_n}$ takes $O(\log |T_{=w_n}|)$ time. Since, by the above discussion, any element is removed from $H_{<w_n}$ or $L_{>w_n}$ at most once, and an element can be inserted into $T_{=w_n}$ at most twice, the total time required to update the data structures is $O(|E| \log |E|)$. $\square$

We now consider the complexity for computing the robustness function of a partition matroid $P = (E, \mathcal{I}, w, c)$ with more than one block.

Theorem 6.1 *The robustness function of a partition matroid $P = (E, \mathcal{I}, w, c)$ can be computed in $O(|E| \log |E|)$ time and using $O(|E|)$ space.*

Proof. Let $E_1, E_2, \ldots, E_\ell$ be the blocks of E. We have shown above how to compute efficiently a set of smallest *rate* in each block E_i. To find the set with overall smallest *rate* in P, we use a heap H in which we store the *rate* of a set of smallest *rate* in each E_i, for all $i = 1, 2, \ldots, \ell$.

The arguments used in the proof of Lemma 6.4 can be extended to show that *uplift* performs only $O(|E|)$ iterations. Hence, *uplift* computes the robustness function of P in $O(|E| \log |E|)$ time. The overall space used by our data structures is $O(|E|)$. $\square$

6.2 Evaluating the Robustness Function at One Point

In the previous section we presented an algorithm that computes all the breakpoints of the robustness function F_P for a partition matroid P. If we do not want to compute all the breakpoints of F_P, but only wish to evaluate F_P at a specific point b, then we can design an algorithm that requires linear time.

Consider a partition matroid $P = (E, \mathcal{I}, w, c)$ with blocks $E_1, E_2, \ldots, E_\ell$. Let $P_i = (E_i, \mathcal{I}_i, w, c)$ be the uniform matroid induced by block E_i, for $i = 1, 2, \ldots, \ell$. For some given budget b, the value of $F_P(b)$ can be computed by determining the optimal way of distributing the budget among the matroids P_i, and by optimally spending the fractional budget b_i assigned to each P_i increasing the weights of its elements. Therefore, we can write

$$F_P(b) = \max\{\sum_{i=1}^{\ell} F_{P_i}(b_i) \mid b_i \geq 0 \text{ for all } 1 \leq i \leq \ell \text{ and } \sum_{i=1}^{\ell} b_i = b\} \quad (5)$$

For arbitrary functions F_{P_i}, problem (5) is known as the *optimal distribution of effort problem* [75] or as the *convex knapsack problem* [31]. There are several efficient algorithms to solve these problems [68, 31], but only under the assumption that the functions F_{P_i} are given in an explicit form that make it possible to compute in constant time the value of $F_{P_i}(x)$ for any $x \geq 0$ and $1 \leq i \leq \ell$. Since we do not have an explicit representation of the robustness functions F_{P_i}, we do not know how to compute the value of $F_{P_i}(x)$ in constant time, and thus, we have not found efficient implementations of these algorithms for our problem. Instead, we present here a prune-and-search algorithm to compute $F_P(b)$ in $O(m)$ time.

Let $\delta = \delta_1, \delta_2, \ldots, \delta_k$ be a sequence of increases on the weights of the elements of a partition matroid P. We say that δ is a *canonical sequence of increases* if each δ_i increases only the weights of the elements in the largest set S of smallest *rate* in P, and each element in S has its weight increased by the same amount $d_i \leq$ *tolerance* (S, P). Algorithm *uplift* can be implemented to determine a canonical sequence of increases, and therefore, a canonical sequence of increases can be used to compute $F_P(b)$ for any $b \geq 0$.

The following property of the largest sets of smallest *rate* in P plays a key role in our algorithm.

Property 6.1 *Let $\delta = \delta_1, \delta_2, \ldots, \delta_k$ be a canonical sequence of increases and S_i be the set chosen by some δ_i, $i < k$. In all subsequent increases δ_j, $j > i$, all elements of S_i will undergo the same weight changes.*

Proof. Let e_1 and e_2 be elements of S_i. Note that the cost of e_1 (e_2) cannot be larger than the *rate* r_{S_i} of S_i, because otherwise $S_i - \{e_1\}$ ($S_i - \{e_2\}$) would have a smaller *rate* than S_i. Suppose that δ_j, $j > i$, selects set $S_j \subseteq E$, with *rate* r_{S_j}, and that $e_2 \in S_j$ but $e_1 \notin S_j$. This means that $r_{S_j} < c(e_1)$, because otherwise $S_j \cup \{e_1\}$ would be a larger set with *rate* at most r_{S_j}. But, then $r_{S_j} < r_{S_i}$ since $c(e_1) \leq r_{S_i}$. This contradicts that the robustness function of P is non-decreasing. □

Property 6.1 can be used to design a slightly more efficient version of *uplift* than that presented in the previous section. The idea is that if in some iteration of *uplift*, set $S \subseteq E$ with $|S| > 1$ is selected as the set of smallest *rate*, then we can replace S by a single meta-element e_S. We can do this, since in the succeeding iterations *uplift* does not need to keep track of the individual weight changes of the elements in S. Although this modified algorithm is more efficient than the original one, its time complexity is still $O(|E| \log |E|)$. We give below a different approach that exploits Property 6.1 to yield a linear time algorithm for computing $F_P(b)$, for a fixed value $b \geq 0$. To introduce our basic strategy, we show first how to compute $F_P(b)$ for a uniform matroid.

6.2.1 Uniform Matroid

Let $\delta(w_i)$, for any value $w_i \geq 0$ be a canonical sequence of increases for the elements of a partition matroid P such that the weights of the elements are increased as much as possible without increasing the weight of any element above w_i. Let $\bar{\delta}(r_i)$, for any value $r_i \geq 0$ be a canonical sequence of increases such that the weights of the elements in P are increased as much as possible without selecting a set of *rate* at least r_i.

Given a uniform matroid $U = (E, \mathcal{I}, w, c)$, our algorithm for evaluating $F_U(b)$, for some $b \geq 0$, does not perform a linear search over the curve F_U, as *uplift* does, but evaluates F_U at some sequence of probe values that converge to the desired one. The algorithm performs two different types of probes, each one implemented by a prune-and-search linear-time routine. The first routine, called *up_to_weight*, takes a weight w_i and computes the weight increases corresponding to $\delta(w_i)$. The routine determines the cost of the weight increases and the *rate* of a set of smallest *rate* according to the increased weights. The second routine, called *up_to_rate*, takes a *rate* r_i and computes the weight increases corresponding to $\bar{\delta}(r_i)$. This routine determines the total cost of the weight increases and the weight of the nth smallest element according to the increased weights.

We present below linear time implementations for these routines. Routine *up_to_weight* described in Figure 20, receives as arguments a weight w_i, the set of elements E, and the weight and cost functions w and c.

Lemma 6.5 *Algorithm up_to_weight finds in linear time the total cost of the weight increases*

Algorithm *up_to_weight* (w_i, E, w, c)
Find the largest set S of smallest *rate* in $E_{\leq w_i}$ assuming that all elements in $E_{\leq w_i}$ have the same weight.
Output the *rate* of S and the cost of increasing the weights of the elements in S to w_i.

Figure 20: Algorithm *up_to_weight.*

made by $\delta(w_i)$ and the rate of a set of smallest rate according to the increased weights.

Proof. We first describe how to compute in linear time the largest set of smallest *rate* in $E_{\leq w_i}$ assuming that all elements in $E_{\leq w_i}$ have the same weight. Perform a binary search on the costs of the elements to find the smallest cost c' for which $c(T)/(|T| - |E_{\leq w_i}| + n) \leq (c(T) + c')/(|T| + 1 - |E_{\leq w_i}| + n)$, where $T \subseteq E_{\leq w_i}$ is formed by all elements in $E_{\leq w_i}$ of cost smaller than c' and $|T| > |E_{\leq w_i}| - n$. If $c' = c(T)/(|T| - |E_{\leq w_i}| + n)$, then add to T all those elements from $E_{\leq w_i}$ of cost c'. By Lemma 6.2, T is the desired set. Using the linear-time algorithm of Blum et al. [10], the above binary search can be performed in $O|E_{\leq w_i}|)$ time.

To show that algorithm *up_to_weight* is correct we have to consider two different cases. Let w_n be the weight of the nth smallest element in E. If $w_i < w_n$, then only the elements of smallest cost in $E_{<w_n}$ have their weights increased by $\delta(w_i)$. Note that the set S computed by *up_to_weight* contains exactly those elements. For the case when $w_i \geq w_n$, suppose that the weights of the elements have been increased by $\delta(w_i)$. Let R be the largest set of smallest *rate* according to the increased weights, and let r_R be its *rate*. Observe that all elements in $E_{\leq w_i} - R$ have cost no smaller than r_R. Hence, if we assume the same initial weight for all elements in $E_{\leq w_i}$, set R would be the largest set of smallest *rate* in it. □

Algorithm *up_to_rate*, described in Figure 21, receives as input a *rate* r, the set of elements E, and the weight and cost functions w and c. The algorithm performs a binary search on the weights, invoking *up_to_weight* on each probe weight $\bar{w}$. If $\bar{w}$ is too large, *up_to_rate* discards all elements of weight $\bar{w}$ or larger since their weights are not increased by $\bar{\delta}(r)$. If $\bar{w}$ is too small, *up_to_rate* tries a larger probe value, but first it reduces the size of E: since all elements of cost at least r and weight at most $\bar{w}$ do not have their weights increased by $\bar{\delta}(r)$, they can be ignored. Note that all elements of cost smaller than r and weight at most $\bar{w}$ will have their weights increased by $\bar{\delta}(r)$. Instead of keeping track of all the individual weight increases of these elements, *up_to_rate* stores them in a set T. When the algorithm determines the maximum weight increase that those elements should have, it performs the increases in a single step.

Algorithm *up_to_rate* maintains the invariant that the value of w^* is an upper bound on the maximum weight w_r that $\bar{\delta}(r)$ can assign to an element of E without selecting a set of *rate* at least r. In each iteration of the repeat-loop, either the value of w^* is decreased to $\bar{w}$, or it is discovered that $\bar{w}$ is a lower bound for w_r. The gap between upper and lower bounds for w_r decreases in each iteration of the repeat-loop. When the loop ends, the value of w^* is equal to w_r.

Algorithm *up_to_rate* (r, E, w, c)
$T \leftarrow \emptyset; \quad w^* \leftarrow \infty$
repeat
Find the weight $\bar{w}$ of the $\lceil \frac{1}{2}|E| \rceil$th smallest element in E.
$(r_1, c_1) \leftarrow$ *up_to_weight*$(\bar{w}, E, w, c)$
if $r_1 \geq r$ **then**
$E \leftarrow E_{<\bar{w}}$
$w^* \leftarrow \bar{w}$
else
$T \leftarrow T \cup \{e \mid w(e) \leq \bar{w} \text{ and } c(e) < r\}$
$E \leftarrow E_{>\bar{w}}$
end if
until $|E| = 0$
$c_T \leftarrow$ cost of increasing the weights of the elements in T to w^*.
$w_n \leftarrow$ weight of the nth smallest element in E according to the modified weights
Output c_T and w_n.

Figure 21: Algorithm *up_to_rate*.

Lemma 6.6 *Algorithm up_to_rate finds in linear time the cost of the weight increases determined by $\bar{\delta}(r)$ and the weight of the nth smallest element according to the increased weights.*

Proof. By the above discussion, *up_to_rate* correctly computes the weight increases determined by $\bar{\delta}(r)$. Each iteration of the repeat-loop takes linear time, and in each iteration the size of E is reduced by at least one half. Therefore, the total time needed by the algorithm is $O(|E|)$. □

We describe in Figure 22 a recursive algorithm for evaluating the robustness function F_U of a uniform matroid $U = (E, \mathcal{I}, w, c)$ at a given budget value $b \geq 0$. We let w_B be the weight of a minimum weight base of U according to the initial weights. In each recursive call the algorithm reduces in linear time the number of elements in E by a fraction of at least one third, hence the overall time complexity of the algorithm is linear in the number of elements. The essential component of each iteration is a pair of tests that allow the algorithm either to find at least one third of the elements in E that have weights or costs that are too large (and, thus, that can be discarded), or to identify at least one third of the elements in E that will end up having the same final weight (and, thus, that can be contracted to a single meta-element).

Function *contract_uniform*, described in Figure 23, identifies a set $S \cup T \subseteq E$ of size at least $\lceil |E|/3 \rceil$ formed by elements that will experience exactly the same weight increases during the computation of $F_U(b)$. Instead of keeping track of the individual weight changes of these elements, they are contracted to a single meta-element $\hat{e}$ and the algorithm computes only the weight increases for $\hat{e}$. After *robustness_uniform* has computed the final weight increases, it is easy to expand the meta-elements to determine the final weight for each element in E.

Observe that every call that algorithm *robustness_uniform* makes to *contract_uniform* is preceded by a call to *up_to_rate*, which determines the cost $\tilde{c}$ of optimally increasing the weights

Algorithm *robustness_uniform* (E, w, c, b)
  if $|E| = 1$ **then**
    Let $E = \{e\}$. Set $w(e) \leftarrow w(e) + b/c(e)$.
    Let w_B^* be the weight of a minimum weight base of U according to the increased weights.
    Output $w_B^* - w_B$.
  else
    Compute w_t, the weight of the $\lceil \frac{2}{3}|E| \rceil$th smallest element in E.
    $(c_t, r_t) \leftarrow$ *up_to_weight*(w_t, E, w, c)
    if $c_t \geq b$ **then**
      $E \leftarrow E - \{e \mid w(e) \geq w_t\}$
      Output (*robustness_uniform* $(E, w, c, , b)$)
    else
      Compute $\bar{c}$, the upper median cost among the elements $e \in E$ of weight $w(e) \leq w_t$.
      $(\tilde{c}, \tilde{w}_n) \leftarrow$ *up_to_rate* $(\bar{c}, E, w, c)$
      if $\tilde{c} \geq b$ **then**
        $E \leftarrow E - \{e \mid w(e) \leq \tilde{w}_n \text{ and } c(e) \geq \bar{c}\}$
        return (*robustness_uniform* (E, w, c, b))
      else
        $(E, w, b) \leftarrow$ *contract_uniform* $(E, w, c, b, \tilde{c}, \tilde{w}_n, \bar{c})$
        if $b = 0$ **then**
          Let w_B^* be the weight of a minimum weight base of U according to the increased weights.
          Output $w_B^* - w_B$.
        else Output (*robustness_uniform* (E, w, c, b)) **end if**
      end if
    end if
  end if

Figure 22: Algorithm *robustness_uniform*.

Algorithm *contract_uniform* $(E, w, c, b, \tilde{c}, \tilde{w}_n, \bar{c})$
    Let $S = \{e \mid w(e) \leq \tilde{w}_n$ and $c(e) < \bar{c}\}$, and $T = \{e \mid w(e) \leq \tilde{w}_n$ and $c(e) = \bar{c}\}$.
    for each $e \in S$ **do** $w(e) \leftarrow \tilde{w}_n$ **end for**
    $b \leftarrow b - \tilde{c}$
    for each $e \in T$ **do**
        $(w(e),\ b) \leftarrow (\min\{\tilde{w}_n, w(e) + b/c(e)\},\ \max\{0, b - (\tilde{w}_n - w(e)) * c(e)\})$
        if $b = 0$ **then** exit the for-loop **end if**
    end for
    if $b > 0$ **then**
        $E \leftarrow (E - S - T) \cup \{\hat{e}\}$, where $\hat{e}$ is meta-element with $w(\hat{e}) = \tilde{w}_n$ and $c(\hat{e}) = c(S \cup T)$.
    end if
    Output (E, w, b).

Figure 23: Algorithm *contract_uniform*.

of the elements in E to the point at which the following weight increase would be made over a set of *rate* at least $\bar{c}$. Algorithm *contract_uniform* continues these optimal weight increases by lifting to $\tilde{w}_n$ the weights of the elements in sets S and T. If the budget is exhausted while performing these increases, then the algorithm stops since it has computed the weight function needed to determine $F_U(b)$. Otherwise, the budget is decreased to account for the new weight increases.

If any budget remains after increasing the weights of the elements in $S \cup T$, then all elements of weight at most $\tilde{w}_n$ and cost at most $\bar{c}$ are contracted. Note that these elements belong to the first set formed by elements of weight $\tilde{w}_n$ that a canonical sequence of increases would select, and thus, by Property 6.1, they can be replaced by a meta-element.

To illustrate how algorithm *robustness_uniform* works, consider the following example. Let $U = (E, \mathcal{I}, w, c)$ be a uniform matroid of rank 4 with set $E = \{a, b, c, d, e, f, g\}$. The initial weights of the elements are $1, 1, 2, 3, 4, 5$, and 6, respectively, and their costs are $1, 3, 5, 1, 3, 1$, and 3. The value of w_B is $1 + 1 + 2 + 3 = 7$. We wish to evaluate $F_U(9)$. The algorithm first computes $w_t = 4$, and invokes routine *up_to_weight*. This routine returns $(c_t, r_t) = (4, 2)$. The value of c_t is the cost of increasing the weights of the elements to $4, 1, 2, 4, 4, 5$, and 6, respectively; with these weights the next set of smallest *rate* is $\{a, d\}$ and it has *rate* $r_t = 2$. Since $c_t < 9$, *robustness_uniform* computes $\bar{c} = 3$, and invokes routine *up_to_rate*. This routine returns $(\tilde{c}, \tilde{w}_n) = (6, 5)$. The value of $\tilde{c}$ is the cost of increasing the weights of the elements to $5, 1, 2, 5, 4, 5$, and 6. Since $\tilde{c} < 9$, then *contract_uniform* is invoked, and it exhausts the budget by increasing the weights of the elements to $5, 2, 2, 5, 4, 5$, and 6. Since upon return from *contract_uniform* b is zero, *robustness_uniform* outputs $F_U(9) = (2 + 2 + 4 + 5) - 7 = 6$.

Lemma 6.7 *Algorithm robustness_uniform computes $F_U(b)$, for any given $b \geq 0$, in $O(|E|)$ time.*

Proof. Each call to algorithm *robustness_uniform* takes linear time and, as we show below, reduces the size of E by at least a fraction of one third. Therefore, the overall time complexity is linear on the number of elements.

In each recursive call, *robustness_uniform* invokes routine *up_to_weight*, which computes the weight increases of the canonical sequence $\delta(w_t)$. If the cost of these weight increases exceeds b, then all elements with weight at least w_t are discarded from E. There are at least $\lceil \frac{1}{3}|E| \rceil$ of these elements, and so, in this case the size of E is reduced to at most $\lfloor \frac{2}{3}|E| \rfloor$.

If the cost of the weight increases is smaller than b, then *robustness_uniform* invokes *up_to_rate* to find the weight increases defined by $\bar{\delta}(\bar{c})$. If the cost of these new increases surpasses the budget, then all elements of weight at most $\tilde{w}_n$ and cost at least $\bar{c}$ are discarded. Since $\tilde{w}_n \geq w_t$, this step removes at least $\lceil \frac{1}{3}|E| \rceil$ elements from E. However, if the cost of the last weight increases is smaller than b, then *robustness_uniform* invokes routine *contract_uniform*. This routine either makes optimal weight increases that exhaust the budget, or it replaces all elements of weight at most $\tilde{w}_n$ and cost at most $\bar{c}$ by a meta-element. These elements represent at least one third of E. Note that if $|E| < 4$ the size of E is still reduced by a fraction of at least one third, even when a meta-element is added to E. The reason for this is that either $\lceil \frac{2}{3}|E| \rceil$ rounds up (in the computation of w_t), or $\lceil \frac{1}{2} \lceil \frac{2}{3}|E| \rceil \rceil$ rounds up (in the computation of $\bar{c}$). □

6.2.2 Partition Matroid

We turn our attention now to the problem of evaluating in linear time $F_P(b)$ for a partition matroid P with ℓ blocks, and $\ell > 1$. This problem is more difficult than the problem for uniform matroids since we have to determine simultaneously how the weights of the elements change in all the blocks E_i. As for the case of a uniform matroid, we compute $F_P(b)$ by a recursive prune-and-search process that combines searches on weights with searches on costs. However, our new algorithm reduces the size E by a fraction of only one tenth in each recursive call. This decrease in performance, compared to *robustness_uniform*, is due to the additional difficulty that multiple blocks E_i impose on finding good probe values.

The algorithm, described in Figure 24, uses an array *upper* of size ℓ, and it stores in *upper*(i) an upper bound on the maximum weight that can be assigned to any element in E_i. Each entry of *upper* is initialized to ∞. We let w_B be the weight of a minimum weight base of P according to the initial weights.

Note the correspondence between the structure of *robustness_uniform* and the structure of *robustness_partition*. The part of *robustness_partition* preceding the test "$\sum_{i=1}^{\ell} c'_i \geq b$" is more complex than the corresponding part of algorithm *robustness_uniform*. The reason is that *robustness_partition* has to consider all blocks E_i, and the weights of the elements in all the blocks are not increased at the same rate. This makes the computation of good probe values more difficult than for the case of a uniform matroid. Observe also that the probe values w_i and $\bar{c}$ are different from the corresponding probe values chosen by *robustness_uniform*. These values were selected to ensure that each call to *robustness_partition* decreases the size of E by a fixed fraction.

Algorithm *robustness_partition* $(E, w, c, b, upper)$
  if $|E| = 1$ **then**
    Let $E = \{e\}$. Set $w(e) \leftarrow w(e) + b/c(e)$.
    Let w_B^* be the weight of a minimum weight base of P according to the increased weights.
    Output $w_B^* - w_B$.
  else
    for $i = 1, 2, \ldots, \ell$ **do**
      Compute w_i, the weight of the $\lceil \frac{4}{5}|E_i| \rceil$th smallest element in E_i.
      $(c_i, r_i) \leftarrow$ *up_to_weight* (w_i, E_i, w, c)
    end for
    Compute r', the weighted median of the rates r_i using, for each i, $|E_i|$ as the weight for r_i.
    for $i = 1, 2, \ldots, \ell$ **do** $(c_i', w_i') \leftarrow$ *up_to_rate* (r', E_i, w, c) **end for**
    if $\sum_{i=1}^{\ell} c_i' \geq b$ **then**
      for $i = 1, 2, \ldots, \ell$ **do**
        $E_i \leftarrow E_i - \{e \mid w(e) \geq w_i'\}$
        $upper(i) \leftarrow w_i'$
      end for
      Output (*robustness_partition* $(E, w, c, b, upper)$)
    else
      Let $S = \cup_{\{i \mid r_i \leq r'\}} \{e \mid e \in E_i \text{ and } w(e) < w_i'\}$
      Compute $\bar{c}$, the cost of the $\lceil \frac{3}{4}|S| \rceil$th smallest cost element in S.
      for $i = 1, 2, \ldots, \ell$ **do** $(\tilde{c}_i, \tilde{w}_{n_i}) \leftarrow$ *up_to_rate* $(\bar{c}, E_i, w, c)$ **end for**
      if $\sum_{i=1}^{\ell} \tilde{c}_i \geq b$ **then**
        for $i = 1, 2, \ldots, \ell$ **do** $E_i \leftarrow E_i - \{e \mid w(e) \leq \tilde{w}_{n_i} \text{ and } c(e) \geq \bar{c}\}$ **end for**
        Output (*robustness_partition* $(E, w, c, b, upper)$)
      else
        $(E, w, b) \leftarrow$ *contract_partition* $(E, w, c, b, \sum_{i=1}^{\ell} \tilde{c}_i, \{\tilde{w}_{n_1}, \ldots, \tilde{w}_{n_\ell}\}, \bar{c}, upper)$
        if $b = 0$ **then**
          Let w_B^* be the weight of a minimum weight base of P according to the increased weights.
          Output $w_B^* - w_B$.
        else Output (*robustness_uniform* $(E, w, c, b, upper)$) **end if**
      end if
    end if
  end if

Figure 24: Algorithm *robustness_partition*.

Algorithm *contract_partition* $(E, w, c, b, \tilde{c}, \{\tilde{w}_{n_1}, \ldots, \tilde{w}_{n_\ell}\}, \bar{c}, upper)$
  Let $S_i = \{e \mid e \in E_i, w(e) \leq \tilde{w}_{n_i}$ and $c(e) < \bar{c}\}$, and
    $T_i = \{e \mid e \in E_i, w(e) < \tilde{w}_{n_i}$ and $c(e) = \bar{c}\}$, for all $i = 1, 2, \ldots, \ell$.
  for $i = 1, 2, \ldots, \ell$ **do**
    Increase to $\tilde{w}_{n_i}$ the weight of every element in S_i.
  end for
  $b \leftarrow b - \tilde{c}$
  for $i = 1, 2, \ldots, \ell$ **do**
    if $|E_i| > 1$ **then**
      for every $e \in T_i$ **do**
        $(w(e),\ b) \leftarrow (\min\{\tilde{w}_{n_i}, w(e) + b/c(e)\},\ \max\{0, b - (\tilde{w}_{n_i} - w(e)) * c(e)\})$
        if $b = 0$ **then** exit the inner for-loop **end if**
      end for
      if $b = 0$ **then** exit the for-loop
      else $E_i \leftarrow (E_i - S_i - T_i) \cup \{\hat{e}\}$, where meta-element $\hat{e}$ has $w(\hat{e}) = \tilde{w}_{n_i}$,
                and $c(\hat{e}) = c(S_i \cup T_i)$.
      end if
    else
      Let $E_i = \{e\}$.
      if $c(e) \leq \bar{c}$ **then**
        $(w(e),\ b) \leftarrow (\min\{upper(i), w(e) + b/c(e)\},$
             $\max\{0, b - (upper(i) - w(e)) * c(e)\})$
        if $b = 0$ **then** exit the for-loop
        else $E \leftarrow E - E_i$ **end if**
      end if
    end if
  end for
  Output E,w, and b.

Figure 25: algorithm *contract_partition*.

Algorithm *contract_partition*, shown in Figure 25, is similar to *contract_uniform*, but it has to deal with one situation that does not appear for the case of uniform matroids. If any set E_i has only one element, then it cannot be further contracted. In this case, *contract_partition* does the following. If the unique element $e \in E_i$ has cost at most $\bar{c}$, then *contract_partition* increases its weight to *upper*(i) if the budget is large enough and then it discards block E_i. This can be done, since the weight of e cannot be increased above *upper*(i), and when it reaches such weight *robustness_partition* does not have to consider it any more. But, if the remaining budget is too small to perform the weight increase, then the weight of e is increased only as much as the budget allows. Since the budget is exhausted, no more weight increases are possible.

Theorem 6.2 *Given a partition matroid $P = (E, \mathcal{I}, w, c)$ and a positive budget b, the value of $F_P(b)$ can be computed in $O(|E|)$ time.*

Proof. To show that the algorithm runs in $O(|E|)$ time, it suffices to show that each iteration of the while-loop reduces the size of E by at least $\frac{1}{10}|E|$. The value r' computed by *robustness_partition* is such that the weight increases defined by $\bar{\delta}(r')$ do not affect the weights of at least $\frac{1}{2} * (1 - \frac{4}{5})|E| = \frac{1}{10}|E|$ elements of E. Hence, if the cost of these weight increases exceeds b, *robustness_partition* discards those elements and reduces the size of E by at least $\lceil \frac{1}{10}|E| \rceil$. If the cost of the weight increases is smaller than b, *robustness_partition* makes a test on $\bar{c}$, the $\lceil \frac{3}{4}|S| \rceil$ smallest element cost in S. It is not difficult to see that $|S| \geq \frac{2}{5}|E|$. If the cost of the new weight increases is at least b, then *robustness_partition* discards at least $\lceil \frac{1}{4}|S| \rceil \geq \lceil \frac{1}{10}|E| \rceil$ elements form E.

If $\sum_{i=1}^{\ell} \tilde{c}_i < b$, then *contract_partition* contracts in each E_i all elements from S of cost at most $\bar{c}$. Let $S_i = E_i \cap S$. There is one situation in which the algorithm cannot contract the elements in S_i of cost at most $\bar{c}$. Suppose that set S_i has only two elements, one of cost at most $\bar{c}$ and the other of cost larger than $\bar{c}$, then *contract_partition* cannot reduce the size of E_i (since it would try to contract the element of cost at most $\bar{c}$ to a meta-element). Since $\bar{c}$ is the $\lceil \frac{3}{4}|S| \rceil$th smallest element cost, then there are at most $\lfloor \frac{1}{4}|S| \rfloor$ elements of cost larger than $\bar{c}$. Hence, there are at most $\lfloor \frac{1}{4}|S| \rfloor$ sets S_i of two elements for which the algorithm cannot contract their sizes as described above. These sets include at most $\lfloor \frac{1}{2}|S| \rfloor$ elements. The other $\lceil \frac{1}{2}|S| \rceil$ elements must belong to sets S_j that *contract_partition* contracts to at most half of their sizes. Therefore, *contract_partition* contracts the size of E by at least $\lceil \frac{1}{4}|S| \rceil \geq \lceil |E|/10 \rceil$ elements. □

7 Conclusions

7.1 Summary of results

We have developed the concept of a robustness function for combinatorial optimization problems, that generalizes the notion of sensitivity analysis. This function can be used to assess the quality of solutions when there are expected changes in the values of the parameters of the problem, or when such values are not known and estimates have to be used.

We have studied the concept of a robustness function using two different models. The discrete model that allows removals of elements from the input of the problem, and the continuous model that permits only finite changes in the weights of these elements. We have shown that the discrete robustness problem is NP-hard even for seemingly simple optimization problems, while the continuous version is polynomially solvable for a large class of combinatorial optimization problems.

We have presented a 2-approximation algorithm for the maximum components problem, that is used as a subroutine in the design of a $O(\log k)$-approximation algorithm for the discrete robustness problem for minimum spanning trees. Given a maximization optimization problem P in which the weights of the elements are 0 or 1, let A be an α-approximation algorithm for it. The techniques presented here can be used to design an $O(\alpha \log n)$-approximation algorithm for problem P when the elements have arbitrary non-negative weights. This algorithm uses A as a key subroutine.

Interestingly, we have been able to design an algorithm for solving the continuous robustness problem for arbitrary matroids. This algorithm has the nice property that all partial solutions that it builds are optimal for some budget value. The complexity of the algorithm inherently depends on that of an oracle to test independence for the matroid. If such an oracle runs in strongly polynomial time, then our algorithm can compute all the breakpoints of its robustness function in strongly polynomial time.

We have studied some interesting classes of matroids. We have presented different techniques, that exploit the special structure of each one of these classes of matroids to design faster algorithms for computing their robustness functions. Specifically, we have designed an $O(n^3m^2\log(n^2/m))$ time algorithm for graphic matroids, an $O(mn(m+n^2)|E|\log(m^2/|E|+2))$ time algorithm for transversal matroids, an $O(m^2n^2)$ time algorithm for scheduling matroids, and an $O(m\log m)$ time algorithm for partition matroids.

We have also studied the problem of evaluating the robustness function of a matroid at a single point, as opposed to generating all the breakpoints. For the case of partition matroids we have designed an optimal algorithm that solves the problem in $O(m)$ time.

7.2 Directions for Future Research

Some problems that remain open are the following. For the case of the discrete robustness problem for minimum spanning trees it would be interesting to find out if it is possible to design a constant factor approximation algorithm for it. A related problem is to find an approximation algorithm for computing the discrete robustness function for the shortest distance between two distinguished vertices. It is not difficult to find a k-approximation algorithm for the latter problem, but one would suspect the existence of an algorithm with a better performance ratio.

The problem of designing a general approximation algorithm for arbitrary matroids resisted our attempts for finding it. A starting point would be to design an approximation algorithm for the problem of finding the smallest number of elements that need to be removed from the ground set of a matroid to reduce its rank. This problem is an interesting generalization of the minimum cut problem.

We have proved that the discrete robustness problem for graphic matroids when the elements have unit destruction cost is NP-hard, but it is polynomially solvable for partition matroids. We also have some preliminary results showing that the problem on some classes of scheduling matroids can be solved efficiently. It would be interesting to find a characterization of the "simplest" matroid for which the problem is NP-hard. This characterization would give us insight on the inherent complexity of the problem.

Our algorithm for computing the continuous robustness function for minimum spanning trees uses as a subroutine an algorithm for computing the strength of a graph. The currently fastest algorithm for solving this latter problem needs to solve n maximum flow computations. It would be interesting to see if a faster algorithm can be designed for the problem.

Our bound for the number of breakpoints of a graphic matroid does not seem to be tight. So far we have not found an example in which the number of breakpoints exceeds m. A tight bound for this number would be reflected in the corresponding reduction of the time complexity of the algorithm for finding the robustness function. The same can be said about the number of breakpoints in the robustness function of arbitrary matroids.

Another way of improving the time complexity of the algorithm for the continuous robustness function for minimum spanning trees would be to design algorithms for efficiently updating maximum flows when some edges of the graph are deleted or new edges are added. This is an interesting problem in its own right.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[2] R.K. Ahuja, J.B. Orlin, C. Stein, and R.E. Tarjan, *Improved algorithms for bipartite network flow*, SIAM Journal on Computing, 23 (1994), pp. 906–933.

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

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

[5] F. Bauer and A. Varma, *Distributed algorithms for multicast path setup in data networks*, IEEE/ACM Transactions on Networking, 1 (1996), pp. 181–191.

[6] O. Berman, *Improving the location of minisum facilities through network modification*, Annals of Operations Research, 40 (1992), pp. 1–16.

[7] K. Bharat-Kumar and J.M. Jaffe, *Routing to multiple destinations in computer networks*, IEEE Transactions on Communications, COM31 (1983), pp. 343–351.

[8] R.E. Bixby, *Matroids and operations research*, pp. 333–458, in *Advanced techniques in the practice of operations research*, edited by H.J. Greenberg, F.H. Murphy, and S.H. Shaw, North-Holland, 1985.

[9] R.E. Bixby and W.H. Cunningham, *Matroid optimization and algorithms*, pp. 551–609, in Handbook of Combinatorics, Vol. 1, edited by R.L. Graham, M. Grötschel, and L. Lovász, The MIT Press, 1995.

[10] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, *Time bounds for selection*, Journal of Computer and System Sciences, 7 (1973), pp. 448–461.

[11] O. Boruvka, *O jistem problemu minimalnim*, Praca Moravske Prirodovedecke Spolcnosti, 3 (1926), pp. 37–58.

[12] O.P. Burdakov, *On using the minimum spanning tree algorithm for optimal secant approximation of derivatives*, Zeitschrift für Angewandte Mathematik und Mechanik, 76 (1996), pp. 389.

[13] L. Cai, *NP-completeness of minimum spanner problems*, Discrete Applied Mathematics, 48 (1994), pp. 187.

[14] L. Cai and D.G. Corneil, *Tree spanners*, SIAM Journal on Discrete Mathematics, 8 (1995), pp. 359–387.

[15] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, Proceedings of the 8th Symposium on Computational Geometry, 1992, pp. 192–201.

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

[17] N. Christofides, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, 1976.

[18] E.G. Coffman Jr., P. Chrétienne, J.K. Lenstra, and Z. Liu, editors, *Scheduling theory and its applications*, John Wiley & Sons, 1995.

[19] E. Cohen, *Fast algorithms for constructing t-spanners and paths with stretch t*, Proceedings of the 34th Annual Symposium on Foundations of Computer Science, 1993, pp. 648–658.

[20] J. Cong, A. Khang, G. Robins, M. Sarrafzadeh, and C.K. Wong, *Provably good performance-driven global routing*, IEEE Transactions on Computer-Aided Design, 11 (1992), pp. 739–752.

[21] W. Cook, L. Lovász, and P. Seymour, editors, *Combinatorial optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Vol. 20, American Mathematical Society, 1995.

[22] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to algorithms*, The MIT Press, 1992.

[23] W.H. Cunningham, *Testing membership in matroid polyhedra*, Journal of Combinatorial Theory, Series B, 36 (1984), pp. 161–188.

[24] W.H. Cunningham, *Minimum cuts, modular functions and matroid polyhedra*, Networks, 15 (1985), pp. 205–215.

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

[26] 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), pp. 864–894.

[27] E.W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[28] K.U. Drangmeister, S.O. Krumke, M.V. Marathe, H. Noltemeier, and S.S. Ravi, *Modifying edges of a network to obtain short subgraphs*, manuscript, 1996.

[29] J. Edmonds, *Submodular functions, matroids and certain polyhedra*, pp. 69–87, in *Combinatorial Structures*, edited by R.K. Guy, Gordon and Beach, New York, 1970.

[30] M.J. Eisner and D.G. Severance, *Mathematical techniques for efficient record segmentation in large shared databases*, Journal of the ACM, 23 (1976), pp. 619–635.

[31] A. Federgruen and P. Zipkin, *Solution techniques for some allocation problems*, Mathematical Programming, 25 (1983), pp 13–24.

[32] R.W. Floyd, *Algorithm 97 (shortest path)*, Communications of the ACM, 5 (1962), pp. 345.

[33] L.R. Ford Jr. and D.R. Fulkerson, *Flows in networks*, Princeton University Press, 1962.

[34] G.N. Frederickson, *Scheduling unit-time tasks with integer release times and deadlines*, Information Processing Letters 16 (1983), pp. 171–173.

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

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

[37] G.N. Frederickson and R. Solis-Oba, *Efficient algorithms for robustness in matroid optimization*, Proceedings of the Eight Annual ACM-SIAM Symposium on Discrete Algorithms, 1997, pp. 659–668.

[38] G.N. Frederickson and R. Solis-Oba, *Efficient algorithms for robustness in resource allocation and scheduling problems*, manuscript, 1997.

[39] G.N. Frederickson and R. Solis-Oba, *Algorithms for robustness in matroid optimization*, manuscript, 1997.

[40] M. Fredman and D.E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, Journal of Computer and System Sciences, 48 (1994), pp. 533.

[41] S. French, *Sequencing and scheduling*, Ellis Horwood Ltd., 1982.

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

[43] H.N. Gabow and R.E. Tarjan, *Efficient algorithms for a family of matroid intersection problems*, Journal of Algorithms, 5 (1984), pp. 80–131.

[44] H.N. Gabow and R.E. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, Journal of Computer and System Sciences, 30 (1985), pp. 209–221.

[45] H.N. Gabow, Z. Galil, T.H. Spencer, and R.E. Tarjan, *Efficient algorithms for finding minimum spanning tree in undirected and directed graphs*, Combinatorica, 6 (1986), pp. 109–122.

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

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

[48] G. Gallo, M.D. Grigoriadis and R.E. Tarjan, *A fast parametric maximum flow algorithm and applications*, SIAM Journal on Computing, 18 (1989), pp. 30–55.

[49] M.R. Garey and D.S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman, 1979.

[50] M.R. Garey, D.S. Johnson, B.B. Simons, and R.E. Tarjan, *Scheduling unit-time tasks with arbitrary release times and deadlines*, SIAM Journal on Computing, 10 (1981), pp. 256–269.

[51] M.X. Goemans and R. Ravi, *The constrained minimum spanning tree problem*, Lecture Notes in Computer Science number 1097, 1996, pp. 66.

[52] B. Golden, *A problem in network interdiction*, Naval Research Logistics Quarterly.

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

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

[55] R.L. Graham, M. Grötschel, and L. Lovász, *Handbook of Combinatorics*, The MIT Press, 1995.

[56] D.M. Gusfield, *Sensitivity analysis for combinatorial optimization*, Ph.D. thesis, University of California, Berkeley, 1980.

[57] D. Gusfield, *Computing the strength of a graph*, SIAM Journal on Computing, 20 (1991), pp. 639–654.

[58] D. Gusfield, L. Wang, and P. Stelling, *Graph traversals, genes and matroids: an efficient special case of the traveling salesman problem*, Technical Report No. CSE-96-3, University of California, Davis, 1996.

[59] P. Hage, F. Harary, and B. James, *The minimum spanning tree problem in archaeology*, American Antiquity, 61 (1996), pp. 149.

[60] F. Harary, *Graph theory*, Addison Wesley, 1969.

[61] F.S. Hillier, *Introduction to operations research*, McGraw Hill, New York, 1995.

[62] J.E. Hopcroft and R.M. Karp, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, SIAM Journal on Computing, 2 (1973), pp. 225–231.

[63] 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), pp. 277–281.

[64] 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), pp. 1143–1155.

[65] F.K. Hwang, *On Steiner minimal trees with rectilinear distance*, SIAM Journal of Applied Mathematics, 30 (1976), pp. 104–114.

[66] F.K. Hwang and D.S. Richards, *Steiner tree problems*, Networks, 22(1992), pp. 55–89.

[67] F.K. Hwang, D.S. Richards, P. Winter, and P. Widmayer, *The Steiner tree problem*, Annals of Discrete Mathematics, 53 (1995), pp. 382.

[68] T. Ibaraki and N. Katoh *Resource allocation problems: algorithmic approaches*, Cambridge Massachusets, MIT Press, 1988.

[69] M. Iri, *Applications of matroid theory*, Mathematical Programming: The State of the Art, Springer, Berlin, 1983, pp. 158–201.

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

[71] J.R. Jackson, *Scheduling a production line to minimize maximum tardiness*, Research Report 43, (1955) Management Science Research Project, University of California, Los Angeles.

[72] D.R. Karger, P.N. Klein, and R.E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees*, Journal of the ACM, 42 (1995), pp. 321.

[73] D. Karger, C. Stein, and J. Wein, *Scheduling algorithms*, to appear in CRC Handbook of Theoretical Computer Science.

[74] R. Karp, *Reducibility among combinatorial problems*, in R.E. Miller and J.W. Tatcher, editors, Complexity of Computer Computations, pp. 85–103. Plenum Press, 1972.

[75] W. Karush, *A general algorithm for the optimal distribution of effort*, Management Science, 9 (1962), pp. 50–72.

[76] K. Kayser, K. Sandau, G. Bohm, and K.D. Kunze, *Analysis of soft tissue tumors by an attributed minimum spanning tree*, Analytical and Quantitative Cytology and Histology, 13 (1991), pp. 329–334.

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

[78] S. Khuller, A. Bar-Noy, B. Schieber, *The complexity of finding most vital arcs and nodes*, Technical Report CS-TR-3539 University of Maryland, 1995.

[79] S. Khuller, B. Raghavachari, and N. Young, *Balancing minimum spanning trees and shortest-path trees*, Algorithmica, 14 (1995), pp. 305.

[80] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, *Multicasting for multimedia applications*, Proceedings of IEEE INFOCOM 1992.

[81] J.B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society, 7 (1956), pp. 48–50.

[82] D. Krznaric and C. Levcopoulos, *Computing hierarchies of clusters from the Euclidean minimum spanning tree in linear time*, Lecture Notes in Computer Science, Number 1026, 1995, pp. 443.

[83] E.L. Lawler, *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, 1976.

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

[85] J. Lee and J. Ryan, *Matroid applications and algorithms*, ORSA Journal of Computing, 4 (1992), pp. 70–98.

[86] A. Lehman, *A solution to the Shannon switching game*, SIAM Journal on Applied Mathematics, 12 (1964), pp. 687–725.

[87] V.G. Leon, S.D. Wu, and R.H. Storer, *Robustness measures and robust scheduling for job shops*, IIE Transactions, 26 (1994), pp. 32–43.

[88] M. Libura, *Sensitivity analysis for minimum weight base of a matroid*, Control and Cybernetics, 20 (1991), pp. 7–24.

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

[90] L. Lovàsz, *Matching theory*, North Holland, 1986.

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

[92] M. Mansour, S. Balemi, and W. Troul (editors), *Robustness of dynamic systems with parameter uncertainties*, Birkhauser Verlag, 1992.

[93] M.V. Marathe, R. Ravi, R. Sundaram, and S.S. Ravi, *Bicriteria network design problems*, Lecture Notes in Computer Science Number 944, 1995, pp. 487.

[94] P. Michael, *Scheduling: theory, algorithms, and systems*, Englewood Cliffs, New Jersey 1995.

[95] B.M.E. Moret and H.D. Shapiro, *Algorithms from P to NP*, The Benjamin/Cummings Publishing Company, 1991.

[96] H. Narayanan, *A rounding technique for the polymatroid membership problem*, Linear Algebra and its Applications, 221 (1995), pp. 41–57.

[97] H. Nagamochi, K. Nishimura, and T. Ibaraki, *Computing all small cuts in an undirected network*, in Proceedings of the 5th International Symposium on Algorithms and Computation, Number 834 in Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 190–198.

[98] J.G. Oxley, *Matroid theory*, Oxford University Press, 1992.

[99] C.H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, New Jersey 1982.

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

[101] R.C. Prim, *Shortest connection networks and some generalizations*, Bell Systems Technical Journal, 36 (1957), pp. 1389–1401.

[102] R. Rado, *Note on independence functions*, Proceedings of the London Mathematical Society, 7 (1957), pp. 300–320.

[103] A. Recski, *Matroid theory and its applications*, Springer Verlag, Berlin, 1989.

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

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

[106] D. Shier, *Arc tolerances in shortest path and network flow problems*, Networks, 10 (1980), pp. 277-291.

[107] H.S. Stone, *Critical load factors in two-processor distributed systems*, IEEE transactions on Software Engineering, 4 (1978), pp. 254–258.

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

[109] R.E. Tarjan, *Finding optimal branchings*, Networks, 1 (1971), pp. 265–272.

[110] R.E. Tarjan, *Efficiency of a good but not linear set union algorithm*, Journal of the ACM, 22 (1975), pp. 215–225.

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

[112] R.E. Tarjan, *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, 1983.

[113] W.T. Tutte, *Matroids and graphs*, Transactions of the American Mathematical Society, 90 (1959), pp. 527–552.

[114] G.M. Weber, *Sensitivity analysis of optimal matchings*, Networks, 11 (1981), pp. 41–56.

[115] D.J.A. Welsh, *Matroid Theory*, Academic Press, London 1976.

[116] H. Whitney, *On the abstract properties of linear dependence*, American Journal of Mathematics, 57 (1935), pp. 509–533.

[117] A. Zelikovsky, *A series of approximation algorithms for the acyclic directed Steiner tree problem*, Algorithmica, 18 (1997), pp. 99.