Efficient Algorithms for Robustness in Matroid Optimization

Greg N. Fredrickson *   Roberto Solis-Oba †

Abstract

The *robustness function* of a matroid measures the maximum increase in the weight of its minimum weight bases that can be produced by increases of a given total cost on the weights of its elements. We present an algorithm for computing this function, that runs in strongly polynomial time for matroids in which independence can be tested in strongly polynomial time. We identify key properties of transversal, scheduling and partition matroids, and exploit them to design robustness algorithms that are more efficient than our general algorithm.

1 Introduction

A fundamental problem in the study of dynamic systems is that of measuring how sensitive a problem is to perturbations in its input [19, 26]. A significant limitation of the current methods for sensitivity analysis in combinatorial optimization is that they measure only changes in the solution of a problem produced by perturbations in the value of a single element in its input [18, 25, 30, 31]. In some situations it is necessary to determine the maximum effect that bounded changes in the whole input of a problem can have over the value of its solution and, thus, in which sensitivity analysis does not suffice [11, 19, 28]. In this paper we consider the important class of matroid optimization problems, and present efficient algorithms to compute the effect that this type of changes has over their solutions.

The *robustness function* of a matroid measures the maximum increase in the weight of its

*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.

†Department of Computer Science, Purdue University, West Lafayette IN 47907. Research of this author partially supported by NSF grant CCR-9322501, by CONACyT-Fulbright and by Universidad Autonoma Metropolitana.

minimum weight bases attainable by increases of a given total cost on the weights of its elements. We assume a cost model that for any element e of a matroid, charges $\delta * c(e)$ to increase the weight of e by δ. Similar cost models have been used to compute the robustness function for minimum spanning trees [11], shortest paths [12], and to approximate the robustness function for maximum flows in planar graphs [28].

Matroid theory provides an elegant structure that captures the essence of a large and important class of problems [23, 24, 29, 32]. There are matroid optimization problems in computational biology [17], graph theory [11, 23], and electrical networks [24, 29] for which only estimates of the input values are available, or for which changes in the input values are expected. The robustness functions for these problems can be used to assess the quality of their solutions. In this paper we present a general algorithm for computing the robustness function for any matroid. Furthermore, we identify key properties of transversal, scheduling, and partition matroids, and exploit them to design robustness algorithms that are more efficient than our general algorithm.

Our algorithm for arbitrary matroids runs in strongly polynomial time for any matroid with a strongly polynomial time independence test. There are two key ideas in this algorithm. The first is a reduction from the problem of computing the robustness function for a weighted matroid to the same problem on unweighted matroids. The second is a transformation from the latter problem to the membership problem on matroid polyhedra. Our algorithm computes the robustness function for the minimum weight bases of a weighted matroid in $O(m^5 n^2 + m^4 n^4 \tau)$ time, where m is the number of elements in the matroid, n is its rank, and τ is the time needed to test independence for a set of at most n elements. As we show, the robustness function is piecewise linear and it has at most mn breakpoints. Our algorithm can compute all the breakpoints of the robustness function within the time bound stated above.

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 (see e.g. [13]). We prove an extension of Hall's Theorem (see e.g. [32]) for minors of transversal matroids, which allows us to solve the membership problem on their matroid polyhedra by performing one min-cut computation over a bipartite graph.

Scheduling matroids are a special case of transversal matroids (see e.g. [13]). We show how to compute the robustness function for a scheduling matroid by solving a series of scheduling-with-preemption subproblems. We present a formulation for these scheduling subproblems that corresponds to a generalization of the off-line min problem [1]. By optimally solving these subproblems we can compute the robustness function for scheduling matroids in $O(mn^2(m + n^2))$ time.

For partition matroids, we give two algorithms. The first identifies increasingly larger clusters of elements as it computes the robustness function. These clusters have a certain "convexity" property that we identify and exploit to design an $O(m \log m)$ time algorithm. Our second algorithm does not compute all the breakpoints of the robustness function, but evaluates the robustness function only at a required point, and does so in $O(m)$ time. This formulation is a version of the optimal distribution of effort problem (see e.g. [22]). Since we do not have explicit representations for the gain functions, we cannot use any of the known methods of solution for the latter problem [9, 22]. Instead, we present a new approach that optimally solves the problem by a sophisticated application of a linear-time selection algorithm [3], that interleaves searches on weights with searches on costs.

The rest of the paper is organized as follows. In Section 2 we give some matroid terminology. In Section 3 we describe our general algorithm. In Sections 4 and 5 we present efficient algorithms for transversal matroids, scheduling matroids, and partition matroids, respectively. In Section 7 we present linear time algorithms for evaluating the robustness function of a uniform matroid and a partition matroid at a given point.

2 Matroid Terminology

A *weighted matroid* $M = (E, \mathcal{I}, w)$ consists of a finite set E of *elements*, a collection $\mathcal{I}$ of subsets of E satisfying well-known axioms (see e.g. [32]), and a non-negative weight function w on E. Set E is the *ground set* of the matroid M, and the subsets in $\mathcal{I}$ are called the *independent sets* of the matroid. An independent set of maximum size is a *base* of M. The number of elements in any base is the *rank* of the matroid. For any set $S \subseteq E$, $rank(S, M)$ denotes the size of the largest independent subset of S. We denote by n the rank of a matroid and by m the size of its ground set.

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 $\{\, S \mid S \in \mathcal{I} \text{ and } S \subseteq T \,\}$. The *contraction* of T from M, denoted as M/T, is the matroid with ground set $E - T$ and independent sets $\{\, S \mid S \subseteq E - T$ and $S \cup Y \in \mathcal{I} \,\}$, where Y is any base of $M|T$.

The convex hull of incidence vectors of the independent subsets of a matroid M form a *matroid polyhedron* $\mathcal{P}(M)$. Let $\mathbb{R}_+$ denote the set of nonnegative real numbers. An explicit representation for the matroid polyhedron of a matroid M is given by $\mathcal{P}(M) = \{z \in \mathbb{R}_+^E \mid \sum_{e \in S} z(e) \leq rank(S, M) \text{ for all } S \subseteq E\}$. Given a vector $x \in \mathbb{R}_+^E$, a *base* of x is a maximal vector $y \in \mathcal{P}(M)$ such that $y(e) \leq x(e)$ for all $e \in E$.

3 The General Algorithm

In this section we present an efficient algorithm for computing the robustness function F_M of a matroid $M = (E, \mathcal{I}, w)$. The basic structure of our algorithm for arbitrary matroids is similar to that presented in [11] for minimum spanning trees; however, the design of the key subroutine requires new ideas that go significantly beyond our work in [11]. We first describe our robustness algorithm, and in the next section we discuss our new ideas.

Given a set $S \subseteq E$, we define *coverage*(S, M) as the minimum number of elements that any minimum weight base of M shares with S. We say that the weights of the elements in a set S are *lifted* when the weight of every element in S is increased by the same amount. Let *tolerance*(S, M) be the minimum amount by which the weights of the elements in S have to be lifted to decrease *coverage*(S, M). The *rate* of S in M, denoted as *rate*(S, M), is defined as $c(S)/coverage(S, M)$, where $c(S)$ is the sum of the costs of the elements in S.

The following greedy algorithm computes $F_M(b)$, the maximum increase in the weight of the minimum weight bases of M achievable by making increases of total cost b on the weights of the elements in its ground set.

Algorithm *uplift*(M, c, b)
orig_wgt $\leftarrow$ weight of a minimum weight base of M
balance $\leftarrow b$
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
return $(new\_wgt^* - orig\_wgt)$

THEOREM 3.1. *Algorithm uplift correctly computes $F_M(b)$ for any value $b \geq 0$.*

Proof. An extension of the arguments presented in [11] proves the correctness of *uplift.* □

To illustrate how the algorithm works, consider the following simple example. Let $M = (E, \mathcal{I}, w)$ be a matroid with ground set $E = \{e_1, e_2, e_3, e_4, e_5\}$ and in which a set $S \subseteq E$ is independent if and only if $|S| \leq 2$. The weights of the elements are $1, 1, 1, 2$, and 4, respectively, and their costs are $1, 4, 1, 1$, and 8. Let $b = 27$. In the first iteration of the while-loop, *uplift* selects subset $S = \{e_1, e_3\}$ with *rate* 2 and *tolerance* 1, and lifts the weights of these elements to 2. The cost of these weight increases is 2, and so the unused budget has value 25. The second set selected by *uplift* is $S = \{e_1, e_3, e_4\}$ with *rate* 3 and *tolerance* 2. Algorithm *uplift* spends 6 units of the budget lifting the weights of the elements to 4. In the following two iterations *uplift* selects set $\{e_2\}$ with *rate* 4 and *tolerance* 3, and then set $\{e_1, e_2, e_3, e_4\}$ with *rate* 7 and *tolerance* ∞. The final weights of the elements are $5, 5, 5, 5$, and 4, and $F_M(27) = (4 + 5) - (1 + 1) = 7$.

LEMMA 3.1. *Algorithm uplift performs at most mn iterations.*

Proof. Consider any iteration of the while-loop of *uplift*. The algorithm first selects some subset S, that intersects every minimum weight base of M, and then it increases the weights of its elements up to some value w_i. Because of these weight increases, the number of elements of weight w_i in every minimum weight base of the matroid grows up by at least one. Therefore, 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 so the algorithm performs at most mn iterations. □

LEMMA 3.2. *The robustness function F_M is piece-wise linear, concave and non-decreasing.*

Proof. We can show in a fashion similar to that in [11] 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 whose slope is 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* $(S_1, M) = \sigma$ and that in the next iteration it chooses a set S_2 with *rate* $(S_2, M) < \sigma$. This would mean that S_1 was not the set of smallest *rate* in the previous iteration because $S_1 \cup S_2$ would have smaller rate. Therefore, the slope of F_M is non-decreasing, and so F_M is concave and non-decreasing. □

3.1 Finding a Set of Smallest Rate

In this section we describe an algorithm for finding a set of smallest *rate* in a matroid $M = (E, \mathcal{I}, w)$. Our approach consists of two stages. In the first stage we form a sub-matroid $M' = (E', \mathcal{I}', w)$ of M in which all of the elements have the same weight, and such that any subset of elements with smallest *rate* in M' also has smallest *rate* in M. In the second stage we exploit the fact that all elements in M' have the same weight to find a set of smallest *rate* in it.

Fix a matroid $M = (E, \mathcal{I}, w)$. Let $w_1, w_2, \ldots, w_p$ be the different element weights in M. For 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 3.3. *Let B be a minimum weight base of M, and $i \in \{1, 2, \ldots, p\}$ such that $B_{=w_i} \neq \emptyset$. The set $B_{=w_i}$ is a minimum weight base of the submatroid $(M|E_{\leq w_i})/E_{<w_i}$.*

Proof. Suppose that B is found using the greedy algorithm. 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 $(M|E_{\leq w_i})/E_{<w_i}$ have weight w_i. □

Lemma 3.4. *Let B be a minimum weight base of M, and $i \in \{1, 2, \ldots, p\}$ such that $B_{=w_i} \neq \emptyset$. If A is a minimum weight base of the submatroid $(M|E_{\leq w_i})/E_{<w_i}$, then $(B - B_{=w_i}) \cup A$ is a minimum weight base of M.*

Proof. By definition of the contraction operation, there is a base A' of $M|E_{\leq w_i}$ such that $A \subseteq A'$. For every $y \in B_{=w_i} - A$ there must be an element $x \in A' - B_{\leq w_i}$ such that the set $S = B_{\leq w_i} - \{y\} \cup \{x\}$ 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 S is also 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 $(B - B_{=w_i}) \cup A$ 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 $(M|E_{\leq w_i})/E_{<w_i}$.

LEMMA 3.5. *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 $(M|E_{\leq w_i})/E_{<w_i}$ 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* $(S_{=w_j}, M) = |B^j \cap S_{=w_j}| = |B^j_{=w_j}|$. By Lemmas 3.3 and 3.4 $B = \cup_{j=1}^{p} B^j_{=w_j}$ is a minimum weight base of M. Thus *coverage* $(S, M) = |S \cap B| = \sum_{j=1}^{p}$ *coverage* $(S_{=w_j}, M)$, and

$$\begin{aligned} \mathit{rate}\,(S, M) &= \frac{\displaystyle\sum_{j=1}^{p} c(S_{=w_j})}{\displaystyle\sum_{j=1}^{p} \mathit{coverage}\,(S_{=w_j}, M)} \\ &= \frac{c(S_{=w_i})}{\mathit{coverage}\,(S_{=w_i}, M)} = \mathit{rate}\,(S_{=w_i}, M) \quad \text{for any } S_{=w_i} \neq \emptyset. \end{aligned}$$

We now prove (2). By Lemmas 3.3 and 3.4, the set of minimum weight bases of submatroid $(M|E_{\leq w_i})/E_{<w_i}$ is $\{ A_{=w_i} \mid A$ is a minimum weight base of $M \}$. Hence, for any set $T \subseteq E_{=w_i}$, *coverage* $(T, (M|E_{\leq w_i})/E_{<w_i}) =$ *coverage* (T, M), and *rate* $(T, M) =$ *rate* $(T, (M|E_{\leq w_i})/E_{<w_i})$. It follows that any set of smallest rate in $(M|E_{\leq w_i})/E_{<w_i}$ has also smallest *rate* in M. □

Lemma 3.5 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 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. Let σ be the smallest *rate* of any subset of E_i. Let $g(\lambda) = \min \{ c(T) - \lambda * \mathit{coverage}\,(T, M_i) \mid T \subseteq E_i \}$. Then, σ is equal to the largest value of λ for which $g(\lambda) = 0$. The slope of the curve g at any given point λ is equal to *coverage* (S, M_i) for some set $S \subseteq E_i$, and thus, g is piecewise linear and has at most n breakpoints. Function g is non-increasing and its leftmost linear piece has slope 0. Hence, σ is equal to the value λ of the first breakpoint of g.

Since $\sigma \leq c(E)/n$, we compute the value of σ by starting at the point $\lambda = c(E)/n$ and moving backwards along the curve g using Newton's method. This procedure for computing the value of σ needs solve at most n parametric problems of the form: $\min \{ c(T)/\lambda - \textit{coverage}\,(T, M_i) \mid T \subseteq E_i \}$.

At this point, we take advantage of the fact that M_i is an unweighted matroid. It can be shown that for any set $S \subseteq E_i$, $\textit{coverage}\,(S, M_i) = \textit{rank}\,(E_i, M_i) - \textit{rank}(E_i - S, M_i)$. The above parametric problem, then, is equivalent to

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

Cunningham [5], shows that problem (1) is equivalent to the problem of finding a base for the vector c/λ in the matroid polyhedron $\mathcal{P}\,(M_i)$.

LEMMA 3.6. *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* $m_i = |E_i|$ *and* τ *is the time required to test whether a set of at most* n_i *elements is independent in* M_i.

Proof. We use the algorithm by Narayanan [27] to solve problem (1) in $O(m_i^4 + m_i^3 n_i^2 \tau)$ time. Since we have to solve at most n_i problems of the form (1) to find a set of smallest *rate* in M_i, the total time needed to solve the problem is $O(m_i^4 n_i + m_i^3 n_i^3 \tau)$. □

The process that we have described for computing a set of smallest *rate* in an arbitrary matroid finds a set S of smallest *rate* in which all of the elements have the same weight, say w_i. To compute *tolerance* (S, M), find the smallest weight w_j to which the weights of the elements in S must be lifted in order to decrease *coverage* (S, M). If such a weight w_j exists, then *tolerance* $(S, M) = w_j - w_i$, otherwise *tolerance* $(S, M) = \infty$.

Since the elements in S have weight w_i, $\textit{coverage}\,(S, M) = \textit{coverage}\,(S, (M|E_{\leq w_i})/E_{<w_i})$. Recall that in an unweighted matroid $M_i = (M|E_{\leq w_i})/E_{<w_i}$, $\textit{coverage}\,(S, M_i) = \textit{rank}\,(E_{=w_i}, M_i) -$

$rank(E_{=w_i} - S, M_i)$. Thus, $coverage(S, M_i)$ 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 3.2. *The robustness function of a weighted matroid $M = (E, \mathcal{I}, w)$ 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 $1 \leq i \leq p$. The total time required to solve these subproblems is $\sum_{w_i} O(|E_{w_i}|^4 n + |E_{w_i}|^3 n^3 \tau) = O(m^4 n + m^3 n^3 \tau)$. This time dominates each iteration of *uplift*, and since by Lemma 3.1*uplift* performs at most mn iterations, the total time is $O(m^5n^2 + m^4n^4\tau)$.

COROLLARY 3.1. *Algorithm uplift can compute all the breakpoints of the robustness function of a matroid in $O(m^5n^2 + m^4n^4\tau)$ time.*

Proof. A breakpoint of the robustness function is detected by *uplift* when the *rate* of the set selected in the current iteration of the while-loop is larger than the *rate* of the set selected in the preceding iteration. □

4 Transversal Matroids

A *transversal matroid* can be defined in terms of matchings in a bipartite graph. Given a bipartite graph $G = (D \cup D', E)$ a *matching* for G is a subset of edges $S \subseteq E$ such that no two edges in S share a common endpoint. A transversal matroid $M = (D, \mathcal{I}, w)$ has as ground set, D, the set of vertices in one side of the bipartite graph G. A subset of D is independent in M if it can be covered by a matching. Let $m = |D|$ and n be the rank of M.

Since most applications of transversal matroids seem to use maximum weight bases [13, 23], we could frame our robustness problem in terms of decreasing weights. However, to avoid redefining notation from the previous section, we keep the same model as before. It is not difficult to check that to decrease the weight of the maximum weight bases of a matroid, by simply changing each weight element w_i to $-w_i$ we get an equivalent problem in which the goal is to increase the weight of the minimum weight bases. Again, we use our algorithm *uplift* to compute the robustness function for a transversal matroid, but design a more efficient algorithm for solving problem (1) in the computation of a set with smallest *rate*.

Let $M_k = (D_k, \mathcal{I}_k) = (M|D_{\leq w_k})/D_{<w_k}$. For any set $S \subseteq D_k$ and vector $x \in \mathbb{R}^{D_k}$ we let $x(S) = \sum_{e \in S} x(e)$. From results in [5, 8], it follows that for any vector $x \in \mathbb{R}^{D_k}$, $\min\{\, x(S) + \mathit{rank}\,(D_k - S, M_k) \mid S \subseteq D_k \,\} = \max\{y(D_k) \mid y \in \mathcal{P}\,(M_k) \text{ and } y \leq x\,\}$, where $\mathcal{P}\,(M_k)$ is the matroid polyhedron for M_k. Problem (1) then can be formulated as

$$\max\{y(D_k) \mid y \in \mathcal{P}\,(M_k) \text{ and } y \leq c/\lambda\,\}. \tag{2}$$

We derive an expression for $\mathcal{P}\,(M_k)$ that allows us to solve efficiently problem (2). Consider again the bipartite graph $G = (D \cup D', A)$ used to define transversal matroid M. For any $S \subseteq D$, let $\mathcal{N}(S) = \{\, v \in D' \mid v \text{ is adjacent to some vertex of } S\,\}$. Hall's Theorem (see e.g. [32]) states that 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 alternative expression for the matroid polyhedron $\mathcal{P}\,(M_k)$ will come from a description of the independent sets of the submatroids M_k of the same flavor as that provided by Hall's Theorem.

Let B be a maximum weight base of transversal matroid M. Let w_k be the weight of an element of B and $\bar{B}_{=w_k} = B - B_{w_k}$. A set B' is independent in M_k if and only if $B' \cup \bar{B}_{=w_k}$ is independent in M. For any set $T \subseteq D_k$, we define the function

$$f(T) = \min\{\, |\mathcal{N}(T \cup S)| - |S| \mid S \subseteq \bar{B}_{=w_k}\,\}. \tag{3}$$

LEMMA 4.1. *A set $S \subseteq D_k$ is independent in M_k if and only if $|S \cap T| \leq f(T)$ for all $T \subseteq D_k$.*

Proof. Let set $S \subseteq D_k$ be independent in M_k. Then, $S \cup \bar{B}_{=w_k}$ is an independent set of M, and by Hall's Theorem, $|N(S')| \geq |S'|$ for all $S' \subseteq (S \cup \bar{B}_{=w_k})$. Let $T \subseteq D_k$ and $F \subseteq \bar{B}_{=w_k}$.

Clearly $((S \cap T) \cup F) \subseteq (S \cup \bar{B}_{=w_k})$ 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 $T \subseteq D_k$.

Now, consider the other direction of the implication. Suppose that for some set $S \subseteq D_k$, $|S \cap T| \leq f(T)$ for all $T \subseteq D_k$. This means that $|S \cap T| \leq |\mathcal{N}(T \cup F)| - |F|$ for all $T \subseteq D_k$ and $F \subseteq \bar{B}_{=w_k}$. In particular, by choosing $T \subseteq S$ 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 $(T \cup F) \subseteq (S \cup \bar{B}_{=w_k})$, and hence, by Hall's Theorem, $S \cup \bar{B}_{=w_k}$ is independent in M. □

A set function $g : 2^F \mapsto \mathbb{R}$ is *submodular* if $g(S) + g(T) \geq g(S \cup T) + g(S \cap T)$ holds for all $S, T \subseteq F$.

LEMMA 4.2. *The function f is submodular.*

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

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

□

The following result by Edmonds [8, 6] allows us to give explicit expressions for the independent sets of M_k and for its matroid polyhedron $\mathcal{P}(M_k)$.

THEOREM 4.1. (EDMONDS) *Let $g : 2^F \mapsto \mathbb{R}$ be a submodular function and $\mathcal{H} = \{ S \mid S \subseteq F$ and $|S \cap F'| \leq g(F')$ for all $F' \subseteq F \}$. Then $\mathcal{H}$ is the family of independent sets of a matroid on F, and the convex hull of incidence vectors of members of $\mathcal{H}$ is $\mathcal{C} = \{ y = (y_1, y_2, \ldots, y_{|F|}) \mid 0 \leq y_i \leq 1$ for all $1 \leq i \leq |F|$ and $y(F') \leq g(F')$ for all $F' \subseteq F \}$.* □

This Theorem and Lemma 4.2 allow us to state the following result.

THEOREM 4.2. *The family of independent sets of the submatroid* $M_k = (D_k, I_k)$ *is* $\mathcal{I}_k = \{ S \mid S \subseteq D_k$ *and* $|S \cap F| \leq f(F)$ *for all* $F \subseteq D_k \}$ *and its matroid polyhedron is* $\mathcal{P}(M_k) = \{ z = (z_1, z_2, \ldots, z_{m_k}) \mid 0 \leq z_i \leq 1$ *for all* $1 \leq i \leq m_k$ *and* $z(F) \leq f(F)$ *for all* $F \subseteq D_k \}$.

Theorem 4.2 allows us to solve problem (2) using the polymatroid greedy algorithm (see e.g. [15, 7]). Let $x = c/\lambda$. The algorithm is the following.

Algorithm *poly_greedy* (M_k, f, x)
$y(e_i) \leftarrow 0$ for all $e_i \in D_k$
for each $e_\ell \in D_k$ **do**
  $\delta \leftarrow \min\{ f(T) - y(T) \mid e_\ell \in T \subseteq D_k \}$
  if $\delta \leq x(e_\ell)$ **then** $y(e_\ell) \leftarrow \delta$
  else $y(e_\ell) \leftarrow x(e_\ell)$ **end if**
end for

In each iteration of the for-loop, algorithm *poly_greedy* needs to solve a minimization problem that, using the definition of f, can be rewritten as $\min \{ |\mathcal{N}(T \cup F)| - |F| - y(T) \mid e_\ell \in T \subseteq D_k$ and $F \subseteq \bar{B}_{=w_k} \}$. This problem can be solved by computing a minimum cut in an auxiliary graph $\hat{G}_\ell = (\{s,t\} \cup D_k \cup \bar{B}_{=w_k} \cup D', E_\ell)$ (see Figure 1). 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 $e_i \in D_k - \{e_\ell\}$. It also has an edge of capacity 1 from s to every vertex in $\bar{B}_{=w_k}$, and an edge of infinite capacity from a vertex $v \in D_k \cup \bar{B}_{=w_k}$ to a vertex $w \in D'$ whenever the edge (v, w) is in G. Finally, $\hat{G}_\ell$ has an edge of capacity 1 from each vertex in D' to t.

LEMMA 4.3. *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,* $c(R, \bar{R}) - y(D_k) - |\bar{B}_{=w_k}| = \min \{ |\mathcal{N}(T \cup F)| - |F| - y(T) \mid e_\ell \in T \subseteq D_k$ and $F \subseteq \bar{B}_{=w_k} \}$.

Proof. Assume that $s \in R$. Clearly the vertex corresponding to the element 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 1 it is apparent that a minimum cut of $\hat{G}_\ell$ has finite capacity.

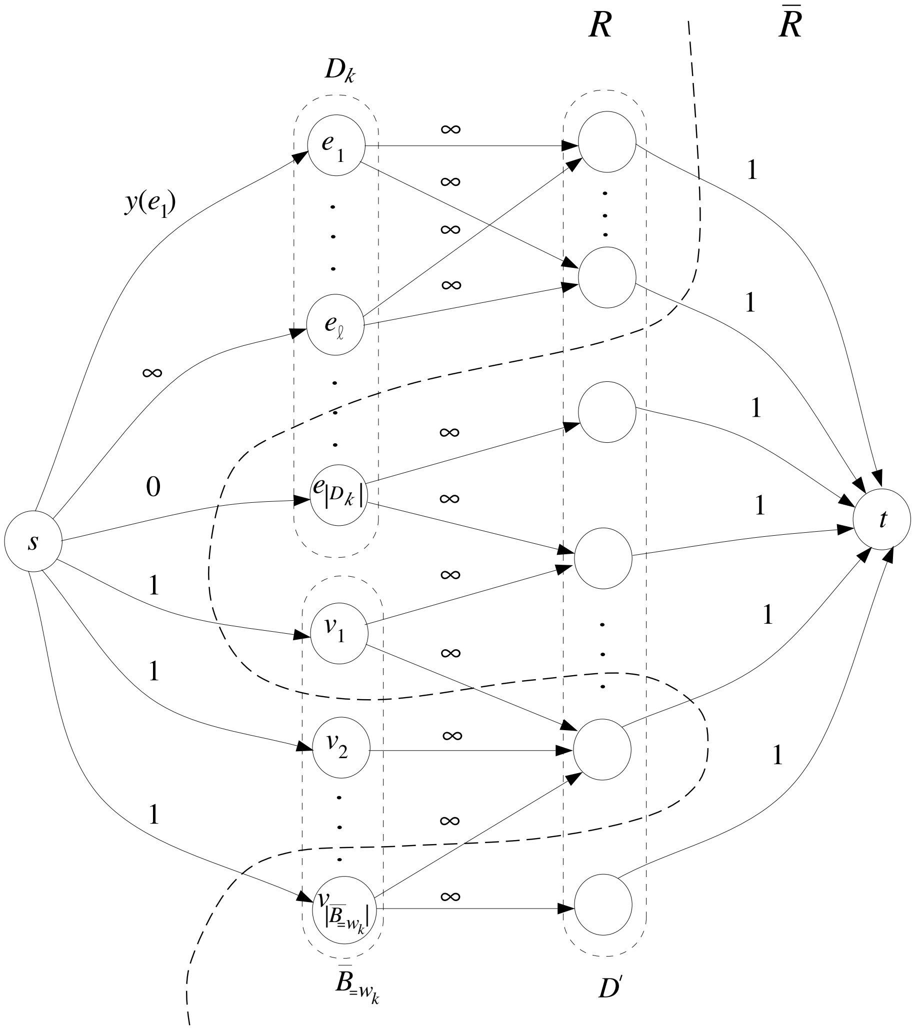

Figure 1: Auxiliary graph $\hat{G}_\ell$ showing a minimum cut $(R, \bar{R})$.

Let $R_{D_k} = R \cap D_k$ and $R_{\bar{B}_{=w_k}} = R \cap \bar{B}_{=w_k}$. We can show that $c(R, \bar{R}) = |\mathcal{N}(R_{D_k} \cup R_{\bar{B}_{=w_k}})| + \sum_{e \in (D_k - R_{D_k})} y(e) + |\bar{B}_{=w_k} - R_{\bar{B}_{=w_k}}|$. Therefore,

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

□

Lemma 4.3 allows us to implement *poly_greedy* by performing $|D_k|$ maximum flow computations on auxiliary graphs $\hat{G}_\ell$, however, we can do better than that. In each iteration of the for-loop of *poly_greedy*, the value of δ can be computed by finding a maximum flow that saturates every edge leaving s, except the edge (s, e_ℓ) of infinite capacity. We can show that δ is equal to the flow going through edge (s, e_ℓ) in this maximum flow. If the flow on edge (s, e_ℓ) is larger than $x(e_\ell)$, then *poly_greedy* reduces it to $x(e_\ell)$. Thus, since each iteration of the for-loop essentially computes the maximum flow that can be sent through edge (s, e_ℓ), we can modify the auxiliary graphs so that the flow computed at the end of all iterations of the for-loop can be determined with a single maximum flow computation on a new auxiliary graph $\bar{G}$.

Build graph $\hat{G} = (\{s, t\} \cup D_{=w_k} \cup \bar{B}_{=w_k} \cup D', \hat{E})$ with the same topology and edge capacities as any of the graphs $\hat{G}_\ell$, except that each edge from s to $e_i \in D_k$ has capacity $x(e_i)$. Find a maximum flow z for $\hat{G}$ that saturates every edge from s to $\bar{B}_{=w_k}$. (Such a flow must exist because the elements in $\bar{B}_{=w_k}$ form an independent set of transversal matroid M, and hence, there has to be a matching in $\hat{G}$ covering them.) Let z' be the vector describing the value of the flow through the edges from s to vertices in D_k.

LEMMA 4.4. *The vector z' is a maximizer for problem* (2).

Proof. It is clear that $z'(s, e_\ell) \le x(s, e_\ell)$ for every edge (s, e_ℓ), $e_\ell \in D_k$. We need only prove that z' is a maximal vector in $\mathcal{P}(M_k)$ with this property. Let F and S be, respectively, subsets of D_k and $\bar{B}_{=w_k}$. Since z is a valid flow function for $\hat{G}$, then it follows that $z(F \cup S) \le \mathcal{N}(F \cup S)$. Also, $z(F \cup S) = z(F) + |S|$ because z saturates every edge incoming to vertices in S. Combining these two inequalities we get that $z'(F) \le \mathcal{N}(F \cup S) - |S|$ for any $F \subseteq D_k$ and $S \subseteq \bar{B}_{=w_k}$. Therefore, by Theorem 4.2, $z' \in \mathcal{P}(M_k)$.

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

Note that during the computation of a set of smallest *rate* in M_k, each iteration of Newton's method decreases the value of the parameter λ in (1) and increases the vector c/λ that defines the capacities of the edges from s to D_k in $\hat{G}$. Hence in two successive iterations of Newton's method, the only change that $\hat{G}$ suffers is an increase in the capacities of some edges leaving the source vertex. The use of parametric flow techniques of [16] can perform all iterations of Newton's method in the time required to perform a single maximum flow computation.

THEOREM 4.3. *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. We use the algorithm FIFO with dynamic trees in [2] to compute the parametric maximum flows for $\hat{G}$ in $O(|D_k \cup \bar{B}_{w_k}| * |\hat{E}|\log(|D_k \cup \bar{B}_{w_k}|^2/|\hat{E}|+2))$ time. This is also the time needed to compute a set of smallest *rate* in one of the submatroids M_k. Then, the total time required to find a set of smallest *rate* in M is $O(\sum_{w_k} |D_k \cup \bar{B}_{=w_k}| * |\hat{E}|\log(|D_k \cup \bar{B}_{=w_k}|^2/|\hat{E}|+2)) = O(|E|\log(m^2/|E|+2)\sum_{w_k}|D_k \cup \bar{B}_{=w_k}|)$. Observe that the sets D_k and $\bar{B}_{=w_k}$ are disjoint and that $\sum_{w_k}|D_k| = |D| = m$. Furthermore, we note that at most n of the matroids M_k 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 and w_i be larger than the weight of any element in B. Then, the only independent set in matroid $M_i = (M|E_{\leq w_i})/E_{<w_i}$ is the empty set. Therefore, $\sum_{w_k}|\bar{B}_{=w_k}| \leq n^2$, and the total time to compute a set of smallest rate in M is $O((m+n^2)|E|\log(m^2/|E|+2)$. Algorithm *uplift* adds an additional factor mn to this time. □

5 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 renders a profit $w(j_i)$ when completed. Job j_i has integer release time r_i and deadline d_i, with $d_i > r_i$. We assume that the largest deadline has value at most m. The problem is to select a subset of jobs that when executed on a single processor maximizes the total profit earned.

This problem can be reduced to a weighted matching problem on a *convex* bipartite graph $G = (J \cup T, A)$ [13]. We think of the i-th vertex of J as the job j_i and the i-th vertex of T as the time interval from $i-1$ to i. Graph G has edges of weight $w(j_i)$ from the vertex corresponding to j_i to each one of the vertices corresponding to the time intervals between r_i and d_i. It can be proved that a matching of maximum weight in G corresponds to a scheduling of maximum profit for J [13].

The set of matchings of a bipartite graph can be modeled with a transversal matroid. For the special case of a convex bipartite graph, the corresponding transversal matroid is called a *scheduling matroid.* A scheduling matroid $M = (J, \mathcal{I}, w)$ has a set of jobs as its ground set and a subset of jobs is considered to be independent in M if and only if there is a valid scheduling for them.

We compute the robustness function for a scheduling matroid using the same approach presented in the previous section, but exploit the special structure of scheduling matroids to get a faster algorithm.

In the previous section we showed how to solve problem (2) by computing a maximum flow z on an auxiliary graph $\hat{G}$. This flow z must saturate all edges from s to $\bar{B}_{=w_k}$. We show that flow z can be computed in linear time for the case of a convex bipartite graph. We describe first how to find in linear time a maximum flow for $\hat{G}$, and then we show how to modify it so that it saturates all edges from s to $\bar{B}_{=w_k}$.

Consider the auxiliary graph $\hat{G} = (\{s,t\} \cup D_k \cup \bar{B}_{=w_k} \cup D', \hat{E})$. The problem of computing a maximum flow of $\hat{G}$ can be interpreted as a scheduling-with-preemption problem on a single processor. Let the vertices $t_k \in D'$ be time slots and the vertices $j_i \in D_k \cup \bar{B}_{=w_k}$ 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, t_k) identifies a time slot t_k where job j_i can be scheduled. Under this interpretation, any flow function for $\hat{G}$ defines a feasible preemptive schedule for various portions of the jobs in $D_k \cup \bar{B}_{=w_k}$: 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, t_k) defines the portion of job j_i that is scheduled in time interval t_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 $D_k \cup \bar{B}_{=w_k}$ on a single processor defines a valid flow for $\hat{G}$.

By the above discussion, any algorithm that finds a preemptive schedule for the jobs in $\bar{B}_{=w_k} \cup D_k$ that maximizes the amount of time that the jobs are executed on a single processor, also computes a maximum flow for $\hat{G}$. We present below an efficient implementation of the *preemptive earliest deadline first rule* [21] that finds in linear time an optimal preemptive schedule for $\bar{B}_{=w_k} \cup D_k$, and thus, a maximum flow for $\hat{G}$.

We assume that the largest deadline among the jobs in $D_k \cup \bar{B}_{=w_k}$ is at most $|D_k \cup \bar{B}_{=w_k}|$. If this condition does not hold, we use the techniques in [10] to modify, in $O(|D_k \cup \bar{B}_{=w_k}|)$ time, the deadlines so that this condition is satisfied. 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* [1]. In this problem we are given a set of integers $\{1, 2, \ldots, p\}$, 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 σ 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 last element before extraction, as necessary. Given a sequence σ 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 σ 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, and 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.

To solve the generalized off-line min problem, write the sequence σ 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 the following.

Algorithm *generalized_off-line*$(\sigma, \{e_1, e_2, \ldots, e_m\})$
Initialize each $Q(i)$ to $\emptyset$, and set j to 1.
while $j < m$ **do**
  $i \leftarrow \mathrm{FIND}(e_j)$.
  Add e_j to the end of list $Q(i)$.
  Find r, the sum of magnitudes of the elements in $Q(i)$.
  if $r \geq 1$ **then**
    $\mathrm{UNION}(i, succ(i), succ(i))$
    $magnitude\,(e_j) \leftarrow r - 1$
  end if

if $r \leq 1$ **then** $j \leftarrow j+1$ **end if**

end while

return (Q)

LEMMA 5.1. *Algorithm generalized_off-line runs in $O(m)$ time.*

Proof. We can use *counting sort* (see e.g. [4]) to sort the elements non-decreasingly by value in $O(m)$ time. So, we can index the elements as required by *generalized_off-line* in linear 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 [14] to perform all the UNION and FIND operations in $O(m)$ time. □

We use algorithm *generalized_off-line* to find an optimal scheduling for the jobs in $D_k \cup \bar{B}_{=w_k}$ as follows. We initialize the data structures of *generalized_off-line* by storing in set i all jobs with the i-th smallest release time. The jobs are indexed non-decreasingly by deadline. The attribute magnitude of each job is set equal to the processing time for the job. Run algorithm *generalized_off-line* with the following slight change. In each iteration of the while-loop, after finding the set i containing some job e_j, *generalized_off-line* inserts e_j in $Q(i)$ only if its deadline is at least i. The schedule can easily be obtained from the information that *generalized_off-line* stores in Q.

LEMMA 5.2. *An optimal scheduling for the jobs $D_k \cup \bar{B}_{=w_k}$ can be computed in $O(|D_k \cup \bar{B}_{=w_k}|)$ time and $O(|D_k \cup \bar{B}_{=w_k}|)$ space.*

Proof. The above algorithm runs in time linear in the number of jobs. □

The maximum flow of $\hat{G}$ corresponding to the schedule generated by *off_line* might not saturate all edges from s to $\bar{B}_{=w_k}$. We show how to find a flow that complies with this condition, so that we can use Lemma 4.4 to solve problem (2).

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 the new scheduling 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 several 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 $D_k \cup \bar{B}_{=w_k}$.

LEMMA 5.3. *The schedule S^* maximizes the amount of time that the jobs in $D_k \cup \bar{B}_{=w_k}$ 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 for the jobs in $D_k \cup \bar{B}_{=w_k}$ 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 such optimal schedule 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 value 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 $f_k < d'_i$. 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. $\square$

THEOREM 5.1. *The robustness function of a scheduling matroid $M = (J, \mathcal{I}, w)$ can be computed in $O(mn^2(m+n^2))$ time.*

Proof. Problem (2) can be solved, using the process described above, in $O(|D_k \cup \bar{B}_{=w_k}|)$ time. Thus, Newton's method finds a set of smallest *rate* in any of the submatroids M_k in $O(n|D_k \cup \bar{B}_{=w_k}|)$ time. The total time used to find a set of smallest *rate* in the scheduling matroid M is $O(\sum_{w_k} n|D_k \cup \bar{B}_{=w_k}|) = O(n(m+n^2))$. Therefore, *uplift* computes the robustness function of M in $O(mn^2(m+n^2))$ time. $\square$

6 Partition Matroids

A *partition matroid* $P = (E, \mathcal{I}, w)$ 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 $1 \leq i \leq \ell$.

As in the case of arbitrary matroids, we use algorithm *uplift* to compute the robustness function of a partition matroid. However, we take advantage of the special structure of partition matroids to design a very efficient algorithm for finding a set of smallest *rate*.

The blocks E_i 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 set 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* in a partition matroid. 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 $P = (E, \mathcal{I}, w)$ of rank n a set $S \subseteq E$ is independent if and only if $|S| \leq n$.

Fix a uniform matroid $P = (E, \mathcal{I}, w)$ with rank n. Let w_n be the weight of the n-th smallest element in E according to the weights. The only elements of E that can belong to a minimum weight base of P are those elements of weight at most w_n. Let $\Delta = |E_{<w_n}| + |E_{=w_n}| - n$.

LEMMA 6.1. *If $S \subseteq E$ is a set of smallest rate in P then*

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

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

Lemma 6.1 states that there is a set of smallest *rate* in P 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 result allows us to compute efficiently the value of k. Let $\{e_1, e_2, \ldots, e_{|E_{=w_n}|}\}$ be the elements in $E_{=w_n}$ indexed in non-decreasing order of cost.

LEMMA 6.2. *The discrete function $f(i) = \sum_{j=1}^{i} c(e_j)/(i - \Delta)$ 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 value where f reaches its minimum.*

Proof. Is it easy to show that if $f(i) \geq f(i-1)$ for any $\Delta + 2 \leq i \leq |E_{=w_n}| - 1$, then $f(i+1) \geq f(i)$. □

To compute efficiently the value of k we use a balanced binary search tree $T_{=w_n}$. 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 $I(x)$ be the set formed by all the elements stored in the leaves of $T_{=w_n}$ that appear before x in an in-order traversal of the tree. Let $\mathit{next}(x)$ 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 the following algorithm.

Algorithm $\mathit{descend}\,(T_{=w_n}, E_{=w_n}, n)$

Let x point to the root of $T_{=w_n}$.

while x does not point to a leaf **do**

  if $|I(x)| \leq \Delta$, or $rate\,(I(x), P) > rate\,(I(succ\,(x)), P)$ **then**

    Move x to the root of its right subtree.

  else Move x to the root of its left subtree. **end if**

end while

if (cost of the element stored in x) $= rate\,(I(x), P)$ **then**

  Find the rightmost leaf y that stores an element of cost equal to $rate\,(I(x), P)$.

  return $(I(y))$

else return $(I(x))$ **end if**

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 $rate\,(I(x), P) = c(I(x))/(|I(x)| - \Delta)$, to compute $rate\,(I(x), P)$ we need to know in each iteration of *descend* the number of elements in $I(x)$ and their total cost. This can easily be found 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 also allows us to compute efficiently $rate\,(I(next(x)), P)$. Since $T_{=w_n}$ is a balanced tree, algorithm *descend* runs in $O(\log |T_{=w_n}|)$ time. □

Algorithm *descend* computes efficiently the set of smallest *rate* in P, when such a set is a subset of $E_{=w_n}$. But, when such a set comes from $E_{<w_n}$ we need an additional data structure to compute it quickly. 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 we can find in constant time the element of smallest cost in $E_{<w_n}$ and, thus, by Lemma 6.1 we can find a set of smallest *rate* in P in $O(\log |T_{=w_n}|)$ time.

We describe now how to update our data structures after each iteration of *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. First, suppose that the singleton $\{e_s\} \subseteq E_{<w_n}$ is chosen as the subset of smallest *rate* in P. In this case, *uplift* increases the weight

of e_s up to w_n, and then, e_s is removed from $H_{<w_n}$ and inserted in $T_{=w_n}$. Second, suppose that set $S \subseteq E_{=w_n}$ is selected as the set of smallest *rate* in P. Algorithm *uplift* increases the weights of the elements in S to w_g, the smallest element weight larger than w_n. Since w_g becomes the n-th smallest element weight in E, all elements in $E_{=w_n} - S$ are moved from $T_{=w_n}$ to $H_{<w_n}$. Each element in $L_{>w_n}$ of weight w_g is removed form $L_{>w_n}$ and inserted in $T_{=w_n}$.

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

Proof. We first show that *uplift* performs fewer than $2|E|$ iterations. In each iteration in which *uplift* selects some $S \subseteq E_{=w_n}$ as the set of smallest *rate* in P, at least one element is removed from $L_{>w_n}$. This can happen at most $|E| - n$ times, since, by the above discussion, no element is ever inserted into $L_{>w_n}$ while updating the data structures.

In each iteration in which a singleton $\{e_s\}$ is chosen as the set of smallest *rate* in P, e_s is moved from $H_{<w_n}$ to $T_{=w_n}$, and it remains in $T_{=w_n}$ until the algorithm ends. To prove this, suppose that in a later iteration j, 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 P. Since *descend* computes the largest set of smallest *rate* in $E_{=w_n}$, it follows that $c(e_s) > \mathit{rate}\,(T, P)$. 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 the total number of iterations that *uplift* performs is at most $2|E| - n$.

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

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 smallest 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 can now state the time complexity for computing the robustness function of a partition matroid $P = (E, \mathcal{I}, w)$ with $\ell \geq 1$.

THEOREM 6.1. *The robustness function of a partition matroid $P = (E, \mathcal{I}, w)$ 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 set E_i. To find the set with overall smallest *rate* in P, we use a heap H in which we store, for each $i = 1, 2, \ldots, \ell$, the smallest *rate* of any subset of E_i.

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 $O(|E|)$. $\square$

7 Evaluating the Robustness Function at One Point

In the previous subsection we presented an algorithm that computes the robustness function F_P for a partition matroid P. This algorithm has the property that every partial solution that it builds gives the value of F_P at some point $b \geq 0$. We exploited this property to compute an explicit representation of the whole robustness function F_P within the same time bound stated in Theorem 6.4. If we are not interested in computing all the breakpoints of F_P, but only want to evaluate F_P at a specific point b, then we can design an algorithm that requires only linear time.

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

Therefore, we can write the value of $F_P(b)$ as

$$F_P(b) = \max\{\sum_{i=1}^{\ell} F_{P_i}(b_i) \mid b_i \geq 0 \text{ for all } 1 \leq i \leq \ell \text{ and } \sum_{i=1}^{\ell} b_i = b\} \tag{4}$$

For arbitrary functions F_{P_i}, problem (4) is known as the *optimal distribution of effort problem* [22] or as the *convex knapsack problem* [9]. There are several efficient algorithms to solve these problems [20, 9], but only under the assumption that the functions F_{P_i} are given in an explicit form that allows us 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 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 S, the largest set of smallest *rate* in P, and each element $e \in S$ has its weight increased by the same amount $\Delta \leq$ *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 7.1. *Let $\delta = \delta_1, \delta_2, \ldots, \delta_k$ be a canonical sequence of increases and S be the set selected by some δ_i, $i < k$. In all subsequent increases δ_j, $j > i$, all elements of S will undergo the same weight changes.*

Proof. Let e_1 and e_2 be elements of S. Note that the cost of e_1 (e_2) cannot be larger than the *rate* r_S of S, because otherwise $S - \{e_1\}$ ($S - \{e_2\}$) would have a smaller *rate* than S. Suppose that δ_j, $j > i$, selects set $S' \subseteq E$, with *rate* $r_{S'}$, as the largest set of smallest *rate* in P, and that $e_2 \in S'$ but $e_1 \notin S'$. This means that $r_{S'} < c(e_1)$, because otherwise $S' \cup \{e_1\}$ would be a larger set with *rate* no larger than $r_{S'}$. But, then $r_{S'} < r_S$ since $c(e_1) \leq r_S$. This contradicts that the robustness function of P is non-decreasing. $\square$

This property can be used to design a 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 algorithm *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 7.1 to yield a linear time algorithm for computing $F_P(b)$, for a given $b \geq 0$. To introduce our basic strategy, we show first how to compute $F_P(b)$ for a uniform matroid P.

7.1 Evaluating the Robustness Function of a Uniform Matroid

Let $\delta(w_i)$ be a canonical sequence of increases for the elements of uniform matroid $P = (E, \mathcal{I}, w)$ such that the weights of the elements are increased as much as possible without increasing the weight of any element above w_i. Let $\delta'(r_i)$ 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.

Our algorithm does not perform a linear search over the curve F_P, as *uplift* does, but evaluates F_P 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 one, called *up_to_weight*, takes a weight w_i and computes the weight increases corresponding to $\delta(w_i)$. The routine outputs the cost of the weight increases and the *rate* of the largest 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 $\delta'(r_i)$. The routine outputs the total cost of the weight increases and the weight of the nth smallest element according to the increased weights.

We describe below a recursive algorithm for computing the value of $F_P(b)$, for a given $b \geq 0$. Each recursive call reduces in linear time the size of the ground set by a fraction of at least one third. Hence, the overall time complexity of the algorithm is linear in the number of elements in the matroid. 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 removed from E), 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).

Algorithm *robustness_uniform* $(P = (E, \mathcal{I}, w), b)$

if $|E| = 1$ **then**

  Let $E = \{e\}$. Set $w(e) \leftarrow w(e) + b/c(e)$.

  return (weight function w)

else

  Compute w_t, the weight of the $\lceil \frac{2}{3}|E| \rceil$th smallest element in E.

  $(c', r') \leftarrow$ *up_to_weight* (w_t, P)

  if $c' \geq b$ **then**

    $E \leftarrow E - \{e \mid w(e) \geq w_t\}$

    return (*robustness_uniform* $(P|E, b)$)

  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* $(\bar{c}, P)$

    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* $(P/E, b)$)

    else

      $(P' = (E', \mathcal{I}', w'), b') \leftarrow$ *contract_uniform* $(P, b, \tilde{c}, \tilde{w}_n, \bar{c})$

      if $b' = 0$ **then return** (weight function w')

      else return (*robustness_uniform* (P', b')) **end if**

    end if

  end if

end if

The algorithm returns the weight function corresponding to optimal weight increases of total cost b. With this function and the initial weight function it is trivial to compute in linear time the value of $F_P(b)$.

The first two recursive calls to *robustness_uniform* are made after reducing the ground set E. In the first call, *robustness_uniform* receives as first argument the submatroid $P|E$ that results after discarding from E all elements of weight larger than w_t. In the second call, the algorithm receives as argument the submatroid P/E, since the elements of weight at most $\tilde{w}_n$ and cost at least $\bar{c}$ will not have their weights increased, and hence, they can be contracted from P. The third recursive call to algorithm *robustness_uniform* is preceded by a call to routine *contract_uniform*, described below. This routine contracts to a single meta-element some subset $S \subseteq E$ of size at least $\lceil |E|/3 \rceil$.

Algorithm *contract_uniform* $(P = (E, \mathcal{I}, w), b, \tilde{c}, \tilde{w}_n, \bar{c})$
Let $S = \{e \mid w(e) \leq \tilde{w}_n \text{ and } c(e) < \bar{c}\}$, and $T = \{e \mid w(e) \leq \tilde{w}_n \text{ and } c(e) = \bar{c}\}$.
Increase to $\tilde{w}_n$ the weight of every element in S, and set $b \leftarrow b - \tilde{c}$
for every $e \in T$ **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 for-loop **end if**
end for
if $b > 0$ **then**
    $E \leftarrow (E - S - T) \cup \{e'\}$, where e' is meta-element with $w(e') = \tilde{w}_n$ and $c(e') = c(S \cup T)$.
end if
Let P' be the matroid corresponding to the new set E and weight function w.
return (P', b)

Every call to algorithm *contract_uniform* is preceded by a call to algorithm *up_to_rate*, which determines the cost $\tilde{c}$ of optimally increasing the weights of the elements in P to the point at which the next weight increase would be 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 set T. If while performing these increases the budget is exhausted, then the algorithm stops since it has computed the weight function needed to determine $F_P(b)$. Otherwise, the budget is decreased to account for the new weight increases.

If after the new weight increases there is any budget left, then *contract_uniform* contracts all elements of weight at most $\tilde{w}_n$ and cost at most $\bar{c}$. Note that these elements belong to the first set formed by elements of weight $\tilde{w}_n$ that a canonical sequence of increases selects, and thus, by Property 7.1 they can be replaced by a meta-element.

To illustrate how algorithm *robustness_uniform* works, consider the following example. Let $P = (E, \mathcal{I}, w)$ be a uniform matroid of rank 4 with ground set $E = \{a, b, c, d, e, f, g\}$. The weights of the elements are $1, 1, 2, 3, 4, 5, 6$, respectively, and their costs are $1, 3, 5, 1, 3, 1, 3$. We want to evaluate $F_P(9)$. The algorithm first computes $w_t = 4$, and invokes routine *up_to_weight*. This routine returns the pair $(c', r') = (4, 2)$ that corresponds to the cost of increasing the weights of the elements to $4, 1, 2, 4, 4, 5, 6$, respectively; with these weights the next set of smallest *rate* has *rate* $r' = 2$. Since $c' < 9$, the algorithm computes $\bar{c} = 3$, and invokes routine *up_to_rate*. This routine returns the pair $(\tilde{c}, \tilde{w}_n) = (6, 5)$ that corresponds to the cost of increasing the weights of the elements to $5, 1, 2, 5, 3, 5, 6$. Since $\tilde{c} < 9$, routine *contract_uniform* is invoked. This routine increases the weights of the elements to $5, 2, 2, 5, 5, 6$, exhausting the budget. Since upon return from routine *contract_uniform*, algorithm *robustness_uniform* finds $b' = 0$, the algorithm outputs these last weights, and stops. A simple calculation shows that $F_P(9) = 11 - 7 = 4$.

Lemma 7.1. *Algorithm robustness_uniform computes $F_P(b)$, for any given $b \geq 0$, in $O(|E|)$ time.*

Proof. Assume linear-time routines *up_to_rate* and *uplift_to_weight*. 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 P.

In each recursive call, algorithm *robustness_uniform* invokes *up_to_weight*, which computes the same weight increases that $\delta(w_t)$ would make. If the cost of these weight increases exceeds b, 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 optimal weight increases defined by $\delta'(\bar{c})$. If the cost of these last weight 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 discards at least $\lceil\frac{1}{3}|E|\rceil$ elements from E. If the cost of the weight increases is smaller than b, *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 $\lceil\frac{2}{3}|E|\rceil$ round 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 $\bar{c}$). □

We describe now an algorithm to compute in linear time the largest set of smallest *rate* in a uniform matroid P. This algorithm is needed to design the linear-time routines *up_to_weight* and *up_to_rate*. The algorithm is as follows. First, compute w_n, the weight of the nth smallest element in E. By Lemma 6.1, the largest set of smallest *rate* in P might consist of all elements of smallest cost in $E_{<w_n}$, the largest optimal subset of $E_{=w_n}$, or the union of both. It is easy to find in linear time all elements of smallest cost in $E_{<w_n}$. To find the largest set of smallest *rate* in $E_{=w_n}$, perform a binary search on the costs of the elements to find the smallest cost c' for which $c(T)/(|T| - |E_{\leq w_n}| + n) \leq (c(T) + c')/(|T| + 1 - |E_{\leq w_n}| + n)$, where $T \subseteq E_{=w_n}$ is formed by all elements in $E_{=w_n}$ of cost smaller than c'. This binary search uses the linear-time algorithm of Blum et al. [3] to find the median element cost. This algorithm computes the largest set of smallest *rate* in P in $O(|E_{\leq w_n}|)$ time.

We present below an implementation of *up_to_weight* using this algorithm for computing the largest set of smallest rate in P. The algorithm receives as arguments a weight w_i and a uniform matroid P.

Algorithm *up_to_weight*$(w_i, P = (E, \mathcal{I}, w))$

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.

Increase to w_i the weights of the elements in S.

return (*rate* of S, cost of weight increases)

LEMMA 7.2. *Algorithm up_to_weight finds in linear time the cost of the weight increases made by $\delta(w_i)$, and the rate of a set of smallest rate according to the increased weights.*

Proof. Clearly, the algorithm runs in time linear on the number of elements in E. To show that the algorithm 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 $E_{<w_n}$ might have their weights increased by *uplift*. 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 are increased as indicated by $\delta(w_i)$. Let S be the largest set of smallest *rate* according to the new weights, and r_S be its *rate*. All elements in $E_{\leq w_i} - S$ have cost no smaller than r_S, because if any $e \in E_{<w_n}$ has $c(e) < r_S$ then $\{e\}$ would be the set of smallest *rate* in P, and if some $e \in E_{=w_n} - S$ has $c(e) < r_S$ then $S \cup \{e\}$ would be a larger set of smallest *rate* in P. Hence, if we assume the same weight for all elements in $E_{\leq w_i}$, then set S would be the set of smallest *rate* in it. $\square$

Algorithm *up_to_rate*,described below, receives as input a *rate* r_i and a uniform matroid P.

Algorithm *up_to_rate*$(r_i, P = (E, \mathcal{I}, w))$

$T \leftarrow \emptyset$;   *upper* $\leftarrow \infty$

$P' = (E', \mathcal{I}', w') \leftarrow P$; $n' \leftarrow$ rank of P'

repeat

  Find the weight $\bar{w}$ of the $\lceil \frac{1}{2}|E| \rceil$th smallest element in E.

$(r', c') \leftarrow up\_to\_weight(\bar{w}, P')$

if $r' \geq r_i$ **then**

$E' \leftarrow E'_{<\bar{w}}$; $n \leftarrow \min\{n', |E'_{<\bar{w}}|\}$; $upper \leftarrow \bar{w}$

else

$T \leftarrow T \cup \{e \mid w(e) \leq \bar{w}$ and $c(e) < r_i\}$

Let $S = \{e \mid w(e) \leq \bar{w}$ and $c(e) \geq r_i\}$.

$E' \leftarrow E'_{>\bar{w}}$; $n \leftarrow \min\{n' - |S|, |E'_{>\bar{w}}|\}$

end if

until $|E'| = 0$

$c_T \leftarrow$ cost of increasing the weights of the elements in T to *upper*.

Let e be the nth smallest element in E

if $e \in T$ **then** $w_n \leftarrow upper$ **else** $w_n \leftarrow w(e)$ **end if**

return $(c_T,\ w_n)$

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 $\delta'(r_i)$. This reduction in the size of E changes the rank of P to $\max\{n, |E_{<\bar{w}}|\}$ since P is restricted to $E_{\bar{w}}$. If $\bar{w}$ is too small, *up_to_rate* tries a larger probe value, but first it reduces the size of E. Since *up_to_rate* stops before it selects a set of *rate* at least r_i, all elements of cost at least r_i and weight at most $\bar{w}$ will not have their weights increased by *up_to_rate*, and so, they can be ignored, i.e. the set S can be contracted from P. It is necessary to adjust the rank n of the matroid to reflect the fact that our algorithm will not consider these elements any more. Since S is independent in P the rank of the matroid decreases to $n - |S|$. Note that all elements of cost smaller than r_i and weight at most $\bar{w}$ will have their weights increased at some point by *up_to_rate*. Hence, instead of keeping track of all the individual weight increases of these elements, *up_to_rate* stores them in a set T. When *up_to_rate* determines the maximum weight increase that those elements should have, it performs the increases in a single step.

The algorithm maintains the invariant that the value of the variable *upper* is an upper bound on the maximum weight $\breve{w}$ that *uplift* can assign to any element of E just before it selects a set of *rate* at least r_i. In each iteration of the repeat-loop, either the value of *upper* is decreased to $\bar{w}$, or it is discovered that $\bar{w}$ is a lower bound for $\breve{w}$. The gap between upper and lower bounds for $\breve{w}$ decreases in each iteration of the repeat-loop. When the loop end, the value of *upper* is equal to $\breve{w}$.

LEMMA 7.3. *Algorithm up_to_rate finds in linear time the cost of the optimal weight increases determined by $\delta'(r_i)$, and the weight of the nth smallest element according to these increased weights.*

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

7.2 Evaluating the Robustness Function of a Partition Matroid

We turn our attention now to the problem of evaluating in linear time $F_P(b)$ for a partition matroid P in which $\ell > 1$. This problem is more difficult than for the case when $\ell = 1$ since we have to determine simultaneously how the weights of the elements change in all the blocks E_i. As for the case of $\ell = 1$, we compute $F_P(b)$ by a prune-and-search process that combines searches on weights with searches on costs. However, in this case each iteration the algorithm reduces the size E by only a fraction of one tenth. 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 uses an array *Upper* of size ℓ, and it stores in *Upper*(i) an upper bound on the maximum weight that any element of E_i can be assigned. Each entry of *Upper* is initialized to ∞. The algorithm is the following.

Algorithm *robustness_partition*$(P = (E, \mathcal{I}, w), b, \mathit{Upper})$
if $|E| = 1$ **then**
  Let $E = \{e\}$. Set $w(e) \leftarrow w(e) + b/c(e)$.

return (weight function w)

else

for $i = 1, 2, \ldots, \ell$ **do**

Compute w_{t_i}, the weight of the $\lceil \frac{4}{5} 4|E_i| \rceil$th smallest element in E_i.

$(c_i, r_i) \leftarrow$ *up_to_weight* $(w_{t_i}, P_i = P|E_i)$

end for

Compute $\bar{r}$, the weighted median of the r_i's using, for each i, $|E_i|$ as the weight for r_i.

for $i = 1, 2, \ldots, \ell$ **do** $(c'_i, t'_i) \leftarrow$ *up_to_rate* $(\bar{r}, P_i)$ **end for**

if $\sum_{i=1}^{\ell} c'_i \geq b$ **then**

for $i = 1, 2, \ldots, \ell$ **do**

$E_i \leftarrow E_i - \{e \mid w(e) \geq t'_i\}$

$Upper(i) \leftarrow t'_i$

end for

return (*robustness_partition* $(P|E, b)$)

else

$S = \cup_{\{i | r_i \leq \bar{r}\}} \{e \mid e \in E_i \text{ and } w(e) < t'_i\}$

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* $(\bar{c}, P_i)$ **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**

return (*robustness_partition* $(P|E, b)$)

else

$(P' = (E', \mathcal{I}', w'), b') \leftarrow$ *contract_partition* $(P, b, \sum_{i=1}^{\ell} \tilde{c}_i, \{\tilde{w}_{n_i}, \ldots, \tilde{w}_{n_\ell}\}, \bar{c}, Upper)$

if $b' = 0$ **then return** (weight function w')

else return (*robustness_uniform* (P', b')) **end if**

end if

end if

end if

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. Note also that the probe values w_{t_i} and $\bar{c}$ are different from the corresponding probe values chosen by *robustness_uniform*. These values were chosen to ensure that each call to *robustness_partition* decreases the size of E by at least the same fraction. Algorithm *contract_partition* is described below.

Algorithm *contract_partition* $(P, b, \tilde{c}, \{\tilde{w}_{n_1}, \ldots, \tilde{w}_{n_\ell}\}, \bar{c}, Upper)$

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 inner for-loop **end if**

    end for

    if $b = 0$ **then** Exit for-loop.

    else $E_i \leftarrow (E_i - S_i - T_i) \cup \{e'\}$, meta-element e' has $w(e') = \tilde{w}_{n_i}$, and $c(e') = c(S_i \cup T_i)$.

    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 for-loop.

      else $E \leftarrow E - E_i$ **end if**

    end if

  end if

end for

Let P' be the matroid corresponding to the new set E and weight function w.

return (P', b)

Algorithm *contract_partition* 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 of the sets E_i has only one element, then it cannot be further contracted. In this case, *contract_partition* does the following. If the element $e \in E_i$ has cost at most $\bar{c}$, then *contract_partition* tries to increase its weight to $Upper(i)$. If the budget is sufficient to increase the weight of e to $Upper(i)$, then the algorithm discards block E_i from E. This can be done, since the weight of e cannot be increased above $Upper(i)$, and hence, *robustness_partition* does not have to consider it any more. But, if the remaining budget is too small to perform the 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 7.1. *Given a partition matroid $P = (E, \mathcal{I}, w)$ 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 $\bar{r}$ computed by *robustness_partition* after the first for-loop, is such that the weight increases defined by $\delta'(\bar{r})$ 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 the cost of the weight increases is smaller than 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 at all the size of E_i (since it tries to contract the element of cost at most $\bar{c}$ to a meta-element). Since $\bar{c}$ is the $\lceil \frac{3}{4}|S| \rceil$ 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. $\square$

8 Conclusions

We have defined the robustness function of a matroid, which measures the sensitivity of the weight of the minimum weight bases of a matroid to increases on the weights of the elements in the matroid. This function generalizes the concept of sensitivity analysis for the minimum weight bases of a matroid. The robustness function of a matroid can be used to assess the quality of the solution of a problem with uncertainties or expected changes in its input.

We have presented an $O(m^5 n^2 + m^4 n^4 \tau)$ general algorithm for computing the robustness function of a matroid. We have also identified key properties of transversal, scheduling, and partition matroids, and exploited them to design robustness algorithms that are more efficient than our general algorithm. All of these algorithms compute the whole robustness function of a matroid. We have also studied the problem of evaluating the robustness function of a matroid only at a given point. We have designed an optimal algorithm for this version of the problem for partition matroids.

Directions for future work include the design of more efficient algorithms, than the ones presented here, for this latter version of the problem for transversal and scheduling matroids. It would also be interesting to find other special matroids for which it is possible to design robustness algorithms that are more efficient that our general algorithm.

An interesting open question worth investigating is to determine whether the techniques presented here can be used to design robustness algorithms for non-matroid optimization problems.

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] 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.

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

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

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

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

[8] J. Edmonds, *Submodular functions, matroids and certain polyhedra*, in Combinatorial Structures, R.K. Guy et al. (Editors), Gordon and Beach, New York 1970, pp . 69–87.

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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