Interdiction of minimum spanning trees and other matroid bases

Noah Weninger and Ricardo Fukasawa

Department of Combinatorics & Optimization, University of Waterloo, Canada

{nweninger,rfukasawa}@uwaterloo.ca

July 16, 2024

Abstract

In the minimum spanning tree (MST) interdiction problem, we are given a graph $G = (V, E)$ with edge weights, and want to find some $X \subseteq E$ satisfying a knapsack constraint such that the MST weight in $(V, E \setminus X)$ is maximized. Since MSTs of G are the minimum weight bases in the graphic matroid of G, this problem is a special case of *matroid interdiction* on a matroid $M = (E, \mathcal{I})$, in which the objective is instead to maximize the minimum weight of a basis of M which is disjoint from X. By reduction from 0-1 knapsack, matroid interdiction is NP-complete, even for uniform matroids.

We develop a new exact algorithm to solve the matroid interdiction problem. One of the key components of our algorithm is a dynamic programming upper bound which only requires that a simpler discrete derivative problem can be calculated/approximated for the given matroid. Our exact algorithm then uses this bound within a custom branch-and-bound algorithm.

For different matroids, we show how this discrete derivative can be calculated/approximated. In particular, for partition matroids, this yields a pseudopolynomial time algorithm. For graphic matroids, an approximation can be obtained by solving a sequence of minimum cut problems, which we apply to the MST interdiction problem. The running time of our algorithm is asymptotically faster than the best known MST interdiction algorithm, up to polylog factors. Furthermore, our algorithm achieves state-of-the-art computational performance: we solved all available instances from the literature, and in many cases reduced the best running time from hours to seconds.

1 Introduction

In an *interdiction problem*, we seek to solve some combinatorial optimization problem on a discrete structure, but there is an adversary that can interdict (i.e., destroy) some parts of the structure before the optimization problem is solved [26]. The adversary aims to make the optimal solution value of the optimization problem as bad as possible. Typically, for each part of the structure that is interdicted, the adversary has to pay a price, and they have a limited budget (i.e., they are restricted by a knapsack constraint).

Interdiction has been studied since the 1950s, and was originally motivated by military applications, but is now seen to be widely applicable. For instance, the minimum s-t cut problem can be seen as the problem of interdicting s-t connectivity in a graph by deleting edges; this now-ubiquitous problem originated in a 1955 US Air Force secret report with the objective of disrupting the Soviet railway system [24]. Another commonly studied interdiction problem is shortest s-t path interdiction, in which the adversary tries to increase the shortest s-t path length in a graph as much as possible by deleting edges [13]. Although the discrete structure is often a graph, other examples exist, such as knapsack interdiction [31], in which the adversary deletes items from a knapsack, or matroid interdiction, the focus of this paper.

In recent years, interdiction problems have typically been interpreted as *bilevel integer programming* (BIP) problems, which can be viewed as a generalization of integer programming (IP) to two-round two-player games. In a BIP, there are two IPs, called the upper level and lower level, between which some variables are shared. An optimal solution to a BIP problem can be seen as the optimal first-player strategy in a two player game. In this game, the first player, called the leader, chooses a point X in the upper level feasible region. The second player, called the follower, then picks an optimal solution Y for the lower level IP, for which feasibility and optimality may depend on the choice of X. In the most general form, the feasibility of X can also depend on the choice of Y. The game then ends and the objective function is calculated as a linear function of X and Y. The goal is for the leader to pick an X which maximizes (or minimizes) this objective function. The terms leader and follower are borrowed from the theory of Stackelberg games [28].

The advantage of viewing interdiction problems as BIPs is that a significant amount of work has been written on general purpose algorithms for solving BIPs [16]. However, a key challenge in this approach is that most BIPs are Σ_2^p-hard and hence cannot be modelled as IPs without using an exponential number of constraints and/or variables. So, typically these algorithms reduce BIP to the problem of solving a series of IPs—exponentially many in the worst case. However, while these general purpose methods can solve small problem instances of most interdiction problems, current solvers are typically very poor at identifying and exploiting the (abundant) structure present in interdiction problems. As such, performance improvements of many orders of magnitude are typically seen when switching from general purpose BIP algorithms to problem-specific interdiction algorithms. This is in contrast to many classical combinatorial optimization problems, where general purpose IP solvers are often very competitive.

Matroid interdiction is only NP-hard (as opposed to Σ_2^p-hard), but there is a lack of algorithms for solving it which are fast in practice. The objective of this work is to further progress our understanding of the structure present in interdiction problems by the development of problem-specific algorithms for matroid interdiction. We hope that these insights can later be integrated into general purpose BIP solvers.

1.1   Problem statement

We assume the reader already has basic familiarity with matroids; the below definitions serve primarily to standardize notation. For a relevant introduction to matroids, see Chapter 8 in [7]. A matroid $M = (E, \mathcal{I})$ consists of a set E, called the *ground set*, and $\mathcal{I}$, a family of subsets of E, called the *independent sets*. If $J \in \mathcal{I}$, we say J is *independent*. For M to be a matroid, we must have:

1. $\emptyset \in \mathcal{I}$,

2. for all $A \in \mathcal{I}$ and $B \subseteq A$, $B \in \mathcal{I}$ (i.e., $\mathcal{I}$ is *downwards closed*), and

3. if A and B are maximal independent subsets of a set $J \subseteq E$, then $|A| = |B|$.

We call maximal sets in $\mathcal{I}$ *bases*. We work in the oracle model: we assume we have a function which, given a set $J \subseteq E$, answers whether $J \in \mathcal{I}$ in a single unit of time. Due to these properties, a minimum (respectively maximum) weight basis can be found using a greedy algorithm which adds elements in order of non-decreasing (non-increasing) weight whenever doing so retains independence [7]. For any $A \subseteq E$, we define the *rank* of A to be $r(A) := \max\{|B| : B \subseteq A, B \in \mathcal{I}\}$. Note that B is a basis if and only if $r(B) = r(E)$.

In the paper we refer explicitly to three types of matroids: uniform, partition, and graphic. The independent sets $\mathcal{I}$ of a *uniform* matroid on a ground set E are defined by $\mathcal{I} = \{S \subseteq E : |S| \leq k\}$ for some given k. Partition matroids generalize uniform matroids: they are defined by a partition $E_1 \cup E_2 \cup \cdots \cup E_\ell = E$ and integers $k_1, k_2, \ldots, k_\ell$; the independent sets are $\mathcal{I} = \{S \subseteq E : |S \cap E_i| \leq k_i \ \forall i \in \{1, \ldots, \ell\}\}$. The *direct sum* of two matroids $M_1 = (E_1, \mathcal{I}_1)$ and $M_2 = (E_2, \mathcal{I}_2)$ where $E_1 \cap E_2 = \emptyset$ is a matroid $M = (E, \mathcal{I})$ where $E = E_1 \cup E_2$ and $\mathcal{I} = \{S : S \cap E_1 \in \mathcal{I}_1 \text{ and } S \cap E_2 \in \mathcal{I}_2\}$. All uniform matroids are partition matroids, but unlike uniform matroids, the class of partition matroids is closed under direct sum. Given an undirected graph $G = (V, E)$, the *graphic* matroid $M = (E, \mathcal{I})$ defined on G has $\mathcal{I} = \{S \subseteq E : S \text{ is acyclic}\}$. Throughout the paper, we assume that G is connected. So, the bases of the graphic matroid are the maximal acyclic subsets of edges of G, i.e., they are spanning trees.

Let $M = (E, \mathcal{I})$ be a matroid and let $m = |E|$. We assume throughout the paper that $E = \{1, 2, \ldots, m\}$. An instance of the matroid interdiction problem is defined by M, a weight vector $w \in \mathbb{Z}^m$, a cost vector $c \in \mathbb{Z}_{\geq 0}^m$, and a capacity $C \in \mathbb{Z}_{\geq 0}$. Let $\mathcal{B} = \{A \subseteq E : r(A) = r(E)\}$ be the set of bases of M. For convenience, we write $x(S) = \sum_{i \in S} x_i$ for a vector x and set S. The matroid interdiction problem can then be stated as follows.

$$\text{OPT} = \max_{X \in \mathcal{U}} \min_{Y \in \mathcal{L}(X)} w(Y) \tag{MI}$$

$$\text{where } \mathcal{U} = \{X \subseteq E : c(X) \leq C\}, \quad \text{(upper level)}$$
$$\text{and } \mathcal{L}(X) = \{Y \subseteq E \setminus X : Y \in \mathcal{B}\}. \quad \text{(lower level)}$$

We often say that the upper-level variables X are selected by a decision maker called the *leader*, and the lower-level variables Y are selected by another decision maker called the *follower*. We call this the *max-min variant* of the problem; the *min-max variant* swaps both max for min and min for max in the objective function. If we are concerned with exact solutions, the choice of working with the max-min variant is arbitrary because negating the weights switches between the two variants:

$$\arg\max\{\min\{w(Y) : y \in \mathcal{L}(X)\} : X \in \mathcal{U}\} = -\arg\min\{\max\{-w(Y) : Y \in \mathcal{L}(X)\} : X \in \mathcal{U}\}. \tag{1.1}$$

It is easily seen that we may assume $w \geq 1$ by adding a sufficiently large constant to all weights, and we may assume $c \geq 1$ because any element e with $c_e = 0$ can be deleted without affecting the optimal solution.

As is typical in bilevel programming, we say that a solution (X, Y) is *feasible* if $X \in \mathcal{U}$ and $Y \in \arg\min\{w(Y) : Y \in \mathcal{L}(X)\}$, that is, to have bilevel feasibility Y must be optimal for the lower-level problem. Since the minimum weight basis of a matroid can be found in polynomial time by the greedy algorithm, feasibility can be verified in polynomial time, and hence the matroid interdiction decision problem is in NP.

We adopt some standard terminology from matroid theory: a matroid which cannot be written as a direct sum of two nonempty matroids is called *connected*, and a maximal connected submatroid of a matroid is called a *component* of that matroid. So, if C_1 and C_2 are components of a matroid M, I_1 is independent in C_1 and I_2 is independent in C_2, then $I_1 \cup I_2$ is independent in M. Throughout the paper, we assume that

$$\text{if } i \text{ and } j \text{ are elements in the same component of a matroid and } i < j, \text{ then } w_i \leq w_j. \qquad \text{(A1)}$$

While there may be multiple minimum weight bases of a matroid, given that $E = \{1, \ldots, m\}$, there is always a unique *lexicographically smallest basis*, and if (A1) is satisfied, then by the greedy algorithm the lexicographically smallest basis is of minimum weight. We assume that the ordering of matroid elements satisfies (A1) throughout the paper. Given a matroid $M = (E, \mathcal{I})$, let $\mathcal{B}_M^{\min}(X)$ be the lexicographically smallest basis $B \subseteq E \setminus X$ of M. This function is only defined for those X where there exists a basis $B \subseteq E \setminus X$. We may omit the subscript M when it is implied by the context.

Given the lexicographically smallest basis B of a matroid M and an element $e \in B$, we say that the *replacement element* for e (if one exists) is the element e' such that $\{e'\} = B' \setminus B$ where B' is the lexicographically smallest basis of $M \setminus \{e\}$. Note that the replacement element can be found by the greedy algorithm.

A closely related problem to matroid interdiction is the *minimum cost matroid blocker problem*, which can be formulated as follows.

$$\min_{X \subseteq E} c(X)$$

$$\text{s.t. } \left[\min_{Y \in \mathcal{L}(X)} w(Y) \right] \geq R$$

Here, R is a given constant which can be though of as the *target weight*. It is easy to see that these two problems are polynomial-time equivalent by performing binary search on C or R. We focus on the matroid interdiction problem for the majority of the paper, but in Section 3.3 we show that our algorithm can easily be adapted to directly solve the minimum cost blocker problem without incurring the run time overhead of binary search.

1.2 Prior work

The simplest and earliest-studied special case of matroid interdiction is the *most vital MST edge* problem, which was introduced by Hsu et al in 1991 [12]. In this variant we wish to find a single edge e which maximizes the MST weight of $G \setminus \{e\}$. This is the special case of matroid interdiction where M is graphic, $c_e = 1$ for all $e \in E$, and $C = 1$. A long line of work led to a recent algorithm which solves this problem in time $O(m \log m + n)$, or $O(m + n)$ if the edges are assumed to already be sorted by weight [2]. A few works have studied variants of the most vital matroid element problem: one paper performs sensitivity analysis on minimum weight matroid bases [18], another studies the variant where the leader can increase the weights of the matroid elements by paying a cost per unit of weight increased [8], and a very recent paper concerns parametric matroid interdiction [10].

A natural generalization of this is the k-most vital MST edges problem, in which M is graphic, $c_e = 1$ for all $e \in E$ and $C = k$ for some constant k. NP-hardness of the k-most vital edges problem was shown in [9] by reduction from minimum k-cut. Later, this hardness result was extended to show NP-hardness for the variant of the problem where vertices are interdicted instead of edges [4]. The same paper also includes various approximation algorithms and inapproximability results. Since the problem is only NP-hard when k is part of the input, an interest developed in algorithms which run in polynomial time for fixed k [25]. After a series of papers, a running time of $O(n^k \log \alpha((k-1)(n+1), n))$ was achieved [3,17]. Most recently, new mixed integer programming formulations and branch-and-bound algorithms were proposed for the k-most vital MST edges problem which have good performance in computational tests for small k (i.e., $k \leq 5$ on instances with hundreds of vertices) [3]. We compare our results to this paper in Section 4.

The first paper to study MST interdiction (then called the *most vital edges in the MST* problem) in full generality presents a simple branch-and-bound algorithm [19]. To the best of our knowledge, this algorithm was never

evaluated computationally. A later paper also established an $O(\log n)$-approximation algorithm [9]. This was subsequently improved to a 14-approximation [32], and eventually a 4-approximation [20]. The paper which shows a 4-approximation also proves the surprising result that the *maximum* spanning tree interdiction problem cannot be approximated to any constant factor under the small set expansion hypothesis. However, a 2-pseudo-approximation (a polytime algorithm that either returns a 2-approximate solution or a solution that exceeds the budget by a factor of 2) is known for min-max matroid interdiction, and hence for maximum spanning tree interdiction [6]. In fact, this pseudo-approximation even works for more general case of interdicting the maximum independent set weight, rather than just the maximum basis weight, as well as for a wide variety of other interdiction problems.

In 2024, Luis Salazar-Zendeja published their PhD thesis on MST interdiction, which proposes and computationally evaluates a variety of MIP-based approaches [23]. The computational results are not compared to prior works, but the largest instances considered (in terms of number of vertices/edges and capacity) are smaller than those studied in previous papers, and the running times are of the same order of magnitude. Important contributions of the thesis include the application of classical but previously unexplored techniques including branch-and-price and Bender's decomposition, and the generalization of all results to the partial interdiction variant of the problem, in which the leader can only slightly increase the weight of MST edges rather than block them entirely.

The MST edge blocker problem is the focus of [30], which studies various integer programming formulations for the problem and evaluates their performance computationally. We compare our results to this paper in Section 4. A follow-up paper generalizes the theoretical aspects of this work to greedoids, among other problems [29].

A number of closely related problems have been studied. One is the *rank reduction* problem, where we wish to find a minimum cardinality subset of the elements of a matroid whose deletion reduces the rank by at least k [14]. Although many parallels can be drawn between the known hardness results for rank reduction and matroid interdiction, we are not aware of any work which connects them explicitly. Another recent paper studies the interdiction problem where both the upper and lower level feasible regions are given by the independent sets of partition matroids over a common ground set [15]. Finally, our recent work [31] on knapsack interdiction also uses branch-and-bound with bounds computed via dynamic programming, and achieves state-of-the-art computational results. The success of those techniques motivated our approach in this work.

1.3 Our contributions

In Section 2, we introduce a new framework for computing upper bounds for interdiction problems. We apply this framework to exactly solve the partition matroid interdiction problem, and derive an upper bound for MST interdiction. In Section 3 we present an exact branch-and-bound algorithm for solving matroid interdiction utilizing the bounds derived in the previous section. We prove that even without the upper bounds, this algorithm matches (up to polylog factors) or improves upon the best known asymptotic running time of an enumerative algorithm for MST interdiction with unit costs. Furthermore, our algorithm is more general as it applies to any matroid interdiction problem. We also introduce a strong heuristic lower bound to initialize the branch-and-bound search. In Section 4 we compare an implementation of our algorithm in computational tests to previously published algorithms. We find that our algorithm is significantly faster than previous approaches, often by a few orders of magnitude. We also generate some new problem instances, which we use to investigate what qualities make an instance hard to solve, and how the various features of our solver contribute to its performance. Finally, in Section 5 we investigate the problem variant in which the leader can force elements to be included in every basis, rather than excluded.

2 Upper bounds

In this section, we describe a general framework for upper bounding interdiction problems with knapsack-like upper level feasibility sets. The key idea is to reduce the interdiction problem to a type of knapsack problem where the item profits bound the change in the lower level objective function when a given element is interdicted. We demonstrate that for many matroids, when the lower level problem is to find the minimum basis weight, this discrete-derivative-like problem is easier to solve than the full interdiction problem and yields good bounds or even exact solutions when integrated with dynamic programming (DP). This way of computing upper bounds also is ideally suited for use in a branch-and-bound scheme, because the bound can be precomputed for every branch-and-bound node and accessed in $O(1)$ time. Our exact branch-and-bound algorithm for matroid interdiction which uses these bounds is described in Section 3.

We assume for now that the set of feasible interdictions is $\mathcal{U} = \{X \subseteq E : c(X) \leq C\}$, as described in Section 1, although later we argue that this can be relaxed somewhat. From this point until the start of Section 2.1, the only assumption that we make about $\mathcal{L}(X)$ is that if $Y \in \mathcal{L}(X)$, then $Y \subseteq E \setminus X$. Let $F(X) = \min\{w(Y) : Y \in \mathcal{L}(X)\}$

be the optimal solution to the follower's problem induced by a given interdiction set $X \in \mathcal{U}$, or ∞ if the follower's problem is infeasible for the given X. The value we desire to upper bound is $\text{OPT} = \max\{F(X) : X \in \mathcal{U}\}$.

For a set $X \subseteq \mathbb{Z}$, let $X_{\leq k}$ be the set $\{j \in X : j \leq k\}$, and similarly for $X_{\geq k}$, $X_{<k}$, etc. Given any $\hat{X} \in \mathcal{U}$ and $i \in E$, we can express $\max\{F(X) : X \in \mathcal{U}, X_{<i} = \hat{X}\}$ as $F(\hat{X}) + \max\{F(X) - F(\hat{X}) : X \in \mathcal{U}, X_{<i} = \hat{X}\}$. In this section, we show that this view offers a clear advantage: upper bounds on $\max\{F(X) - F(\hat{X}) : X \in \mathcal{U}, X_{<i} = \hat{X}\}$ can be computed for all $i \in E$ and $\hat{X} \in \mathcal{U}$ as the entries of a DP table. Moreover, this structure will be exploited in our branch-and-bound algorithm (presented in Section 3), in which branch-and-bound nodes are identified by $\hat{X}$ and i.

Before we present our DP algorithm, we comment that we formalize only the essential aspects that are needed for such description, for the sake of generality. In particular, we choose to be informal regarding some of the desired aspects, using terms like 'small enough' or 'large enough', since these are guiding principles that need to be made concrete depending on the specific lower level problem.

The dynamic program will be over a set of possible states T and requires mapping each $X \in \mathcal{U}$ to some *state* $s \in T$ by a function $\phi(X)$. The goal is for $|T|$ to be small enough that we can use $s \in T$ to index the DP table (without using too much memory), but large enough that $\phi(\hat{X})$ captures enough information about some $\hat{X} \in \mathcal{U}$ to be able to compute a good bound for $\max\{F(X) - F(\hat{X}) : X \in \mathcal{U} : X_{<i} = \hat{X}\}$. We also define a function $\pi(i, s)$ which describes how a state $s \in T$ transitions to a new state if element i is interdicted when in state s. We formalize this property below as P_{succ}.

Finally, we need a function $\delta(i, r, s)$ which upper bounds how much the follower's objective can increase by if element i is interdicted with r knapsack capacity remaining when in state s. We formalize this property below as P_{bound}. The assumption that this function can be computed faster than solving the entire interdiction problem is key to our approach. We devote most of the second half of this section to defining a suitable δ for various matroids.

Formally, these functions have the following signatures:

$$\phi : \mathcal{U} \to T$$
$$\pi : E \times T \to T$$
$$\delta : E \times \{0, 1, \ldots, C\} \times T \to \mathbb{Z}$$

These will be defined precisely later, depending on the specifics of the follower's problem. We now formally define the two conditions that these functions should satisfy to be able to apply our results. In the below definitions, $\mathcal{U}$ and T are implicit based on the definitions of the given functions. Condition $P_{\text{bound}}(\delta, \phi, \pi)$ is satisfied if and only if for all $i \in E$, $r \in \{0, 1, \ldots, C\}$ and $s \in T$,

$$c_i \leq r \implies \delta(i, r, s) \geq \max \begin{cases} F(X_{<i} \cup \{i\}) - F(X_{<i}) \\ \text{s.t. } X \in \mathcal{U}, c(X_{<i}) = C - r, \phi(X_{<i}) = s \end{cases}.$$

This condition is the most important since it ultimately defines what the upper bound is. The other conditions and functions are defined to make sure that DP transitions from one state to another are consistent and can be computed for all possible states. The next condition, $P_{\text{succ}}(\phi, \pi)$, is satisfied if and only if for all X such that $X \cup \{i\} \in \mathcal{U}$,

$$\phi(X_{<i} \cup \{i\}) = \pi(i, \phi(X_{<i})).$$

This condition intuitively says that transitioning from the state $\phi(X_{<i})$ by interdicting i should lead us to the state $\phi(X_{<i} \cup \{i\})$. We now define a dynamic program $f : \{1, \ldots, m + 1\} \times \{0, \ldots, C\} \times T \to \mathbb{Z}$.

$$f(i, r, s) = \begin{cases} 0 & \text{if } i > m, \\ f(i + 1, r, s) & \text{if } i \leq m \text{ and } c_i > r, \\ \max \begin{cases} f(i + 1, r, s), \\ f(i + 1, r - c_i, \pi(i, s)) + \delta(i, r, s) \end{cases} & \text{otherwise.} \end{cases}$$

Note that this is a generalization of the standard Bellman recursion for 0-1 knapsack (e.g., see Section 2.3 of [21]); the only difference is that we keep track of additional state s, which is managed by the update function π, and that the item profit values $\delta(i, r, s)$ are a function of both i, r, and s, as opposed to only i. For example, suppose we wish to solve a knapsack problem with costs c_i, capacity C and profits p_i. Then we can simply define $T = \{\epsilon\}$, $\phi(X) = \epsilon$, $\pi(i, \epsilon) = \epsilon$, and $\delta(i, r, \epsilon) = p_i$ (where ϵ is just a null element used to ensure T is nonempty); then $f(1, C, \epsilon)$ will solve the knapsack problem.

The following theorem formalizes the above intuition using this definition. It follows immediately that OPT $\leq$ $f(1, C, \phi(\emptyset)) + F(\emptyset)$ by applying the theorem with $i = 1$ and $\hat{X} = \emptyset$. To simplify notation in the next two proofs we define $\Phi(X) = (C - c(X), \phi(X))$ and $\Pi(i, (r, s)) = (r - c_i, \pi(i, s))$. The intuition for this is that the remaining knapsack capacity $r = C - c(X)$ and the state $s = \phi(X)$ can both be treated as parts of a larger state $\Phi(X)$ with transition function Π. Then, it is easy to see that for all X such that $X \cup \{i\} \in \mathcal{U}$,

$$\Phi(X_{<i} \cup \{i\}) = \Pi(i, \Phi(X_{<i})).$$

Some slight abuse of notation aside, this is equivalent to saying that $P_{\text{succ}}(\Phi, \Pi)$ holds, assuming that $P_{\text{succ}}(\phi, \pi)$ does. By doing this, we do not need to individually reason about both r and s in the proof, and can instead treat them as a unit (r, s). To simplify notation we also interchangeably use $f(i, r, s)$ and $f(i, (r, s))$, and similarly for δ and π.

Theorem 2.1. *Given an upper-level feasible set $\mathcal{U}$, state set T, and functions δ, ϕ, and π satisfying $P_{bound}(\delta, \phi, \pi)$ and $P_{succ}(\phi, \pi)$, we have that for all $i \in E$ and $\hat{X} \in \mathcal{U}$,*

$$f(i, \Phi(\hat{X}_{<i})) \geq \max\{F(X) - F(X_{<i}) : X \in \mathcal{U}, X_{<i} = \hat{X}_{<i}\}.$$

Proof. We prove by induction on i from $m + 1$ to 1 that for all $r \in \{0, 1, \ldots, C\}$ and $s \in T$,

$$f(i, r, s) \geq \max\left\{F(X) - F(X_{<i}) : X \in \mathcal{U}, \Phi(X_{<i}) = (r, s)\right\}. \tag{2.1}$$

Since $X_{<i} = \hat{X}_{<i}$ implies $\Phi(X_{<i}) = \Phi(\hat{X}_{<i})$, this proves the theorem statement by taking $(r, s) = \Phi(\hat{X}_{<i})$:

$$(2.1) \geq \max\left\{F(X) - F(X_{<i}) : X \in \mathcal{U}, X_{<i} = \hat{X}_{<i}\right\}.$$

For the base case of the induction, suppose that $i = m + 1$. Since $E \setminus X = E \setminus X_{<m+1}$ for any $X \subseteq E$, we have $F(X) = F(X_{<i})$ and hence $f(i, r, s) = 0$ satisfies Eq. (2.1). Now assume $1 \leq i \leq m$ and that Eq. (2.1) holds for $i + 1$ with arbitrary $r \in \{0, \ldots, C\}$ and $s \in T$. First suppose that $c_i > r$. Then, applying the induction hypothesis (Eq. (2.1)),

$$f(i, r, s) = f(i + 1, r, s) \geq \max\left\{F(X) - F(X_{<i+1}) : X \in \mathcal{U}, \Phi(X_{<i+1}) = (r, s)\right\}$$

Since $c_i > r$ we have that $i \notin X$ for any $X \in \mathcal{U}$ with $C - c(X_{<i}) = r$. Therefore, for any such X, $X_{<i+1} = X_{<i}$, so we have $\Phi(X_{<i+1}) = (r, s)$. Hence, the result follows. Now assume $c_i \leq r$. We claim that

$$f(i + 1, \Pi(i, r, s)) + \delta(i, r, s) \geq \max\left\{F(X) - F(X_{<i})) : i \in X \in \mathcal{U}, \Phi(X_{<i}) = (r, s)\right\}. \tag{2.2}$$

To see this, first observe that by the induction hypothesis and $P_{\text{bound}}(\delta, \phi, \pi)$,

$$f(i + 1, \Pi(i, r, s)) + \delta(i, r, s)$$
$$\geq \max\left\{F(X) - F(X_{<i+1}) : X \in \mathcal{U}, \Phi(X_{<i+1}) = \Pi(i, r, s)\right\} \tag{2.3}$$
$$+ \max\left\{F(X_{<i} \cup \{i\}) - F(X_{<i}) : X \in \mathcal{U}, \Phi(X_{<i}) = (r, s)\right\}. \tag{2.4}$$

Now, by taking the intersection of the feasible sets X in Eqs. (2.3) and (2.4) and adding the objectives, we get that

$$(2.3) + (2.4) \geq \max\left\{\begin{matrix} F(X) - F(X_{<i+1}) + F(X_{<i} \cup \{i\}) - F(X_{<i}) \\ \text{s.t. } X \in \mathcal{U}, \Phi(X_{<i+1}) = \Pi(i, r, s), \Phi(X_{<i}) = (r, s) \end{matrix}\right\}. \tag{2.5}$$

If $X \in \mathcal{U}$ and $i \in X$, then since $P_{\text{succ}}(\Phi, \Pi)$ holds we have $\Phi(X_{<i+1}) = \Pi(i, \Phi(X_{<i}))$. Hence,

$$(2.5) \geq \max\left\{F(X) - F(X_{<i}) : i \in X \in \mathcal{U}, \Phi(X_{<i}) = (r, s)\right\}. \tag{2.6}$$

This proves the claim. We now complete the inductive step. The following is immediate from the induction hypothesis and Eq. (2.2).

$$f(i, r, s) = \max\{f(i + 1, r, s), f(i + 1, \Pi(i, r, s)) + \delta(i, r, s)\}$$
$$\geq \max\left\{\begin{matrix} \max\left\{F(X) - F(X_{<i+1}) : X \in \mathcal{U}, \Phi(X_{<i+1}) = (r, s)\right\}, \\ \max\left\{F(X) - F(X_{<i}) : i \in X \in \mathcal{U}, \Phi(X_{<i}) = (r, s)\right\} \end{matrix}\right\}. \tag{2.7}$$

Now, by restricting X in the first inner max of Eq. (2.7) to exclude i, we get

$$(2.7) \geq \max \left\{ \begin{matrix} \max \left\{ F(X) - F(X_{<i}) : i \notin X \in \mathcal{U}, \ \Phi(X_{<i}) = (r, s) \right\}, \\ \max \left\{ F(X) - F(X_{<i}) : i \in X \in \mathcal{U}, \ \Phi(X_{<i}) = (r, s) \right\} \end{matrix} \right\}. \tag{2.8}$$

Finally, we can merge the two inner max terms into a single term by noticing that the first considers all X with $i \notin X$ and the second considers all X with $i \in X$, but they are otherwise identical.

$$(2.8) = \max \left\{ F(X) - F(X_{<i}) : X \in \mathcal{U}, \ \Phi(X_{<i}) = (r, s) \right\}.$$

This proves Eq. (2.1), as desired. $\qquad\square$

We will use this theorem in Section 3 to compute upper bounds for our branch-and-bound scheme. A natural question to ask is how loose/tight this upper bound can be. The next theorem gives some answer to that question by showing that if we have additive approximation bound of α for δ, there we have an additive approximation bound of $m\alpha$ for f. We formalize the notion of having an additive approximation bound of α for δ with the condition $P_{\mathrm{approx}}(\delta, \phi, \pi, \alpha)$, which is satisfied if and only if for all $X \in \mathcal{U}$ and $i \in E$,

$$c_i \leq C - c(X_{<i}) \implies \delta(i, C - c(X_{<i}), \phi(X_{<i}))) \leq \alpha + F(X_{<i} \cup \{i\}) - F(X_{<i}).$$

Theorem 2.2. *Suppose we have an upper-level feasible set $\mathcal{U}$, state set T, and functions δ, ϕ, and π. Assume that $P_{succ}(\phi, \pi)$ is satisfied, and that there exists some $\alpha \geq 0$ such that $P_{approx}(\delta, \phi, \pi, \alpha)$ is satisfied. Then, for all $i \in E$ and $\hat{X} \in \mathcal{U}$,*

$$f(i, \Phi(\hat{X}_{<i})) \leq (m - i + 1)\alpha + \max\{F(X) - F(X_{<i}) : X \in \mathcal{U}, \ \Phi(X_{<i}) = \Phi(\hat{X}_{<i})\}.$$

Proof. To prove the theorem, we define a new function f_X in which the outcome of the max operation in f is decided by a special parameter X (rather than actually choosing the maximum).

$$f_X(i, r, s) = \begin{cases} 0 & \text{if } i > m \\ f_X(i + 1, r, s) & \text{if } i \leq m \text{ and } i \notin X \\ f_X(i + 1, \Pi(i, r, s)) + \delta(i, r, s) & \text{if } i \leq m \text{ and } i \in X \end{cases}$$

Let $\hat{X} \in \mathcal{U}$ and $i \in E$. By recursively expanding out the definition of f, we can see that

$$f(i, \Phi(\hat{X}_{<i})) = \sum_{j=1}^{k} \delta(x_j, r_j, s_j)$$

for some k and some sequences $x_1, \ldots, x_k \in E$ and $(r_1, s_1), \ldots, (r_k, s_k) \in \{0, \ldots, C\} \times T$ with $i \leq x_1 < \cdots < x_k$ and $(r_1, s_1) = \Phi(\hat{X}_{<i})$. There may be multiple valid choices of these sequences. We may assume that for all $j \in \{1, \ldots, k\}$,

$$f(x_j + 1, r_j, s_j) \neq f(x_j + 1, \Pi(x_j, r_j, s_j)) + \delta(x_j, r_j, s_j)$$

because if not, there is an alternative choice for the sequences which exclude x_j and (r_j, s_j), respectively. Let $X = \hat{X}_{<i} \cup \{x_1, \ldots, x_k\}$. Then, by definition we have that $f(x_j, r_j, s_j) = f_X(x_j, r_j, s_j)$ for all $j \in \{1, \ldots, k\}$. Intuitively, X encodes the 'decisions' made in $f(i, \Pi(\hat{X}_{<i}))$.

Note that since $(r_1, s_1) = \Phi(\hat{X}_{<i}) = \Phi(X_{<i})$, by $P_{\mathrm{succ}}(\Phi, \Pi)$, $(r_2, s_2) = \Pi(x_1, \Phi(X_{<x_1})) = \Phi(X_{<x_2})$ and inductively for all ℓ,

$$(r_{\ell+1}, s_{\ell+1}) = \Pi(x_\ell, \Phi(X_{<x_\ell})) = \Phi(X_{<x_{\ell+1}}). \tag{2.9}$$

We claim that $X \in \mathcal{U}$. If not, then by Eq. (2.9) and the fact that $\hat{X} \in \mathcal{U}$, there is some j such that $c_{x_j} > C - c(X_{<x_j}) = r_j$. However,

$$x_j \in X \implies f_X(x_j, r_j, s_j) = f_X(x_j + 1, \Pi(x_j, s_j)) + \delta(x_j, r_j, s_j), \text{ and}$$
$$c_{x_j} > r_j \implies f(x_j, r_j, s_j) = f(x_j + 1, r_j, s_j).$$

This contradicts $f(x_j, r_j, s_j) = f_X(x_j, r_j, s_j)$ given our earlier observation that we may assume $f(x_j + 1, r_j, s_j) \neq f(x_j + 1, \Pi(x_j, r_j, s_j)) + \delta(x_j, r_j, s_j)$. Therefore, $X \in \mathcal{U}$. We can now finish the proof.

$$
\begin{aligned}
f_X(i, \Phi(\hat{X}_{<i})) &= \sum_{j=1}^{k} \delta(x_j, r_j, s_j) && \text{(by the definition of } f_X) \\
&= \sum_{j=1}^{k} \delta(x_j, \Phi(X_{<x_j})) && \text{(by Eq. (2.9))} \\
&\leq k\alpha + \sum_{j=1}^{k} F(X_{<x_j} \cup \{x_j\}) - F(X_{<x_j}) && \text{(by } P_{\text{approx}}(\delta, \phi, \pi, \alpha)) \\
&= k\alpha + \sum_{j=1}^{k} F(X_{<x_j+1}) - F(X_{<x_j}) && \text{(by the definition of } X) \\
&= k\alpha + F(X_{<x_k+1}) - F(X_{<x_1}) && \text{(since the sum telescopes)} \\
&= k\alpha + F(X) - F(X_{<i}) && \text{(by the definition of } X) \\
&\leq (m - i + 1)\alpha + \max\{F(X') - F(X'_{<i}) : X' \in \mathcal{U}, \Phi(\hat{X}_{<i}) = \Phi(X'_{<i})\} \\
&&& \text{(since } X \in \mathcal{U}, \hat{X}_{<i} = X_{<i}, \text{ and } k \leq m - i + 1.)
\end{aligned}
$$

$\square$

We remark that while this guarantee may appear to be quite poor in the worst case (losing an additive factor of m), the strength of this approach is that it computes bounds for many subproblems, using only the amount of time required to compute $\delta(i, r, s)$ for each subproblem. These subproblem bounds can be precomputed and efficiently accessed by our branch-and-bound scheme. Furthermore, for graphic matroids, our upper bounds are typically very strong in practice, outperforming other methods such as the LP relaxation of the extended formulation [3,30] or Lagrangian relaxation [20]. For partition matroids, our upper bound actually solves the interdiction problem exactly (i.e., we can take $\alpha = 0$ in Theorem 2.2).

The techniques in this section can be generalized considerably beyond the given setting of matroid interdiction. We already noted that $\mathcal{L}(X)$ does not necessarily need to be the set of bases of a matroid which exclude elements in X; it suffices that an appropriate function δ can be computed. However, $\mathcal{U}$ does not need to be $\{X \subseteq E : c(X) \leq C\}$ either; the only important quality that $\mathcal{U}$ should have is that it should be possible to keep track of whether a set is feasible under additions of elements to the set using a 'small' amount of state (we want it to be 'small' so that memory usage is reasonable). In the above, we used the variable r to do this. As an example generalization, if $\mathcal{U} = \{X \subseteq E : c^1(X) \leq C^1, c^2(X) \leq C^2\}$, then we could use variables $r_1 \in \{0, \ldots, C^1\}$ and $r_2 \in \{0, \ldots, C^2\}$ to keep track of feasibility. The proofs require only trivial modifications.

2.1 Uniform matroids

We start with perhaps the easiest nontrivial case: uniform matroids. Recall that the independent sets of a uniform matroid over E are defined by $\{S \subseteq E : |S| \leq k\}$ for some given k. Intuitively, to determine the replacement element when an element e is interdicted in a uniform matroid, we only need to know the number of elements before e which were interdicted, because independence of a set S depends only on $|S|$. The rest of this section is spent formalizing this intuition and developing an exact algorithm for uniform matroid interdiction.

We now define a set T and the functions δ, π, and ϕ, as discussed in the previous subsection. Let $T = \{0, 1, \ldots, m\}$. We define $\pi : E \times T \to T$ and $\phi : \mathcal{U} \to T$ as follows:

$$
\begin{aligned}
\pi(i, n) &= n + 1, \\
\phi(X) &= |X|.
\end{aligned}
$$

Here, the state T keeps track of the number of interdicted elements, in line with our intuition. It is trivial to see that $P_{\text{succ}}(\phi, \pi)$ is satisfied. The proof of the following theorem leads to our definition of δ and the proof of $P_{\text{bound}}(\delta, \phi, \pi)$.

Theorem 2.3. *Suppose that M is uniform. Then there exists a function $\delta(i, r, n)$ satisfying $P_{approx}(\delta, \phi, \pi, 0)$ with equality, i.e., for all $X \in \mathcal{U}$, $r \in \{0, \ldots, C\}$ and $i \in E$,*

$$
c_i \leq C - c(X_{<i}) \implies \delta(i, C - c(X_{<i}), \phi(X_{<i})) = F(X_{<i} \cup \{i\}) - F(X_{<i}).
$$

8

Proof. Consider some $X \subseteq \{1, \ldots, i - 1\}$ such that $X \cup \{i\} \in \mathcal{U}$ and $\phi(X) = n$. We want to determine $F(X \cup \{i\}) - F(X)$. Let $B = \mathcal{B}^{\min}(X)$ be the lexicographically smallest basis of $E \setminus X$. As discussed in Section 1, the lexicographically smallest basis is always of minimum weight, given our item ordering assumption.

If $i \notin B$, then $F(X \cup \{i\}) = F(X)$ because B is also the lexicographically smallest basis of $E \setminus (X \cup \{i\})$. Otherwise, if $i \in B$, then there are two cases. If there is no $j > i$ with $j \notin B$, then $F(X \cup \{i\}) - F(X) = \infty$ because there is no element that can replace element i in B. Otherwise, if such a j exists, then the minimum such element j replaces element i, so $F(X \cup \{i\}) - F(X) = w_j - w_i$.

We claim that these conditions can be tested for only by knowing i, k and n (recall $n = |X|$). Observe that B, the lexicographically smallest basis of $E \setminus X$, is defined by $B = \{j \in E : j \notin X, \, j \leq k + n\}$ because uniform matroids are connected so, by (A1), the elements are sorted by weight. Let $j = k + n + 1$. There are three cases.

1. If $j \leq i$, then $i \notin B$ because $i \geq j = k + n + 1$. So, we take $\delta(i, r, n) = 0$.

2. If $i < j \leq |E|$, then $i \in B$ because $i \notin X$ and $i < j = k + n + 1$. We have $j \in E \setminus B$ by definition, and for any $i < j' < j$ we know $j' \in B$. So $B \setminus \{i\} \cup \{j\}$ is the lexicographically smallest basis of $E \setminus (X \cup \{i\})$. Hence, we take $\delta(i, r, n) = w_j - w_i$.

3. If $j > |E|$, then again $i \in B$, but there is no $j' > i$ with $j' \in E \setminus B$. So, there is no element that can replace i, and hence we take $\delta(i, r, n) = \infty$. $\qquad\square$

To summarize, $\delta(i, r, n)$ is defined as follows. Note that it does not actually depend on r.

$$\delta(i, r, n) = \begin{cases} 0 & \text{if } k + n < i, \\ w_{k+n+1} - w_i & \text{if } i \leq k + n < |E|, \\ \infty & \text{if } |E| \leq k + n. \end{cases}$$

Property $P_{\text{bound}}(\delta, \phi, \pi)$ follows easily from this. Let $i \in E$ and $s \in T$. If there exists some $X \in \mathcal{U}$ such that $C - c(X_{<i}) = r$ and $\phi(X_{<i}) = s$, then by Theorem 2.3 we have

$$c_i \leq r \implies \delta(i, r, s) = F(X_{<i} \cup \{i\}) - F(X_{<i})$$
$$= \max \left\{ \begin{matrix} F(X'_{<i} \cup \{i\}) - F(X'_{<i}) \\ \text{s.t. } X' \in \mathcal{U}, \, C - c(X_{<i}) = r, \, \phi(X'_{<i}) = s \end{matrix} \right\}.$$

Otherwise, if no such X exists, then the max in $P_{\text{bound}}(\delta, \phi, \pi)$ is taken over the empty set, so the condition is vacuously satisfied.

Since uniform matroids are so simple, we can further show that with δ defined as above, $F(\emptyset) + f(1, C, \phi(\emptyset))$ is not just an upper bound, but actually solves the matroid interdiction problem exactly.

Corollary 2.4. *For uniform M, we have*

$$F(\emptyset) + f(1, \phi(\emptyset)) = \max\{F(X) : X \in \mathcal{U}\} = \text{OPT}.$$

The proof follows immediately from Theorems 2.1 to 2.3. The following is then also immediate from the observation that δ can be computed in $O(1)$ time and $|T| = O(m)$, so the DP table for f has size $O(m^2 C)$.

Corollary 2.5. *Uniform matroid interdiction can be solved in time $O(m^2 C)$.*

Finally, to complement this, we prove that the decision problem version of uniform matroid interdiction is NP-complete, and hence no better running time than pseudopolynomial is possible unless P = NP. We define this decision problem as follows: given an instance of uniform matroid interdiction and an integer t, determine if there is a feasible solution to the uniform matroid interdiction problem of value at least t.

Theorem 2.6. *The uniform matroid interdiction decision problem is weakly NP-complete.*

Proof. The problem is easily seen to be in NP, because determining feasibility requires only checking whether the knapsack constraint is satisfied and determining the minimum basis weight in a uniform matroid.

Given a knapsack instance with n items, costs a, budget C and profits p, we define a uniform matroid $M = (E, \mathcal{I})$ where $E = \{1, 2, \ldots, 2n\}$, $\mathcal{I} = \{S \subseteq E : |S| \leq n\}$, and we define weights and interdiction costs as follows. Let $p_{\max}$ be the largest item profit from p. For $e \in \{1, \ldots, n\}$, set $w_e = p_{\max} - p_e$ and $c_e = a_e$. For $e \in \{n + 1, \ldots, 2n\}$ set $w_e = p_{\max}$ and $c_e = C + 1$. The uniform matroid interdiction instance has the same capacity C as the knapsack. It

9

is easily seen that interdicting any element $e \in \{1, \ldots, n\}$ costs a_e and causes the minimum basis weight to increase by p_e, because the replacement element is always after element n. No elements after n can be interdicted, because they have interdiction cost greater than C. This is exactly the same change to profit and cost which are seen when adding e to the knapsack. So, a set X is an optimal solution for the knapsack problem if and only if X is an optimal solution to the corresponding uniform matroid interdiction problem. $\qquad \square$

In Section 2.2, we extend these results to partition matroids, which are the direct sum of uniform matroids.

2.2   Direct sums of matroids

In this section, we show how to derive an upper bound for the interdiction problem on a matroid $M = (E, \mathcal{I})$ which is the direct sum of two matroids (or more, by induction) for which we have upper bounds. Furthermore, any additive approximation bound for the matroids being summed is preserved. The most immediate consequence of this result is a pseudopolynomial time algorithm for solving the partition matroid interdiction problem. This result is fairly easy to intuitively understand, but a rigorous proof ends up being quite technical due to the desire to keep the number of DP states small.

Let $m = |E|$, and let $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ be matroids. We assume without loss of generality that there exists some p such that if $J \in \mathcal{I}_1$ then $J \subseteq \{1, \ldots, p\}$ and if $J \in \mathcal{I}_2$ then $J \subseteq \{p+1, \ldots, m\}$, i.e., that the matroids are disjoint. We assume that M is represented as follows:

$$E = \{1, \ldots, m\},$$
$$\mathcal{I} = \{J : J \cap \{1, \ldots, p\} \in \mathcal{I}_1 \text{ and } J \cap \{p+1, \ldots, m\} \in \mathcal{I}_2\}.$$

We assume that the subsets $\{1, \ldots, p\}$ and $\{p+1, \ldots, m\}$ of E are individually sorted as necessary for the constructions on the individual matroids. For example, if M_1 and M_2 are both uniform or graphic, then we would have $w_1 \leq w_2 \leq \cdots \leq w_p$ and $w_{p+1} \leq w_{p+2} \leq \cdots \leq w_m$. Notice that this satisfies our element ordering assumption (A1) since there are no two elements $i \in \{1, \ldots, p\}$ and $j \in \{p+1, \ldots, m\}$ which are in the same component of M. Let $F_M(X)$ be the minimum w-weight of a basis B of $E \setminus X$ in M, or ∞ if no such basis exists.

Suppose that T_i, ϕ_i, π_i and δ_i, correspond to M_i for $i \in \{1, 2\}$ and that they satisfy $P_{\text{bound}}(\delta_i, \phi_i, \pi_i)$ and $P_{\text{succ}}(\phi_i, \pi_i)$. In this section we make the additional assumption that for all $S \subseteq \{1, \ldots, p\}$,

$$\phi_2(S) = \phi_2(\emptyset). \tag{2.10}$$

This is a natural assumption to make because elements 1 through p are not contained in any independent set of M_2, and hence should be irrelevant to ϕ_2. This assumption arises because we represent the matroids as having the same ground set, which simplifies the main proof. Let $T = T_1 \sqcup T_2$ (the disjoint union of T_1 and T_2). We define the functions ϕ, π, and δ as follows.

$$\phi(X) = \begin{cases} \phi_1(X) & \text{if } X \subseteq \{1, \ldots, p\} \\ \phi_2(X) & \text{otherwise} \end{cases}$$

$$\pi(i, s) = \begin{cases} \pi_1(i, s) & \text{if } i \leq p \\ \pi_2(i, \phi_2(\emptyset)) & \text{if } i > p \text{ and } s \in T_1 \\ \pi_2(i, s) & \text{if } i > p \text{ and } s \in T_2 \end{cases}$$

$$\delta(i, r, s) = \begin{cases} \delta_1(i, r, s) & \text{if } i \leq p \\ \delta_2(i, r, \phi_2(\emptyset)) & \text{if } i > p \text{ and } s \in T_1 \\ \delta_2(i, r, s) & \text{if } i > p \text{ and } s \in T_2 \end{cases}$$

We remark that while these functions may appear to be asymmetrical, the desired properties hold independent of which matroid is chosen as M_1 or M_2. The asymmetry is necessary to handle the fact that in the recursive definition of f, a state s can only transition upon an item being interdicted. So, when we transition from processing elements of M_1 to elements of M_2, we may still have a state corresponding to M_1 if the boundary element is not interdicted, and hence we need to transition to a state from M_2 at the earliest point that an element from M_2 is interdicted. This construction can be simplified if we allow state transitions in f when items are not interdicted, but it comes at the cost of heavier notation in the rest of the paper. The following establishes that this works as intended.

Theorem 2.7. *If T, ϕ, π, and δ are defined as above, then $P_{bound}(\delta, \phi, \pi)$ and $P_{succ}(\phi, \pi)$ are satisfied. Furthermore, if there exists some $\alpha \geq 0$ such that $P_{approx}(\delta_1, \phi_1, \pi_1, \alpha)$ and $P_{approx}(\delta_2, \phi_2, \pi_2, \alpha)$ are satisfied, then $P_{approx}(\delta, \phi, \pi, \alpha)$ is satisfied.*

1 Let $\delta(i,r)$ be defined as a table;
2 **for** $i = 1, \ldots, m$ **do**
3   **for** $r = 0, \ldots, C$ **do** $\delta(i,r) \leftarrow \infty$;
4   **if** *there exists a replacement element* k *for* i *in* $M \setminus \{1, \ldots, i-1\}$ **then**
5   ⌊ **for** $r = 0, \ldots, C$ **do** $\delta(i,r) \leftarrow w_k - w_i$;
6 $E' \leftarrow \emptyset, c' \leftarrow 0$;
7   **for** $j = 1, \ldots, i-1$ **do**
8   ⌊ $E' \leftarrow E' \cup \{j\}$;
9   ⌊ $c'_j \leftarrow c_j$;
10   Let u and v be the endpoints of edge i;
11 $x \leftarrow$ `MinCut`(E',c',u,v);
12   **for** $r = \max\{0, C - x + 1\}, \ldots, C$ **do** $\delta(i,r) \leftarrow 0$;
13   **for** $j = i+1, \ldots, m$ **do**
14 $E' \leftarrow E' \cup \{j\}$;
15 $c'_j \leftarrow \infty$;
16 $x \leftarrow$ `MinCut`(E',c',u,v);
17   **for** $r = \max\{0, C - x + 1\}, \ldots, C$ **do** $\delta(i,r) \leftarrow \min\{\delta(i,r), w_j - w_i\}$;
18   **if** $x > C$ **then break**;

Algorithm 1: Algorithm to compute $\delta(i,r)$ for all i and r, when M is graphic.

The proof is technical and not very interesting, so we defer it to Appendix A. Since partition matroids are the direct sum of uniform matroids, we can then establish the following.

Corollary 2.8. *The partition matroid interdiction problem can be solved in time $O(km^2C) = O(m^3C)$, where k is the number of disjoint uniform matroids that need to be summed to construct the partition matroid.*

Proof. Applying our method to construct a state set T for a partition matroid yields $T = \{0, \ldots, C\} \times (T_1 \sqcup T_2 \sqcup \cdots \sqcup T_k)$ where the T_i correspond to uniform matroids. Any element in the set $T_1 \sqcup \cdots \sqcup T_k$ can be addressed by using a number in the set $\{1, \ldots, k\}$ to indicate which T_j the element is in, along with a number from $\{1, \ldots, m\}$ to represent the element itself in T_j. So, considering the i parameter of f, which is in $\{1, \ldots, m\}$, and the r parameter, which is in $\{0, \ldots, C\}$, the total number of DP states of f is $O(km^2C)$, and clearly each can be computed in constant time. $\qquad\square$

2.3 Graphic matroids

In this section, we describe our upper bound for the case when M is graphic. This case is also known as the MST interdiction problem (e.g., see [20,30]), or the most vital edges in the MST problem (e.g., see [19]).

We follow again the framework laid out in Section 2, wherein we must define a set T and functions δ, π, and ϕ. For now we define $T = \{\epsilon\}$ (i.e., an arbitrary 1-element set) as no additional state is needed to get a good bound; later we will show how to extend T to get stronger bounds. So, we can simply define $\pi(i, \epsilon) = \epsilon$ and $\phi(X) = \epsilon$. It is trivial to verify the desired conditions hold. To simplify notation, we simply write $\delta(i,r)$ to mean $\delta(i,r,\epsilon)$.

It remains to define a function δ satisfying $P_{\text{bound}}(\delta, \phi, \pi)$. However, we will not actually use the condition that $\phi(X_{<i}) \geq r$ from the definition of P_{bound}, so our function δ will actually satisfy the following stronger condition:

$$\delta(i,r) \geq \max\left\{F(X_{<i} \cup \{i\}) - F(X_{<i}) : X \in \mathcal{U}, \phi(X_{<i}) \leq r\right\}. \tag{2.11}$$

Algorithm 1 computes the value of $\delta(i,r)$ for all values of i and r. In the algorithm, `MinCut`(E,c,u,v) computes the minimum u-v cut in the subgraph of G with edges E and edge weights c.

Theorem 2.9. *The values of $\delta(i,r)$ after running Algorithm 1 satisfy Eq. (2.11).*

Proof. Notice that each iteration i of the outer loop can be computed independently of the other iterations. So, fix i arbitrarily. If there exists a replacement element k for i in $M \setminus \{1, \ldots, i-1\}$, then on Line 5, the algorithm sets $\delta(i,r)$ to $w_k - w_i$ for every i and r, This satisfies the desired inequality because it effectively assumes $X = E$, which does not necessarily satisfy $X \in \mathcal{U}$ and $\phi(X_{<i}) \leq r$, but certainly upper bounds $F(X_{<i} \cup \{i\}) - F(X_{<i})$. In fact, this simple bound for $\delta(i,r)$ works with any matroid, not just graphic matroids.
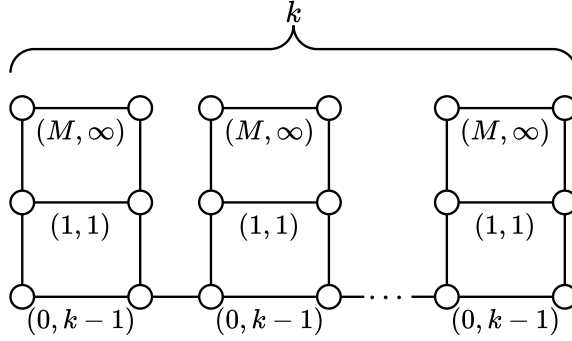
Figure 1: Comb instance with k 'teeth'. The unlabelled edges have $w_e = 0$ and $c_e = \infty$; all other edges are labelled with (w_e, c_e). The capacity C is $2k - 1$.

Consider the case where the algorithm sets $\delta(i, r) = 0$ on Line 12. Notice that we set $\delta(i, r) = 0$ precisely when $C - r <$ MinCut(E',c',u,v). In order for i to be an edge in the lexicographically smallest spanning tree, we must cut u and v in (V, E'); otherwise, an earlier edge could replace i in the lexicographically smallest spanning tree. So, if $C - r <$ MinCut(E',c',u,v), then there is no X for which i is an edge in the lexicographically smallest spanning tree of $E \setminus X$, and hence there will be no gain from interdicting i. That is, $F(X \cup \{i\}) - F(X) = 0$ for all $X \subseteq \{1, \dots, i - 1\}$ with $c(X) \leq C - r$. Therefore, the condition is satisfied.

We now show that the assignment to $\delta(i, r)$ on Line 17 is correct. Let $X \subseteq \{1, \dots, 1-i\}$ be such that $c(X) \leq C-r$. Consider the lexicographically smallest spanning tree T of $G \setminus X$. If edge i is not in T, then $F(X \cup \{i\}) - F(X) = 0$, and since $w_j \geq w_i$, we set $\delta(i, r) \geq 0$ and the condition is satisfied.

So, we may assume i is in T. Fix r, let j be the earliest iteration where we set $\delta(i, r) = w_j - w_i$ on Line 17, and let E' and c' be as they are when Line 17 executes on iteration j. Since we set $\delta(i, r) = w_j - w_i$, there is no u-v cut of cost at most $C - r$ in the graph (V, E') with edge costs c'. However, in order for us to have $i \in T$, X must be a u-v cut in $(V, E' \cap \{1, \dots, i - 1\})$; otherwise i could be replaced in T with another edge e' from the cut with $e' < i$, yielding an lexicographically smaller spanning tree. However, since $c(X) \leq C - r$ and every u-v cut in (V, E') has cost greater than $C - r$, in particular X is not a u-v cut in (V, E'). So, some edge $e \in \{i + 1, \dots, j\}$ crosses the cut defined by X. Therefore, adding e to $T \setminus \{i\}$ produces a spanning tree, and $F(X \cup \{i\}) - F(X) \leq w_j - w_i$ as desired. $\square$

2.3.1 Worst-case performance

We have now established how to compute some upper bounds for graphic matroid interdiction, but the above algorithm and proof gives little intuition about how strong the bounds are. In this section we define a family of instances for which $f(1, \phi(\emptyset)) = \Theta(m)$ OPT. Although this result is negative—for example, the Lagrangian upper bound is known to be at most $2 \cdot$ OPT [20]—we argue in Section 4 that the poor worst-case performance is outweighed by other attractive features of this bound in the context of a branch-and-bound scheme.

The intuition is as follows. For each $k \geq 1$ and $M \geq 1$, we define a *comb* instance $\mathcal{I}(k, M)$. Each comb instance can be divided into k sections, called *teeth*, which act like independent subproblems, in the following sense: the choice of which edges to include in the spanning tree from any one of the k sections has no impact on which edges can be added in any of the other sections. In each tooth, there is a high-cost edge and a low-cost edge which must both be interdicted to significantly increase the MST weight. However, in the upper bound computation, after the leader pays for a single high-cost edge from one of the teeth, it can then significantly increase the MST weight in every tooth by interdicting only the low-cost edge, because there is no way to distinguish which tooth the interdicted high-cost edge is from. The instances $\mathcal{I}(k, M)$ are formally described in Fig. 1.

Proposition 2.10. *The optimal solution to the MST interdiction problem on a comb instance $\mathcal{I}(k, M)$ has weight $M + 1$.*

Proof. Notice that of the edges with non-infinite interdiction cost, they all either have cost 1 or cost $k - 1$, and there is one of each in each tooth. Furthermore, in each tooth, it is necessary to interdict the edge of cost $k - 1$ to gain any advantage from interdicting the edge of cost 1, because otherwise the cost 1 edge will not be in the MST. Since the capacity is $2k - 1$, we can interdict at most two of the edges of cost $k - 1$. If we interdict one of them, say

in the first tooth, then we can additionally interdict the edge of cost 1 in that tooth which will increase the MST weight from 0 to M. This leaves $k-1$ capacity remaining, with which we can interdict the $k-1$ cost edge in one more tooth, which increases the MST weight to $M+1$. □

Proposition 2.11. *The value of the upper bound $f(1,C)+F(\emptyset)$ on a comb instance $\mathcal{I}(k,M)$ is at least $k(M-1)+1 = \Theta(m) \cdot \text{OPT}$.*

Proof. Recall that edges are sorted by increasing weight; hence, all of the weight 0 edges come first, followed by the weight 1 edges, and then the weight M edges. Notice that the MST of a comb instance has weight 0 because the subgraph with weight 0 edges is a spanning tree.

Suppose that in the DP we first interdict some edge i with cost $k-1$. This causes the MST weight to increase by 1 (since the edge of weight 1 in that tooth must now be used), and hence $\delta(i,C) \geq 1$. We claim that interdicting this one edge allows us to interdict every edge of cost 1, while gaining an upper bound weight of at least $M-1$ from each edge, despite the fact weight from interdicting these edges does not contribute any actual weight. This plan costs $k-1+k = 2k-1 = C$, and produces an upper bound of $f(1,C) + F(\emptyset) = 1 + k(M-1) + 0$ as desired.

To show that the claim holds, it suffices to show that for every edge j of cost 1 and weight 1 and for every $1 \leq r \leq C - (k-1)$, we have $\delta(j,r) \geq M-1$. Let j' be the edge of cost $k-1$ on the same tooth as j. We define $X = \{j'\}$. Then $\sum_{k \in X} c_k = k-1 \leq C-r$. Observe that $F(X \cup \{j\}) = M$ and $F(X) = 1$. This completes the proof. □

It is easy to see that the maximum spanning tree of a comb instance has weight kM (this corresponds to including the edge of weight M in the spanning tree from each of the k teeth). So, Proposition 2.11 shows that the upper bound is nearly returning the maximum spanning tree weight and hence the analysis is reasonably tight.

It is interesting to note that the same proof would work even if δ satisfied $P_{\text{bound}}(\delta, \phi, \pi)$ with equality. So, the reason why the bound is poor is because our state space T is not large enough. The existence of a pseudopolynomial sized T for which the upper bound has worst-case performance of $o(m)\,\text{OPT}$ is an open question. While the worst-case performance may appear to be quite poor, our computational experiments in Section 4 suggest that the average case is significantly better.

2.4 Strengthening

The strength of our upper bounds depend upon the set T and the ability to exploit T to get strong bounds on the discrete derivative problem. In this section we present a generic way to strengthen the bounds, which comes at an exponential cost in running time and memory usage, but has the practical advantage that it allows us to compute stronger bounds for hard instances, which significantly improves solver performance.

The idea explored in this section is to extend T to store *complete* information about whether the first p elements of the matroid were interdicted, for some small p. Evidently, if $p = m$, then this extended T would completely describe the interdiction set X, and hence the interdiction problem could be solved exactly just by the upper bound. But, this requires $|T| = \Omega(2^p)$, so we have to choose p to be small in order for the DP table to be of a practical size.

Given the set T and functions ϕ, π, and δ defined for a matroid M, we define T_p, ϕ_p, and π_p for all $p \in \{0,1,\ldots,m\}$ as follows:

$$T_p = T \times \{X \cap \{1,2,\ldots,p\} : X \in \mathcal{U}\}$$

$$\pi_p(i,(s,X)) = \begin{cases} (\pi(i,s), X \cup \{i\}) & \text{if } i \leq p \\ (\pi(i,s), X) & \text{otherwise} \end{cases}$$

$$\phi_p(X) = (\phi(X), X \cap \{1,2,\ldots,p\})$$

The function $\delta_p(i,r,(s,X))$ is a bit less straightforward to define. In essence, we want to delete all elements of X from the matroid, and for all $j \notin X$ with $j \leq p$, forbid j from being interdicted. In Algorithm 1, this amounts to modifying the loop on Line 7 to skip over elements $j \leq p$ such that $j \in X$ while subtracting their capacity from C, and setting elements $j \leq p$ such that $j \notin X$ to have infinite cost c'_j. Other matroids must be handled on a case-by-case basis depending on the definition of δ.

We then define $f_p(i,r,(s,X))$ to be the same as $f(i,r,s)$, but using the functions ϕ_p, π_p, and δ_p. It is easy to see that for all i, r, and s, we have $f_0(i,r,(s,\emptyset)) = f(i,r,s)$, and

$$f_p(i, C - c(X_{<i}), (\phi(X_{<i}), X_{\leq p})) \leq f(i, C - c(X_{<i}), \phi(X_{<i}))$$

for $p \geq 0$ and $X \in \mathcal{U}$. However, f_p is still a valid upper bound by Theorem 2.1, as desired.

13

```
 1  function LowerBound()
 2    │  X ← ∅;
 3    │  Let Y be a minimum weight basis of M \ X;
 4    │  while true do
 5    │  │  b ← −∞; f ← ⊥;
 6    │  │  for e ∈ Y such that c_e ≤ C − c(X) do
 7    │  │  │  Let r_0, r_1, …, r_s be the replacement chain for e in Y;
 8    │  │  │  v ← max{ (w_{r_i} − w_e)/∑_{j=0}^{i} c_{r_j} : i = 1, 2, … s };
 9    │  │  │  if b < v then b ← v; f ← e;
10    │  │  if f = ⊥ then break;
11    │  │  X ← X ∪ {f};
12    │  │  Y ← Y \ {f} ∪ {R(Y, f)};
13    │  return (X, Y)
```

Algorithm 2: Greedy lower bound.

3 Exact algorithms

In this section, we describe a branch-and-bound algorithm for exactly solving the matroid interdiction problem. The performance of our algorithm crucially depends on the upper bounds derived in Section 2. We also use a heuristic greedy lower bound to improve computational performance further.

3.1 Greedy lower bound

We use a simple greedy heuristic to find an initial feasible solution (i.e., lower bound) to seed the branch-and-bound algorithm with. Although this heuristic does not have any performance guarantee, we find that for MST interdiction, it improves performance in practice as it is able to find good solutions faster than is possible by branching alone. In fact, the solution returned by the heuristic has an average optimality gap of 1.48% over the entire data set. A full performance analysis can be found in Section 4.2.

The basic idea behind the heuristic is to repeatedly select an element to interdict which maximizes the ratio of basis weight increase to interdiction cost. To formalize this, suppose we start with some feasible solution (X, Y). For $e \in Y$ let $R(Y, e)$ be the replacement element for e if e is interdicted from Y. We could select the element $e \in Y$ to interdict (i.e., add to X) which maximizes $(w_{R(e)} − w_e)/c_e$ (we call this quantity the *efficiency*). X and Y are then updated accordingly and this process repeated until there is no item which can be added to X without going over capacity.

This idea is not completely new; a similar algorithm appeared previously for the case where all costs are 1 [3]. However, we modify the element selection slightly based on analyzing some cases where we noticed poor performance. The fundamental issue with this algorithm is that it may be necessary to interdict some element(s) with low efficiency in order to have the opportunity to interdict an element with large efficiency. So, by always choosing the item with the highest immediate benefit, we are in some sense behaving too greedily and could potentially miss out on the largest efficiency gains.

To address this issue, we modify the selection rule to look ahead along *replacement chains*. We define the replacement chain for an element $e \in Y$ as the sequence $r_0, r_1, \ldots r_s$ where $r_0 = e$ and $r_i = R(Y \cup \{r_0, \ldots, r_{i−1}\}, r_{i−1})$ for $i \geq 1$. The sequence ends when there is not enough capacity to interdict the next element that would have been be added, i.e., s is the smallest index such that $c(X \cup \{r_0, \ldots, r_s, r_{s+1}\}) > C$. The replacement chain for an element $e \in Y$ can be computed using the independence oracle in polynomial time with a single scan through the elements, because we know that $r_0 < r_1 < \cdots < r_s$ due to our element ordering assumption.

We utilize the replacement chains as follows. In each step of the heuristic, instead of selecting the element $e \in Y$ which maximizes $(w_{R(e)} − w_e)/\max\{1, c_e\}$, we instead select the element $e \in Y$ which maximizes

$$\max\left\{ (w_{r_i} − w_e)/\textstyle\sum_{j=0}^{i} c_{r_j} : i = 1, 2, \ldots s \right\}.$$

The full algorithm is described in Algorithm 2. It is easy to see that the output is a feasible solution because edges are only added to X when they fit within the capacity, and we maintain that Y is a minimum weight basis of $M \setminus X$ at each iteration. To integrate this heuristic into our branch and bound scheme, we compute `LowerBound()` at the beginning of the search, and use it as the initial lower bound.
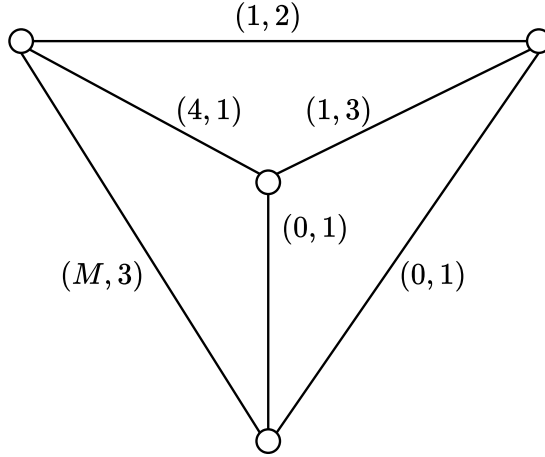
Figure 2: Worst-case instance for the greedy lower bound. Each edge e is labelled with (w_e, c_e), and $C = 2$.

In Fig. 2 we show a family of instances parameterized by an integer M for which the heuristic returns an arbitrarily bad solution (with approximation ratio $\Theta(M)$). Specifically, the greedy lower bound only interdicts the edge labelled $(1, 2)$—yielding an MST of weight 4—but the optimal solution is to interdict both edges labelled $(0, 1)$, which increases the MST weight to $M + 2$. There are countless possible variations of this heuristic and surely better variants to be found. However, we found this algorithm to be remarkably effective in practice—it often finds the optimal solution—so we did not attempt to improve it further in this work. Discussion regarding the strength of the heuristic in practice can be found in Section 4.2.

3.2 Branch-and-bound algorithm

Our branch-and-bound algorithm for solving matroid interdiction exactly is now straightforward to describe, given the upper and lower bounds from previous sections. Throughout this section we assume elements are ordered according to (A1). In essence, we recursively enumerate feasible solutions by branching on whether $e \in X$ (i.e., whether e is interdicted) for $e = 1, \ldots, m$. At each step, an optimal $Y \in \mathcal{L}(X)$ is maintained by the use of a dynamic data structure. Since the elements are sorted by non-decreasing weight within each component, when an element is interdicted from Y it is always replaced by a later element. So, we do not need to branch on every element: branching on the next element which is also in Y suffices. The algorithm is formalized in Algorithm 3 and correctness is proved in Theorem 3.1. With branching alone our algorithm is asymptotically faster than the worst-case running time of previous enumerative algorithms up to polylog factors; this is formalized in Theorem 3.2. However, the biggest improvement in our algorithm over previous algorithms is the pruning we can accomplish using the upper bounds from Section 2.

The `BranchAndBound` function recursively enumerates solutions. It has one parameter i, and refers to global variables d, X^* and Y^*. The parameter i is the number of basis elements that have been skipped (i.e., not interdicted). When $i = r(E)$, there are no more basis elements to interdict. The global variable d is an instance of `DynamicInterdiction`, a dynamic data structure which keeps track of the optimal lower-level solution as elements are added and removed from the interdiction set, and the variables X^* and Y^* hold the current best known incumbent solution. Given an instance d of the `DynamicInterdiction` structure, the following variables and operations are available:

- $d.X$ is the current set of interdicted elements. Initially this is $\emptyset$.

- $d.Y$ is the lexicographically smallest basis of $M \setminus d.X$.

- $d.e$ returns the current element being considered for interdiction. Initially, this is the first (smallest weight) element of $d.Y$.

- $d.\texttt{Skip}()$ advances $d.e$ to point to the next element of $d.Y$ following $d.e$.

- $d.\texttt{UnSkip}()$ undoes a `Skip()` operation. It is only valid to call this if the last (non-undo) operation was a `Skip()` operation.

1 In a background thread, compute the upper bound $f_p(1, C, \phi(\emptyset), \emptyset)$ with a progressively increasing number
  of prefix bits $p = 0, 1, \ldots$ until the memory usage limit is reached;

2 $X^*, Y^* \leftarrow$ LowerBound();

3 $d \leftarrow$ DynamicInterdiction.Init();

4 **function** BranchAndBound(i)

5   **if** $w(d.Y) > w(Y^*)$ **then** $X^*, Y^* \leftarrow (d.X, d.Y)$;

6   **if** $i = r(E)$ *or* $d.e > m$ *or* $\min\{c_e : e \geq d.e\} > C - c(d.X)$ **then return**;

7   **if** *some upper bound has finished computation* **then**

8   Let $p = \max\{p : \text{computation of } f_p \text{ has finished}\}$;

9   **if** $f_p(d.e, C - c(d.X), d.X \cap \{1, \ldots, p\}) + w(d.Y) \leq w(Y^*)$ **then return**;

10   **if** $c_{d.e} \leq C - c(d.X)$ **then**

11   **if** d.Interdict() $= false$ **then output** error;

12   BranchAndBound(i);

13 d.UnInterdict();

14 d.Skip();

15   BranchAndBound($i + 1$);

16 d.UnSkip();

17 BranchAndBound(0);

18 **output** (X^*, Y^*);

<div align="center">

Algorithm 3: Branch and bound algorithm to exactly solve matroid interdiction.

</div>

- d.Interdict() adds $d.e$ to $d.X$, and replaces $d.e$ in $d.Y$ with the replacement element for $d.e$ (given the current $d.X$). If no such replacement element exists, the function should return false. It then advances $d.e$ to point to the smallest element of $d.Y$ after the original $d.e$ (note that this next element is not necessarily the replacement element for the original $d.e$).

- d.UnInterdict() undoes an Interdict() operation. It is only valid to call this if the last (non-undo) operation was an Interdict() operation.

We will describe the actual implementation of this data structure that we use later; for now it is enough to know that the operations satisfy this description. We now prove the two main results of this section. Theorem 3.1 establishes correctness of the algorithm, and Theorem 3.2 establishes a worst-case running time bound.

Theorem 3.1. *Algorithm 3 correctly solves the matroid interdiction problem.*

Proof. Before BranchAndBound(0) is called, X^* and Y^* are initialized to a feasible solution by the LowerBound() function. We show that the algorithm maintains the invariant that (X^*, Y^*) is feasible, and for every feasible solution, the algorithm either enumerates it or proves that it is no better than the current best known solution (X^*, Y^*). It follows that the algorithm always outputs an optimal solution.

Assuming the correct implementations of the operations on the DynamicInterdiction structure, it is easy to see that an element is only interdicted (added to $d.X$) if there is enough capacity to do so, and it is maintained that $d.Y$ is the lexicographically smallest basis of $M \setminus d.X$. So, $(d.X, d.Y)$ is always a feasible solution. Since (X^*, Y^*) is only ever updated to hold the value of $(d.X, d.Y)$, (X^*, Y^*) also remains feasible at every step of the algorithm.

Now, fix some $X \in \mathcal{U}$, and observe that if $Y = \mathcal{B}^{\min}(X)$, then for any X' such that $X \subseteq X' \subseteq E \setminus Y$, we also have $Y = \mathcal{B}^{\min}(X')$. So, since $c \geq 0$, it suffices to show that the algorithm enumerates all X' such that $\mathcal{B}^{\min}(X') \neq \mathcal{B}^{\min}(X)$ for every $X \subset X'$. We call such X' *undominated*.

For now, assume that no upper bound has been computed, so that the bound test on Line 9 does not prune any nodes. We also assume that the leader is unable to reduce the rank of the matroid for any $X \in \mathcal{U}$, that is, $\mathcal{B}^{\min}(X)$ always exists. We first show that with these assumptions, the algorithm enumerates all feasible undominated sets. After establishing this, we prove that the bound test only prunes a node when enumerating it could not lead to an improvement in the incumbent solution (X^*, Y^*), and that if the leader is able to reduce the matroid rank, this situation is detected.

Suppose $X' = \{x_1, x_2, \ldots, x_q\}$ is undominated, where $x_1 < x_2 < \cdots < x_q$. We claim that $X'_p := \{x_1, x_2, \ldots, x_p\}$ is also undominated for all $p \in \{0, 1, \ldots, q\}$. Assume for the sake of contradiction that X' is undominated but $X'_{q-1} = X' \setminus \{x_q\}$ is not. So, there is some $f \in X' \setminus \{x_q\}$ such that $\mathcal{B}^{\min}(X' \setminus \{x_q, f\}) = \mathcal{B}^{\min}(X' \setminus \{x_q\})$. This implies that f is not in the lexicographically smallest basis $\mathcal{B}$ of $M \setminus (X' \setminus \{x_q\})$ (otherwise, interdicting it would

cause the basis to change). But, if x_q is interdicted from $\mathcal{B}$, it will be replaced with an element after x_q because the elements are sorted by weight within each component. In particular, x_q cannot be replaced with f. So, f is also not in the lexicographically smallest basis of $M \setminus X'$, but this implies that $\mathcal{B}^{\min}(M \setminus X') = \mathcal{B}^{\min}(M \setminus (X' \setminus f))$, contradicting that X' is undominated.

Assume that $X' \in \mathcal{U}$. We claim that the algorithm enumerates X'; that is, there is some point where BranchAndBound is called when $d.X = X'$. We show by induction on p that each X'_p is enumerated. Clearly, $X'_0 = \emptyset$ is enumerated, because $\emptyset$ is the initial value of $d.X$. Now suppose $p \geq 1$ and that X'_{p-1} is enumerated. Consider the values of d and i when X'_{p-1} is enumerated; if it is enumerated multiple times then break the tie by choosing i as small as possible. So, $d.X = X'_{p-1}$, $d.Y$ is the lexicographically smallest basis of $M \setminus X'_{p-1}$, and, because we chose i as small as possible, $d.e$ is the smallest element of $d.Y$ which comes after x_{p-1}. Since X'_p is undominated, $\mathcal{B}^{\min}(X'_p) \neq \mathcal{B}^{\min}(X'_{p-1})$. So, x_p must be in $d.Y$; otherwise, $d.Y$ would also be the lexicographically smallest basis of $M \setminus X'_p$, contradicting that it is undominated. Hence, $d.e \leq x_p$. If $d.e = x_p$, then we are done, because the algorithm will enumerate X'_p via the recursive call on Line 12. Therefore, if we don't already have $d.e = x_p$ then there is a sequence of recursive calls on Line 15 that advance $d.e$ to point to x_p, at which point the recursive call on Line 12 will enumerate X'_p.

So, without the bound test on Line 9, the algorithm would have enumerated every undominated $X' \in \mathcal{U}$. It remains to show that if some X' is pruned by the bound test, it could not have led to an improved solution (X^*, Y^*). Suppose that for some X^*, Y^*, $d.X$, $d.Y$, and $d.e$ we have $f_p(d.e, C - c(d.X), d.X \cap \{1, \ldots, p\}) + w(d.Y) \leq w(Y^*)$. Recall that $f_p(d.e, C - c(d.X), d.X \cap \{1, \ldots, p\})$ is an upper bound on how much the minimum basis weight $w(d.Y)$ can increase by adding edges from $\{d.e, d.e + 1, \ldots, m\}$ to $d.X$. So, if the bound test prunes this node, then no child of this node can have minimum basis weight larger than Y^*, as desired.

Finally, we claim that if there is some $X \in \mathcal{U}$ such that there is no basis using elements in $E \setminus X$, then the algorithm correctly detects this case and outputs an error. First, notice that if there is such an X, we may assume it is minimal with this property, and hence undominated. Then, deleting the largest element from X yields an undominated set X', so by the same argument used earlier, X' is enumerated by the algorithm. When Interdict() is invoked during the recursive call where X' is enumerated, it will return false because there is no replacement edge for $d.e$ (otherwise, $\mathcal{B}^{\min}(X)$ would exist). So, the algorithm will raise an error. $\square$

Theorem 3.2. *Assume that the elements of M are already sorted according to* (A1). *Then the worst-case running time of* BranchAndBound(0) *in Algorithm 3 is*

$$O\left(\binom{r(E) + k}{\min\{r(E), k\}} \cdot D\right) + I$$

where $k = \max\{|X| : X \in \mathcal{U}\}$, D is the time required for each operation of the dynamic data structure, and I is the time required to initialize the dynamic data structure.

Proof. We claim that the total number of leaves in the recursion tree is at most $t(r(E), k)$ where $t(0, j) = t(i, 0) = 1$, and $t(i, j) = t(i - 1, j) + t(i, j - 1)$ for $i, j \geq 1$. To see this, consider the following.

- The i parameter of BranchAndBound is incremented whenever we recurse on Line 15, and when $i = r(E)$, the recursion terminates.

- Since $k = \max\{|X| : X \in \mathcal{U}\}$, the algorithm can recurse at most k times on Line 12 before we are guaranteed that $\min\{c_e : e \geq d.e\} > C - c(d.X)$, and the recursion terminates in this case. We assume that $\min\{c_e : e \geq f\}$ is precomputed for all $f \in E$, so that this test takes $O(1)$ time.

So, we can think of the recursive call on Line 15 as decrementing the i parameter of $t(i, j)$, and the recursive call on Line 12 as decrementing the j parameter. Hence, the total number of leaves is at most $t(r(E), k)$.

Now, we show that $t(i, j) = \binom{i+j}{\min\{i,j\}}$. If $i = 0$ or $j = 0$, this is clearly the case. So, suppose $i, j \geq 1$. Then, by induction,

$$t(i, j) = t(i - 1, j) + t(i, j - 1) = \binom{i + j - 1}{\min\{i - 1, j\}} + \binom{i + j - 1}{\min\{i, j - 1\}} = \binom{i + j}{\min\{i, j\}},$$

where the last equality follows by splitting into cases depending on whether i or j is larger and applying the recurrence relation for binomial coefficients. Since the number of nodes in a binary tree is at most 2 times the number of leaves, the total number of recursive calls is also $O(t(i, j))$. Each call takes $O(D)$ time (ignoring time taken in child nodes), so the total running time is as desired. $\square$

17

In the case of MST interdiction, by using the best-known decremental dynamic MST data structure (i.e., only supporting edge deletion operations) of Holm et al [11], we can get (amortized) $D = O(\log^2 n)$, with initialization time $I = O(m \log^2 n)$. We remark that the time required to sort the matroid elements and initialize the data structure will typically be dominated by the other terms. It may not be immediately obvious why it suffices to use a *decremental* dynamic MST data structure. The reason for this is that we only need to be able to support edge deletions and undo operations. The undo operation can easily be implemented to run in the same time as the edge deletion operation by keeping track of all changes made to the data structure during the edge deletion operations.

For the special case of MST interdiction when all interdiction costs are 1, the previous best known running time for an enumerative algorithm is $O(n^k \log \alpha((k-1)(n+1), n))$ [3,17]. By applying standard bounds on binomial coefficients to the result of Theorem 3.2, we find that for MST interdiction our algorithm runs in time $O(\min\{(2en/k)^k, (2ek/n)^n\} \cdot D) + I$. If we treat n and k as variables, this is asymptotically faster than the previous best algorithm up to polylog factors. Furthermore, our algorithm can handle arbitrary interdiction costs and generalizes to arbitrary matroids. Our algorithm also prunes many nodes using the upper bounds, and since accessing the upper bound table takes $O(1)$ time, it does so without any increase in the asymptotic running time. Further improvements in algorithms for the decremental dynamic MST problem would immediately yield improvements to our algorithm as well.

The dynamic data structure we use is described formally in Algorithm 4. Correctness is easy to see: it is just an adaptation of Kruskal's algorithm. While our dynamic data structure needs $D = O(m)$ and $I = O(m)$, and hence is theoretically slower than the data structure of Holm et al, we found it to have excellent performance in practice, while also being considerably simpler. This follows a common theme in the literature on dynamic MST in which simple algorithms are repeatedly found to have better computational performance than complex algorithms with good theoretical guarantees, for typical graphs [1,5,22].

Unlike other dynamic MST data structures, Algorithm 4 should be easy to adapt for other matroids: only the union-find structure needs to be replaced with a different mechanism for testing whether a set is independent in the given matroid. As a starting point towards such an adaptation, it is easy to modify the data structure to work for any matroid given an independence oracle. First, we remove all references to uf from Skip, UnSkip, and Interdict. Then, to find the replacement edge for e in the Interdict function, we find the lexicographically smallest basis of $E \setminus X$, remove e from it, and query the independence oracle for each edge after e to see if it can be added to the basis. Depending on the specifics of the matroid, it may be possible to improve this algorithm further.

3.3 Adaptation for the minimum cost blocker problem

Recall that in the minimum cost blocker problem, the objective is to minimize the cost required to increase the minimum basis weight to at least R. In the introduction, we mentioned that this problem is polynomial-time equivalent to the matroid interdiction problem by performing binary search on C or R. In this section, we show that our branch-and-bound scheme can in fact be modified to directly solve the minimum cost blocker problem without incurring the overhead of performing binary search. In Section 4, we compare the new algorithm (Algorithm 5) against Algorithm 3 with binary search over C in computational experiments, and find that Algorithm 5 is faster, as expected.

Since minimum cost blocker instances do not have a C value, for the purposes of adapting the algorithm, in the graphic matroid case we define $C = \text{MinCut} - 1$ where MinCut is the global minimum cut of G with edge costs c. The motivation for this is that $\text{MinCut} - 1$ is always an upper bound on C in the MST interdiction problem (setting it any larger would allow the leader to cut the graph). For general matroids the trivial bound $C = c(E)$ can be used. Algorithm 5 describes the modified branch-and-bound algorithm. We omit a full proof of correctness because the proof is largely the same as the proof of Theorem 3.1. However, a few observations are in order.

In the edge blocker problem, a minimum c-cost set X^* which intersects every basis is always a trivial solution, because it causes the minimum basis weight to increase to infinity. So, we start with such an X^* as an initial incumbent solution. For graphic matroids this X^* is just a global minimum cut, but it may be NP-hard to find X^* for general matroids, which we discuss later in Section 3.4. We then enumerate all undominated X (adopting the term *undominated* from the proof of Theorem 3.1) which could potentially lead to an improved incumbent solution (X^*, Y^*). Notice that for (X, Y) to be an improved incumbent solution, $w(Y)$ must reach the target weight R, and the cost of X must be at most the cost of X^*. Furthermore, no child node of a node with weight at least R can have lesser cost, so we can prune any such node. Similarly, no child of a node with cost at least $c(X^*)$ can have lesser cost, and since we assume all costs are at least 1, it cannot have higher weight without also having higher cost, so we can prune it.

The bound test $f_p(d.e, c(X^*) - c(d.X), d, X \cap \{1, \ldots, p\}) + w(d.Y) < R$ works correctly because in order to reach an improved incumbent solution, we must use at most $c(X^*) - c(d.X)$ additional capacity, and we must reach a

```
 1  structure DynamicInterdiction
 2  │   function Init()
 3  │   │   X ← ∅;
 4  │   │   Y ← lexicographically smallest basis of M;
 5  │   │   e ← element of Y with the smallest index;
 6  │   │   uf ← empty union-find structure on n elements;
 7  │   │   bst ← empty self-balancing binary search tree;
 8  │   └   stack ← empty stack;
 9  │   function Skip()
10  │   │   uf.Union(e.s, e.t);                                    /* Union the endpoints of edge e */
11  │   └   e ← bst.Next(e);                              /* Move to the next element in bst after e */
12  │   function UnSkip()
13  │   │   e ← bst.Prev(e);                            /* Move to the element in bst before e */
14  │   └   uf.Undo();                                  /* Undo the most recent Union operation */
15  │   function Interdict()
16  │   │   if e = m then return false;
17  │   │   for f = e + 1, . . . , m do                        /* Find the replacement edge for e */
18  │   │   │   if f ∈ Y then uf.Union(f.s, f.t);
19  │   │   │   else if uf.Find(f.s) ≠ uf.Find(f.t) then break;
20  │   │   └   if f = m then f ← ⊥;
21  │   │   Undo all Union operations from the above loop;
22  │   │   if f = ⊥ then return false;
23  │   │   X ← X ∪ {e};
24  │   │   Y ← Y \ {e} ∪ {f};
25  │   │   bst.Insert(f);
26  │   │   e ← bst.Erase(e);   /* Erase e from bst, then replace e with the next element after the
          location where e was in bst */
27  │   │   stack.Push((e, f));
28  │   └   return true;
29  │   function UnInterdict()
30  │   │   e′, f ← stack.Pop();
31  │   │   X ← X \ {e′};
32  │   │   Y ← Y ∪ {e′} \ {f};
33  │   │   bst.Erase(f);
34  └   └   e ← bst.Insert(e′);                                   /* Insert e′ into bst, then set e to e′ */
```

Algorithm 4: Dynamic interdiction data structure, specialized for MST interdiction.

1 In a background thread, compute the upper bound $f_p(1, r, \phi(\emptyset), \emptyset)$ for all $r = 0, 1, \ldots, C$ with a progressively increasing number of prefix bits $p = 0, 1, \ldots$ until the memory usage limit is reached;

2 Let X^* be a minimum c-cost set which intersects every basis of M and let $Y^* = \bot$;

3 $d \leftarrow$ `DynamicInterdiction.Init()`;

4 **function** `BranchAndBound`(i)

5   **if** $w(d.Y) \geq R$ *and* $c(d.X) \leq c(X^*)$ **then** $X^*, Y^* \leftarrow (d.X, d.Y)$;

6   **if** $i = \mathrm{r}(E)$ *or* $d.e > m$ *or* $w(d.Y) \geq R$ *or* $c(d.X) \geq c(X^*)$ **then return**;

7   **if** *some upper bound has finished computation* **then**

8   Let $p = \max\{p :$ computation of f_p has finished$\}$;

9   **if** $f_p(d.e, c(X^*) - c(d.X), d.X \cap \{1, \ldots, p\}) + w(d.Y) < R$ **then return**;

10   **if** $d.$`Interdict()` **then**

11   `BranchAndBound`(i);

12 $d.$`UnInterdict()`;

13 $d.$`Skip()`;

14   `BranchAndBound`$(i + 1)$;

15 $d.$`UnSkip()`;

16 `BranchAndBound(0)`;

17 **output** (X^*, Y^*);

Algorithm 5: Modification of the branch and bound algorithm for solving the minimum cost blocker problem.

solution of value at least R. The background computation of the upper bound differs from the interdiction case in that we compute $f_p(1, r, \emptyset)$ for *all* $r = 0, 1, \ldots, C$. This is necessary because the algorithm calls $f_p(e, c(X^*) - c(d.X), d.X \cap \{1, \ldots, p\})$, and $c(X^*) - c(d.X)$ is not necessarily equal to $C - c(X)$ for any $X \in \mathcal{U}$.

Finally, instead of interdicting the element $d.e$ whenever there is capacity to do so (as in the interdiction problem), we instead interdict whenever a replacement element for $d.e$ exists (i.e., when $d.$`Interdict()` returns true). This ensures that we enumerate all possible undominated solutions.

3.4 Preprocessing

In this section, we discuss two preprocessing steps: one which can simplify or even solve some matroid interdiction instances, and one which can speed up computation of the upper bound at the expense of bound quality. The former preprocessing step has been used previously in the case of MST interdiction [20,32]. Here, we generalize the ideas to arbitrary matroids, and describe how to perform the preprocessing in polynomial time for graphic and partition matroids.

To begin, observe that if there is some $X \in \mathcal{U}$ such that $\mathrm{r}(E \setminus X) < \mathrm{r}(E)$, then $\mathcal{L}(X) = \emptyset$, so the optimal value of (MI) is ∞. Our branch-and-bound algorithm signals an error if this case is detected. We would instead like to have a way of detecting this case as a preprocessing step. The problem of finding such an X is equivalent to the problem of finding a minimum cost set X which intersects every basis, known as a *cocircuit*. If we have such a set X which intersects every basis and $c(X) \leq C$, then $X \in \mathcal{U}$ and $\mathrm{r}(E \setminus X) < \mathrm{r}(E)$. For graphic and partition matroids, there are polynomial time algorithms that can find a minimum cost cocircuit, which we discuss below. However, this is NP-hard even for binary matroids (see [27], which shows NP-hardness for the equivalent problem of finding the minimum distance of a binary linear code). So, in general we cannot do this preprocessing in polynomial time unless P = NP.

For graphic matroids, we can easily find a minimum cost cocircuit in polynomial time: such a set is simply a global minimum cut in the graph. Now we consider partition matroids. Suppose that the partition matroid $M = (E, \mathcal{I})$ is represented by $\mathcal{I} = \{J \subseteq E : |J \cap S_i| \leq k_i \ \forall i\}$ for some disjoint sets S_i and integers k_i. Then, a cocircuit is any set X which has $|X \cap S_i| > |S_i| - k_i$ for some i, because any basis B has $|B \cap S_i| = k_i$, so $|X \cap S_i| + |B \cap S_i| > |S_i|$ and hence $X \cap B \neq \emptyset$. Since $c_e \geq 1 \ \forall e \in E$, any minimum cost cocircuit must only intersect a single set S_i, and hence we can find such a cocircuit by using a greedy algorithm to find the cheapest i and X such that $|X \cap S_i| = |S_i| - k_i + 1$.

An extension of this idea can be used to simplify some problem instances, as observed in prior works [20,32]. Let $w_1', w_2', \ldots, w_\ell'$ be the distinct weights from w, and assume $w_1' < w_2' < \cdots < w_\ell'$. For all i, let $E_{\leq i} = \{e \in E : w_e \leq w_i'\}$. Let k be the smallest index such that $\mathrm{r}(E_{\leq k} \setminus X) = \mathrm{r}(E)$ for all $X \in \mathcal{U}$. Then, we claim that for any $X \in \mathcal{U}$, $\mathcal{B}^{\min}(X \cap E_{\leq k-1}) = \mathcal{B}^{\min}(X)$. The following lemma is a trivial generalization of Lemma 2.2 from [20], which concerns only graphic matroids. The proof is almost identical, but we include it for completeness. To derive

it from the original proof, just substitute $\mathrm{r}(E) - \mathrm{r}(S) + 1$ for every occurrence of $\sigma(S)$, where S is an arbitrary subset of E (here, $\sigma(S)$ is just the number of connected components of the graph (V, S)).

Lemma 3.3. *For convenience, define $w_0' := 0$. If $X \in \mathcal{U}$, then*

$$\min\{w(Y) : Y \in \mathcal{L}(X)\} = \sum_{i=0}^{k-1} (w_{i+1}' - w_i')(\mathrm{r}(E) - \mathrm{r}(E_{\leq i} \setminus X)).$$

Proof. Consider running the greedy algorithm to obtain a minimum weight basis of $M \setminus X$. For every $1 \leq j \leq \ell$, the greedy algorithm includes $\mathrm{r}(E_{\leq j} \setminus X) - \mathrm{r}(E_{\leq j-1} \setminus X)$ elements of weight w_j'. Note that $\mathrm{r}(E_{\leq j} \setminus X) - \mathrm{r}(E_{\leq j-1} \setminus X) = 0$ for $j > k$ because the rank of both sets is $\mathrm{r}(E)$ by the definition of k. Now, we have

$$
\begin{aligned}
\min\{w(Y) : Y \in \mathcal{L}(X)\} &= \sum_{j=1}^{\ell} w_j'(\mathrm{r}(E_{\leq j} \setminus X) - \mathrm{r}(E_{\leq j-1} \setminus X)) = \sum_{j=1}^{k} w_j'(\mathrm{r}(E_{\leq j} \setminus X) - \mathrm{r}(E_{\leq j-1} \setminus X)) \\
&= \sum_{i=0}^{j-1} (w_{i+1}' - w_i') \sum_{j=1}^{k} (\mathrm{r}(E_{\leq j} \setminus X) - \mathrm{r}(E_{\leq j-1} \setminus X)) \\
&= \sum_{i=0}^{k-1} (w_{i+1}' - w_i') \sum_{j=i+1}^{k} (\mathrm{r}(E_{\leq j} \setminus X) - \mathrm{r}(E_{\leq j-1} \setminus X)) \\
&= \sum_{i=0}^{k-1} (w_{i+1}' - w_i')(\mathrm{r}(E_{\leq k} \setminus X) - \mathrm{r}(E_{\leq i} \setminus X)) = \sum_{i=0}^{k-1} (w_{i+1}' - w_i')(\mathrm{r}(E) - \mathrm{r}(E_{\leq i} \setminus X)). \qquad \square
\end{aligned}
$$

As a consequence, it suffices to consider only interdiction sets X such that $X \subseteq E_{\leq k-1}$, because the objective value of the lower level problem has no dependence on whether elements in $E_{\geq k}$ are interdicted. In other words, if we define

$$\bar{m} = \max\{e : e \in E_{\leq k-1}\},$$

then we can assume every $e \in X$ has $e \leq \bar{m}$. Furthermore, we can delete all elements of weight larger than w_k', because there is always a (minimum weight) basis using only elements in $E_{\leq k} \setminus X$. For matroids where we can efficiently find a minimum cost cocircuit, we compute $\bar{m}$ by performing binary search to find the smallest i such that minimum cost of a cocircuit in $E_{\leq i}$ is greater than C. Algorithm 3 can be modified to benefit from this observation by replacing the condition $d.e > m$ with $d.e > \bar{m}$ on Line 6. Algorithm 5 can be modified similarly (using the assumption that $C = \texttt{MinCut} - 1$ for the purposes of computing $\bar{m}$).

We now discuss the second preprocessing step. Memory usage requirements can be an issue for computing the upper bound f (as described in Section 2), especially if C is large. To address this we describe a way to round the costs and capacities used in the upper bound computation while ensuring that it remains an upper bound. This reduces the memory and time required at the expense of the quality of the bounds. When C is particularly large, performing this rounding step can be beneficial to the overall running time.

The way we accomplish this is as follows. We pick some constant K, replace all interdiction costs c_e by $\lfloor c_e/K \rfloor$, and replace the capacity C by $\lceil C/K \rceil$. Since we round down the costs and round up the capacity, all feasible upper-level solutions remain feasible, but some new upper-level solutions may become feasible which were not previously. The result is a relaxation of the original feasible region, and since we are dealing with a maximization problem, this produces an upper bound. In practice, we only perform this rounding on a group of instances from the literature where the costs are on the order of 10^8, and hence it would be completely impractical to use the DP algorithm without rounding. The details can be found in Section 4.1.

4 Computational results

We evaluated our algorithm computationally for the MST interdiction problem and the minimum cost MST edge blocker problem. We compare our results to two previous papers which also published computational results. Overall, our algorithm performs exceedingly well compared to past algorithms. For many instances, we improved the best run time from hours to seconds. This was enough to solve all previously unsolved instances from the literature.

We implemented our algorithm in C++, compiled it using the Clang 17 compiler, and tested it on an Intel Gold 6148 Skylake CPU @ 2.4 GHz. Our solver was limited to use at most 32 GB of RAM. Our code and instances are

open source and are available at `https://github.com/nwoeanhinnogaehr/mstisolver`. The correctness of our implementation was verified by comparing the output against algorithms such as the MIP extended formulation [3,30] or a basic brute-force algorithm whenever it was possible for these slower algorithms to solve the instances.

This section is divided into two parts. In the first, we compare the performance of our algorithms to previously published algorithms. The second examines in detail how various solver features and instance parameters affect the performance of our algorithms.

4.1   Comparison with prior works

We compare our algorithm directly to two previous works: the (mixed) integer programming formulations for the minimum cost MST edge blocker problem by Wei, Walteros, and Pajouh [30], and the branch-and-bound algorithms for the special case of MST interdiction where all interdiction costs are 1 by Bazgan, Toubaline, and Vanderpooten [3]. As the original implementations were not available, the comparison is made directly to the running times reported in the original papers. Despite this, we believe the gap in performance is significant enough to rule out the possibility of the improvements coming from compute speed or implementation quality alone. Note that with default settings, our solver uses two threads: one for branch-and-bound, and one for computing upper bounds, so the CPU time is roughly double the wall-clock time. The reason why we compute upper bounds on a separate thread is that it is difficult to determine ahead of time how difficult an instance is and thus how strong of an upper bound we should invest in computing. By doing it in parallel, we ensure that strong bounds are eventually computed for hard instances and that time is not wasted on strong bounds for easy instances. In Section 4.2, we show that using two threads in this way does not offer any significant advantage over periodically switching between branch-and-bound search and upper bound computation on a single thread, and hence it is fair to compare our algorithm to other serial algorithms. Since it is unclear if the previous papers used a parallelized implementation or whether they measured CPU time or wall-clock time, we report all running times as CPU time to ensure as fair a comparison as possible.

The minimum cost MST edge blocker tests performed by Wei, Walteros, and Pajouh were run on a 12-core Intel Xeon E5-2620 v3 CPU at 2.4GHz with 128 GB of RAM, using C++ and Gurobi 8 [30]. We tested our algorithm on the instances available for download in the online supplementary material for the paper. For these instances, the authors used a parameter $\gamma \in \{0.05, 0.3, 0.7, 0.95\}$ to set the value of R as a function of the minimum and maximum spanning tree weight. Specifically, they set $R = (\overline{w} - \underline{w})\gamma + \underline{w}$ where $\overline{w}$ is the maximum spanning tree weight and $\underline{w}$ is the minimum spanning tree weight. In our tables we only report results for $\gamma = 0.05$, because we found that for $\gamma \in \{0.3, 0.7, 0.95\}$, the optimal solution for all instances is for the leader to use a minimum cut. Our algorithm is able to very quickly handle this trivial case, solving all such instances in just a few milliseconds. The instances are grouped by number of vertices n and approximate number of edges $\tilde{m}$. Exactly how $\tilde{m}$ is determined is described in the original paper [30]. We remark that the instances available in the supplementary material do not exactly match what is described in the paper. Specifically, the data appears to include 20 instances for each value of n and $\tilde{m}$, but the paper only references 12. We tested all 20, as we found no indication of which subset of these instances were originally tested. We also adjusted the weights and costs of the downloaded instances to fall in the range $[20, 80]$, as reported in the paper. In any case, this adjustment had no measurable impact on performance.

Since these instances have weights and costs specified with 6 digits of decimal precision but our implementation only supports integer weights and costs, we scaled up all values by 10^6 so that they became integers. However, costs of this magnitude are not suitable for use with our pseudopolynomial time DP bounds, so to handle this we rounded the costs and capacities used in the DP bound as described in Section 3.4, with $K = 10^5$. This reduced the costs to integers less than 1000, which are of an appropriate magnitude for the DP algorithm. We emphasize that this does not affect the accuracy of the solution output because only the bounds are weakened; the costs and capacity used in the branching part of the algorithm are not rounded.

We report our results on these instances in Table 1. In this table, each row corresponds to the instances with a specific value of n and $\tilde{m}$. Under the headings 'BnB' and 'BS-BnB' we report the average CPU time in seconds for Algorithm 5 and binary search over the value of C in Algorithm 3, respectively. Both of our algorithms solved all instances to optimality. Under the heading 'Best from [30]' we include the best result from any algorithm described in the paper by Wei, Walteros, and Pajouh. The original data can be found in Table 2 of their paper. The 'Time' column is the best average running time in seconds for any of the three algorithms EXT, CST-1, or CES (or 7200 seconds if the time-out is reached), and the 'Opt%' column is the best percentage of instances solved to optimality. Excluded from the table are the real-world instances, for which the optimal solution to all instances is the minimum cut. The real-world instances are solved within a few seconds by our algorithms, which is slower than the time reported in [30]; the reason is that determining $\bar{m}$ for these instances is slow because the graphs are very large. If computation of $\bar{m}$ is disabled, they are solved almost instantly.

		BnB	BS-BnB	Best from [30]	
n	$\tilde{m}$	Time	Time	Time	Opt%
40	78	0.004	0.018	0.04	100
40	118	0.008	0.047	0.26	100
80	160	0.014	0.082	0.49	100
80	240	0.063	0.433	1,943.23	75
80	305	0.147	0.547	1,469.49	83
160	318	0.051	0.115	97.28	100
160	476	0.115	0.308	3,829.95	48
160	624	0.206	0.903	6,066.43	20
200	398	0.075	0.136	280.76	100
200	596	0.176	0.226	3,757.74	48
200	784	0.327	0.761	7,200	0

Table 1: Optimality percentage and average running times for instances by Wei, Walteros, and Pajouh [30]. Our algorithms (BNB and BS-BnB) solved all instances to optimality.

		BnB	Best from [3]
n	C	Time	Time
20	3	0.004	0
20	5	0.012	0.032
20	7	0.036	0.38
20	9	0.127	3.047
25	3	0.005	0
25	5	0.019	0.089
25	7	0.071	1.617
25	8	0.214	3.566
30	3	0.007	0
30	5	0.02	0.231
30	7	0.143	3.553
50	3	0.029	0.026
50	5	0.102	1.856
50	7	1.924	81.707
75	3	0.055	0.096
75	5	0.245	10.459
75	7	9.062	650.008
100	3	0.108	0.21
100	5	0.576	49.895
100	7	21.512	2,016.41
200	5	8.057	572.557
300	5	57.512	1,793.46
400	5	117.556	7,265.85

Table 2: Average running times for instances by Bazgan, Toubaline, and Vanderpooten [3]. Both approaches solved all instances to optimality.

The MST interdiction tests by Bazgan, Toubaline, and Vanderpooten were performed on a 3.4GHz processor with 3 GB of RAM, using the C programming language [3]. The exact instances they used were not available for download, but the method used to generate the instances was described precisely, so we were able to recreate the data set (up to the choice of random values). Specifically, all graphs are complete graphs on n vertices, the weights are integers chosen uniformly at random in the range $[0, 100]$, and all interdiction costs are 1. For each choice of n and interdiction budget C, the original paper generated 10 instances. To reduce any bias from the random choice of weights on the average running time we instead generated 100 instances for each choice of parameters.

We report our results on these instances in Table 2. In this table, each row corresponds to the instances with a specific value of n and C. The columns under the heading 'BnB' describe the results for Algorithm 3. Under the 'Best from [3]' heading, we include the best result reported from the original paper, across each of their four algorithms. The original data for these columns can be found in Table 2 of their paper. The meaning of the individual columns is the same as in Table 1. Note that our algorithm solved all instances to optimality, as did the best algorithm from the original paper.

In both Tables 1 and 2, our branch and bound algorithms can clearly be seen to offer a substantial improvement in running time over all previous mixed integer programming and branch and bound algorithms.

4.2 Detailed performance evaluation

While the previous section has established the competitive performance of our algorithm, the instances which appeared in the literature do not test some important cases. For example, all of the minimum cost MST edge blocker instances are relatively sparse, and all of the MST interdiction instances are complete graphs with unit costs and a very small interdiction budget. In this section we test some newly generated instances to better understand how the performance of our algorithm depends on various qualities of an instance. We also perform a feature knockout test to determine which algorithm features are most important for good performance.

We chose to focus on generating MST interdiction instances which are small (in terms of n and m) but which take a long time to solve using current solvers. In fact, many of these new instances with only 20 vertices take significantly longer to solve than instances from the literature with 400 vertices. The 1024 new instances were generated as follows. For each choice of $n \in \{10, 15, 20, 25\}$, $\gamma \in \{0.25, 0.5, 0.75, 1\}$, $d \in \{0.5, 0.66, 0.83, 1\}$, $c_{\max} \in$

$\{1, 100, 1000, 10000\}$, and $w_{\max} \in \{2, 100, 10000, 1000000\}$, we generated one instance. The instance has n vertices, $m = \lfloor d\binom{n}{2} \rfloor$ edges (chosen uniformly at random), integer costs c uniformly distributed in $[1, c_{\max}]$, integer weights w uniformly distributed in $[1, w_{\max}]$ and capacity $\lfloor \gamma(M-1) \rfloor$ where M is the cost of the global minimum cut for costs c. Note that $M - 1$ is the largest nontrivial capacity, as any larger would produce an instance in which the leader is able to cut the graph.

In Table 3, we report the results for these new instances. The rows of the table are split into five sections, each with four rows; in each section the instances are grouped based on the value of one of the five instance parameters n, γ, d, $c_{\max}$, or $w_{\max}$. That is, each section reports results for all instances, but instances are aggregated differently in different sections.

The first four data columns, 'Opt%', 'Time', 'MaxTime' indicate the percentage of instances solved to optimality, average CPU time, and maximum CPU time, respectively. All instances were tested with a 1 hour (3600 second) CPU time limit. For instances where the limit is reached, the time limit is used in place of the running time. The next four columns describe the quality of bounds: 'LB' and 'MaxLB' are the average and maximum percent gap between the greedy lower bound and the optimal solution, and 'UB' and 'MaxUB' are the average and maximum percent gap between the (strengthened) upper bound and the optimal solution. The gap percentage is computed as $100 \cdot |z^* - z|/z^*$ where z is the approximate solution weight (lower or upper bound) and z^* is the optimal solution weight. The 'UBTime' column reports the average number of seconds required to compute the upper bound $f(1, C, \phi(\emptyset))$. For any p, the time required to compute $f_p(1, C, \phi(\emptyset), \emptyset)$ can be accurately estimated by multiplying this value by 2^p. Only instances which were solved to optimality are included in the average for the last four columns.

The results indicate that the number of vertices n is the best predictor of instance difficulty: while instances with $n = 10$ are all solved within 20 milliseconds, around 7% of the instances with $n = 25$ could not be solved within 1 hour. However, to be hard an instance must have high density d and high capacity factor γ. This matches the behaviour seen in instances from the literature. The instances in Table 1 are solved very quickly, and the graphs are very sparse (i.e., d is small). On the other hand, if γ is small, as is the case in Table 2, instances are still largely tractable as well. The parameters $c_{\max}$ and $w_{\max}$ have less impact on the performance. Instances with smaller $c_{\max}$ are typically solved faster because the DP table is smaller and can be computed more quickly. On the other hand, instances with smaller $w_{\max}$ are somewhat harder, but only by a small factor; we are not sure why this is the case. We remark that the number of branch-and-bound nodes is often on the order of 10^9 for the hardest instances. This is expected since our algorithm only needs $O(m)$ time per node.

Surprisingly, we found that across all instances, the greedy lower bound is 1.48% from optimal on average, and is never worse than 17.82% from optimal. These statistics hold for instances from the literature as well. In Section 3.1 we saw that the worst case can be arbitrarily bad even for a graph on four vertices, but evidently the performance on random graphs is much better. Therefore, this greedy algorithm alone could serve as a very strong heuristic in applications where an exact algorithm is not needed.

As for the strengthened DP upper bounds, they have an average optimality gap of 14.76% at the root node, and are never worse than 68.24%. The parameter which has the largest correlation with the time required to compute the bounds is $c_{\max}$, because the running time is directly a function of it. The running time also depends on m, but as the magnitude of m is much smaller, it has less of an impact. While there is no clear connection between n and the bound strength, all other parameters do appear to be correlated with the bound strength. In Section 2.3.1 we showed an example with an optimality gap linear in n, but evidently this case is not commonly seen in random graphs. This offers some indication as to why the branch-and-bound algorithm performs so well: although it is possible for the upper bounds to be weak, the bounds are used across a large number of subproblems. Due to the good average-case performance, it appears to be quite challenging to construct an instance where the bounds are consistently weak for all subproblems. The hardest instances we currently know of have $d = 1$, $\gamma = 1$, and very large $c_{\max}$. However, these instances are still far from the theoretical worst case.

We now examine the impact of different solver features on performance in a knockout test. The features considered in the test are: (1) the DP upper bound, (2) the strengthened DP upper bound, (3) the greedy lower bound, and (4) the computation of branch-and-bound search and upper bounds in parallel. For these tests we again used a 3600 CPU second time limit. The results are summarized in Table 4. As usual, the 'Opt%' column is the percentage of instances solved to optimality, and the 'Time' column is the average CPU time in seconds. Instances which reached the time limit are counted as having taken 3600 seconds. The aggregation of instances into sections is the same as in Table 3.

It can easily been seen from the table that disabling the upper bound entirely or disabling the strengthened upper bound causes a clear reduction in performance, especially on more difficult instances. This is true even considering that without the upper bound test, our branch-and-bound implementation is able to enumerate nodes

n	Opt%	Time	MaxTime	LB	MaxLB	UB	MaxUB	UBTime
10	100	0	0.02	1.11	16	9.09	44.75	0.0003
15	100	0.13	10.04	1.61	15.88	16.78	68.24	0.0116
20	99.61	70.31	3,600	1.52	17.25	16.75	54.96	0.0388
25	92.97	380.99	3,600	1.67	17.82	16.4	45.95	0.0666
γ								
0.25	100	0.01	0.32	0.84	10	4.94	25.71	0.0038
0.5	100	4.49	266.43	1.28	15.88	13.14	45.76	0.0275
0.75	98.44	116.25	3,600	1.36	17.25	18.05	60.27	0.0469
1.0	94.14	330.69	3,600	2.47	17.82	20.97	68.24	0.0554
d								
0.5	100	0.12	8.7	0.91	15.92	9.3	43.48	0.0138
0.66	100	5.67	381.7	1.11	14.15	13.57	45.95	0.0291
0.83	99.22	77.84	3,600	1.78	17.25	16.82	46.01	0.045
1.0	93.36	367.82	3,600	2.15	17.82	20.03	68.24	0.0523
$c_{\max}$								
1	98.83	69.47	3,600	1.34	17.82	9.35	29.92	0.0018
100	98.83	106.2	3,600	1.52	16	15.38	45.76	0.0029
1000	97.66	129.98	3,600	1.58	17.25	19.36	60.27	0.0183
10000	97.27	145.79	3,600	1.45	11.4	24.59	68.24	0.2227
$w_{\max}$								
2	96.48	177.85	3,600	0.63	10	8.88	31.25	0.0179
100	98.44	116.14	3,600	1.78	15.92	17.67	68.24	0.0301
10000	99.22	68.68	3,600	1.86	17.82	18.79	54.96	0.0593
1000000	98.44	88.77	3,600	1.61	14.15	19.37	60.27	0.0508

Table 3: Aggregated statistics for new instances.

	All features		No UB		Weak UB		No LB		Single thread	
n	Opt%	Time	Opt%	Time	Opt%	Time	Opt%	Time	Opt%	Time
10	100	0	100	0	100	0	100	0	100	0
15	100	0.13	100	0.24	100	0.1	100	0.13	100	0.15
20	99.61	70.31	97.66	166.93	99.61	77.06	100	68.09	100	74.86
25	92.97	380.99	84.31	680.48	88.67	515.15	92.58	383.81	91.8	352.55
γ										
0.25	100	0.01	100	0.01	100	0.01	100	0.01	100	0.01
0.5	100	4.49	100	12.63	100	6.64	100	4.54	100	4.63
0.75	98.44	116.25	93.75	260.86	96.88	167.94	98.05	118.12	97.66	104.67
1.0	94.14	330.69	88.24	573.75	91.41	417.74	94.53	329.36	94.14	318.27
d										
0.5	100	0.12	100	0.28	100	0.07	100	0.11	100	0.11
0.66	100	5.67	100	24.75	100	6.77	100	5.55	100	5.69
0.83	99.22	77.84	96.09	219.21	97.66	130.56	99.22	79.26	98.44	79
1.0	93.36	367.82	85.88	603.11	90.63	454.93	93.36	367.12	93.36	342.77
$c_{\max}$										
1	98.83	69.47	94.14	234.28	98.05	90.65	98.83	69.7	98.83	72.22
100	98.83	106.2	95.7	215.68	97.27	165.89	98.83	107.26	98.05	99.83
1000	97.66	129.98	96.09	200.52	96.88	168.02	97.27	131.08	97.27	120.84
10000	97.27	145.79	96.08	195.27	96.09	167.76	97.66	144	97.66	134.69
$w_{\max}$										
2	96.48	177.85	96.47	178.72	96.48	171.28	96.48	177.75	96.88	174.61
100	98.44	116.14	94.92	231.17	96.48	155.69	98.44	116.67	97.66	112.21
10000	99.22	68.68	94.92	226.93	97.27	133.75	99.22	68.83	99.22	60.93
1000000	98.44	88.77	95.7	208.89	98.05	131.6	98.44	88.79	98.05	79.82

Table 4: Knockout test.

about two times faster. On the other hand, disabling the greedy lower bound has no clear impact on the optimality percentage and the running time. This may be surprising considering how we found in Table 3 that the greedy lower bound has exceptionally good performance. The lack of any significant impact on the exact algorithm suggests that typically the branch-and-bound scheme very quickly finds solutions of quality comparable to the greedy lower bound.

For the single-threaded variant of the solver (feature column 'Single thread'), the implementation specifically does the following. A timer is set initially to expire after 0.01 seconds. We then repeat the following until the problem is solved. First, branch-and-bound search begins. When the timer expires, it is reset and the solver switches to performing upper bound computation. When the timer expires again, the upper bound computation is interrupted, the length of the timer is doubled, and the process repeats (i.e., branch-and-bound resumes, but with double the amount of time on the timer). This method of alternating between search and bounds ensures easy instances are solved quickly by branching (without spending a lot of time on unnecessary bound computations) but for hard instances we spend sufficient time on improving the bounds. As can be seen from the table, this change to the solver only has a minimal impact on the performance and hence it is fair to compare our (technically multi-threaded) algorithm to other single-threaded algorithms, especially considering the wide gaps in performance seen in Section 4.1.

5 The matroid inclusion-interdiction problem

Typically, interdiction problems consider an adversary who is excluding some structure from being used in the solution to the lower level problem. However, besides being motivated by applications, the choice of excluding rather then *including* is perhaps arbitrary. In this section we consider this dual problem in which the leader forces certain elements to be included in the follower's basis, and show that the two problems are connected through matroid duality.

Let $M = (E, \mathcal{I})$ be a matroid and let $M^* = (E, \mathcal{I}^*)$ be its dual. Let $\mathcal{B}$ be the set of bases of M, and let $\mathcal{B}^*$ be the set of bases of M^*. Let $w \in \mathbb{Z}^E$. By the definition of the dual matroid, we know that $Y \in \mathcal{B}^*$ if and only if $E \setminus Y \in \mathcal{B}$. This immediately yields the following.

Proposition 5.1.

$$\max\{\min\{w(Y) : Y \in \mathcal{B}^*, Y \subseteq E \setminus X\} : X \in \mathcal{U}\} \tag{5.1}$$

$$= w(E) - \min\{\max\{w(Y) : Y \in \mathcal{B}, X \subseteq Y\} : X \in \mathcal{U}\}. \tag{5.2}$$

This proposition connects these two types of interdiction problems on matroids: (5.1), in which the leader can force elements to be excluded from a basis, and (5.2), in which in which the leader can force elements to be included in a basis. We will call this inclusion variant of the problem the *matroid inclusion-interdiction problem*.

Since our exact algorithms do not require non-negative weights, Proposition 5.1 implies that we can solve the matroid inclusion-interdiction problem on a matroid M whenever we can solve the matroid interdiction problem on the dual of M. The same holds for the minimum cost blocker variants. The following is now immediate from the fact that both partition matroids and graphic matroids from planar graphs are self-dual.

Corollary 5.2. *There is a pseudopolynomial time algorithm for the partition matroid inclusion-interdiction problem.*

Corollary 5.3. *Algorithm 3 (with bounds computed by Algorithm 1) can be used to solve the planar graphic matroid inclusion-interdiction problem.*

We do not know how to apply our techniques from Section 2 to get bounds when M is the bond matroid of a non-planar graph, so it remains open to derive good bounds for matroid inclusion-interdiction on non-planar graphic matroids or in even more general settings.

6 Conclusion

We have presented new combinatorial algorithms and theoretical results for matroid interdiction and related problems. In the special case of MST interdiction, our algorithms achieve state-of-the-art computational performance, improving on the previous best results by a few orders of magnitude.

There are a number of interesting directions for future work. It is natural to consider the generalization of these methods to the min-max problem where the follower wants to find a maximum weight *independent set* (as

opposed to a maximum weight basis, as has been the focus of this paper); this would provide a connection with the rank reduction problem [14]. We hope that our bound framework will be applied to other matroids (some obvious next targets are laminar matroids, bond matroids, transversal matroids, and gammoids), and to other interdiction problems more broadly.

Acknowledgements

References

[1] Amato G, Cattaneo G, Italiano GF (1997) Experimental analysis of dynamic minimum spanning tree algorithms. In: SODA, Citeseer, vol 97, pp 314–323

[2] Bader DA, Burkhardt P (2019) A simple and efficient algorithm for finding minimum spanning tree replacement edges. arXiv preprint arXiv:190803473

[3] Bazgan C, Toubaline S, Vanderpooten D (2012) Efficient determination of the k most vital edges for the minimum spanning tree problem. Computers & Operations Research 39(11):2888–2898

[4] Bazgan C, Toubaline S, Vanderpooten D (2013) Critical edges/nodes for the minimum spanning tree problem: complexity and approximation. Journal of Combinatorial Optimization 26:178–189

[5] Cattaneo G, Faruolo P, Petrillo UF, Italiano GF (2010) Maintaining dynamic minimum spanning trees: An experimental study. Discrete Applied Mathematics 158(5):404–425

[6] Chestnut SR, Zenklusen R (2017) Interdicting structured combinatorial optimization problems with {0,1}-objectives. Mathematics of Operations Research 42(1):144–166

[7] Cook WJ, Cunningham WH, Pulleyblank WR, Schrijver A (1998) Combinatorial optimization. John Wiley and Sons, New York

[8] Frederickson GN, Solis-Oba R (1998) Algorithms for measuring perturbability in matroid optimization. Combinatorica 18:503–518

[9] Frederickson GN, Solis-Oba R (1999) Increasing the weight of minimum spanning trees. Journal of Algorithms 33(2):244–266

[10] Hausbrandt N, Bachtler O, Ruzika S, Sch äfer LE (2024) Parametric matroid interdiction. Discrete Optimization 51:100823

[11] Holm J, De Lichtenberg K, Thorup M (2001) Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. Journal of the ACM (JACM) 48(4):723–760

[12] Hsu LH, Jan RH, Lee YC, Hung CN, Chern MS, et al. (1991) Finding the most vital edge with respect to minimum spanning tree in weighted graphs. Information Processing Letters 39(5):277–281

[13] Israeli E, Wood RK (2002) Shortest-path network interdiction. Networks: An International Journal 40(2):97–111

[14] Joret G, Vetta A (2015) Reducing the rank of a matroid. Discrete Mathematics and Theoretical Computer Science 17(2):143–156

[15] Ketkov SS, Prokopyev OA (2024) On a class of interdiction problems with partition matroids: complexity and polynomial-time algorithms. arXiv preprint arXiv:240112010

[16] Kleinert T, Labbé M, Ljubić I, Schmidt M (2021) A survey on mixed-integer programming techniques in bilevel optimization. EURO Journal on Computational Optimization 9:100007

[17] Liang W (2001) Finding the k most vital edges with respect to minimum spanning trees for fixed k. Discrete Applied Mathematics 113(2-3):319–327

[18] Libura M (1991) Sensitivity analysis for minimum weight base of a matroid. Control and Cybernetics 20(3):7–24

[19] Lin KC, Chern MS (1993) The most vital edges in the minimum spanning tree problem. Information Processing Letters 45(1):25–31

[20] Linhares A, Swamy C (2017) Improved algorithms for MST and metric-TSP interdiction. arXiv preprint arXiv:170600034

[21] Pisinger D, Toth P (1998) Knapsack problems. Handbook of Combinatorial Optimization: Volume 1–3 pp 299–428

[22] Ribeiro CC, Toso RF (2007) Experimental analysis of algorithms for updating minimum spanning trees on graphs subject to changes on edge weights. In: Experimental Algorithms: 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007. Proceedings 6, Springer, pp 393–405

[23] Salazar-Zendeja L (2022) Models and algorithms for the minimum spanning tree interdiction problem. PhD thesis, Centrale Lille Institut

[24] Schrijver A (2002) On the history of the transportation and maximum flow problems. Mathematical programming 91:437–445

[25] Shen H (1999) Finding the k most vital edges with respect to minimum spanning tree. Acta Informatica 36:405–424

[26] Smith JC, Song Y (2020) A survey of network interdiction models and algorithms. European Journal of Operational Research 283(3):797–811

[27] Vardy A (1997) The intractability of computing the minimum distance of a code. IEEE Transactions on Information Theory 43(6):1757–1766

[28] Von Stackelberg H (1952) The theory of the market economy. Oxford University Press, England

[29] Wei N, Walteros JL (2022) Integer programming methods for solving binary interdiction games. European Journal of Operational Research 302(2):456–469

[30] Wei N, Walteros JL, Pajouh FM (2021) Integer programming formulations for minimum spanning tree interdiction. INFORMS Journal on Computing 33(4):1461–1480

[31] Weninger N, Fukasawa R (2023) A fast combinatorial algorithm for the bilevel knapsack problem with interdiction constraints. In: International Conference on Integer Programming and Combinatorial Optimization, Springer, pp 438–452

[32] Zenklusen R (2015) An O(1)-approximation for minimum spanning tree interdiction. In: IEEE 56th Annual Symposium on Foundations of Computer Science, IEEE, pp 709–728

A   Proof of Theorem 2.7

Proof. First we show that $P_{\text{bound}}(\delta, \phi, \pi)$ is satisfied. Let $r \in \{0, \ldots, C\}$, $s \in T$ and $i \in E$ such that $c_i \leq r$. We claim that for all $X \in \mathcal{U}$,

$$i \leq p \implies F_{M_1}(X_{<i} \cup \{i\}) - F_{M_1}(X_{<i}) = F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}), \tag{A.1}$$

$$i > p \implies F_{M_2}(X_{<i} \cup \{i\}) - F_{M_2}(X_{<i}) = F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}). \tag{A.2}$$

To see this, observe that for any $X' \subseteq E$, $F_M(X') = F_{M_1}(X') + F_{M_2}(X')$ because the independent sets of M_1 and M_2 are disjoint. Now, considering what elements can be independent in each matroid, if $i \leq p$, then $F_{M_2}(X_{<i} \cup \{i\}) = F_{M_2}(\emptyset) = F_{M_2}(X_{<i})$ and if $i > p$ then $F_{M_1}(X_{<i} \cup \{i\}) = F_{M_1}(X \cap \{1, \ldots, p\}) = F_{M_1}(X_{<i})$, establishing the claim.

Now we establish $P_{\text{bound}}(\delta, \phi, \pi)$. If $i \leq p$, then

$$\delta(i, r, s) = \delta_1(i, r, s)$$
$$\geq \max\{F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}) : X \in \mathcal{U}, C - c(X_{<i}) = r, \phi_1(X_{<i}) = s\}$$
$$\text{(by } P_{\text{bound}}(\delta_1, \phi_1, \pi_1) \text{ and Eq. (A.1))}$$
$$= \max\{F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}) : X \in \mathcal{U}, C - c(X_{<i}) = r, \phi(X_{<i}) = s\}.$$
$$\text{(because } i \leq p \text{ implies } \phi(X_{<i}) = \phi_1(X_{<i}))$$

So, assume $i > p$. If $s \in T_2$ then

$$\delta(i, r, s) = \delta_2(i, r, s)$$
$$\geq \max\{F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}) : X \in \mathcal{U}, C - c(X_{<i}) = r, \phi_2(X_{<i}) = s\}$$
$$\text{(by } P_{\text{bound}}(\delta_2, \phi_2, \pi_2) \text{ and Eq. (A.2))}$$
$$\geq \max\{F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}) : X \in \mathcal{U}, C - c(X_{<i}) = r, \phi(X_{<i}) = s\}$$

where the last inequality follows because if $X \in \mathcal{U}$ satisfies $\phi(X_{<i}) = s$, then since $s \in T_2$, we know $\phi(X_{<i}) = \phi_2(X_{<i})$. Finally, if $i > p$ and $s \in T_1$, then

$$\delta(i, r, s) = \delta_2(i, r, \phi_2(\emptyset))$$
$$\geq \max\{F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}) : X \in \mathcal{U}, C - c(X_{<i}) = r, \phi_2(X_{<i}) = \phi_2(\emptyset)\}$$
$$\text{(by } P_{\text{bound}}(\delta_2, \phi_2, \pi_2) \text{ and Eq. (A.2))}$$
$$\geq \max\{F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}) : X \in \mathcal{U}, C - c(X_{<i}) = r, \phi(X_{<i}) = s\}$$

where the last inequality follows because if $X \in \mathcal{U}$ satisfies $\phi(X_{<i}) = s$, then since $s \in T_1$, we have that $X \subseteq \{1, \ldots, p\}$. Hence, $\phi_2(\emptyset) = \phi_2(X_{<i})$ by Eq. (2.10).

Now we show that $P_{\text{succ}}(\phi, \pi)$ holds. We consider four cases:

1. If $X_{<i} \subseteq \{1, \ldots, p\}$ and $i \leq p$: since $P_{\text{succ}}(\phi_1, \pi_1)$ holds,

$$\phi(X_{<i} \cup \{i\}) = \phi_1(X_{<i} \cup \{i\}) = \pi_1(i, \phi_1(X_{<i})) = \pi(i, \phi(X_{<i})).$$

2. If $X_{<i} \subseteq \{1, \ldots, p\}$ and $i > p$: by $P_{\text{succ}}(\phi_2, \pi_2)$ and Eq. (2.10),

$$\phi(X_{<i} \cup \{i\}) = \phi_2(X_{<i} \cup \{i\}) = \pi_2(i, \phi_2(X_{<i})) = \pi_2(i, \phi_2(\emptyset)) = \pi(i, \phi(X_{<i})).$$

3. If $X_{<i} \not\subseteq \{1, \ldots, p\}$ and $i \leq p$: this is impossible, as $X_{<i} \not\subseteq \{1, \ldots, p\}$ implies $i > p$.

4. If $X_{<i} \not\subseteq \{1, \ldots, p\}$ and $i > p$: since $P_{\text{succ}}(\phi_2, \pi_2)$ holds,

$$\phi(X_{<i} \cup \{i\}) = \phi_2(X_{<i} \cup \{i\}) = \pi_2(i, \phi_2(X_{<i})) = \pi(i, \phi(X_{<i})).$$

Finally, we show that if there exists some $\alpha \geq 0$ such that $P_{\text{approx}}(\delta_j, \phi_j, \pi_j, \alpha)$ is satisfied for both $j = 1$ and $j = 2$, then $P_{\text{approx}}(\delta, \phi, \pi, \alpha)$ is satisfied. Let $X \subseteq \mathcal{U}, i \in E$. If $i \leq p$, then by $P_{\text{approx}}(\delta_1, \phi_1, \pi_1, \alpha)$ and Eq. (A.1),

$$\delta(i, C - c(X_{<i}), \phi(X_{<i})) = \delta_1(i, C - c(X_{<i}), \phi(X_{<i}))$$
$$\leq \alpha + F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}).$$

So, assume $i > p$. Then, if $X \not\subseteq \{1, \ldots, p\}$, then by $P_{\text{approx}}(\delta_2, \phi_2, \pi_2, \alpha)$ and Eq. (A.2),

$$\delta(i, C - c(X_{<i}), \phi(X_{<i})) = \delta_2(i, C - c(X_{<i}), \phi(X_{<i}))$$
$$\leq \alpha + F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}).$$

Otherwise, if $X \subseteq \{1, \ldots, p\}$, then by Eq. (2.10), $P_{\text{approx}}(\delta_2, \phi_2, \pi_2, \alpha)$ and Eq. (A.2),

$$\delta(i, C - c(X_{<i}), \phi(X_{<i})) = \delta_2(i, C - c(X_{<i}), \phi_2(\emptyset))$$
$$= \delta_2(i, C - c(X_{<i}), \phi_2(X_{<i}))$$
$$\leq \alpha + F_M(X_{<i} \cup \{i\}) - F_M(X_{<i}). \qquad \square$$