Faster exact and approximation algorithms for packing and covering matroids via push-relabel

Kent Quanrud*

Abstract

Matroids are a fundamental object of study in combinatorial optimization. Three closely related and important problems involving matroids are maximizing the size of the union of k independent sets (that is, *k-fold matroid union*), computing k disjoint bases (a.k.a. *matroid base packing*), and covering the elements by k bases (a.k.a. *matroid base covering*). These problems generalize naturally to integral and real-valued capacities on the elements. This work develops faster exact and/or approximation problems for these and some other closely related problems such as optimal reinforcement and matroid membership. We obtain improved running times both for general matroids in the independence oracle model and for the graphic matroid. The main thrust of our improvements comes from developing a faster and unifying *push-relabel* algorithm for the integer-capacitated versions of these problems, building on previous work by Frank and Miklós [24]. We then build on this algorithm in two directions. First we develop a faster augmenting path subroutine for k-fold matroid union that, when appended to an approximation version of the push-relabel algorithm, gives a faster exact algorithm for some parameters of k. In particular we obtain a subquadratic-query running time in the uncapacitated setting for the three basic problems listed above. We also obtain faster approximation algorithms for these problems with real-valued capacities by reducing to small integral capacities via randomized rounding. To this end, we develop a new randomized rounding technique for base covering problems in matroids that may also be of independent interest.

1 Introduction

Matroids are a fundamental object of study in combinatorial optimization. Three closely related and important problems involving matroids are maximizing the size of the union of k independent sets (also known as *matroid union* where we are taking k copies of the same matroid), computing k disjoint bases (a.k.a. *matroid base packing*), and covering the groundset by k bases (a.k.a. *matroid base covering*). These problems generalize naturally to integral and real-valued capacities as we explain later. This work develops faster exact and/or approximation problems for these and some other closely related problems. The main thrust of our improvements comes from developing a faster and unifying *push-relabel* algorithm for the integer-capacitated versions of these problems.

The push-relabel framework is most commonly associated with maximum flow. The push-relabel framework for flow, introduced by [37] and then improved and generalized by [36] (cf. [38]), is an elegant framework that has lead to new algorithms and perspectives for max-flow (e.g., [1, 12, 63]) and related problems such as minimum cost flow [35], parametric flow [33] and directed minimum cuts [42, 43] (to name a few). Empirical work has also shown that push-relabel algorithms can work well in practice (e.g., [13, 14]). Loosely speaking, the push-relabel algorithm for flow departs from previous algorithms by making local improvements, "pushing" flow along one edge at a time, rather than augmenting along paths. Rather than maintaining an (s,t)-flow (conserving flow at non-terminals), the push-relabel algorithm maintains a relaxation of flow called *preflows* that allow for positive surplus at non-terminal vertices. Vertex labels, assigning integer levels to each vertex, are introduced to guide the push operations and obtain polynomial running times.

The push-relabel concept has been extended (in a much more abstract form) to more general combinatorial problems including submodular flow [25], submodular intersection [26], submodular minimization [21, 22, 48], and other related problems [46, 47]. More recently, Frank and Miklós [24] developed simpler strongly polynomial-time push-relabel algorithms for abstract combinatorial optimization problems ranging from matroid partition to submodular flow. Their work builds on and helps unify some of these past results. Our work directly builds on [24]. Focusing on the class of matroid problems listed above, we contribute both structural observations and algorithmic techniques to their framework that accelerate exact algorithms and also extend the framework to yield fast approximation algorithms.

*Dept. of Computer Science, Purdue University, West Lafayette, IN 47907. krq@purdue.edu. kentquanrud.com. Supported in part by NSF grant CCF-2129816.

We complement these techniques (that fall within the push-relabel framework) in two different ways. The first is to design an augmenting path subroutine specialized to integer-capacitated k-fold matroid union, which can be used augment an approximate solution produced (very quickly) by the push-relabel framework to an optimal one. We point out that this augmenting-path subroutine requires additional structure provided by the push-relabel framework to work. The augmenting paths lead to faster running time for certain ranges of parameters, including a subquadratic-query running time for the uncapacitated setting. The second extension is based on random sampling, which is used to reduce approximation problems for real-capacitated problems to approximation problems with (small) integer capacities. Some of these randomized rounding techniques are new while others are provided by previous work [51]. The reduced setting with integer capacities is particularly well-suited for the push-relabel algorithms that we develop and this leads to a series of approximation algorithms with nearly linear oracle and running time complexity for real-valued capacities.

Given that many of these problems have classically been studied primarily from the perspective of exact algorithms, we take a moment to motivate our additional interest in fast approximation algorithms. One motivation comes from the trend of large datasets and the need for algorithms that can scale with the data. For sufficiently large inputs, one may be more willing to exchange exact correctness for a polynomial reduction in the running time, especially if we can still control the worst-case approximation factor. Fast approximation algorithms are also utilized as subroutines in other algorithms in situations where weaker approximation factors suffice. We list a few examples from graph algorithms. Benczúr and Karger's [4] graph sparsification algorithm needs only a loose approximation of so-called "edge strengths" to generate its sampling probabilities. The linear time minimum cut algorithm of [50] begins with an approximate tree packing. A large wave of recent graph algorithms are based on expander decompositions [59], which is based on fast approximation algorithms for sparsest cut. The recent breakthrough in max flow uses fast subroutines approximating minimum ratio cycles [10]. These applications do not require exact solutions to their subproblems. (We note that for some of these applications, there are other considerations besides static speed, such as fast dynamic updates.) Of direct relevance to this work is a recent result [57] which generalizes graph and hypergraph sparsifiers to quotient-preserving sparsifiers to matroids and submodular functions. The sparsification algorithm builds on the ideas in this paper and uses fast matroid push-relabel approximation algorithms to build a hierarchical decomposition of the elements of a matroid, from which the sampling probabilities are then inferred.

1.1 Outline of results We now outline the improved running times obtained in this work. For the sake of brevity we defer more detailed descriptions of each problem to later in the paper when they are analyzed in full. For each we state the new running time and only the most competitive and directly comparable running times in the literature (at the time of this work). We provide pointers to the full theorem statements for each result. Additional background is described in appendix A. The problems we discuss are all basic and well-studied problems in matroid optimization and can be found in [60], which we refer to for additional background. We pose these problems for general matroids in the oracle model and for the graphic matroid (i.e., forests of an undirected graph).

We briefly mention some preliminaries needed to describe the results; additional preliminaries including further notation and relevant definitions are provided in section 2. For matroid problems, we let n denote the number of elements, and r the rank of the matroid. We assume access to an *independence oracle* to which we can query if a set of elements S is independent. We adopt the standard and simplifying convention of counting, as part of the independence query, the $O(|S|)$ work one typically needs to assemble and transmit the set S to the oracle. For graph problems, we let m denote the number of (distinct) edges and n the number of vertices in the graph. When the graph has integer edges capacities, we let U denote the total capacity.

The results come in one of three flavors: (1) exact algorithms for integral capacities that produce integral solutions, (2) integral approximation algorithms for integral capacities that produce integral solutions, and (3) approximate decision algorithms for real-valued capacities. Type (1) usually comes with two (incomparable) running times — one given directly by the push-relabel algorithm, and one combining push-relabel with augmenting paths. Algorithms of the second type are typically truncated versions of the corresponding push-relabel algorithms of type (1). Algorithms for the third type are typically obtained by reducing to problems of the second type via random sampling. All the algorithms for integer capacities — of type (1) and type (2) — are deterministic and construct integral and mutually certifying primal and dual solutions. The algorithms for real-valued capacities (type (3)) are randomized Monte Carlo algorithms that succeed with high probability. They construct an approximate and integral dual solution but not a primal solution.

We proceed to present the results. We will present the results for matroids in the independence oracle model first and then a parallel set of results for the graphic matroid.

In the uncapacitated setting, the *k-fold matroid union* problem asks for k bases $B_1, \ldots, B_k$ maximizing the size of their union. This naturally extends to integer capacities where we now count each element in $B_1, \ldots, B_k$ with multiplicity up to its capacity. We first obtain a running time of $O(n + \mathrm{OPT}\, r \log(kr))$ independence queries, where OPT denotes the optimum value, via the push-relabel algorithm (theorem 3.2). We also obtain a running time of $O\left(n + \mathrm{OPT}\sqrt{n' \log(kr)}\right)$ independence queries, where $n' = \min\{n + \mathrm{OPT}\log(r), rk\log(kr)\}$, by combining the push-relabel algorithm with augmenting paths (theorem 8.1). Note that in the uncapacitated setting, $\mathrm{OPT} \leq n$ and $n' \leq n\log(r)$, so the second running time is at most $O(n^{3/2})$ queries for any choice of parameters k and r. These running times are to be compared to $O\left(\left(\mathrm{OPT}^{3/2} + k\right)nQ + \mathrm{OPT}^{1/2}\, kn\right)$ for the uncapacitated setting [15], where Q denotes the time for a query to an independence oracle. One can also reduce uncapacitated k-fold matroid union to unweighted matroid intersection (with nk elements and rank OPT); this yields a randomized running time of $\tilde{O}\left(nk\sqrt{\mathrm{OPT}}\right)$ queries [5].

We also consider approximation algorithms for capacitated k-fold matroid union. Let $\epsilon \in (0, 1)$ be a given parameter; the goal is to achieve an objective value of at least $(1 - \epsilon)\,\mathrm{OPT}$. For integer capacities we obtain a running time of $O(n + \mathrm{OPT}\log(kr)/\epsilon)$ independence queries (theorem 3.3). The algorithm produces both integral primal and dual solution which mutually certify that they are $(1 \pm \epsilon)$-approximately optimal. As per comparable running times, while [15] does not consider approximations explicitly, [15] implicitly contains a $(1 - \epsilon)$-approximation algorithm for the uncapacitated setting with time $O(knQ + (\mathrm{OPT}\, nQ + kn)/\epsilon)$. One can also reduce uncapacitated k-fold matroid union to approximate matroid intersection, yielding a running time of $\tilde{O}\left(nk\sqrt{\mathrm{OPT}}/\epsilon\right)$ queries [5]. The k-fold matroid union problem generalizes to real-valued capacities where one instead optimizes over fractional combinations of k bases. For this setting we develop a randomized algorithm with randomized running time bounded by $O\left(n + (\mathrm{OPT}/k)\log(n)\log\left(r\log(n)/\epsilon^2\right)/\epsilon^3\right)$ independence queries. We are not aware of comparable algorithms in the literature.

The next problem we consider is base packing, which we describe for integer capacities. The goal is to compute k bases so that no element appears in more bases than its capacity. It is easy to see that (exactly solving) the problem reduces to k-fold matroid union, hence we obtain the same running times as listed above. In addition to the [15] result mentioned above, one can also compare to a Las Vegas randomized algorithm for the uncapacitated setting that runs in randomized $O\left(n + r^3k^{5/2}\log^{3/2}(kr)\right)$ independence queries with high probability [51]. [51] also gives a deterministic algorithm running in $O\left(n + (rk)^3\log^{O(1)}(kr)\right)$.

We also consider approximate base packing. The goal is to either pack $(1 - \epsilon)k$ bases or certify that there is no packing of $(1 + \epsilon)k$ bases, for a parameter $\epsilon \in (0, 1)$. Here approximate k-base packing does not reduce directly to approximate k-fold matroid union, although we use similar techniques. We obtain a running time of $O(n + kr\log(n)\log(kr)/\epsilon)$ (theorem 4.1). This is to be compared to a randomized Monte Carlo algorithm for the uncapacitated setting running in $\tilde{O}\left(n + r^3k/\epsilon^3\right)$ independence queries [51]. For real-valued capacities we obtain a randomized Monte Carlo algorithm running in $O\left(n + r\log^2(n)(\log\log(n) + \log(1/\epsilon))/\epsilon^3\right)$ independence queries that succeeds with high probability (theorem 7.4, page 20). This can be compared to a deterministic $\tilde{O}\left(nk/\epsilon^2\right)$-query time algorithm from [9] or a randomized Monte Carlo algorithm running in $O\left(n + r^3\log^{O(1)}(kr)/\epsilon^5\right)$ independence queries [51]. We believe that ideas from [9, 51] can be combined to also obtain a randomized Monte Carlo algorithm running in $\tilde{O}\left(n + r/\epsilon^4\right)$ independence queries. Our randomized algorithm for fixed k can be converted to a randomized algorithm to compute the maximum value k (which we call the *matroid strength*) via a modified binary search. The running time we obtain is slightly better than one gets from a straightforward application of binary search (theorem 7.5, page 21).

Next we discuss base covering. Here the capacities are interpreted as lower bounds. The goal is to compute k bases so that for each element, the number of bases containing each element is at least the lower capacity of that element. If that is not possible then one expects a dual certificate of feasibility. As with packing, exact base covering can be solved by k-fold matroid union so we inherit those running times. To the best of our knowledge, the best comparable running time is that of the k-fold matroid union algorithm of [15] for the uncapacitated setting, mentioned above. (In particular we are not aware of developments for base covering analogous to [51]

or [9] which focus on base packing.) For $(1-\epsilon)$-approximate base covering with integer lower capacities, which we note does not reduce to approximate k-fold matroid union, we obtain a $O(n\log(n)\log(kr)/\epsilon)$ query running time (theorem 4.2). (Note that $n \leq kr$ without loss of generality for base covering.) For real-valued capacities we may assume without loss of generality $k = 1$. In this case we have the *matroid membership* problem where the goal is to decide if a real-valued vector is in the independent set polytope of a matroid. [17] gave the first strongly-polynomial time algorithm and there is an exact algorithm running in $O\big(n^3r^2\big)$ independence queries [53]. We obtain a randomized Monte Carlo algorithm running in $O\big(n + r\ln(n)\ln(r/\epsilon)/\epsilon^3\big)$ queries (theorem 7.6). Again there is not much literature explicitly on approximate base covering or matroid membership with real capacities. However we believe that the techniques in [9] could have also obtained a deterministic $\tilde{O}\big(nk/\epsilon^2\big)$-query time algorithm; we note that such an algorithm does not produce an integral packing when the lower capacities are integral.

The final problem we discuss for matroids is optimal reinforcement. In the integral version, given a matroid with integer element capacities, real-valued element costs, and an integer parameter k, the goal is to compute a minimum cost extension of the capacities so that the resulting capacitated matroid has matroid strength k. [16] proposed and analyzed this problem in graphic matroids. Generalized to matroids, his algorithm is a reduction to $\tilde{O}(n)$ calls to k-fold matroid union. Multiplied against the running time for [15] mentioned above (here $\mathrm{OPT} = kr$) gives a running time of $O\big(\big((kr)^{3/2} + k\big)n^2Q + k^{3/2}rn^2\big)$. ([16] also considers real capacities which we do not address.) We show how to compute the optimum reinforcement in time equal to 1 call to our first push-relabel algorithm for k-fold matroid union, running in $O(n + \mathrm{OPT}\, r\log(kr))$ independence queries. More precisely, we show that the push-relabel algorithm directly solves the optimal reinforcement with only a minor modification to how we initialize the algorithm.

We now outline the corresponding results for the graphic matroid. (For a graph $G = (V, E)$, we denote $m = |E|$ and $n = |V|$.) Generally speaking, we take the algorithms above for general matroids and replace the independence oracle with appropriate data structures that can answer queries directly. This generally replaces the query in the running times with a polylogarithmic overhead (or better, such as the inverse Ackermann $\alpha(n)$.) For k-fold matroid union — that is, maximizing the total size of a packing of k forests — we obtain a running time of $O(m\alpha(n) + n\,\mathrm{OPT}(\log(n) + \log(k)\alpha(n)))$ by push-relabel alone (theorem 5.2) and $O\Big(m\alpha(n) + \mathrm{OPT}^{3/2}\log^2(nk)\alpha^2(n)\Big)$ (in connected graphs) by combining push-relabel with augmenting paths (theorem 8.2). (Note that $\mathrm{OPT} \leq nk$.) Comparable running times are (a) $O\big(n^2k\log k\big)$ and (b) $O\Big(\min\{U, nk\}\sqrt{k(U + n\log n)}\Big)$. [32]. In particular our first running time improves (a) and our second running time improves (b) because $\mathrm{OPT} \leq \min\{U, nk\}$. For $(1-\epsilon)$-approximations, we obtain an $O(m\alpha(n) + \mathrm{OPT}\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$ for integer capacities and randomized $O\big(m\alpha(n) + (\mathrm{OPT}/k)\log^2(n)(\log(n) + \log(k)\alpha(n))/\epsilon^3\big)$ time for real capacities.

Packing bases translates to packing spanning trees. This can be solved by the algorithms for k-fold union and we obtain the same running times. In addition to the running times listed above for k-fold matroid union, one can also compare to running times of $O(Un\log(m/n))$ [32] and $O\big(kn\sqrt{U + n\log n}\big)$ [27]. For $(1-\epsilon)$-approximations, we obtain $O(m\alpha(n) + nk(\log(n) + \log(k)\alpha(n))/\epsilon)$ for approximate integer tree packings. This can be compared to a Las Vegas randomized algorithm that runs in $\tilde{O}\big(kn^{3/2}/\epsilon^2\big)$ time with high probability. The maximum number of spanning trees that can be packed into a graph is called the network strength. For real-valued capacities we obtain a randomized $O\big(m\alpha(n) + (\mathrm{OPT}/k)\log(n)\log\big(n\log(m)/\epsilon^2\big)\big)$ time for $(1\pm\epsilon)$-approximately testing the network strength and a slightly greater running time for approximating the network strength up to a $(1\pm\epsilon)$-factor. This result result can be compared with a deterministic $\tilde{O}\big(m/\epsilon^2\big)$ time algorithm or a randomized $\tilde{O}\big(m + n/\epsilon^4\big)$ time algorithm in [9].

Covering by bases corresponds to covering by spanning trees. Again new exact running time for integer lower capacities are obtained via the k-fold union algorithms listed above. Additional comparable running times besides k-fold union are $\tilde{O}\big(U^{5/3}\big)$ and $O(Un\log(n))$ [32]. For $(1-\epsilon)$-approximations with integer lower capacities, we obtain a running time of $O(m\alpha(n) + nk\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$. The minimum number of spanning trees required to cover a graph is called the *arboricity*. The arboricity can be computed exactly in $O\big(mn\log\big(n^2/m\big)\big)$ time [29]. We obtain a randomized $O\big(m\alpha(n) + n\ln(n)(\log(n) + \log(\log(n)/\epsilon)\alpha(n))/\epsilon^3\big)$ time Monte Carlo algorithm for $(1\pm\epsilon)$-approximately testing the arboricity and slightly greater running time for approximating the arboricity up to a $(1\pm\epsilon)$-factor (theorems 7.7 and 7.9). As was the case for matroids, there are not as many developments for approximate covering by spanning trees as for packing spanning trees. However we believe the techniques in [9]

lead to $\tilde{O}(m/\epsilon^2)$ and randomized $\tilde{O}(m + n/\epsilon^4)$-time algorithms to $(1 \pm \epsilon)$-approximate the arboricity (value).

Lastly, by the same reduction as for matroids, the optimal reinforcement problem in graphs is solved by a single call to the push-relabel algorithm for k-fold union, running in $O(m\alpha(n) + n\,\mathrm{OPT}(\log(n) + \log(k)\alpha(n)))$ time where OPT refers to the optimum for k-fold union, and is at most nk. This improves [16]'s reduction to n calls to k-fold union, as well as the running time of $O(n^2 m \log(n^2/m))$ by [29] for small k. ([29]'s algorithm and [16]'s reduction also extend to real-valued capacities.)

1.2 Discussion and overview of technical ideas As mentioned above, our improved running times start with a common "matroid push-relabel" algorithm building on ideas from [24]. Before describing our contributions, it may be helpful to first describe their framework at a high-level, especially given the unusual perspective for those coming for flow.

We informally describe their algorithm for k-fold matroid union for illustrative purposes. We assume the uncapacitated setting for simplicity; the goal is to compute a set of k bases $B_1, \ldots, B_k$ maximizing the size of their union. Call an element e *uncovered* if it is not in any B_i, *covered* if it is in some B_i, and *overpacked* if it is in more than one B_i. We want to cover as many elements as possible, and loosely speaking, overpacked elements represent wasted slots among the bases. The push-relabel algorithm of [24] manipulates $B_1, \ldots, B_k$ by repeatedly selecting an uncovered element e and trying to exchange it into some B_i so that it is covered. To make direct progress one would have to exchange it out for an overpacked element d; otherwise the size of the union stays the same. However, such a profitable exchange may not be available, even if $B_1, \ldots, B_k$ is not yet an optimal solution. While one can exchange e for other covered (but not overpacked) elements, it is not clear how this helps. This is analogous to pushing flow from one non-terminal vertex with surplus to another non-terminal vertex; it is not clear that we are making progress towards a sink.

This is where the *relabel* aspect of the push-relabel framework comes in. Each element is labeled by an integer *level*. Overpacked elements are kept at level 0, analogous to sinks in flow. Exchanges are restricted so that an element e is exchanged into a base B_i for an element d that is one level below B_i. In some sense, the "excess" represented by e being uncovered is transferred down one level to d, and thus closer to the overpacked elements at level 0. These ideas eventually lead to a more elaborate argument in [24] about why the algorithm terminates in polynomial time.

Above is a sketch omitting details, proofs of correctness, and even a complete description of the algorithm (deferring a more technical treatment to later). However it starts to form an analogy between the familiar push-relabel framework for max flow, and the abstract version presented by [24]. Instead of labeling vertices of a graph, we label elements of a matroid. In flow, we push flow along edges from one vertex to another; with matroids, we "push" exchanges of one element for another that maintain feasibility. Similar to flow, pushes are restricted to go "down" a level, and when no pushes are available, there is a *relabel* operation where some elements have their level increased. Doing so may reveal a violating constraint induced by the level sets of vertices, similar to how minimum cuts emerge from the labels in flow.

In analyzing their push-relabel framework, for problems ranging from matroid partition to submodular flows, [24] focuses on demonstrating that the algorithms are strongly polynomial while keeping the algorithms and analysis as simple as possible. (Historically, obtaining strongly polynomial running times for these abstract problems was highly non-trivial.) To this end, rather than directly bound the running time, [24] gave worst-case polynomial bounds on the number of "basic operations" — the number of push and relabel operations — made by the framework. (E.g., [24] gives a bound of $O(n^5)$ basic operations for the k-fold matroid union problem above.) One can show that it takes polynomial time (and queries to an independence oracle) to identify and execute a basic operation, but we caution that this is far less straightforward to do this than for flow, due to the abstract nature of matroids and the oracle model. For example, in flow, the edges explicitly specify where we can "push". With matroids, finding an exchangeable pair of elements e_1 and e_2 (as described above) may require nested loops over the ground set of elements and an independence query for each inner iteration. There are additional technical issues that [24] addresses which have no obvious analogy for flow.

We have taken to calling this framework *matroid push-relabel*, to distinguish from push-relabel for flow. Initially we were drawn by the conceptual appeal of [24], and started developing algorithms for some more specific problems hoping at best for some simpler or more practical alternatives to existing algorithms (similar to the role now assumed by push-relabel for flow). Given the large bounds and high level of abstraction in [24], it was not at all clear that competitive bounds could be obtained from matroid push-relabel for basic, long-studied problems where

there are alternative approaches that seem more direct. We were surprised to discover that, upon developing several more ideas within the matroid push-relabel framework, one can actually improve the best known bounds for several of these problems.

As mentioned above, the push-relabel algorithm manipulates a collection of bases and assigns levels to each element. The algorithm modifies the bases and levels while obeying a set of "push-relabel invariants", proposed by [24], which impose a discipline on the exchanges made to the bases. We extend these invariants slightly by introducing an integer-valued parameter called the *height*. The height is the minimum level of any uncovered element. One motivation for the height is to facilitate the analysis of fast approximation algorithms, as discussed below. The running time of our core matroid push-relabel algorithm is expressed as a function of height. The height parameter is chosen based on the problem and whether we seek exact or approximate solutions.

As mentioned above, the algorithms in [24] are not very concrete (let alone efficient), and the algorithms were analyzed to the point of bounding the number of push/relabel operations, as opposed to bounding the running time. We fill in the running time analysis and introduces several more ideas to improve the running time. Some of these ideas are somewhat subtle, detailed and local; such as a refined bound on the height for k-fold matroid union, or more careful application of data structures for spanning tree problems. There are also some broader ideas that are easier to isolate and which we now highlight below.

Level-wise decreasing order of bases. Recall that the push-relabel algorithms maintains a collection of k bases for an input parameter k. Every time we want to exchange an element e into the solution (so to speak), the multitude of bases raises an algorithmic issue of quickly identifying a suitable base in which to exchange e. A naive approach loops through all the bases, which is very slow, especially with large convex combinations. An important, new idea introduced in this work, that seems very specific both to matroids and the push-relabel framework, is to maintain the bases in "level-wise decreasing order". The definition of this ordering is based on the upper level sets of the bases induced by the labels. We require and maintain the bases in such an order that for every level, the level set of one base spans the level set of the next one. Speaking intuitively and abstractly, it turns out that the monotonic nature of this order cooperates nicely with the way that elements are relabeled and exchanged in the push-relabel framework. In particular, some simple and necessary greedy rules for selecting between different choices of push and relabel operations are shown to be sufficient for maintaining the descending order. Moreover, given the bases in level-wise descending order, we no longer have to loop over the bases as (loosely) described above. Instead, we can apply a binary search to identify the right base in logarithmic time and queries. The descending order also allows for a binary search along the levels when relabeling an element, which also improves the running time.

Approximation via truncation. An important technique for obtaining fast approximations in the push-relabel framework is the idea that a truncated height, depending primarily on the desired accuracy, suffices to obtain an approximate solution. This is analogous to the well-known connection; in problems such as edge/vertex disjoint paths, bipartite matching, and matroid intersection; between the length of the shortest augmenting path and the quality of the current solution (e.g., [15, 20, 44]). In this work, the matroid push-relabel algorithms frame matroid optimization problems in a perspective more amenable to these types of arguments. For many problems, we show that either $O(1/\epsilon)$ or $O(\log(n)/\epsilon)$ levels suffice to obtain a $(1 \pm \epsilon)$-factor approximation. We note that there are different arguments that lead to either the $O(1/\epsilon)$ or $O(\log(n)/\epsilon)$ bounds, and some additional analysis was required to identify which was appropriate for each problem.

As mentioned earlier, besides the enhancements to the matroid push-relabel algorithm, we develop two more techniques that extend the applicability of the push-relabel algorithm.

Randomized rounding To extend the push-relabel framework for integer capacities to real-valued capacities, for the sake of fast approximation algorithms, we employ randomized rounding. More specifically, we use random sampling to discretize the capacities and effectively reduce the capacities to small integer values (at most $O\big(\ln(n)/\epsilon^2\big)$). For packing problems, the randomized techniques we need are already provided by [51]. For k-fold matroid union and covering problems, such techniques were not known, and the arguments from [51] did not seem to extend. We develop a new analysis for these remaining problems. Interestingly, this analysis also recovers the results of [51], via an arguably simpler proof. (In particular, the new analysis does not depend on the random contraction algorithm.) See theorem 7.1, section 7.1. The randomized techniques for matroid base covering and k-fold matroid union may be of independent interest.

Augmenting paths. Historically the most common approach to the problems considered here is via augmenting paths. Augmenting paths can also be used to extend an approximate (integral) solution to an optimal one (which is our application). With augmenting paths for (say) k-fold matroid union, one maintains a packing of k independent sets $I_1, \ldots, I_k$ and tries to extend it one element at a time. Extending such a packing is non-trivial, and may require a complicated sequence of exchanges to open up a slot for a new element, so to speak. A shortest path between a designated source and sink in this graph can be shown to give an augmenting path. Now, the most straightforward approach is to build out the entire auxiliary graph explicitly by testing for the presence of each possible arc, and then running BFS in the resulting graph. However the graph is potentially dense and building out the graph becomes the bottleneck.

We instead explore techniques that look for the desired path implicitly. At a high-level, one recognizes that identifying all the vertices reachable from the designated source does not require exploring all the arcs. Indeed, we need not test the existence of an arc where the head is already "marked" as explored. That said, we do need to be able to identify arcs to "unmarked" auxiliary vertices. To help us search for these useful arcs we impose additional invariants. One invariant is to keep $I_1, \ldots, I_k$ in "decreasing order", in a way similar to the level-wise order of decreasing bases in the matroid push-relabel algorithm. This restricts the family of augmentations we allow in each iteration. The second invariant comes within a single search for an augmenting path. As we mark auxiliary vertices as they are explored, we also require the subset of elements in $I_1, \ldots, I_k$ corresponding to marked auxiliary vertices to also be in "decreasing order". This is addressed by introducing a "pre-search" subroutine that is called on an auxiliary vertex before marking it. With this additional structure — keeping the $I_1, \ldots, I_k$ and the marked elements of the $I_1, \ldots, I_k$ (so to speak) in descending order — it becomes much easier to navigate the auxiliary graph implicitly. The overall running time for one search comes out to roughly a logarithmic number of independence queries per vertex in the auxiliary graph. (Lemma 8.1, section 8.)

We use the augmenting path subroutine to extend an $(1-\epsilon)$-approximate solution produced by the push-relabel algorithm to an optimum solution, for an appropriate choice of ϵ. However, to apply our augmenting path algorithm we require the output of the push-relabel algorithm to satisfy certain invariants: namely, the bases $B_1, \ldots, B_k$ should contain a packing $I_1, \ldots, I_k$ of the same total capacity and in descending order. Fortunately this is the case, and the proof critically depends on the fact that the $B_1, \ldots, B_k$ were in level-wise decreasing order already.

We note that some similar ideas pertaining to implicitly navigating the auxiliary graph have been applied recently to accelerating augmenting path algorithms for matroid intersection [5, 6, 8, 56], which is a closely related problem. Still, additional ideas specific to k-fold matroid union as well as the added structure from the matroid push-relabel algorithm are required to obtain our running time.

Future work. We recognize that some of the ideas here can be useful for exact push-relabel algorithms for real capacities and defer this to future work. We have also continued to develop algorithms from the push-relabel perspective for more abstract problems such as polymatroid intersections and submodular flow. These topics require much more abstract machinery and the conceptual focus is different from the presentation here which is more specialized to matroids. We defer these developments to future work.

Independent work. Independent work by [7] also develops fast exact algorithms for some of the matroid packing and covering problems considered here, with similar running times. The techniques are different, as [7] focuses on augmenting path algorithms, and takes a more data-structure-centric point of view highlighting the use of dynamic rank oracles that can accelerate these algorithms. Our algorithms are instead based on independence oracles. One interesting structural observatio n (in hindsight) is that in some situations where the algorithms in [7] use fully dynamic data structures, the algorithms here only leverage partially dynamic data structures. (E.g., the use of fully dynamic connectivity oracles [34, 49] in [7] versus the use of the disjoint union data structure here, for the graphic matroid.)

1.3 Organization.

The rest of this work is organized as follows.

- In section 2 we present preliminary definitions and notation.
- In section 3, we present and analyze the matroid push-relabel algorithm for k-fold matroid union with integer capacities.
- In section 4, we analyze the base packing and covering problems for integer capacities.

- In section 5, we implement the matroid push-relabel algorithms for the graphic matroid.
- In section 6 we analyze the minimum cost reinforcement problem.
- In section 7 we develop the randomized algorithms for real-valued capacities.
- In section 8 we develop the augmenting path algorithm.
- Additional background is given in A.

Acknowledgements. We thank András Frank for his helpful feedback which has improved this paper.

2 Preliminaries

We briefly introduce matroids and refer to [23, 60] for additional background. A matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ consists of a finite ground set $\mathcal{N}$ and a collection of *independent sets* $\mathcal{I} \subseteq 2^{\mathcal{N}}$ that satisfy three properties: (i) $\emptyset \in \mathcal{I}$ (ii) $A \in \mathcal{I}$ and $B \subset A$ implies $B \in \mathcal{I}$ and (iii) $A, B \in \mathcal{I}$ and $|B| > |A|$ implies that there is $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$. The rank function of a matroid is an integer valued function over the subsets of $\mathcal{N}$ where $\operatorname{rank}(S)$ is the cardinality of the largest independent set contained in S. We let $\operatorname{span}(S) = \{e : \operatorname{rank}(S + e) = \operatorname{rank}(S)\}$ denote the set of elements spanned by S. For an independent set $I \in \mathcal{I}$, and an element $e \in \operatorname{span}(I) \setminus I$, there is a unique minimal set in $I + e$, called the *circuit* of e in I and denoted $C(I + e)$.

The *independence polytope* is the set of vectors $x \in \mathbb{R}^{\mathcal{N}}_{\geq 0}$ that can be expressed as a convex combination of indicator vectors of independent sets. We let $P_{\mathcal{I}}$ denote the independence polytope. For $k > 0$, we let $kP_{\mathcal{I}}$ scale up $P_{\mathcal{I}}$ by a factor of k; equivalently, $kP_{\mathcal{I}}$ denotes the the set of vectors x such that $x/k \in P_{\mathcal{I}}$.

In all our problems the elements are equipped with *capacities* $u \in \mathbb{R}^{\mathcal{N}}_{>0}$. The capacities are usually integral except in section 7 where we consider real-valued capacities. Abusing notation, for a set $S \subseteq \mathcal{N}$, we denote the sum of capacities over S by

$$u(S) \stackrel{\text{def}}{=} \sum_{e \in S} u(e).$$

Many of our problems manipulate a set of k bases $B_1, \ldots, B_k$ which may have overlapping elements. In such a context, for an element $e \in \mathcal{N}$, let

$$x(e) \stackrel{\text{def}}{=} |\{i \in [k] : e \in B_i\}|$$

We say that an element is *uncovered* if $x(e) < u(e)$, *covered* if $x(e) \geq u(e)$, *feasibly packed* if $x(e) \leq u(e)$, and *overpacked* if $x(e) > u(e)$.

3 Matroid push-relabel with integer capacities

This section presents the matroid push-relabel algorithm. Our discussion centers on the k-fold matroid union problem, presenting an algorithm that accelerates an algorithm for k-fold matroid union presented in [24]. The k-fold matroid union problem was briefly introduced in section 1.1 and we reintroduce the problem here.

The input consists of a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, integer capacities $u : \mathcal{N} \to \mathbb{N}$, and an integer k. This input defines the following dual min-max problems shown by [55] to have equal objective values:

$$\text{maximize} \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \text{ over } k \text{ bases } B_1, \ldots, B_k \in \mathcal{I}; \tag{3.1}$$

$$\text{minimize } k \operatorname{rank}(S) + \sum_{e \in \bar{S}} u(e) \text{ over all sets } S \subseteq \mathcal{N}. \tag{3.2}$$

($x(e)$ is defined in section 2.) (3.1) is called the *k-fold matroid union* problem; we refer to (3.2) simply as its dual problem. As mentioned earlier, [15] gave a $O\Big((\mathrm{OPT}^{3/2} + k)nQ + \mathrm{OPT}^{1/2}\, kn\Big)$-time algorithm for the uncapacitated version (i.e., $u(e) = 1$ for all e) of the problem. Additionally, [24] gave a bound of $O(n^5)$ "basic operations" (which are defined below) in the uncapacitated setting. We note that both [15, 24] consider the more general matroid union setting, where each B_i is a base in a different matroid (over the same ground set).

We first introduce the high-level components of the matroid push-relabel framework, based on [24], in section 3.1. We then analyze how optimality is obtained with the framework, improving bounds in [24] as well as extending the analysis to approximations, in section 3.2. Finally we present and analyze the faster algorithm in section 3.3.

3.1 Components of the matroid push-relabel framework The matroid push-relabel maintains a map $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ assigning *levels* to each element, initially set uniformly to 0. The algorithm also maintains k bases $B_1, \ldots, B_k \in \mathcal{B}$, all initialized to be the same base chosen arbitrarily.

Our discussion frequently groups elements by their levels and to this end it is convenient to introduce the following notation. For a fixed level j, we let $\mathcal{N}_j$ denote the set of elements at level j. We also let $\mathcal{N}_{\leq j}$ denote the set of elements at level less than or equal to j; similarly we have $\mathcal{N}_{<j}$, $\mathcal{N}_{>j}$, and $\mathcal{N}_{\geq j}$.

As the algorithm updates the bases $B_1, \ldots, B_k$ by inserting elements, for each base B_i and each element $e \in B_i$, the algorithm tracks the level of e when it was inserted into B_i. For $i \in [k]$ and $j \in \mathbb{Z}_{\geq 0}$, we let $B_{i,j} \subseteq B_i$ be the subset of elements e in B_i that were inserted into B_i when e was at level j. (Note that $B_{i,j}$ does *not* equal $B_i \cap \mathcal{N}_j$.) We let $B_{i,\geq j}$ denote the set of elements e inserted into B_i when $\ell(e)$ was at least j. Similarly we have $B_{i,\leq j}$, $B_{i,>j}$, and $B_{i,<j}$.

In the matroid push-relabel algorithm, the k bases $B_1, \ldots, B_k$ form a candidate solution for the maximization problem (3.1). The sub-level sets $\mathcal{N}_{\leq j}$ (where $j \in \mathbb{N}$) represent candidate solutions for the minimization problem (3.2). The framework is designed to ensure that

$$\sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \leq k \operatorname{rank}(\mathcal{N}_{\leq j}) + u(\mathcal{N}_{>j})$$

for all j. As shown in section 3.2 below, at termination, the algorithm identifies a level j for which the inequality above is (exactly or approximately) tight. Thereby $B_1, \ldots, B_k$ and $\mathcal{N}_{\leq j}$ certify one another to be (exactly or approximately) optimal for their respective problems.

The matroid push-relabel algorithm obeys the following invariants proposed by [24]. Here we say that an element e is *covered* if it is contained in at least $u(e)$ bases, and otherwise *uncovered*.

(I) $\ell(e) = 0$ *for all elements* e *that are in (strictly) more than* $u(e)$ *bases.*

(II) For $i = 1, \ldots, k$, *and all levels* j, $B_{i,\geq j}$ *spans* $\mathcal{N}_{>j}$.

(III) All uncovered elements e *have* $\ell(e) \geq h$ *for a parameter* $h \in \mathbb{Z}_{\geq 0}$.

Given a configuration of bases and level assignments, the *height* is defined as the minimum level of any uncovered element. That is, the height is the minimum value of h satisfying invariant (III).

The matroid push-relabel algorithm is composed of essentially two operations, which [24] calls *basic operations*.

1. *Push (exchange):* Given an uncovered element e, a base B_i, and an element $d \in B_{i,\ell(e)-1}$, such that $B_i - d + e \in \mathcal{I}$, replace B_i with $B_i - d + e$.[1]
2. *Relabel / lift:* Given an uncovered element e, increase $\ell(e)$ to $\ell(e) + 1$.

To preserve invariant (II), an uncovered element e can only be relabeled if there is no push operation available. As mentioned in section 1.2, identifying a feasible push for an element e is a computational bottleneck, in contrast to flow. [24] proved that the push and relabel operations preserve invariants (I)–(III) which we assume as a fact.

3.2 Optimality via the matroid push-relabel invariants We will eventually develop an algorithm that tries to obtain, as quickly as possible, a configuration of bases and levels satisfying invariants (I)–(III) above for a given height parameter $h \in \mathbb{N}$. First we show how particular values of h correlate with good solutions to the dual min-max problems in eqs. (3.1) and (3.2). Here we have analyses for both exact and approximate solutions.

3.2.1 Exact solutions We first consider exact solutions to eqs. (3.1) and (3.2). Previously, [24] showed that height $\Omega(n)$ suffices to derive an exact solution in the more general setting of matroid union (with different matroids). For the specific case of k-fold matroid union we have the following stronger bound of $r + 2$.

LEMMA 3.1. *Suppose* $B_1, \ldots, B_k$ *and* $\ell : \mathcal{N} \to \mathbb{R}_{\geq 0}$ *satisfy the invariants invariants (I)–(III) with height* $h > r + 2$. *Then there exists a level* j *such that*

$$\sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \geq u(\mathcal{N}_{<j}) + k \operatorname{rank}(\mathcal{N}_{\geq j}).$$

[1] Note that d may be the same as e, inserted earlier when e was at a lower level.

This certifies that $B_1 \cup \cdots \cup B_k$ is a maximum solution and that $\mathcal{N}_{\geq j}$ is a minimum dual solution.

Proof. As a function of $j \in \mathbb{Z}_{\geq 0}$, $\operatorname{rank}(\mathcal{N}_{\geq j})$ is integral and nondecreasing from 0 to r. By the pigeonhole principle, there exists a level $j \in \{1, \ldots, h\}$ such that $\operatorname{rank}(\mathcal{N}_{\geq j}) = \operatorname{rank}(\mathcal{N}_{>j})$. We will prove the claim for this choice of j. By invariant (II), $B_{i,\geq j}$ spans $\mathcal{N}_{>j}$, hence

$$\sum_{i=1}^{k} |B_{i,\geq j}| \geq k \operatorname{rank}(\mathcal{N}_{\geq j}) = k \operatorname{rank}(\mathcal{N}_{>j}).$$

By invariant (I), no element in $\mathcal{N}_{\geq j}$ is overpacked, so

$$\sum_{i=1}^{k} |B_{i,\geq j}| \leq \sum_{i=1}^{k} |B_i \cap \mathcal{N}_{\geq j}| = \sum_{e \in \mathcal{N}_{\geq j}} \min\{x(e), u(e)\}.$$

Lastly, by invariant (III), all elements in $\mathcal{N}_j$ are covered, so

$$u(\mathcal{N}_{<j}) = \sum_{e \in \mathcal{N}_{<j}} \min\{u(e), x(e)\}.$$

Altogether we have

$$u(\mathcal{N}_{<j}) + k \operatorname{rank}(\mathcal{N}_{\geq j}) \leq u(\mathcal{N}_{<j}) + \sum_{i=1}^{k} |B_{i,\geq j}| \leq \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\},$$

as desired. ∎

3.2.2 Approximate solutions We now turn to approximations. Here we show that height $O(1/\epsilon)$ suffices to obtain an $(1+\epsilon)$-approximations for both the primal and dual problems.

LEMMA 3.2. *Let $\epsilon \in (0,1)$. Suppose $B_1, \ldots, B_k$ and $\ell : \mathcal{N} \to \mathbb{R}_{\geq 0}$ satisfy invariants (I)–(III) with height $h > 1/\epsilon + 2$. Then there exists a level j such that*

$$u(\mathcal{N}_{<j}) + k \operatorname{rank}(\mathcal{N}_{\geq j}) \leq (1+\epsilon) \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\}.$$

This certifies that $B_1, \ldots, B_k$ is a $1/(1+\epsilon)$-approximately maximum solution, and that $\mathcal{N}_{\geq j}$ is a $(1+\epsilon)$-approximately minimum dual solution.

Proof. Since $h > 1/\epsilon + 2$, there exists an index $j \in \{1, \ldots, h-1\}$ such that

$$|B_{1,j}| + \cdots + |B_{k,j}| \leq \epsilon(|B_{1,\geq 1}| + \cdots + |B_{k,\geq 1}|). \tag{3.3}$$

We will prove the claim for this choice of j. Since each $B_{i,\geq j}$ spans $\mathcal{N}_{>j}$ (per (II)),

$$k \operatorname{rank}(\mathcal{N}_{>j}) \leq \sum_{i=1}^{k} |B_{i,\geq j}|.$$

Additionally, by choice of j per eq. (3.3), we have

$$\sum_{i=1}^{k} |B_{i,\geq j}| \leq \sum_{i=1}^{k} |B_{i,>j}| + \epsilon \sum_{i=1}^{k} |B_{i,\geq 1}| \leq x(\mathcal{N}_{>j}) + \epsilon x(\mathcal{N}_{\geq 1}).$$

Now, since $x(e) \leq u(e)$ for all $e \in \mathcal{N}_{>0}$ and $u(e) \leq x(e)$ for all $e \in \mathcal{N}_{<h}$ by invariants (I) and (III), we have

$$u(\mathcal{N}_{\leq j}) + k \operatorname{rank}(\mathcal{N}_{>j}) \leq u(\mathcal{N}_{\leq j}) + x(\mathcal{N}_{>j}) + \epsilon x(\mathcal{N}_{\geq 1}) \leq (1+\epsilon) \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\},$$

as desired. ∎

3.3 A faster matroid push-relabel algorithm Having now established how the push-relabel invariants (I)–(III) imply exact or approximate, we turn to the algorithmic question of computing a configuration that satisfies the invariants for a prescribed value of height h. The running times we obtain, as a function of h, are described in the following theorem. Below, we let OPT denote the common optimum value of eqs. (3.1) and (3.2).

THEOREM 3.1. *Given a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, integer capacities $u : \mathcal{N} \to \mathbb{N}$, and parameters $k, h \in \mathbb{N}$, there is an algorithm that, in running time bounded by $O(n + h\,\mathrm{OPT}\log(kr))$ calls to an independence oracle, computes bases $B_1, \ldots, B_k$ and levels $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ that satisfy invariants (I)–(III) with height h.*

As a point of comparison, [24] showed that $O(nh)$ basic operations suffice to obtain height h for the unweighted setting (which does not account for other computational factors such as identifying basic operations). The rest of this subsection is devoted to proving theorem 3.1.

As mentioned above, the algorithm initializes $B_1, \ldots, B_k$ to be any arbitrary base. Here the greedy algorithm can compute a base in time proportional to $O(n)$ independence queries. Initially we set $\ell(e) = 0$ for all elements e. In addition to invariants (I)–(III), we impose the following monotonicity condition on $B_1, \ldots, B_k$ and ℓ.

DEFINITION 1. *Let $B_1, \ldots, B_k$ be a sequence of independent sets, and let $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ assign integer levels to each element. We say that $B_1, \ldots, B_k$ is* monotone decreasing *(or just* decreasing*) if $B_{i,\geq j}$ spans $B_{i+1,\geq j}$ for all indices $i = 1, \ldots, k-1$ and all levels $j \in \mathbb{Z}_{\geq 0}$.*

Given h, the goal of the algorithm is to reach a configuration where $\ell(e) \geq h$ for all uncovered elements e. To this end, the push-relabel algorithm (described by [24]) repeatedly selects an uncovered element e with $\ell(e) < h$, and in principle, wants to either push e into some B_i to cover e, or relabel e and bring $\ell(e)$ closer to h. For the faster algorithm, we describe a new procedure, called *greedy insertion*, that employs binary search along both the levels and the bases, to more aggressively place e in the first available base (so to speak). Some justification is required to argue that this process simulates a legal sequence of push-relabel operations; we prove this after describing the procedure.

Greedily inserting an element e: Let $e \in \mathcal{N}$ be uncovered with $\ell(e) < h$.

1. If e is spanned by $B_{k,h-1}$ then set $\ell(e) = h$ and return.
2. Otherwise identify the first level j such that $B_{k,>j}$ does not span e.
3. Otherwise search for the first index i such that $B_{i,>j}$ does not span e. Set $\ell(e) = j + 1$, and exchange e into B_i for an element $d \in B_{i,j}$ such that $B - d + e \in \mathcal{I}$.

All put together, the overall algorithm is as follows. We initially set all bases $B_1, \ldots, B_k$ to an arbitrary base, and $\ell(e) = 0$ for all e. As long as there is an uncovered element e with $\ell(e) < h$, we greedily insert e, which either places e in a base, or sets $\ell(e) = h$.

It remains to analyze both the correctness and the running time of this algorithm. We start with correctness, and in particular, we first show that `greedy-insertion` maintains invariants (I)–(III). The proof will require the fact that the bases were in decreasing order prior to greedy insertion; hence we also prove that greedy insertion maintains the decreasing order of bases.

LEMMA 3.3. *Suppose $B_1, \ldots, B_k$ and $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ satisfy invariants (I)–(III), and $B_1, \ldots, B_k$ are in descending order. Then greedily inserting an element e maintains invariants (I)–(III) and keeps the bases in decreasing order.*

Proof. We first show that greedy insertion maintains the descending order. We need only consider the case where we execute an exchange in step 3, as otherwise there is no change to the bases. Thus, suppose we exchange an element e into a base B_i at level j, in exchange for an element d at level $j-1$. We let $B_1, \ldots, B_k$ denote the bases before the exchange. We let $B'_i = B_i - d + e$ denote the updated base after the exchange; this is the only change to the sequence of bases. It suffices to compare B'_i to the bases B_{i-1} and B_{i+1} that precede and succeed B_i (assuming B_i is not the first or last base, respectively). There is no need to verify levels j' strictly larger than j since these level sets do not change.

Consider first B_{i-1} (when $i > 1$). For any level $j' \leq j$, we have

$$B'_{i,\geq j'} \subseteq B_{i,\geq j'} \cup \{e\} \overset{(a)}{\subseteq} \mathrm{span}(B_{i-1,\geq j'}),$$

and taking the span of both sides gives the desired subset inequality. Here (a) is because $B_{i,\geq j'} \subseteq \text{span}(B_{i-1,\geq j'})$ by the fact that $B_1, \ldots, B_k$ is in descending order, and also because $e \in \text{span}(B_{i-1,\geq j})$ by choice of the index i.

Consider now B_{i+1} (when $i < k$). At level j, we have

$$B_{i+1,\geq j} \overset{\text{(b)}}{\subseteq} \text{span}(B_{i,\geq j}) \overset{\text{(c)}}{\subseteq} \text{span}\big(B'_{i,\geq j}\big),$$

and taking the span of both sides gives the desired subset inequality. Here (b) is by the existing descending order. (c) is because $B'_{i,\geq j} = B_{i,\geq j} + e$. For levels $j' < j$, we have

$$B_{i+1,\geq j'} \overset{\text{(d)}}{\subseteq} \text{span}(B_{i,\geq j'}) \subseteq \text{span}(B_{i,\geq j'} + e) = \text{span}\big(B'_{i,\geq j'} + d\big) \overset{\text{(e)}}{=} \text{span}\big(B'_{i,\geq j'}\big),$$

and taking the span of both sides gives the desired subset inequality. Here (d) is by the descending order, (e) is by monotonicity, (f) is because $B'_{i,\geq j'} + d = B_{i,\geq j'} + e$, and (g) is because d is spanned by $B'_{i,\geq j'} = B_{i,\geq j'} + e - d$.

Next we show that greedy insertion maintains the push-relabel invariants (I)–(III). More directly, we will show that greedy insertion simulates a legal sequence of basic operations; as mentioned, [24] has already proven that basic operations maintain the invariants.

Given e, suppose we repeatedly try to push or relabel e until we either (a) execute a push, or (b) increase $\ell(e)$ to h. In event (a), of all possible choices of base B_i in which to exchange e at a fixed level, we specifically select the base B_i with the smallest index i. This describes a valid sequence of basic operations, hence would preserve invariants (I)–(III). We will show that greedy insertion simulates this process.

Fix an uncovered element e with $\ell(e) = j$. First, we claim that an element e can be pushed into a base B_i (at that level) iff e is not spanned by $B_{i,\geq j}$. Indeed, if $B_{i,\geq j}$ spans e, then all elements $d \in B_i$ that could be exchanged for e are in $B_{i,\geq j}$, and in particular, not in $B_{i,j-1}$ as required for a push. Conversely, if $B_{i,\geq j}$ does not span e, then the unique circuit of $B_i + e$ must contain an element d with $d \in B_{i,<j}$. Moreover, by invariant (II), $B_{i,\geq j-1}$ spans e, so this circuit is also the unique circuit of $B_{i,\geq j-1} + e$. This implies that $d \in B_{i,j}$, and can be exchanged for e.

Second, we observe that for all i, if $e \notin \text{span}(B_{i,\geq j})$, then $e \notin \text{span}(B_{i',\geq j})$ for all $i' \geq i$. This follows from the decreasing order of bases which implies that the sets $\text{span}(B_{i,\geq j})$ forms a nested, descending sequence of sets. This observation implies the following two points. First, if $e \in \text{span}(B_{k,\geq j})$, then e is spanned by all $B_{i,\geq j}$, and we can safely increase $\ell(e)$. Second, if $e \notin \text{span}(B_{i,\geq j})$ for *some* B_i, then we can binary search for the base B_i with smallest index i such that $e \notin \text{span}(B_{i,\geq j})$.

Putting everything together, recall that we want to argue that greedy insertion simulates a push/relabel process that repeatedly relabels e until either $\ell(e) = h$ or we can exchange e into a base B_i for an element $d \in B_{i,\ell(e)-1}$, at which point it makes the exchange into the first such B_i. As observed above, such an exchange is possible iff $e \notin \text{span}\big(B_{i,\geq \ell(e)}\big)$. Moreover, as observed above, the latter is possible iff $e \notin \text{span}\big(B_{k,\geq \ell(e)}\big)$. Since the sets $B_{k,\geq j}$ forms a nested, decreasing sequence of sets in j, and $\text{span}(\cdots)$ is a monotonically increasing set function, we can binary search for the first (smallest) index j such that $e \notin B_{k,\geq j}$. This index j is exactly the level that the simulated push-relabel process would have eventually set $\ell(e)$ to. Assuming $j < h$, the simulated push-relabel process would then identify the first B_i into which we can exchange e. By the observations above, this base B_i is the same as that identified via binary search in step 3. ∎

Lemma 3.3 establishes the correctness of the algorithm via the invariants (I)–(III). To complete the proof of theorem 3.1 it remains to prove the running time bound.

Each instance where we greedily insert an uncovered element e can be charged to either (a) setting $\ell(e) = h$, or (b) increasing the size of $B_{i,>j}$ for some $i \in [k]$ and $0 \leq j < h$. Each element has its level set to h once, so there are n insertions of type (a). Each insertion of type (a) takes one independence query. To bound the number of insertions of type (b), we observe that for a fixed level $j > 0$, the sets $B_{i,>j}$ across i form a feasible solution to (3.1), hence have total size at most OPT. Since there are h levels, we have at most $h\,\text{OPT}$ insertions of type (b). So to recap, we have at most n greedy insertions of the first type and $O(\text{OPT}\, h)$ of the second type.

The first type of greedy insertion takes one oracle call. Consider the second type. With binary search, the first search in step 2 takes $O(\log h)$ probes. Better yet, by standard doubling tricks, we can adjust the binary search so that the first search also takes at most $O(1 + \ell)$ probes where ℓ is the number of levels the element moves forward. We will be able to charge these off to increasing the ranks of at least ℓ sets $B_{i,>j}$. The search in step 3 takes $O(\log k)$ probes. In both cases, each probe takes one independence query.

When executing an exchange, we also need to identify an element d to remove quickly. To this end, we can maintain a balanced binary tree over $B_{i,j}$ in insertion order, and use binary search to quickly identify the last

possible choice of d. (This is the first element d such that all the elements in $B_{i,j-1}$ before d, along with the elements in $B_{i,\geq j}$, do not span e). This takes $\log(r)$ independence queries. We note that step 3 is only invoked for one of the $\mathrm{OPT}\, h$ queries of type (b). This gives the total running time.

This completes the presentation of the faster matroid push-relabel algorithm for integer capacities.

3.4 Putting it all together By combining the running time of theorem 3.1 with the required heights per lemmas 3.1 and 3.2, we obtain the following running times for exact and approximate k-fold matroid union with integer capacities.

THEOREM 3.2. *For integer capacities, a maximum k-fold matroid union and a dual solution can be computed in $O(n + \mathrm{OPT}\, r \log(kr))$ independence queries.*

THEOREM 3.3. *For integer capacities, a $(1-\epsilon)$-approximately maximum k-fold matroid union, and a $(1+\epsilon)$-approximately minimum dual solution, can be computed in time bounded by $O(n + \mathrm{OPT} \log(kr)/\epsilon)$ independence queries.*

4 Base Packing and Covering

We now turn to the problems of packing and covering a matroid in bases. The problems were briefly introduced in section 1.1 and we now describe them in greater detail.

In the base packing problem, given an integer k, the goal is to compute k bases $B_1, \ldots, B_k$ such that $x(e) \leq u(e)$ for all $e \in \mathcal{N}$. We say that $B_1, \ldots, B_k$ is a *packing* when $x(e) \leq u(e)$ for all e. Edmonds [18] proved there is a packing $B_1, \ldots, B_k$ of k bases iff

$$u(\bar{S}) \geq k(r - \mathrm{rank}(S))$$

for all sets S.

In the base covering problem, the goal is to compute k bases $B_1, \ldots, B_k$ such that $x(e) \geq u(e)$ for all $e \in \mathcal{N}$. Such a set of $B_1, \ldots, B_k$ is called a *covering*. Edmonds [19] proved there is a covering $B_1, \ldots, B_k$ of k bases iff

$$k\, \mathrm{rank}(S) \geq u(S)$$

for all sets $S \subseteq \mathcal{N}$.

Both of the dual characterizations above, for base packing and for base covering, can be obtained via the dual characterization for k-fold matroid union of [55] that was presented in section 3.

As mentioned there, exact base packing and covering reduces directly to k-fold matroid union. For example, for packing, there is a base packing of k bases iff there are k bases whose union has size kr. For covering, there is a base covering of k bases iff there are k bases whose union has size n. The dual solutions given by the k-fold matroid union algorithm certify infeasibility for base packing or covering when no packing or covering is found.

One might expect the approximation algorithms for k-fold matroid union to also be an approximation algorithms for packing and covering, but this is not the case. Consider, for example, uncapacitated packing of k bases. The approximation algorithm for k-fold matroid union will output k bases whose union has $(1-\epsilon)$-times the maximum size of any union. If there exists k disjoint bases, then in particular the union has size at least $(1-\epsilon)kr$ total elements. However this is not the same as $(1-\epsilon)k$ disjoint bases. A union of size $(1-\epsilon)kr$ neither confirms that there are at least $(1-\epsilon)k$ disjoint bases, nor denies that there exist k disjoint bases.

Similar disparities arise for capacitated packing, and capacitated and uncapacitated covering. For all of these problems, we give a slightly stronger analysis to obtain the desired form of approximation. The main difference here is that height necessary to obtain $(1 \pm \epsilon)$-approximations increases by a logarithmic factor. Consequently all the running times for approximating packing and covering are a logarithmic greater than for approximating k-fold matroid union.

4.1 Packing We first consider approximations for integer base packing, for which theorem 4.1 claimed a running time of $O(n \log(1/\epsilon) + n \log(kr/\epsilon))$ independence queries. Below we prove that height $O(\log(n)/\epsilon)$ height implies a $(1-\epsilon)$-approximation. The running time in theorem 4.1 then follows from running the integer matroid push-relabel algorithm for height $O(\log(n)/\epsilon)$, by theorem 3.1.

LEMMA 4.1. *Let $\epsilon \in (0,1)$ and $k \in \mathbb{N}$. Let $B_1, \ldots, B_k$ be a family of k bases and $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ an assignment of levels satisfying the matroid push-relabel invariants (I)–(III) with height $h = O(\log(n)/\epsilon)$. Then either (a) the bases $B_1, \ldots, B_k$ forms a feasible packing of k bases, or (b) $(\mathcal{M}, u)$ has no feasible packing of $(1+\epsilon)k$ bases.*

Proof. Suppose that $B_1, \ldots, B_k$ is not a proper packing. By invariant (I), this implies that $\mathcal{N}_0 \neq \emptyset$. Now, $u(\mathcal{N}_{\leq j})$ is nondecreasing in $j \in \{0, \ldots, h\}$, bounded below by 1 for $j = 0$ (as noted above) and at most n for $j = h$. Consequently there must be a level $j \in \{1, \ldots, h-1\}$ such that $u(\mathcal{N}_j) < (\epsilon/(1+\epsilon))u(\mathcal{N}_{\leq j})$. Fix j as such. By choice of j we have

$$u(\mathcal{N}_{\leq j}) + (1+\epsilon)k \operatorname{rank}(\mathcal{N}_{>j}) < (1+\epsilon)(u(\mathcal{N}_{\leq j}) + k \operatorname{rank}(\mathcal{N}_{>j})) - (1+\epsilon)u(\mathcal{N}_j) \tag{4.4}$$

By invariant (III), all elements in $\mathcal{N}_{\leq j}$ are covered, hence

$$u(\mathcal{N}_{\leq j}) \leq (1+\epsilon)u(\mathcal{N}_{<j}) \leq (1+\epsilon)x(\mathcal{N}_{\leq j}).$$

By invariant (II), each $B_{i,\geq j}$ covers $\mathcal{N}_{>j}$, and none of the elements in $\mathcal{N}_{\geq j}$ are overpacked, hence

$$k \operatorname{rank}(\mathcal{N}_{>j}) \leq \sum_{i=1}^{k} |B_{i,\geq j}| = x(\mathcal{N}_{\geq j}).$$

Altogether we have

$$u(\mathcal{N}_{\leq j}) + (1+\epsilon)k \operatorname{rank}(\mathcal{N}_{>j}) \leq (1+\epsilon)x(\mathcal{N}) \leq (1+\epsilon)kr.$$

Rearranging, we have

$$u(\mathcal{N}_{\leq j}) \leq (1+\epsilon)k(r - \operatorname{rank}(\mathcal{N}_{>j})),$$

hence $\mathcal{N}_{>j}$ certifies that the strength is at most $(1+\epsilon)k$. ∎

Applying the matroid push-relabel algorithm (theorem 3.1) with the height parameter $h = O(\log(n)/\epsilon)$, per lemma 4.1, gives the following $(1-\epsilon)$-approximation algorithm for base packing.

THEOREM 4.1. *For integer capacities, a $(1-\epsilon)$-approximate base packing, or a $(1+\epsilon)$-approximate certificate of infeasibility, can be computed in running time bounded by $O(n + kr \log(n) \log(kr)/\epsilon)$ independence queries.*

4.2 Covering We now move on to covering problems, starting with integer covering. Here we show that $O(\log(r)/\epsilon)$ height suffices to obtain a $(1+\epsilon)$-approximation; note that this bound is (slightly) better than for approximating integer packing above.

LEMMA 4.2. *Let $k \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let $B_1, \ldots, B_k \in \mathcal{B}$ be a collection of k bases and $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ a set of levels that satisfy the push-relabel invariants (I)–(III) with height $h = O(\log(r)/\epsilon)$. Then either:*

1. *$B_1, \ldots, B_k$ is a covering, or*
2. *For some index j, $\mathcal{N}_{>j}$ certifies that more than $(1-\epsilon)k$ bases are required in any covering.*

Proof. Suppose $B_1, \ldots, B_k$ is not a covering. Then there is at least one uncovered element; moreover, any uncovered element is in $\mathcal{N}_{\geq h}$ by invariant (III). As a function of $j \in \{0, \ldots, h\}$, $\operatorname{rank}(\mathcal{N}_{\geq j})$ is nonincreasing, bounded above by r at $j = 0$, and bounded below by 1 at $j = h$ because $\mathcal{N}_{\geq h}$ is nonempty. Since h is at least $O(\log(r)/\epsilon)$, there must be a level $j \in \{0, \ldots, h\}$ such that $\operatorname{rank}(\mathcal{N}_{>j}) \geq (1-\epsilon)\operatorname{rank}(\mathcal{N}_{\geq j})$. Fix j as such. We have

$$u(\mathcal{N}) > \sum_{e \in \mathcal{N}} \min\{u(e), x(e)\} \tag{4.5}$$

because $B_1, \ldots, B_k$ is not a covering. Since elements in $\mathcal{N}_{\geq j}$ are not overpacked, we have

$$\sum_{e \in \mathcal{N}_{\geq j}} \min\{u(e), x(e)\} = \sum_{e \in \mathcal{N}_{\geq j}} x(e) = \sum_{i=1}^{k} |B_{i,\geq j}|.$$

By invariant (II) each $B_{i,\geq j}$ spans $\mathcal{N}_{>j}$ hence

$$\sum_{i=1}^{k} |B_{i,\geq j}| \geq k \operatorname{rank}(\mathcal{N}_{>j}).$$

On the other hand, since all elements in $\mathcal{N}_{<j}$ are covered, we have

$$\sum_{e \in \mathcal{N}_{<j}} \min\{u(e), x(e)\} = u(\mathcal{N}_{<j}).$$

Plugging back into (4.5) we now have

$$u(\mathcal{N}) > k \operatorname{rank}(\mathcal{N}_{>j}) + u(\mathcal{N}_{<j}).$$

Finally, since $\operatorname{rank}(\mathcal{N}_{>j}) \geq (1-\epsilon)\operatorname{rank}(\mathcal{N}_{\geq j})$ by choice of j, we obtain

$$u(\mathcal{N}) > (1-\epsilon)k \operatorname{rank}(\mathcal{N}_{\geq j}) + |\mathcal{N}_{<j}|.$$

Rearranging we have that $(1-\epsilon)k \operatorname{rank}(\mathcal{N}_{\geq j}) < |\mathcal{N}_{\geq j}|$, which by the dual characterization above implies that more than $(1-\epsilon)k$ bases are required in any covering. ∎

Plugging in $h = O(\log(r)/\epsilon)$ to the matroid push-relabel algorithm (theorem 3.1), we obtain the following approximation algorithm for base covering. (When plugging into theorem 3.1, we note that $\mathrm{OPT} = n$, and $n \leq kr$.)

THEOREM 4.2. *For integer capacities, one can compute either a covering of $(1+\epsilon)k$ bases or a certificate of infeasibility for any covering of $(1-\epsilon)k$ bases in time bounded by $O(n \log(n) \log(kr)/\epsilon)$ independence queries.*

5 Graphic Matroid Push-Relabel

In this section, we consider the matroid push-relabel framework for the special case of the graphical matroid; i.e., forests of an undirected graphs. This leads to algorithms for the several graph problems mentioned in section 1.1.

For a given graph $G = (V, E)$, we denote $m = |E|$ and $n = |V|$.

THEOREM 5.1. *Given a graph with integer capacities and integers k and h, in $O(m\alpha(n) + \mathrm{OPT}\, h(\log(n) + \log(k)\alpha(n)))$ time, one can compute a sequence of k spanning trees $T_1, \ldots, T_k$ and levels $\ell : E \to \mathbb{Z}_{\geq 0}$ satisfying the push-label invariants (I)–(III) with height h.*

Proof. We implement the matroid push-relabel algorithm with the following data structures. We maintain, for each spanning tree T_i, a link-cut tree [61], with edges labeled by their levels. Given an edge e, a tree T_i, we can query for the biggest level j such that $T_{i,\geq j}$ spans e by querying for the minimum level edge on the cycle induced by e. This also allows us to retrieve an edge d to exchange out in $O(\log n)$ time. We make a total of $O(\mathrm{OPT}\, h)$ such exchanges.

We also maintain, for each base B_i and each level $j \in \{1, \ldots, h\}$, a disjoint union data structure representing the connected components of $B_{i,\geq j}$. This allows us to query if an edge e is spanned by a forest $B_{i,\geq j}$ in $\alpha(n)$ time. We make $O(\mathrm{OPT}\, h)$ total insertions into thee disjoint union data structures over all i and j. We make at most $O(m + \mathrm{OPT}\, h \log(k))$ such queries. ∎

Theorem 5.1, combined with the optimality conditions given by lemmas 3.1, 3.2, 4.1 and 4.2 for k-fold matroid union, base packing, and base covering, both exact and approximate, give the follow running times for the graphic matroid.

We start with the k-fold matroid union problem. For graphs it is more natural to state this as computing a packing of forests $F_1, \ldots, F_k$ of maximum total capacity.[2] We have the following exact and approximate running times.

[2] As with trees, a set of forests is a packing if no element e appears in more than $u(e)$ of the forests.

THEOREM 5.2. *A maximum capacity packing of k forests, and the dual minimization problem, can be solved in $O(m\alpha(n) + n\,\mathrm{OPT}(\log(n) + \log(k)\alpha(n)))$ time, where* OPT *denotes the optimum size.*

THEOREM 5.3. *An $(1-\epsilon)$-approximately maximum capacity packing of k forests, and an $(1+\epsilon)$-approximately minimum dual solution, can be computed in $O(m\alpha(n) + \mathrm{OPT}\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$ time.*

For packing and covering spanning trees, in addition to the exact algorithms implied by theorem 5.2, we have the following approximation algorithms.

THEOREM 5.4. *For integer-capacitated graphs, there is an algorithm that, in $O(m\alpha(n) + nk(\log(n) + \log(k)\alpha(n))/\epsilon)$ time, outputs either a packing of k spanning trees, or a certificate that the network strength is less than $(1+\epsilon)k$.*

THEOREM 5.5. *In integer-capacitated graphs, there is an algorithm that, in $O(m\alpha(n) + nk\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$ time, outputs either a covering by k spanning trees, or a certificate that the strength is less than $(1+\epsilon)k$.*

6 Minimum Cost Reinforcement

In this section we consider the minimum cost reinforcement problem, introduced in section 1.1. We primarily discuss the more general matroid setting; the graphic setting follows as a special case.

Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with n elements and rank r, let $u : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ be a set of integer capacities, let $c : \mathcal{N} \to \mathbb{R}_{>0}$ be a set of real-valued costs, and let $k \in \mathbb{N}$. The cost $c(e)$ represents the cost of augmenting $u(e)$ by 1. The goal is to compute the minimum cost augmentation of u to obtain strength k.

Our argument is closely tied to Cunningham's algorithm [16], which was the first strongly polynomial time algorithm for this problem. Cunningham focused on network strength for undirected graphs and here we describe a straightforward generalization of his algorithm to matroids.

1. Compute a maximum point $y \in kP_{\mathcal{I}}$ subject to $y \leq u$, via k-fold matroid union.
2. Greedily extend y to an integral point $y + z$ in $kP_{\mathcal{I}}$, where $z \in \mathbb{Z}^{\mathcal{N}}_{\geq 0}$ is computed as follows. Initially, we set $z(e) = 0$ for all elements e. Then for each element e in increasing order of costs, set $z(e)$ as large as possible subject to $y + z \in kP_{\mathcal{I}}$. The maximum value for $z(e)$ can be obtained by binary search, where each probe invokes a k-fold matroid union algorithm to see if the candidate value for $z(e)$ is feasible.

We refer to [16] for the full justification of this algorithm. Note that the algorithm requires many calls to a matroid partition algorithm; first to compute the initial point y, and then logarithmically many times for every element e to obtain the right $z(e)$.

In what is perhaps a surprising coincidence, the k-fold matroid union algorithm developed in section 3 actually solves the reinforcement problem in one shot (so to speak). There is just one minor adjustment: the initial bases (which was allowed to be arbitrary in section 3) must all be set to the minimum cost base which we denote B_0. Recall that the push-relabel algorithm applied to a set of integer capacities u and a parameter k produces a set of k bases $B_1, \ldots, B_k$ that maximizes

$$\sum_{e \in \mathcal{N}} \min\{u(e), x(e)\},$$

where $x(e)$ denotes the number of bases B_i containing e. The vector y defined by $y(e) = \min\{u(e), x(e)\}$ (for all e) fulfills step 1 of Cunningham's algorithm. The key claim, proven below, is that if the bases in the k-fold matroid union algorithm are all initially set to the minimum cost base B_0, then at the end of the algorithm, x describes $y + z$ for an optimum reinforcement solution z. Thus z can be read off directly from x and y. This gives an overall running time that is exactly the same as for k-fold matroid union. Here we have two speed-ups compared to [16] – one from a faster k-fold matroid union algorithm, and the second from omitting the second stage altogether. The following lemma formalizes the key claim.

LEMMA 6.1. *Let B_0 be the minimum cost base w/r/t c, and consider the exact push-relabel k-fold matroid union algorithm adjusted so that the initial bases are all set to B_0. Let $B_1, \ldots, B_k$ be the k bases output by the push-relabel*

matroid-partition algorithm, and let $x \in \mathbb{Z}^{\mathcal{N}}_{\geq 0}$ *be the vector where* $x(e)$ *is the number of bases* B_i *containing* e *for each* $e \in \mathcal{N}$*. Define* $z \in \mathbb{R}^{\mathcal{N}}_{\geq 0}$ *by*

$$z(e) = \min\{0, x(e) - u(e)\} \text{ for } e \in \mathcal{N}.$$

Then z *is a minimum cost reinforcement.*

Proof. In addition to $B_1, \ldots, B_k$ and x as described above, let ℓ be the set of levels produced by the capacitated k-fold matroid union algorithm. Then $B_1, \ldots, B_k$, x, and ℓ satisfy invariants (I)–(III) from section 3.1. Let $C = \{e : x(e) > u(e)\}$. We have $\ell(e) = 0$ for all $e \in C$ by invariant (I).

Claim 1. $C \subseteq B_0$.

To this end, observe that initially we have $x(e) > u(e)$ only for elements in the initial base, $e \in B_0$. Thereafter, a coordinate $x(e)$ is only increased if $x(e) < u(e)$, and never exceeding $u(e)$.

Claim 2. For any element $e \in B_0$*, and any base* $B_i \in \{B_1, \ldots, B_k\}$*, we have either* $e \in B_i$*, or* $e \in \text{span}(B_i \setminus C)$.

Fix any element $e \in B_0$ and a base B_i from $B_1, \ldots, B_k$. If $e \notin B_i$, then it was exchanged out by an element d with $\ell(d) = 1$, such that $B_{i,\geq 1} = B_{i,>0}$ spans e. That is, if $e \notin B_i$, then $e \in \text{span}(B_{i,>0})$. Since $C \subseteq \mathcal{N}_0$, we have $B_{i,>0} \subseteq B_i \setminus C$, hence $e \in \text{span}(B_i \setminus C)$.

Now, let y be the pointwise minimum of x and u; y is maximum in $kP_{\mathcal{I}}$ subject to $y \leq u$. Recall that a minimum cost base can be produced by a greedy algorithm adding feasible elements in nondecreasing order of cost. Number the elements $\mathcal{N} = \{e_1, e_2, \ldots\}$ in nondecreasing order of cost, breaking ties so that a greedy algorithm processing elements in this order produces B_0. In Cunningham's greedy augmentation algorithm, a minimum cost reinforcement z is obtained by processing the e_i's in order, taking setting $z(e_i)$ to the maximum quantity subject to $y + z \in kP_{\mathcal{I}}$. (This equals the minimum quantity subject to $(y + z)/k$ spanning e_i in the independent set polytope $P_{\mathcal{I}}$, and in Cunningham's algorithm it is identified via binary search and a call to k-fold matroid union for each probe). Cunningham has already shown that this algorithm produces an optimum solution. Therefore it suffices to prove the following claim.

Claim 3. Cunningham's greedy augmentation selects $z(e) = \max\{x(e) - u(e), 0\}$ *for all* $e \in \mathcal{N}$.

We analyze each element in the greedy order. Consider the ith iteration (where $i \in [n]$), in which the greedy algorithm processes e_i. We assume by induction that $(y+z)/k$ spans $\{e_1, \ldots, e_{i-1}\}$ (in $P_{\mathcal{I}}$). (The base case, where $i = 1$, holds vacuously.) If $e_i \notin B_0$, then $e_i \in \text{span}(\{e_j \in B_0 : j < i\})$, so e_i is spanned by $(y + z)/k$.

Now suppose $e_i \in B_0$. We claim that the greedy augmentation algorithm sets $z(e_i) = x(e_i) - u(e_i)$. To see this, let z' be the vector obtained from z by setting $z'(e_i) = x(e_i) - u(e_i)$. We know that $y + z'$ is feasible because $y + z' \leq x$ and $x \in kP_{\mathcal{I}}$. To show that $(y + z')/k$ spans e_i, we first observe that one can pack into $y + z'$ the k independent sets

$$B'_j \stackrel{\text{def}}{=} B_j \setminus (C - e_i) \text{ for } j = 1, \ldots, k.$$

We claim that B'_j spans e_i for each j which shows that $(y + z')/k$ spans e_i. We have two cases. In the first case, if $e_i \in B'_j$, then of course $e_i \in \text{span}(B'_j)$. In the second case, if $e_i \notin B'_j$, then $e_i \notin B_j$. By claim 2, $e_i \in \text{span}(B_j \setminus C)$. Since $B'_j \subseteq B_j \setminus C$, $e_i \in B'_j$.

This shows that the Cunningham's greedy augmentation algorithm takes $z(e_i) = x(e_i) - u(e_i)$. This establishes the claim, and completes the proof. ∎

7 Approximations for problems with general capacities

This section develops fast approximation algorithms for matroid problems for general capacities. All of the algorithms in the section is based on using *randomized rounding* to reduce problems with real-valued capacities and a real-valued parameter k to problems with integer capacities and an integer parameter k on the order of $\ln(n)/\epsilon^2$, with high probability. We then apply the approximate push-relabel algorithms developed in prior sections which are particularly well suited to the reduced setting.

One cost of this convenience is that we will no longer obtain primal solutions for the original input. However we will still be able to obtain dual solutions which at least provide a certificate for one side of the corresponding decision problem.

7.1 Randomized rounding of real-valued capacities Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with n vertices and rank r, and let $u : \mathcal{N} \to \mathbb{R}_{>0}$ be a set of capacities. Let $k > 0$ be a parameter specified by the context. Let $\tau > 0$ also be a given parameter with $\tau \leq ck\epsilon^2/\ln(n)$ for a sufficiently small constant c. Decreasing c as needed, we may assume that k/τ is an integer without loss of generality. Let $\tilde{u} \in \mathbb{Z}_{\geq 0}^{\mathcal{N}}$ be the a randomized set of integral capacities by randomly rounding u/τ to an integral vector. That is, for each element $e \in \mathcal{N}$, we independently set

$$\tilde{u}(e) = \begin{cases} \lceil u(e)/\tau \rceil & \text{with probability } u(e)/\tau - \lfloor u(e)/\tau \rfloor, \\ \lfloor u(e)/\tau \rfloor & \text{with (remaining) probability } \lceil u(e)/\tau \rceil - u(e)/\tau. \end{cases}$$

The scaled down capacitated matroid $(\mathcal{M}, u)$ have some immediately appealing properties. First we have $\mathbf{E}[\tilde{u}(e)] = u(e)/\tau$ for all $e \in \mathcal{N}$. By linearity of expectation we also have $\mathbf{E}[\tilde{u}(S)] = u(S)/\tau$ for all sets S. Second, $(\mathcal{M}, \tilde{u})$ has an integer capacities, that are (in expectation) a $\tilde{O}(k)$-factor smaller than $(\mathcal{M}, u)$. We are interested in applying this randomized rounding for problems such as maximizing the total capacity covered by a packing of k independent sets, packing k bases, or covering by k bases, and these problem-specific values of k are used for the value of k in randomly rounding to $\tilde{u}$. Therefor it is helpful that k/τ is an integer for the corresponding scaled down problems over $(\mathcal{M}, \tilde{u})$. Additionally, for all these problems, k is a natural upper bound or near-upper bound on the capacities, hence $\tilde{u}$ will have relatively small capacities bounded above by $O\big(\ln(n)/\epsilon^2\big)$ in expectation (and with high probability).

We would like to show that the $(\mathcal{M}, \tilde{u})$ is (with high probability) a good representative sample of $(\mathcal{M}, u)$ for these problems. Now, while $\tilde{u}$ reflects u/τ in expectation, in general the values $\tilde{u}(e)$ for $e \in \mathcal{N}$ and $\tilde{u}(S)$ for $S \subseteq \mathcal{N}$ are too numerous to assume they are all concentrated at their expectation in expectation. (In fact, a value $\tilde{u}(S)$ will never be close to its expectation, multiplicatively speaking, when $u(S)$ is significantly smaller than τ.) Nonetheless we have the following theorem which leverages the dual characterizations of these problems to show that $(\mathcal{M}, \tilde{u})$ is a good (problem-specific) representation of $(\mathcal{M}, u)$ with high probability.

THEOREM 7.1. *Given the setup described above, the following all hold with high probability.*

(i) Letting M denote the maximum total capacity of any fractional packing of k independent sets in $(\mathcal{M}, u)$, and $\tilde{M}$ denote the maximum total capacity of any fractional packing of k/τ independent sets in $(\mathcal{M}, \tilde{u})$, we have $\big|\tilde{M} - M/\tau\big| \leq \epsilon M/\tau$.

(ii) If $(\mathcal{M}, u)$ can fractionally pack k bases, then $(\mathcal{M}, \tilde{u})$ can pack at least $(1-\epsilon)k/\tau$ bases.

(iii) If $(\mathcal{M}, u)$ cannot fractionally pack k bases, then $(\mathcal{M}, \tilde{u})$ cannot pack $(1+\epsilon)k/\tau$ bases.

(iv) If $(\mathcal{M}, u)$ can be covered by k bases, then $(\mathcal{M}, \tilde{u})$ can be covered by $(1+\epsilon)k/\tau$ bases.

(v) If $(\mathcal{M}, u)$ cannot be covered by k bases, then $(\mathcal{M}, \tilde{u})$ cannot be covered by $(1-\epsilon)k/\tau$ bases.

Proof. Recall that a set $S \subseteq \mathcal{N}$ is *closed* if $S = \operatorname{span}(S)$. We claim that with high probability, we have

$$|u(S) - \tau\tilde{u}(S)| \leq \frac{\epsilon}{2}(u(S) + k\operatorname{rank}(S)) \text{ and } |u(\bar{S}) - \tau\tilde{u}(\bar{S})| \leq \frac{\epsilon}{2}(u(\bar{S}) + k\operatorname{rank}(S)) \tag{7.6}$$

for all closed sets S (simultaneously).

For ease of notation, call a closed set S *bad* if $\tilde{u}(S)$ or $\tilde{u}(\bar{S})$ violates the inequalities above. We want to prove that there are no bad closed sets with high probability.

First, fix a closed set S with $\operatorname{rank}(S) = q$. Consider the first (leftmost) of the inequalities we seek. By standard Chernoff inequalities, we have

$$\mathbf{P}\Big[|\tilde{u}(S) - u(S)/\tau| > \frac{\epsilon}{2}(u(S)/\tau + kq/\tau)\Big] \leq 2e^{-\epsilon^2 kq/4\tau} = 2n^{-q/4c}.$$

Likewise the second inequality (for $\tilde{u}(\bar{S})$) has probability of error is at most $2n^{-q/4c}$. Taking the union bound,

$$\mathbf{P}[S \text{ is bad}] \leq 4n^{-q/4c} \leq n^{-c_1 q}$$

for a sufficiently large constant c_1.[3]

Now, fix q. Each closed set S is defined by any base of S, which consists of q elements. Therefore, there are at most n^q closed sets of rank q. Taking the union bound over all sets S of rank q,

$$\mathbf{P}[\text{any closed set of rank } q \text{ is bad}] \leq 2n^q n^{-c_1 q} \leq n^{-c_2 q}$$

for a sufficiently large constant c_2.

Finally, taking the union bound over all ranks $q \in [r]$, we have

$$\mathbf{P}[\text{any closed set is bad}] \leq \sum_{q=1}^{r} n^{-c_2 q} \leq n^{-c_3 q}$$

for a sufficiently large constant c_3. This proves the claim.

For the rest of the proof we assume the high probability event where the inequalities in (7.6) hold for all closed S. We will use these inequalities to prove each of results (i)–(v).

Consider first result (i). Recall that M equals the minimum of $u(\bar{S}) + k\,\mathrm{rank}(S)$ over all sets S, and similarly for M' except with respect to $\tilde{u}$ and k/τ. Since replacing S with its closure $\mathrm{span}(S)$ can only decrease this quantity, it suffices to consider only the closed sets.

For all closed sets S

$$\begin{aligned}\tilde{u}(\bar{S}) + \frac{k}{\tau}\,\mathrm{rank}(S) &\geq \frac{1-\epsilon}{\tau}u(\bar{S}) - \frac{\epsilon k}{\tau}\,\mathrm{rank}(S) + \frac{k}{\tau}\,\mathrm{rank}(S) \\ &= \frac{1-\epsilon}{\tau}(u(\bar{S}) + k\,\mathrm{rank}(S)) \geq \frac{(1-\epsilon)M}{\tau}\end{aligned}$$

for all closed sets S. Thus $M' \geq (1-\epsilon)M/\tau$ with high probability.

Next we upper bound M'. There exists a set S be a closed set such that $M = u(\bar{S} + k\,\mathrm{rank}(S))$. We have

$$\begin{aligned}M' &\leq \tilde{u}(\bar{S}) + \frac{k}{\tau}\,\mathrm{rank}(S) \leq \frac{1+\epsilon}{\tau}u(\bar{S}) + \frac{\epsilon k}{\tau}\,\mathrm{rank}(S) + \frac{k}{\tau}\,\mathrm{rank}(S) \\ &= \frac{1+\epsilon}{\tau}(u(\bar{S}) + k\,\mathrm{rank}(S)) \leq \frac{(1+\epsilon)M}{\tau},\end{aligned}$$

as desired. This proves result (i).

Consider now results (ii) and (iii). By the matroid base packing theorem, the packing number of $(\mathcal{M}, u)$ is at least k iff for all sets $S \subseteq \mathcal{N}$,

$$u(\bar{S}) \geq k(r - \mathrm{rank}(S)).$$

Thus the packing number is exactly k if the inequality holds for all S, and is tight for some set S with $\mathrm{rank}(S) < r$.

Now consider $(\mathcal{M}, \tilde{u})$. By theorem 7.1, with high probability, we have

$$\begin{aligned}\tilde{u}(\bar{S}) &\geq \frac{1-\epsilon/2}{\tau}u(S) - \frac{\epsilon k}{2\tau}\,\mathrm{rank}(S) \geq \frac{(1-\epsilon/2)k}{\tau}(r - \mathrm{rank}(S)) - \frac{\epsilon k}{2\tau}\,\mathrm{rank}(S) \\ &\geq \frac{(1-\epsilon)k}{\tau}(r - \mathrm{rank}(S)),\end{aligned}$$

so the packing number is at least $(1-\epsilon)k/\tau$. This proves result (ii).

For the opposite direction in result (iii), we know there exists a closed set S with $\mathrm{rank}(S) < r$ and $u(\bar{S}) = k(r - \mathrm{rank}(S))$. Note that $u(\bar{S}) \geq k$. By the Chernoff inequality we have

$$\mathbf{P}[\tilde{u}(\bar{S}) \geq (1+\epsilon)u(\bar{S})/\tau] \leq e^{-\epsilon^2 u(\bar{S})/3\tau} \leq e^{-\epsilon^2 k/3\tau} = n^{-1/3c}.$$

(We point out that $1/3c$ represents an arbitrarily large constant.) Thus with high probability we have

$$\tilde{u}(\bar{S}) \leq (1+\epsilon)u(\bar{S})/\tau = \frac{(1+\epsilon)k}{\tau}(r - \mathrm{rank}(S))$$

[3]By which we mean that c_1 can be made an arbitrarily large constant by making c sufficiently small.

and so the packing number of $(\mathcal{M}, \tilde{u})$ is at most $(1+\epsilon)k/\tau$. This proves result (iii).

Lastly we prove results (iv) and (v). Recall that $(\mathcal{M}, u)$ can be fractionally covered by k bases iff for all sets $S \subseteq \mathcal{N}$, $u(S) \leq k \operatorname{rank}(S)$. Since the capacities are nonnegative, it suffices to verify the inequality $u(S) \leq k \operatorname{rank}(S)$ for all closed sets.

Suppose $(\mathcal{M}, u)$ can be fractionally covered by k bases. We have

$$\tilde{u}(S) \leq \left(1 + \frac{\epsilon}{2}\right)\frac{u(S)}{\tau} + \frac{\epsilon k \operatorname{rank}(S)}{2\tau} \leq \frac{(1+\epsilon)k}{\tau}\operatorname{rank}(S)$$

for all closed sets S. Thus $(\mathcal{M}, \tilde{u})$ can be fractionally covered by $(1+\epsilon)k/\tau$ bases. This proves result (iv).

For result (v), suppose $(\mathcal{M}, u)$ cannot be fractionally covered by less than k bases. Then there is a closed set S such that $u(S) \geq k \operatorname{rank}(S)$. We have

$$\tilde{u}(S) \geq \frac{1-\epsilon/2}{\tau}u(S) - \frac{\epsilon k}{2\tau}\operatorname{rank}(S) \geq \frac{(1-\epsilon)k}{\tau}u(S).$$

Thus $(\mathcal{M}, \tilde{u})$ cannot be fractionally covered by $(1-\epsilon)k/\tau$ bases. This establishes result (v) and completes the proof. ∎

REMARK 7.1. *As mentioned above, for approximating the base packing problem specifically, [51] already provides a lemma that allows us to reduce real-valued capacities to small integer capacities. The construction in [51] is slightly different; in [51], each random capacity $\tilde{u}(e)$ is sampled independently from a Poisson distribution of mean $p \cdot u(e)$, for a parameter $p > 0$ with $p \geq c\epsilon^2/k \ln(n)$ for a sufficiently large constant c. Overall the net effect is the same as the rounding-based construction for $\tilde{u}$ that we analyze here. Despite the overlap with [51] we include the proofs of results (ii) and (iii) as we find them interesting for the following reasons. First, the proof techniques here are unified with the proofs for the other matroid problems in results (i), (iv) and (v). (Conversely, the proof techniques in [51] did not seem as useful for these other problems.) Second, the proofs here are different and arguably simpler than in [51] as it does not depend on the random contraction algorithm.*

7.2 Maximum capacity packings of independent sets and forests

THEOREM 7.2. *For real-valued capacities, a $(1-\epsilon)$-approximation to the value of the maximum (fractional) k-fold matroid union, along with an $(1+\epsilon)$-approximately minimum dual solution, can be computed with high probability in $O\big(n + (\mathrm{OPT}/k)\log(n)\log\big(r\log(n)/\epsilon^2\big)/\epsilon^3\big)$ randomized time. (Note that $\mathrm{OPT}/k \leq r$.)*

Proof. We apply theorem 7.1 to reduce the problem to integer capacities and optimum value $(\mathrm{OPT}/k)\log(n)/\epsilon^2$ with high probability. We then apply the $(1-\epsilon)$-approximation algorithm for integer capacities from theorem 3.3. The running time follows from theorem 3.3. ∎

The same reduction but for graphic matroids gives the following.

THEOREM 7.3. *In an undirected graph with real-valued edge capacities, an $(1\pm\epsilon)$-approximation to the maximum total capacity that can be covered by a fractional packing of k forests can be computed in $O\big(m\alpha(n) + (\mathrm{OPT}/k)\log^2(n)(\log(n) + \log(k)\alpha(n))/\epsilon^3\big)$ randomized time. (Note that $\mathrm{OPT}/k \leq n-1$.)*

7.3 Matroid base packing, matroid membership, and network strength

THEOREM 7.4. *For real-valued capacities, an $(1\pm\epsilon)$-approximation to deciding if the matroid strength is (greater or less than) k can be computed with high probability in randomized time bounded by $O\big(n + r\log^2(n)(\log\log(n) + \log(1/\epsilon))/\epsilon^3\big)$ independence queries.*

Proof. By either theorem 7.1 or the techniques in [51], we can reduce the problem to packing k/τ bases into integer capacities, with high probability. We then apply the $(1\pm\epsilon)$-approximation algorithm for packing $k/\tau = O\big(\ln(n)/\epsilon^2\big)$ bases with integer capacities given by theorem 4.1. The running time follows from theorem 4.1. ∎

The approximate algorithm for deciding matroid strength can be extended to an approximation algorithm for approximating the matroid strength via binary search. Here we present a modified algorithm that carefully modifies the error parameters to reduce the standard logarithmic overhead.

THEOREM 7.5. *For real-valued capacities between* 1 *and* U*, an* $(1 \pm \epsilon)$*-approximation to the matroid strength can be computed in time bounded by* $O\big((n + r\log(n)\log(r))\log(nU/r) + r\ln(n)\ln(r/\epsilon)/\epsilon^3\big)$ *independence queries.*

Proof. At the outset, we know that the matroid strength is between 1 and nU/r.

Now, for $i \in \mathbb{Z}_{\geq 0}$, let $\epsilon_i = 2^{-i}$. A $(1+\epsilon_0)$-approximation to the strength can be obtained with high probability by combining a binary search of depth $O(\log(nU/r))$ with the approximate decision algorithm in theorem 7.4 with error parameter a constant factor small than ϵ_0. For $i \geq 1$, given a $(1+\epsilon_{i-1})$-approximation for the strength, we can compute a $(1+\epsilon_i)$-approximation via a binary search of constant depth, with each probe making a call to theorem 7.4 with error parameter $\Omega(\epsilon_i)$. Eventually we obtain a $(1+\epsilon_i)$-approximation where $\epsilon_i = 2^{-i}$ is at most the input error parameter ϵ, as desired.

We now bound the running time. The first set of $\log(n)$ calls to theorem 7.4 with constant error parameter takes

$$O(n\log(nU/r) + r\log(n)\log(r)\log(nU/r))\text{-query} \tag{7.7}$$

time. Thereafter we have a constant number of calls to theorem 7.4 for each ϵ_i between 1 and $\epsilon/2$. For the leading $O(n)$ term, all these calls add up to $O(n\log(1/\epsilon))$ work which is dominated by $O(n\log(n))$ above. For the second term of the form $r\ln(n)\ln(r/\epsilon)/\epsilon^3$, the sum over all ϵ_i's is dominated by the smallest ϵ_i which is $\Omega(\epsilon)$, given

$$O\big(r\ln(n)\ln(r/\epsilon)/\epsilon^3\big) \tag{7.8}$$

work in total. Summing together eqs. (7.7) and (7.8) gives the claimed running time. ∎

7.4 Matroid base covering, matroid membership, and arboricity Fractional base covering can be posed as a decision problem where, given a capacitated matroid and an additional parameter k, the goal is to decide if the lower capacities can be fractionally covered by k bases. The important special case of $k = 1$ is equivalent to testing if a fractional point $x \in \mathbb{R}^{\mathcal{N}}_{\geq 0}$ lies in the independent set polytope. This problem is called matroid membership.

For deciding fractional base covering, we may assume that $k = 1$ without loss of generality. For a fixed error parameter $\epsilon \in (0,1)$, a $(1 \pm \epsilon)$-*approximation* to the matroid membership problem is defined as a correct output that either (a) the matroid can be covered by $1+\epsilon$ bases, or (b) the matroid can be covered by $1-\epsilon$ bases. Note that either option is allowed when the fractional covering number is between $1-\epsilon$ and $1+\epsilon$. We obtain the following running time for approximating matroid membership.

THEOREM 7.6. *An* $(1 \pm \epsilon)$*-approximation to the matroid membership problem can be computed with high probability in* $O\big(n + r\ln(n)\ln(r/\epsilon)/\epsilon^3\big)$ *randomized time.*

Proof. [Proof sketch] By theorem 7.1, we can reduce $(1 \pm \epsilon)$ matroid membership (with real capacities) to $(1+\epsilon)$-approximate integral base covering with $k = O\big(\ln(n)/\epsilon^2\big)$ bases. The running time now follows from theorem 4.2. ∎

For graphic matroids, recall that the fractional covering number is called the arboricity. The following matches the theorem for matroid membership above except for graphic matroids. The reduction is the same except now we apply the corresponding algorithm for the graphic matroid.

THEOREM 7.7. *An* $(1 \pm \epsilon)$*-approximation to deciding if a point is in the forest polytope can be computed with high probability in* $O\big(m\alpha(n) + n\ln(n)(\log(n) + \log(\log(n)/\epsilon)\alpha(n))/\epsilon^3\big)$ *randomized time.*

One may also want to find the maximum value k for which a capacitated matroid can be covered by k fractional bases. The following uses theorem 7.6 as a black box and is slightly faster than one obtains by directly plugging into a straightforward binary search.

THEOREM 7.8. *For real-valued capacities, an* $(1 \pm \epsilon)$*-approximation to the minimum* k *by which a matroid can be covered by* k *fractional bases can be computed with high probability in running time bounded by* $O\big((n + r\ln(r))\ln(n) + r\ln(n)\ln(r/\epsilon)/\epsilon^3\big)$ *independence queries.*

Proof. At the outset, we know that the arboricity is between the maximum capacity of any element and the sum of capacities over all elements, which are within a factor n of each other.

Similar to theorem 7.5, for $i \in \mathbb{Z}_{\geq 0}$, let $\epsilon_i = 2^{-i}$. A $(1+\epsilon_0)$-approximation to the strength can be obtained with high probability by combining a binary search of depth $O(\log(n))$ with the approximate decision algorithm in theorem 7.6 with constant error parameter. For $i \geq 1$, given a $(1+\epsilon_{i-1})$-approximation for the strength, we can compute a $(1+\epsilon_i)$-approximation with a binary search of constant depth. Each probe making a call to theorem 7.6 with error parameter $\Omega(\epsilon_i)$. Eventually we obtain a $(1+\epsilon_i)$-approximation where $\epsilon_i = 2^{-i}$ is at most the input error parameter ϵ, as desired.

We now bound the running time. The first set of $\log(n)$ calls to theorem 7.6 with constant error parameter take

$$O(n\log(n) + r\log(n)\log(r)) \tag{7.9}$$

time. Thereafter we have a constant number of calls to theorem 7.6 for each ϵ_i between 1 and $\epsilon/2$. For the leading $O(n)$ term, all these calls add up to $O(n\log(1/\epsilon))$ work which is dominated by $O(n\log(n))$ above. For the second term of the form $r\ln(n)\ln(r/\epsilon)/\epsilon^3$, the sum over all ϵ_i's is dominated by the smallest ϵ_i which is $\Omega(\epsilon)$, hence

$$O\big(r\ln(n)\ln(r/\epsilon)/\epsilon^3\big) \tag{7.10}$$

work in total. Adding together eqs. (7.9) and (7.10) gives the claimed running time. ∎

Applying the same modified binary search to the graphic matroid gives the following randomized algorithm for estimating the arboricity of a graph.

THEOREM 7.9. *An $(1\pm\epsilon)$-approximation to the strength of a graph can be computed with high probability in $O\big(m\log(n)\alpha(n) + n\log(n)(\log(n) + (\log\log(n) + \log(1/\epsilon))\alpha(n))/\epsilon^3\big)$ randomized time.*

8 Faster exact algorithms via augmentation

In this final section, we describe an augmenting path subroutine for k-fold matroid union and use it obtain a faster exact algorithms when $r \geq k^{1+o(1)}$.

LEMMA 8.1. *A packing of k independent sets produced by the $(1-\epsilon)$-approximate k-fold matroid union data structure can be extended to an optimum solution in time bounded by $O(\min\{n + \mathrm{OPT}\log(r), rk\log(kr)\})$ queries per additional element.*

Combining lemma 8.1 with the $(1-\epsilon)$-approximation algorithm, for appropriate choice of ϵ, leads to the following running time which is faster in the regime where $r \geq k^{1+o(1)}$. In particular, in the unweighted setting where $\mathrm{OPT} \leq kr \leq n$, we have a subquadratic upper bound of $\tilde{O}\big(n^{3/2}\big)$ independence queries.

THEOREM 8.1. *A maximum capacity packing of k independent sets can be computed in time bounded by $O\Big(n + \mathrm{OPT}\sqrt{n'\log(kr)}\Big)$ independence queries, where $n' = \min\{n + \mathrm{OPT}\log(r), rk\log(kr)\}$.*

Proof. Let $\epsilon > 0$ be a parameter to be determined. A $(1-\epsilon)\,\mathrm{OPT}$-capacity packing can be computed in $O(n + \mathrm{OPT}\log(kr)/\epsilon)$ independence queries. This can be augmented to an optimal solution in $O(n' + \mathrm{OPT}\log(r))$ time per augmentation. Thus the total running time is

$$O(n + \mathrm{OPT}\log(kr)/\epsilon + \epsilon\,\mathrm{OPT}(n' + \mathrm{OPT}\log(r))).$$

The last two terms are balanced by taking $\epsilon = \sqrt{(n' + \mathrm{OPT}\log(r))/\log(kr)}$ gives the claimed running time. Here we note that a constant factor approximation for OPT can be obtained by running the approximation matroid union algorithm with constant ϵ, and this suffices to balance the terms up to constant factors. ∎

It remains to prove lemma 8.1.

8.1 Initialization from matroid push-relabel We need to initialize our algorithm with a packing of k independent sets $I_1, \ldots, I_k$, whereas the k-fold matroid union algorithm from section 3 directly produces k bases $B_1, \ldots, B_k$. Such a packing $I_1, \ldots, I_k$ with the same total capacity can be easily obtained from $B_1, \ldots, B_k$ by dropping overpacked elements from some of the bases until there are no overpacked elements. However to preserve certain useful structures of $B_1, \ldots, B_k$ we carefully remove overpacked elements from the bases in the following greedy fashion.

Initially, we set $I_1 = B_1, \ldots, I_k = B_k$. While there is an overpacked element e (with respect to $I_1, \ldots, I_k$), we remove e from then independent set I_j of maximum index j. It is easy to see that at termination there are no overpacked elements while the objective value is preserved. By removing overpacked elements in such a fashion we also gain the following critical properties which we now define.

DEFINITION 2. *Let $I_1, \ldots, I_k$ be a packing of k independent sets. We say that $I_1, \ldots, I_k$ is* maximal *if for all uncovered elements e and all independent sets I_i we have $e \in I_i$. We say that $I_1, \ldots, I_k$ are in* decreasing order *if their spans are; that is, $I_{i+1} \subseteq \text{span}(I_i)$ for $i = 1, \ldots, k-1$.*

LEMMA 8.2. *Let $I_1, \ldots, I_k$ be a packing of k independent set obtained from the bases $B_1, \ldots, B_k$ output by the k-fold matroid union push-relabel algorithm in the greedy fashion described above. Then:*

(a) $I_1, \ldots, I_k$ is a maximal packing.

(b) $I_1, \ldots, I_k$ are in decreasing order.

Proof. Fix the configuration of $B_1, \ldots, B_k$ and $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$ at the end of the matroid push-relabel algorithm.

We first claim that any overpacked element e has $\ell(e) = 0$. Indeed, initially all elements have $\ell(e) = 0$, and elements can only be relabeled when they are uncovered. Meanwhile, an element can only be overpacked by the initial configuration, so an element that is overpacked at termination was overpacked – and never uncovered – all along.

Now, each I_i will contain all elements from B_i that are not overpacked. Since overpacked elements have level 0, we have $B_{i,\geq 1} \subseteq I_i \subseteq B_i$ for all i. Meanwhile, any uncovered element has level > 1. By invariant (II), for all uncovered elements e we have

$$e \in \mathcal{N}_{>1} \subseteq \text{span}(B_{i,\geq 1}) \subseteq \text{span}(I_i)$$

This establishes property (a).

Next we show property (b), which claims that the independent sets are in decreasing order. For each independent set I_i, we can express I_i as the disjoint union of $B_{i,\geq 1}$ and $I_{i,0} \stackrel{\text{def}}{=} I_i \cap B_{i,0}$. For each index $i \in [k-1]$, we have $B_{i+1,\geq 1} \subseteq \text{span}(B_{i,\geq 1})$ by the decreasing order of bases (see defintion 1) and $I_{i+1,0} \subseteq I_{i,0}$ because overpacked elements are removed from the independent sets of maximum index. Thus

$$I_{i+1} = B_{i+1,\geq 1} \cup I_{i+1,0} \subseteq \text{span}(B_{i,\geq 1}) \cup I_{i,0} \subseteq \text{span}(I_i),$$

as desired. ∎

8.2 Greedy sparsification Before proceeding to describe the augmenting path algorithm, we point out that by techniques by [51], one can assume $n \leq O(kr \ln(r))$.

LEMMA 8.3. *Let $I_1, \ldots, I_\ell$ be a maximal packing of $\ell \geq k(1 + \ln(r))$ sets. Then the size of the maximum k-fold in the (smaller) capacitated matroid induced by $I_1, \ldots, I_\ell$ is the same as in the input matroid.*

Proof. [Proof sketch] The proof is essentially the same as [51] which focused on base packing instead. In the proof, one replaces the role of dual characterization for base packing [18] with the dual characterization for matroid union [55]. ∎

[51] described how to construct such a packing greedily in $O(n + kr \ln(r))$ independence queries. The same construction extends here except we start with the maximal packing $I_1, \ldots, I_k$ given by the push-relabel algorithm and then extend it greedily. Thus we have the following.

LEMMA 8.4. *With a running time overhead of $O(\min\{n, kr \ln(r)\})$ independence queries, we may assume that $n \leq O(kr \ln(r))$.*

8.3 Setting up the auxiliary graph At a high-level, we have a packing of independent sets $I_1, \ldots, I_k$, and the immediate goal is to increase their total size. However it is not as simple as finding an uncovered element e to add to a set I_k — all the uncovered elements are spanned by all the independent sets because $I_1, \ldots, I_k$ is maximal. To extend the total size of $I_1, \ldots, I_k$ we may have to shuffle many of the elements from among the $I_1, \ldots, I_k$ to make room for one more element.

As is well-known, such an augmentation can be found be searching for a path in a directed auxiliary graph where each arc encodes a local exchange such as replacing one element with another in an independent set I_i, or moving an element from one I_i to another. Here we describe the auxiliary graph which is standard [39, 52].[4] We have two auxiliary vertices s and t which will act as the beginning and end of our search. For each element e, we have k auxiliary vertices $e^{(1)}, \ldots, e^{(k)}$. Each $e^{(i)}$ represents e in relation to B_i as will be made clear by the arcs which we now describe. We have four types of arcs.

1. $(s, e^{(i)})$ where e is uncovered and $e^{(i)} \notin B_i$. This arc represents trying to exchange $e^{(i)}$ into B_i.
2. $(e^{(i)}, d^{(i)})$ where $d \in B_i$, $e \notin B_i$, and $B_i - d + e$ is independent. This arc represents exchanging e for d in B_i.
3. $(d^{(i)}, d^{(j)})$ where $d \in B_i$, $d \in \text{span}(B_j) \setminus B_j$, and d is covered. This arc represents removing d from B_i and using the freed up capacity to initiate an exchange for d into B_j.
4. $(e^{(i)}, t)$ where e is not spanned by B_i. This arc represents inserting e into B_i.

In total, the auxiliary graph has $O(nk)$ vertices, and at most $O(nrk)$ arcs.

Paths from s to t in this graph have a very specific graph. Between s and t, the path consists of auxiliary vertices of the form $e^{(i)}$ that alternate between those where $e \notin B_i$ and where $e \in B_i$. There are always an odd number of such internal vertices, with one more of the former type. Arc-wise, the first arc is of type 1, the last arc is of type 4, and in-between the arcs alternate between type 2 and type 3, starting with type 2 and ending with type 2. All put together, an (s,t)-path in the auxiliary graph above corresponds to a sequence of exchanges, plus one final insertion, where the net effect is to increase the total size of $I_1, \ldots, I_k$ by 1. We call such a path an *augmenting path* if it maintains feasibility. That is, the sets $I_1, \ldots, I_k$ remain independent, and we do not overpack elements.

(s,t)-paths are not necessarily augmenting paths. On one hand, it is easy to see that these operations will not violate any capacity constraints, since any time a covered element is inserted into a set, it is preceded by removing the same element from another set. As per the feasibility of the $I_1, \ldots, I_k$, while each exchange (encoded by an arc of type 2) would individually maintain the independence of each set, all the exchanges taken together do not necessarily maintain independence. However, the following criteria outlines conditions in which a sequence of exchanges to the same independent set does maintain independence. This criteria is standard and have long been used to justify matroid intersection and partition algorithms. (Here the wording is from [17]).

FACT 8.1. *Given a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, let $I \in \mathcal{I}$, and let $e_1, \ldots, \notin I$, $d_1, \ldots, d_p \in I$, and optionally $e_{p+1} \notin I$ be distinct elements such that:*

(a) For $i = 0, \ldots, p$, $I - d_i + e_i \in \mathcal{I}$.

(b) For $0 \le j < i \le p$, $I - d_i + e_j \notin \mathcal{I}$.

(c) $I + e_{p+1} \in \mathcal{I}$ (when including e_{p+1}).

Then $I' = I - d_1 - \cdots - d_p + e_1 + \cdots + e_{p+1} \in \mathcal{I}$.

The exchanges corresponding to an (s,t)-path in the auxiliary graph can only violate property (b) out of the three properties in fact 8.1. Below, we design a subroutine that, given an (s,t)-path in the auxiliary graph, extracts a subpath that is an augmenting path, by efficiently identifying and removing violations to property (b), as follows.

LEMMA 8.5. *Given an (s,t)-path of length ℓ in the auxiliary graph, one can compute an augmenting subpath in running time bounded by $O(\ell \log(r))$ independence queries.*

[4] This auxiliary graph can also be interpreted through the lens of matroid intersection with some modifications.

Proof. Fix an independent set I_i. Suppose the path encodes inserting $e_1, \ldots, e_p \notin I_i$ in exchange for $d_1, \ldots, d_p \in I_i$, respectively and in sequence. This means that the auxiliary arcs $(e_j^{(i)}, d_j^{(i)})$ for $j = 1, \ldots, p$ appear in the auxiliary path in that order. If $I_i - d_1 - \cdots - d_p + e_1 + \cdots + e_p \notin \mathcal{I}$, then by fact 8.1, there must be indices $1 \leq j_1 < j_2 \leq p$ such that $I_i - d_{j_2} + e_{j_1} \in \mathcal{I}$. This exchange implies $\left(e_{j_1}^{(i)}, d_{j_2}^{(i)}\right)$ is an arc in the auxiliary graph, and so we can shorten our (s,t)-path by replacing all arcs between (and including) $(e_{j_1}^{(i)}, d_{j_1}^{(i)})$ and $(e_{j_2}^{(i)}, d_{j_2}^{(i)})$ with $\left(e_{j_1}^{(i)}, d_{j_2}^{(i)}\right)$. Let us call such a pair (e_{j_1}, d_{j_2}) a *chordal pair* as it represents a chord with respect to our (s,t)-path. Our goal is to repeatedly identify chordal pairs (j_1, j_2) and shorten the path until we arrive at a shorter sequence of exchange for I_i that is feasible.

A pair of indices (j_1, j_2) as described above can be identified efficiently as follows. For $j_1 = p$ down to 1, we binary search for the first index j_2 such that $I - d_1 \cdots - d_{j_1} - d_{j_2} + e_{j_1} \in \mathcal{I}$. Then j_2 is the first index such that d_{j_2} is in the circuit of $I + e_{j_1}$. Now, if $j_1 = j_2$, then this verifies that $I - d_j + e_{j_2} \notin \mathcal{I}$ for all $j < j_2$, as desired. Otherwise (j_1, j_2) represents a chord which can be used to shorten the (s,t)-path. We shortcut the path at (j_1, j_2). We then decrease j_1 and continue the search, short cutting or validating each e_{j_1} until we have certified that there are no chordal pairs remaining. The total running time to prune all chordal pairs for I_i is bounded above by $O(p \log(r))$, where p refers to the number of exchanges originally encoded in the path, before pruning.

The description above was for the case where the path encoded a sequence of exchanges and not an additional insertion. However the discussion extends immediately to the case where the path encodes a sequence of p exchanges plus an additional insertion. Again we can prune all chordal pairs for a fixed independent set I_i, and guarantee that the remaining sequence of exchanges and insertion maintains the independence of I_i, in running time bounded above by $O(p \log(r))$ independence queries.

Overall the algorithm process each independent set I_i one at a time. For each I_i we prune chordal pairs and shorten the (s,t)-path so that the remaining exchanges maintain the independence of I_i. The total running time over all sets I_i is bounded above $O(\ell \log(r))$ independence queries, since $O(\ell)$ counts the total number of exchanges in the original path over all independent sets I_i. ∎

8.4 Efficiently searching for (s,t)-paths in the auxiliary graph We have now set up an auxiliary graph and shown that any (s,t)-path can be efficiently converted to an augmenting path. It remains to design an efficient method to find an (s,t)-path. Of course one could construct the graph explicitly by testing for the presence for every arc above, but as remarked above the size of the overall graph is bigger than the desired running time.

Recall that our goal is to find an (s,t)-path or conclude that no such paths exist. In particular we are not strictly required to test and traverse all the arcs in the graph. Our goal is to develop a search our algorithm with running time bounded by a number of independence queries proportional (up to logarithmic terms) to the total number of vertices in the graph.

We will show how to modify BFS to this purpose.[5] Recall that BFS marks vertices when they are first visited before adding them to the queue of elements to be searched next. Of course, the marks record that a vertex has been visited and prevents infinite loops. In the context of our implicit auxiliary graph we can also avoid testing for the existence of an arc if the head of the arc is already marked.

For the sake of efficiency we will impose the following invariant on the set of marked vertices. For each independent set I_i in our packing, let M_i denote the set of auxiliary vertices $e^{(i)}$, where $e \in I_i$, that have been marked. We will strictly adhere to the invariant that the marked sets $M_1, \ldots, M_k$ are in decreasing order (in the same sense as $I_1, \ldots, I_k$, cf. definition 2). To maintain this invariant we introduce the following subroutine.

Predecessor search. Whenever we are about to mark an element $e^{(i)}$ where $e \in B_i$, we need to ensure that $e \in \text{span}(M_j)$ for all $j < i$. In this case we launch a *predecessor search*, or *pre-search* for short, at e_i, which executes the following steps.

Pre-searching an auxiliary element $e^{(i)}$: Let $e \in B_i$.

1. While $e \notin \text{span}(M_{i-1})$:
   A. For each element $d \in I_{i-1} \setminus \text{span}(M_j)$ such that $I_{i-1} - d + e \in \mathcal{I}$.
   B. Recursively pre-search d_j.
   C. Mark d_j, and record the arc (e_i, d_j), and add d_j to the (outer, BFS) search queue.

[5] Other marking based search algorithms such as DFS could have been used instead.

The steps above are described at a high-level and we will discuss concrete details and issues of efficiency later. First we establish why the (high-level) steps above maintain $M_1, \ldots, M_k$ in decreasing order. (The reason we order this fact before analyzing concrete implementation details is because we will use the decreasing order of the $M_1, \ldots, M_k$ to make the implementation more efficient.)

LEMMA 8.6. *Suppose $I_1, \ldots, I_k$ and $M_1, \ldots, M_k$ are both in decreasing order, and let $e \in B_i$. Then pre-searching $e^{(i)}$ maintains the $M_1, \ldots, M_k$ in decreasing order. If $i > 1$, we also have $M_i + e \subseteq \text{span}(M_{i-1})$.*

Proof. We prove the claim by induction on i. If $i = 1$ then there is nothing to do and the claim is vacuous. Now, let $i > 1$ and assume the claim holds for $i - 1$. Consider a pre-search to an auxiliary vertex $e^{(i)}$ where $e \in I_i$. Recall that I_{i-1} spans e because $I_1, \ldots, I_k$ is in decreasing order. Consequently if M_{i-1} does not span e then there are still other elements d in the circuit of $I_{i-1} + e$ that are not in M_{i-1}. As long as M_{i-1} does not span e, the pre-search subroutine repeated selects such an element d and calls pre-search on $d^{(i-1)}$ before marking $d^{(i-1)}$. By induction the pre-search on $d^{(i-1)}$ ensures that we can safely mark $d^{(i-1)}$ while preserving $M_1, \ldots, M_k$ in decreasing order. The pre-search at e ends with $M_1, \ldots, M_k$ still in decreasing order, and with M_{i-1} spanning e. ∎

The implicit search algorithm. We now describe the search algorithm which employs pre-search as a black box. Here the challenge is to search the auxiliary graph without knowing all the arcs explicitly. At a high-level, by leveraging the decreasing order of both $I_1, \ldots, I_k$ and $M_1, \ldots, M_k$, up to logarithmic factors, we are able to limit our queries to those that produce arcs to as yet univisited vertices.

The search algorithm behaves differently depending on the type of auxiliary vertex we're search. Here we have three types. The first is at the source s. The second is at an at auxiliary vertices $e^{(i)}$ where either $e \in I_i$ or $i = k$. In particular we do not call search directly on auxiliary vertices $e^{(i)}$ where $e \notin I_i$ and $i < k$, for efficiency reasons made clearer below.

We start with the search routine for s, which simply searches every uncovered element e.

Searching at the source s:

1. For each uncovered element e, if $e^{(k)}$ is unmarked, then mark $e^{(k)}$, record the arc $(s, e^{(k)})$, and call search on $e^{(k)}$.

Next we describe the search algorithm for elements of the form $e^{(i)}$.

Searching at an auxiliary vertex $e^{(i)}$:

1. If $e \notin \text{span}(I_k)$:
   (a) Let j be the first index such that $e \notin \text{span}(I_j)$.
   (b) Mark $e^{(j)}$, mark t, and record the arcs $(e^{(i)}, e^{(j)})$ and $(e^{(j)}, t)$. Signal that we have found an (s, t)-path.
2. Otherwise, while $e \notin \text{span}(M_k)$:
   (a) Let $d \in I_k \setminus M_k$ be such that $I - d + e \in \mathcal{I}$.
   (b) Pre-search $d^{(k)}$.
   (c) Mark $e^{(k)}$ and $d^{(k)}$. Record the arc $(e^{(i)}, e^{(k)})$ and $(e^{(k)}, d^{(k)})$. Add $d^{(k)}$ to the search queue.

LEMMA 8.7. *After searching an auxiliary vertex $e^{(i)}$, where $e \in I_i$, we have the following:*

1. *If there is an index j such that $e \notin \text{span}(I_j)$, then for the first such index j, the path $(e^{(i)}, e^{(j)}, t)$ is recorded and the vertices $e^{(j)}$ and t are marked.*
2. *If not, then we have $e \in \text{span}(M_j)$ for all indices j.*

Proof. The first case is straightforward from the code. In the second case, we have $e \in \text{span}(I_i)$ for all i. The search at e exits only when $e \in \text{span}(M_k)$. Because $M_1, \ldots, M_k$ are in decreasing order, we then have $e \in \text{span}(M_i)$ for all i. ∎

LEMMA 8.8. *After searching $e^{(k)}$ for an uncovered element e, we have $e \in \text{span}(M_i)$ for all i.*

Proof. We know that $e in \text{span}(I_k)$ because e is uncovered and $I_1, \ldots, I_k$ is a maximal packing. The search exits only when $e \in \text{span}(M_k)$. Because $M_1, \ldots, M_k$ are in decreasing order, we then have $e \in \text{span}(M_i)$ for all i. ∎

LEMMA 8.9. *The search finds a path from s to t if one exists.*

Proof. The algorithm records arcs that jointly contain paths to all marked vertices. Thus the algorithm marks t and signals that an (s,t)-paths is found, then the record arcs contain the desired arc. (Parent pointers, or a graph search through the recorded arcs, will produce an (s,t)-path.)

Observe that if the search does not signal an (s,t)-path, it will still mark all reachable auxiliary vertices except possibly for an auxiliary vertex $e^{(i)}$ where e is uncovered and $e \notin I_i$. However the preceding lemma ensures that $e \in \text{span}(M_i)$, so any of the auxiliary vertices reachable from $e^{(i)}$ have still been explored. Thus if t is reachable, then it will be marked and an (s,t)-path will be found, as desired. ∎

Efficiency of search and pre-search:

LEMMA 8.10. *A search from s takes running time bounded by $O(n + \text{OPT}\log(r) + \log(k))$ independence queries.*

Proof. Let ℓ denote the total number of auxiliary vertices of the $e^{(i)}$ that are marked. Note that $\ell \leq O(n + \text{OPT})$ because if we mark an auxiliary vertex $e^{(i)}$ with $e \notin I_i$ and $i < k$, then there is also an outgoing arc to some $d^{(j)}$ were $d \in I_j$. Each search or pre-search routine at an element e consists of a constant number of queries plus an unspecified number of queries per auxiliary vertex that gets marked, in order to search for the element that is marked. The constant number of queries per search add up to at most $O(n + \ell)$ in total, because each auxiliary vertex is pre-searched or searched at most once (before or after it is marked). Next we explain how to implement the latter category of queries in a logarithmic number of queries per marked element. Here we have two types of searches for the next marked element.

The first is to identify elements $d \in I_i \setminus M_i$ such that $I - d + e \in \mathcal{I}$, given e and an index i. This can be done by maintaining I_i in any order such that M_i comes first, and searching for the first prefix of this ordering that spans e. The last element in this ordering gives an element d. The search requires $O(\log(r))$ oracle queries. (Note that it is easy to maintain this ordering as elements are added to M_i.)

The second type is to identify an index j, as small as possible, such that $e \notin \text{span}(I_j)$. Since $I_1, \ldots, I_k$ is in decreasing order, we can binary search for the first index j with $O(\log k)$ probes. Each probe corresponds to 1 independence query. Additionally, this search also ends the search, so we only search for such an index j once. ∎

Preserving $I_1, \ldots, I_k$ in decreasing order. Next we address the fact that our particular choice of augmentations keeps $I_1, \ldots, I_k$ in decreasing order.

LEMMA 8.11. *An augmentation induced by the search algorithm keeps $I_1, \ldots, I_k$ in decreasing order.*

Proof. Suppose the end of the path encodes inserting an element e in I_i. We first note that the preceding exchanges do not effect the span. Then, when inserting e into I_i, we know that $e \in \text{span}(I_{i-1})$ for all $j < i$ by choice of i. Thus $I_i + e \in \text{span}(I_{i-1})$ and we preserve the decreasing order. ∎

Putting it all together. Together, lemmas 8.5 and 8.10 gives the overall running time to find an augmenting path. (To apply lemma 8.5, we note that the length of any (s,t)-path is at most $O(\text{OPT})$.) This completes the proof of lemma 8.1.

8.5 Graphic matroids We conclude the section by translating the oracle-based running time into concrete running times for the graphic matroid. The ideas here are similar to those in section 5. By maintain disjoint union data structures over I_i, we can maintain and implement independence queries for I_i with $\alpha(n)$ overhead. By also managing each I_i in a link-cut trees, we can retrieve unmarked edges $d \in I_k \setminus M_k$ for exchanging in $O(\log n)$ time (bypassing the binary search from the oracle model). Retracing the proofs of lemmas 8.5 and 8.10, and keeping in mind that the maximum length of a path is OPT, shows that it takes $O(m'\alpha(n) + \text{OPT}\log(nk))$ time per augmenting path for $m' = \min\{m, n\log(nk)\}$. Balancing the choice of ϵ with the $O(m\alpha(n) + \text{OPT}\log(n)(\log(n) + \log(k)\alpha(n))/\epsilon)$ running time for the $(1-\epsilon)$-approximation, we obtain the following.

THEOREM 8.2. *For graphs with integer edge capacities, a forest packing of maximum total size can be computed in*

$$O\Big(m\alpha(n) + \sqrt{\mathrm{OPT}(\min\{m, n\log(nk)\}\alpha(n) + \mathrm{OPT}\log(nk))\log(n)(\log(n) + \log(k)\alpha(n))}\Big)$$

time.

A simplified running time for connected graphs, observing that $\mathrm{OPT} \geq \Omega(n)$ for $k \geq 1$, is

$$O\Big(m\alpha(n) + \mathrm{OPT}^{3/2}\log^2(nk)\alpha^2(n)\Big).$$

References

[1] Ravindra K. Ahuja and James B. Orlin. "A Fast and Simple Algorithm for the Maximum Flow Problem". In: *Oper. Res.* 37.5 (1989), pp. 748–759.

[2] Francisco Barahona. "Packing Spanning Trees". In: *Math. Oper. Res.* 20.1 (1995), pp. 104–115.

[3] Francisco Barahona. "Separating from the dominant of the spanning tree polytope". In: *Oper. Res. Lett.* 12.4 (1992), pp. 201–203. URL: https://www.sciencedirect.com/science/article/abs/pii/0167637792900455.

[4] András A. Benczúr and David R. Karger. "Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs". In: *SIAM J. Comput.* 44.2 (2015), pp. 290–319.

[5] Joakim Blikstad. "Breaking O(nr) for Matroid Intersection". In: *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 31:1–31:17. URL: https://doi.org/10.4230/LIPIcs.ICALP.2021.31.

[6] Joakim Blikstad, Jan van den Brand, Sagnik Mukhopadhyay, and Danupon Nanongkai. "Breaking the quadratic barrier for matroid intersection". In: *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. Ed. by Samir Khuller and Virginia Vassilevska Williams. ACM, 2021, pp. 421–432. URL: https://doi.org/10.1145/3406325.3451092.

[7] Joakim Blikstad, Sagnik Mukhopadhyay, Danupon Nanongkai, and Ta-Wei Tu. "Fast Algorithms via Dynamic-Oracle Matroids". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*. Ed. by Barna Saha and Rocco A. Servedio. ACM, 2023, pp. 1229–1242. ISBN: 978-1-4503-9913-5. URL: https://doi.org/10.1145/3564246. Full version available at https://arxiv.org/abs/2302.09796.

[8] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, Sahil Singla, and Sam Chiu-wai Wong. "Faster Matroid Intersection". In: *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. Ed. by David Zuckerman. IEEE Computer Society, 2019, pp. 1146–1168. URL: https://doi.org/10.1109/FOCS.2019.00072.

[9] Chandra Chekuri and Kent Quanrud. "Near-Linear Time Approximation Schemes for some Implicit Fractional Packing Problems". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. SIAM, 2017, pp. 801–820.

[10] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. "Maximum Flow and Minimum-Cost Flow in Almost-Linear Time". In: *CoRR* abs/2203.00671 (2022). arXiv: 2203.00671. URL: https://doi.org/10.48550/arXiv.2203.00671.

[11] Eddie Cheng and William H. Cunningham. "A Faster Algorithm for Computing the Strength of a Network". In: *Inf. Process. Lett.* 49.4 (1994), pp. 209–212.

[12] Joseph Cheriyan and S. N. Maheshwari. "Analysis of Preflow Push Algorithms for Maximum Network Flow". In: *SIAM J. Comput.* 18.6 (1989), pp. 1057–1086. URL: https://epubs.siam.org/doi/abs/10.1137/0218072?journalCode=smjcat. Preliminary version in FSTTSC, 1988.

[13] Boris V. Cherkassky and Andrew V. Goldberg. "On Implementing the Push-Relabel Method for the Maximum Flow Problem". In: *Algorithmica* 19.4 (1997), pp. 390–410. Preliminary version in IPCO, 1995.

[14] Boris V. Cherkassky, Andrew V. Goldberg, and Paul Martin. "Augment or Push: A Computational Study of Bipartite Matching and Unit-Capacity Flow Algorithms". In: *ACM J. Exp. Algorithmics* 3 (1998), p. 8. Preliminary version in WAE, 1997.

[15] William H. Cunningham. "Improved Bounds for Matroid Partition and Intersection Algorithms". In: *SIAM J. Comput.* 15.4 (1986), pp. 948–957.

[16] William H. Cunningham. "Optimal Attach and Reinforcement of a Network". In: *J. ACM* 32.3 (1985), pp. 549–561.

[17] William H. Cunningham. "Testing membership in matroid polyhedra". In: *J. Comb. Theory, Ser. B* 36.2 (1984), pp. 161–188. URL: https://www.sciencedirect.com/science/article/pii/0095895684900236.

[18] Jack Edmonds. "Lehman's switching game and a theorem of Tutte and Nash-Williams". In: *Journal of Research National Bureau of Standards Section B* 69 (1965), pp. 72–77.

[19] Jack Edmonds. "Minimum partition of a matroid into independent subsets". In: *Journal of Research National Bureau of Standards Section B* 69 (1965), pp. 67–72.

[20] Shimon Even and Robert Endre Tarjan. "Network Flow and Testing Graph Connectivity". In: *SIAM J. Comput.* 4.4 (1975), pp. 507–518.

[21] Lisa Fleischer and Satoru Iwata. "A push-relabel framework for submodular function minimization and applications to parametric optimization". In: *Discret. Appl. Math.* 131.2 (2003), pp. 311–322. URL: https://doi.org/10.1016/S0166-218X(02)00458-4.

[22] Lisa Fleischer and Satoru Iwata. "Improved algorithms for submodular function minimization and submodular flow". In: *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*. Ed. by F. Frances Yao and Eugene M. Luks. ACM, 2000, pp. 107–116. URL: https://doi.org/10.1145/335305.335318.

[23] András Frank. *Connections in combinatorial optimization*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2011.

[24] András Frank and Zoltán Miklós. "Simple push-relabel algorithms for matroids and submodular flows". In: *Japan Journal of Industrial and Applied Mathematics* 29 (2012), pp. 419–439. URL: http://web.cs.elte.hu/egres/tr/egres-10-05.pdf.

[25] Satoru Fujishige and Xiaodong Zhang. "A Push/Relabel framework for submodular flows and its definement for 0-1 submodular flows". In: *Optimization* 38.2 (1996), pp. 133–154.

[26] Satoru Fujishige and Xiaodong Zhang. "New algorithms for the intersection problem of submodular systems". In: *Japan Journal of Industrial and Applied Mathematics* 9 (1992), pp. 368–392.

[27] Harold N. Gabow. "A Matroid Approach to Finding Edge Connectivity and Packing Arborescences". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*. Ed. by Cris Koutsougeras and Jeffrey Scott Vitter. ACM, 1991, pp. 112–122. URL: https://doi.org/10.1145/103418.103436.

[28] Harold N. Gabow. "A Matroid Approach to Finding Edge Connectivity and Packing Arborescences". In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*. Ed. by Cris Koutsougeras and Jeffrey Scott Vitter. ACM, 1991, pp. 112–122.

[29] Harold N. Gabow. "Algorithms for Graphic Polymatroids and Parametric s-Sets". In: *J. Algorithms* 26.1 (1998), pp. 48–86. Preliminary version in SODA, 1995.

[30] Harold N. Gabow and K. S. Manu. "Packing algorithms for arborescences (and spanning trees) in capacitated graphs". In: *Math. Program.* 82 (1998), pp. 83–109. Preliminary version in IPCO 1995.

[31] Harold N. Gabow and Matthias F. M. Stallmann. "Efficient Algorithms for Graphic Matroid Intersection and Parity (Extended Abstract)". In: *Automata, Languages and Programming, 12th Colloquium, Nafplion, Greece, July 15-19, 1985, Proceedings*. Ed. by Wilfried Brauer. Vol. 194. Lecture Notes in Computer Science. Springer, 1985, pp. 210–220. URL: https://doi.org/10.1007/BFb0015746.

[32] Harold N. Gabow and Herbert H. Westermann. "Forests, Frames, and Games: Algorithms for Matroid Sums and Applications". In: *Algorithmica* 7.5&6 (1992), pp. 465–497. Preliminary version in STOC, 1988.

[33] Giorgio Gallo, Michael D. Grigoriadis, and Robert Endre Tarjan. "A Fast Parametric Maximum Flow Algorithm and Applications". In: *SIAM J. Comput.* 18.1 (1989), pp. 30–55. URL: https://doi.org/10.1137/0218003.

[34] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. "Dynamic graph connectivity with improved worst case update time and sublinear space". In: *CoRR* abs/1509.06464 (2015). arXiv: 1509.06464. URL: http://arxiv.org/abs/1509.06464.

[35] Andrew V. Goldberg and Robert E. Tarjan. "Finding Minimum-Cost Circulations by Successive Approximation". In: *Math. Oper. Res.* 15.3 (1990), pp. 430–466. Preliminary verison in STOC, 1987.

[36] Andrew V. Goldberg and Robert Endre Tarjan. "A new approach to the maximum-flow problem". In: *J. ACM* 35.4 (1988), pp. 921–940. Preliminary version in STOC, 1986.

[37] Andrew Vladislav Goldberg. *A New Max-Flow Algorithm*. Tech. rep. MIT/LCS/ TM-291. Cambridge, Massachusetts: Laboratory for Computer Science, Massachusetts Institute of Technology, 1985.

[38] Andrew Vladislav Goldberg. "Efficient graph algorithms for sequential and parallel computers". PhD thesis. Massachusetts Institute of Technology, Cambridge, MA, USA, 1987. URL: http://hdl.handle.net/1721.1/14912.

[39] C. Greene and T.L. Magnanti. "Some abstract pivot algorithms". In: *SIAM Journal on Applied Mathematics* 29 (1975), pp. 530–539.

[40] Dan Gusfield. "Computing the Strength of a Graph". In: *SIAM J. Comput.* 20.4 (1991), pp. 639–654.

[41] Dan Gusfield. "Connectivity and Edge-Disjoint Spanning Trees". In: *Inf. Process. Lett.* 16.2 (1983), pp. 87–89. URL: https://doi.org/10.1016/0020-0190(83)90031-5.

[42] Jianxiu Hao and James B. Orlin. "A Faster Algorithm for Finding the Minimum Cut in a Directed Graph". In: *J. Algorithms* 17.3 (1994), pp. 424–446. Preliminary version in SODA, 1992.

[43] Monika Rauch Henzinger, Satish Rao, and Harold N. Gabow. "Computing Vertex Connectivity: New Bounds from Old Techniques". In: *J. Algorithms* 34.2 (2000), pp. 222–250. Preliminary version in FOCS, 1996.

[44] John E. Hopcroft and Richard M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM J. Comput.* 2.4 (1973), pp. 225–231. Preliminary version in FOCS, 1971.

[45] Hiroshi Imai. "Network-Flow Algorithms for Lower-Truncated Transversal Polymatroids". In: *Journal of the Operations Research Society of Japan* 26.3 (1983), pp. 186–211.

[46] Satoru Iwata, S. Thomas McCormick, and Maiko Shigeno. "A fast cost scaling algorithm for submodular flow". In: *Inf. Process. Lett.* 74.3-4 (2000), pp. 123–128. URL: https://doi.org/10.1016/S0020-0190(00)00052-1.

[47] Satoru Iwata, Kazuo Murota, and Maiko Shigeno. "A Fast Parametric Submodular Intersection Algorithm for Strong Map Sequences". In: *Math. Oper. Res.* 22.4 (1997), pp. 803–813. URL: https://doi.org/10.1287/moor.22.4.803.

[48] Satoru Iwata and James B. Orlin. "A simple combinatorial algorithm for submodular function minimization". In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*. Ed. by Claire Mathieu. SIAM, 2009, pp. 1230–1237. URL: http://dl.acm.org/citation.cfm?id=1496770.1496903.

[49] Bruce M. Kapron, Valerie King, and Ben Mountjoy. "Dynamic graph connectivity in polylogarithmic worst case time". In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. Ed. by Sanjeev Khanna. SIAM, 2013, pp. 1131–1142. ISBN: 978-1-61197-251-1. URL: https://doi.org/10.1137/1.9781611973105.81.

[50] David R. Karger. "Minimum cuts in near-linear time". In: *J. ACM* 47.1 (2000), pp. 46–76. Preliminary version in STOC, 1996.

[51] David R. Karger. "Random sampling and greedy sparsification for matroid optimization problems". In: *Math. Program.* 82 (1998), pp. 41–81. Preliminary version in FOCS 1993.

[52] Donald E. Knuth. *Matroid Partitioning*. Tech. rep. Report STAN-CS-73-342. Stanford, California: Computer Science Department, Stanford University, 1974.

[53] Harihar Narayanan. "A rounding technique for the polymatroid membership problem". In: *Linear Algebra and its Applications* 221 (1995), pp. 41–57.

[54] C. St. J. A. Nash-Williams. "Edge-disjoint spanning trees of finite graphs". In: *J. London Math. Soc.* 36 (1961), pp. 445–450.

[55] Crispin St. John Alvah Nash-Williams. "An application of matroids to graph theory". In: *Theory of Graphs — International Symposium — Théorie des graphes — Journées internationales d'étude (Rome, 1966)*. Ed. by R. Rosenstiehl. Gordan, Breach, New York, and Dunod, Paris, 1967, pp. 263–265.

[56] Huy L. Nguyen. "A note on Cunningham's algorithm for matroid intersection". In: *CoRR* abs/1904.04129 (2019). arXiv: 1904.04129. URL: http://arxiv.org/abs/1904.04129.

[57] Kent Quanrud. *Quotient sparsification for submodular functions*. Nov. 2022. URL: https://kentq.s3.amazonaws.com/submodular-sparsification.pdf.

[58] James Roskind and Robert E. Tarjan. "A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees". In: *Math. Oper. Res.* 10.4 (1985), pp. 701–708. URL: https://doi.org/10.1287/moor.10.4.701.

[59] Thatchaphol Saranurak and Di Wang. "Expander Decomposition and Pruning: Faster, Stronger, and Simpler". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 2616–2635.

[60] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Vol. 24. Algorithms and Combinatorics. Springer, 2003.

[61] Daniel Dominic Sleator and Robert Endre Tarjan. "A Data Structure for Dynamic Trees". In: *J. Comput. Syst. Sci.* 26.3 (1983), pp. 362–391. Preliminary version in STOC 1981.

[62] V. A. Trubin. "Strength of a graph and packing of trees and branchings". In: *Cybernetics and Syst. Analysis* 29.3 (1993), pp. 379–384.

[63] Levent Tunçel. "On the Complexity of Preflow-Push Algorithms for Maximum-Flow Problems". In: *Algorithmica* 11.4 (1994), pp. 353–359.

[64] W. T. Tutte. "On the problem of decomposing a graph into n connected components". In: *J. London Math. Soc.* 36 (1961), pp. 221–230.

A Additional background

In this section, we provide some additional background omitted from section 1.1 and not appearing elsewhere in the paper.

Recall the problem of maximizing the size of a forest packing (i.e., k-fold matroid union problem in the graphic matroid). There are several previous works on this problem [31, 32, 45, 58]. [32] obtained several running times that are each optimal for some range of parameters, of which we highlighted the most comparable in section 1.1.

Packing spanning trees and network strength have important additional connections via the classical theorem of Nash-Williams [54] and Tutte [64] which we now describe. Given a partition of $(V_1, \ldots, V_k)$ of vertex set V, the *strength* of the partition is defined as the ratio

$$|\partial(V_1, \ldots, V_k)|/(k-1),$$

where $\partial(V_1, \ldots, V_k)$ denotes the set of edges cut by the partition $V_1, \ldots, V_k$. One interpretation of the strength of a partition, given by Cunningham [16] in the context of network vulnerability, is that the strength reflects a cost per additional connected component created by the cut. The *(network) strength* of a graph is defined as the minimum strength over all partitions. Network strength was proposed as a measure of network vulnerability by Gusfield [41]. The definitions extend naturally to positive edge capacities. [54, 64]'s theorem shows that the network strength equals the maximum size of any (fractional) tree packing; the maximum size of any integral tree packing is the floor of the network strength.

There are several works on packing spanning trees in capacitated and uncapacitated graphs [2, 17, 28, 30, 32, 45, 58, 62]. Besides the running times section 1.1, [30] obtains a strongly polynomial running time of

$O(n^3 m \log(n^2/m))$. Beside via packing spanning trees, there is further work on computing the strength directly [3, 11, 16, 29, 40] via flow or parametric flow, culminating in $O(n^2 m \log(n^2/m))$-time algorithms for computing the strength via parametric flow (and related techniques) [11, 29]. For covering by trees, besides the running times mentioned in section 1.1, [30] also obtains an $O(n^3 m \log(n^2/m))$ running time for covering by trees in the capacitated graphs, the same as for packing.