Quotient sparsification for submodular functions

Kent Quanrud*

Abstract

Graph sparsification has been an important topic with many structural and algorithmic consequences. Recently hypergraph sparsification has come to the fore and has seen exciting progress. In this paper we take a fresh perspective and show that they can be both be derived as corollaries of a general theorem on sparsifying matroids and monotone submodular functions.

Quotients of matroids and monotone submodular functions generalize k-cuts in graphs and hypergraphs. We show that a weighted ground set of a monotone submodular function f can be sparsified while approximately preserving the weight of every quotient of f with high probability in randomized polynomial time.

This theorem conceptually unifies cut sparsifiers for undirected graphs [7] with other interesting applications. One basic application is to reduce the number of elements in a matroid while preserving the weight of every quotient of the matroid. For hypergraphs, the theorem gives an alternative approach to the hypergraph cut sparsifiers obtained recently in [12], that also preserves all k-cuts. Another application is to reduce the number of points in a set system while preserving the weight of the union of every collection of sets. We also present algorithms that sparsify hypergraphs and set systems in nearly linear time, and sparsify matroids in nearly linear time and queries in the rank oracle model.

1 Introduction

Graph sparsification has been an important topic with many structural and algorithmic consequences. Recently hypergraph sparsification has come to the fore and has seen exciting progress. In this paper we take a fresh perspective and show that they can be both be derived as corollaries of a general theorem on sparsifying matroids and monotone submodular functions.

To formalize these ideas we require some preliminary definitions. A *matroid* is a set system $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ where $\mathcal{N}$ is a *ground set* of elements, and $\mathcal{I} \subseteq 2^{\mathcal{N}}$ is the family of *independent sets* satisfying the following properties:

(a) $\emptyset \in \mathcal{I}$.
(b) If $J \in \mathcal{I}$, and $I \subseteq J$, then $I \in \mathcal{I}$.
(c) If $I, J \in \mathcal{I}$, and $|I| < |J|$, then there exists an element $e \in J \setminus I$ such that $I + e \in \mathcal{I}$.

The *rank* of a set S is the maximum cardinality of any independent subset of S, and denoted $\operatorname{rank}(S)$. The rank of $\mathcal{M}$ is defined as the maximum cardinality of any independent set. The *span* of a set S, denoted $\operatorname{span}(S)$, is the set of elements e such that $\operatorname{rank}(S + e) = \operatorname{rank}(S)$ (including those in S). A set S is *closed* if $S = \operatorname{span}(S)$.

An illustrative example of a matroid is the *graphic matroid*. Given an undirected graph $G = (V, E)$, the graphic matroid is the matroid with ground set $\mathcal{N} = E$, and independent sets consisting of all edge sets $F \subseteq E$ that form a forest in G (or equivalently, that contain no cycles). For $F \subseteq E$, $\operatorname{span}(F)$ is the set of all edges $\{u, v\}$ where u and v are in the same connected component in the graph induced by G. If G is connected and has n vertices, the rank of the graphic matroid is $n - 1$.

Our work is centered on the notion of *quotients* of matroids and submodular functions. We first illustrate the notion for the graphic matroid. Consider a set of edges $F \subseteq E$, which induces $n - \operatorname{rank}(F)$ connected components in G. The edges spanned by F are the edges with both endpoints in the same connected component of F. The remaining edges, $E \setminus \operatorname{span}(F)$, are the edges cut by the connected components of F.

$E \setminus \operatorname{span}(F)$ is an example of a *quotient* in the graphic matroid. In general, for a matroid $\mathcal{M}$, a quotient is any set that is the complement of a closed set. Equivalently, $Q \subseteq \mathcal{N}$ is a quotient iff $Q = \mathcal{N} \setminus \operatorname{span}(S)$ for some set S. In the graphic matroid, the quotients are exactly the sets of edges cut by partitions $(U_1, \dots, U_k)$ of V (for arbitrary k), i.e., the *k-cuts* of G. This includes, but is not limited to, the edge cuts of G in the standard sense where $k = 2$.

This work extends beyond matroid quotients to quotients of monotone submodular function. To introduce the latter, let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be a real-valued set function over $\mathcal{N}$. f is *monotone* if $f(S)$ is nondecreasing in S; i.e.,

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

$f(S) \leq f(T)$ for $S \subseteq T$. f is *submodular* if

$$f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$$

for all $S, T \subseteq \mathcal{N}$. Equivalently, f is submodular if it expresses decreasing marginal values. This means that $f(S \,|\, T) \overset{\text{def}}{=} f(S \cup T) - f(T)$ is nonincreasing in T, for all $S, T \subseteq \mathcal{N}$.

One example of a monotone submodular function is the rank function. Another is the coverage function with nonnegative weights. Here we have a family of n sets $S_1, \ldots, S_n \subseteq \mathcal{U}$ and a weight function $w : \mathcal{U} \to \mathbb{R}_{\geq 0}$. We define a set function f over $[n]$ by $f(I) = w\bigl(\bigcup_{i \in I} S_i\bigr)$ for $I \subseteq [n]$; i.e., the weight of all the elements covered by some set indexed by I.

The matroid terminology above extends to set functions $f : 2^{\mathcal{N}} \to \mathbb{R}$ as follows. For a set $S \subseteq \mathcal{N}$, the span of S (with respect to f) is defined as the set of elements with marginal value 0 with respect to S:

$$\operatorname{span}(S) \overset{\text{def}}{=} \{e : f(e \,|\, S) = 0\}.$$

If f is monotone and submodular, then $\operatorname{span}(S)$ is monotonically increasing. A set S is *closed* if $S = \operatorname{span}(S)$. A *quotient of* f is defined as any set that is the complement of a closed set. Equivalently, a set $Q \subseteq \mathcal{N}$ is a quotient iff $Q = \mathcal{N} \setminus \operatorname{span}(S)$ for some set S. This generalizes quotients in matroids by taking f to be the rank function. We call $f(\mathcal{N})$ the *rank of* f.

Cut-preserving sparsification of graphs has been an important topic of study with many applications. The random sampling work of Karger culminated in the influential work of Benczúr and Karger [7]. They proved the following. Given any fixed $\epsilon \in (0,1)$, an undirected graph can be randomly sparsified to $O(n \log(n)/\epsilon^2)$ reweighted edges while approximately preserving the weight of every (2-)cut up to a $(1+\epsilon)$-factor with high probability. Since every k-cut can be expressed as the sum of the 2-cuts of each component (divided by 2), this also preserves the weight of every k-cut of G. In the language of matroid theory, then, [7] takes as input a weighted graphic matroid of rank r, and produces a randomly reweighted subset of $O\bigl(r \log(r)/\epsilon^2\bigr)$ elements that preserves the weight of every quotient of the graphic matroid up to a $(1+\epsilon)$-factor.

We show a more general result for the quotients of a monotone submodular function f. Henceforth, let f be a monotone submodular function and w a set of nonnegative weights over a ground set $\mathcal{N}$ of n elements. We assume f is normalized so that $f(\emptyset) = 0$ and all nonzero marginal values are at least 1. (The latter holds automatically for integral f.) The rank function of a matroid is one example of a function f satisfying these properties. Let $\epsilon \in (0,1)$ be a fixed input parameter. Let $r \overset{\text{def}}{=} f(\mathcal{N})$ be the maximum value of f. The main theorem is as follows.

Theorem 1.1. *Let $f : 2^{\mathcal{N}} \to \mathbb{R}$ be a monotone submodular function with $f(\emptyset) = 0$ and where all nonzero marginal values are at least 1. Let $r = f(\mathcal{N})$. Let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be a set of nonnegative weights. There exists a set of weights $\tilde{w} : \mathcal{N} \to \mathbb{R}_{\geq 0}$ such that:*
(a) The support of $\tilde{w}$ has size $O\bigl(r \log(nr)/\epsilon^2\bigr)$.
(b) The weight of every quotient of f is preserved up to a $(1+\epsilon)$-factor; i.e., $|w(Q) - \tilde{w}(Q)| \leq \epsilon w(Q)$ for all quotients Q.
These weights $\tilde{w}$ can be computed with high probability in randomized polynomial time assuming oracle access to f.

Theorem 1.1 was motivated by matroid sparsification as a natural generalization of graph sparsification. For matroids, theorem 1.1 translates to the following by taking f to be the rank function. We let Q_{RANK} denote the running time of a rank query. $\tilde{O}(\cdots)$ hides logarithmic factors.

Theorem 1.2. *For any weighted matroid with n elements and rank r, and $\epsilon \in (0,1)$, there exists a set of weights supported by $O\bigl(r \ln(n)/\epsilon^2\bigr)$ elements that preserves the weight of every quotient up to a $(1+\epsilon)$-factor. These weights can be computed with high probability in $\tilde{O}(nQ_{\text{RANK}})$ randomized time assuming oracle access to the rank function.*

One application of theorems 1.1 and 1.2 of recent interest is to hypergraph cuts. A hypergraph is a generalization of a graph where the edges (called *hyperedges*) may have any number of endpoints. A *cut* is defined as the set of edges properly crossing a set of vertices S. [12] recently gave a randomized polynomial time algorithm that, given an n-vertex hypergraph $H = (V,E)$, with high probability, computes a reweighted subgraph of H with $O\bigl(n \log(n)/\epsilon^2\bigr)$ edges that preserves the weight of every cut up to a $(1+\epsilon)$-factor. Their algorithm runs in

$\tilde{O}\big(p + n^{10}/\epsilon^7\big)$ time, where $p = \sum_{e\in E}|e|$ is the total size of the hypergraph. We note that while several follow up works have improved on [12] in various dimensions (discussed further below), none of these produce a cut sparsifier with $O\big(n\log(n)/\epsilon^2\big)$ edges in nearly linear time.

Meanwhile, given a hypergraph $H = (V, E)$, one can define a matroid over the edge set E called the *hypergraphic matroid*, where cuts in the hypergraph map to quotients in this matroid [43]. For algorithmic reasons we introduce a submodular function different from the rank function of the hypergraphic matroid, which we call the *hypergraphic polymatroid function*. This monotone submodular set function takes as input a set of hyperedges S and returns n minus the number of the connected components in the subgraph induced by S. The quotients of the hypergraphic polymatroid function consists of all k-cuts of the hypergraph (for all k). Similar to graphs, a k-cut is defined by a partition $(U_1, \ldots, U_k)$ of V, and consists of all hyperedges e properly crossing $(U_1, \ldots, U_k)$ (i.e., e is not contained in any single U_i). Thus hypergraph cuts (as first introduced above) are 2-cuts. Unlike for (normal) graphs, a hypergraph sparsifier preserving all (2-)cuts does not necessarily preserve all k-cuts.

Along these lines we re-derive the $O\big(n\log(n)/\epsilon^2\big)$-edge hypergraph cut sparsifiers of [12], and also strengthen the sparsifiers to preserve all k-cuts. We present a new algorithm that is the first to run in nearly linear time in the input size among those producing cut-preserving hypergraphs with $O\big(n\log(n)/\epsilon^2\big)$ edges.

Theorem 1.3. *For any hypergraph with n vertices, there is a reweighted sub-hypergraph with at most $O\big(n\ln(n)/\epsilon^2\big)$ edges that preserves the weight of every k-cut up to a $(1+\epsilon)$-factor (for all k). This sub-hypergraph can be computed with high probability in randomized $\tilde{O}(p)$ time, where p is the total size of the hypergraph.*

Another example of a matroid is the linear matroid. Here, the ground set consists of a set of vectors in a fixed vector space, and a set of vectors is independent iff they are linearly independent. The Gram-Schmidt process gives a polynomial time rank function for this matroid, and the rank of the matroid is at most the dimension of the vector space. Via the linear matroid, theorem 1.2 leads to the following theorem for weighted collections of vectors in a fixed vector space.

Corollary 1.1. *Let X be a finite set of m weighted vectors in a vector space of dimension n. There exists a reweighted subset of $O\big(n\log(m)/\epsilon^2\big)$ vectors that, for every subspace S, preserves the total weight of all vectors lying outside S up to a $(1+\epsilon)$-factor. This reweighted subset of vectors can be computed with high probability in randomized polynomial time.*

There are other monotone submodular functions f of interest besides matroid rank functions. One example is coverage, where the ground set is a collection of sets over a weighted universe of points, and f takes as input a collection of sets and evaluates the weight of their union. Previous work by [2] proved the following. Given a family of n over a finite universe of weighted points, one can randomly sample a reweighted subset of $O\big(n^2\big)$ points that, for every subcollection of sets, preserves the total weight of their union up to a $(1+\epsilon)$-factor. Their algorithm runs in nearly linear time. Below, the *total size* of a set system is the sum of cardinalities of all the sets in the set system, and represents the input size.

Theorem 1.4. *Given a family of n sets over a finite universe of m weighted points, there exists a reweighted subset of at most $O\big(n\log(n)/\epsilon^2\big)$ points that, for every subcollection of sets, preserves the total weight of their union up to a $(1+\epsilon)$-factor. This reweighted subset of points can be computed with high probability in randomized $O\big(N\log^2(m+n)\big)$ time, where N is the total size of the set system.*

Another example of a monotone submodular function arises in SAT. Given a boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunction normal form (CNF), let the ground set $\mathcal{N}$ consist of the literals $x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n$. We have a monotone submodular function $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ that returns the number of clauses satisfied by setting a collection of literals to be true.

Corollary 1.2. *Given a boolean formula $\varphi(x_1, \ldots, x_n)$ in CNF with n variables and m weighted clauses, one can compute a reweighted subset of at most $O\big(n\log(n)/\epsilon^2\big)$ clauses that, for every assignment of variables, preserves the total weight of the number of satisfied clauses up to a $(1+\epsilon)$-factor. This reweighted subset of clauses can be computed in randomized $O\big(N\log^2(m+n)\big)$ time, where N is the total size of the formula φ.*

Theorem 1.4 and corollary 1.2 are closely related and we present two different ways to obtain then. One way is to directly apply theorem 1.1 to a dual *hitting set function*. A second way is via a reduction from CNF-SAT sparsification to hypergraph sparsification described by [39]. Via this reduction, the bound on the number of clauses in corollary 1.2 is implied by [39, 12]; the remaining contribution is the running time. The reduction from [39] can be extended to coverage. The direct approach based on the hitting set function yields a faster and simpler algorithm, but the second approach via hypergraph sparsification is conceptually simpler if we take hypergraph sparsification for granted.

Joint entropy defines a monotone submodular function over random variables of a fixed probability space. For n random variables $X_1, \ldots, X_n$ in a fixed probability space, and a subset of random variables S, the quotient Q induced by S is the set of variables that are not determined by S. Therefore, applying theorem 1.1 to joint entropy gives the following.

Corollary 1.3. *Let $X_1, \ldots, X_n$ be random variables. Let $\alpha > 0$ be the minimum nonzero marginal value of the joint entropy function over $X_1, \ldots, X_n$, and let $\beta > 0$ be the total joint entropy of $X_1, \ldots, X_n$. Given weights over $X_1, \ldots, X_n$, there is a reweighted subset of at most $O\big(\beta \log(n\beta/\alpha)/\alpha\epsilon^2\big)$ variables that, for every subset S of $X_1, \ldots, X_n$, preserves the total weight of all variables that are not determined by S up to a $(1+\epsilon)$-factor. This reweighted subset of variables can be computed with high probability in randomized polynomial time assuming oracle access to the joint entropy of any subset of random variables.*

These applications give just a few concrete examples of theorem 1.1. Given the generality of submodular functions and matroids, and their important roles both in theory (e.g., [16, 50]) and practice (e.g., [8]), one hopes that further interesting applications will emerge.

High-level overview of techniques. The techniques in this work are most directly inspired by previous work on cut sparsifiers for undirected graphs, particularly [32, 7]. We highlight the most technically relevant aspects of these works. [32] introduced the randomized contraction algorithm for undirected minimum cut and critically observed that there is only a polynomial number of minimum cuts and near-minimum cuts. In particular, [32] implies that randomly rounding all the edge capacities uniformly relative to the weight of the minimum cut approximately preserves the minimum cut in the graph with high probability. However, this random sample is not necessarily sparse. For example, any edge with capacity larger than the minimum cut is automatically retained. The uniform sample is not sensitive to the fact that edges that are only contained in large cuts can be randomly sampled with greater variance.

[7] gave a hierarchical decomposition of a weighted graph into a sequence of induced subgraphs (called "strong components") with the following structural property: if one randomly rounds each edge capacity in proportion to the minimum cut of the last subgraph/strong component containing that edge, then all cuts in the input graph are approximately preserved with high probability. This decomposition effectively identifies which edges are contained in small cuts, and have to be sampled conservatively, and which edges are only contained in stronger cuts, which can tolerate the greater variance of a sparser sample. With this decomposition guiding the sampling, [7] can preserve all cuts up to a $(1+\epsilon)$-factor with high probability with a random sample of $O\big(n \log(n)/\epsilon^2\big)$ reweighted edges.

Moreover, [7] showed how to approximate the strong component decomposition in nearly linear time. Their algorithm is based on an algorithm of [44] that iteratively removes spanning forests from the graph. This process reflects the cut structure of the graph; for example, if two vertices s, t are still connected after removing (say) x forests, then the minimum (s,t)-cut is at least x. [7] shows that a careful application of this fast iterative process leads to a good enough approximation of the strong component decomposition.

As mentioned above, graph cuts are quotients of the rank function of the graphic matroid. We would like to generalize the high-level approach of [32, 7] to quotients of matroids and more generally of monotone submodular functions. A first step is provided by [34] for unweighted matroids. For a quotient $Q = \mathcal{N} \setminus S$, let us define the *ratio* of Q as the quantity $|Q|/(\text{rank}(\mathcal{N}) - \text{rank}(S))$. Analogous to [32] for graph cuts, [34] gave a randomized contraction argument to bound the number of near-minimum ratio quotients. Similar to minimum cuts, this shows that random sampling relative to the weight of the minimum quotient ratio approximately preserves the size of all quotients (up to scaling). This leads to some speedups for matroid packing problems in [34], but does not imply a sparse set of (reweighted) elements that preserves all quotients of a weighted matroid.

In this work, we generalize the contraction argument further to the quotients of a monotone submodular function, and then take the next step, analogous to [7], of hierarchically decomposing the elements to produce nonuniform sampling probabilities that lead to quotient-preserving sparsifiers. While we are guided by [32, 7] at a high-level, several components did not transfer at a more technical level to matroids and submodular functions, and here we try to motivate and explain the differences.

The decomposition of [7] classifies the edges $\{u, v\}$ of an undirected graph based on the weight of the minimum (u, v)-cut. For matroids and submodular function, the analogous approach would classify each element e based on the weight of the minimum quotient Q containing e. We failed to make this approach work; for starters, the sampling techniques in [34] are based on the ratio of Q, and not just the weight / cardinality of Q.

We describe our alternative approach by first describing it in undirected graphs as a concrete example that a broader audience can relate to. We first recall the Tutte–Nash-Williams tree packing theorem. For a k-cut $C = \delta(S_1, \ldots, S_k)$ with total edge weight $W = w(C)$, let us define the "ratio" of the k-cut as the quantity $W/(k-1)$. One can interpret the ratio of a k-cut as the average capacity removed per additional component generated by the k-cut. The Tutte–Nash-Williams theorem states that the maximum number of fractional spanning trees that can be packed in a graph is equal to the minimum ratio over all k-cuts in the graph (over all k). This quantity is called the *network strength* of the graph.

To be clear, the notion of "strength" in [7], which is based on edge connectivity, is different from network strength. That said, it happens that the network strength is always within a factor of 2 of the minimum cut in the graph. (This connection is exploited, for example, in the nearly linear time minimum cut algorithm of [33].) One can now try to (liberally) reinterpret [32, 7] in terms of the Tutte–Nash-Williams theorem [55, 45]. [32] implies that randomly rounding edge weights in proportion to the network strength preserves the ratio (hence the weight) of every k-cut. The decomposition in [7] can be interpreted as loosely approximating a different decomposition based on recursively removing the minimum ratio k-cut. The iterative process of removing spanning trees can be interpreted as a greedy tree packing algorithm that, in the dual, is also a greedy (inexact) algorithm for the minimum ratio k-cut.

For us, the advantage of the Tutte–Nash-Williams perspective is that it classically extends to matroids. Let us extend the notion of quotient ratios to weighted matroids in the natural way by replacing the cardinality of the quotient in the numerator with the weight of the quotient. Edmonds [14] showed that the maximum quantity of fractional bases that can be packed in a weighted/capacitated matroid is equal to the minimum quotient-ratio over all quotients Q. We call this quantity the *strength* of the matroid. We can now try to generalize the Tutte–Nash-Williams interpretation of the [7] algorithm to matroids as follows. First, decompose the ground set by repeatedly computing and removing minimum ratio quotients. This gives a hierarchical decomposition of the elements of the matroid. Then, randomly round each element based on the ratio of the quotient that removed the element from the matroid. One can now prove that randomly rounding based on this decomposition preserves all quotients of the matroid. This approach and analysis for matroids generalizes (with technical care) to quotients of monotone submodular functions. (The generalization from matroids to monotone submodular functions is perhaps not surprising given, for example, the classical theory of polymatroids).

The decomposition based on minimum ratio quotients can be computed in polynomial time via submodular minimization. However, we are also interested in fast running times akin to [7] for matroids and concrete special cases such as hypergraphs. The analog to [7]'s use of [44] would be to iteratively remove bases from the matroid. Unfortunately, it seems that the [7] argument leverages graph structure in a non-generalizable way, as we could not make this algorithm work for matroids. Instead, we develop new algorithms that simultaneously compute approximately maximum base packings and (in the dual) approximately minimum ratio quotients. These algorithms are based on the push-relabel framework proposed in [17] and further developed more recently in [46]. Let us call this the "matroid push-relabel" framework to distinguish it from push-relabel for flow. Similar to push-relabel for flow, matroid push-relabel assigns labels to the elements, and uses these labels to guide exchanges that gradually transform an arbitrary collection of bases into a feasible packing (respecting the capacities). Similar to the labels inducing a min cut in flow, the labels in the matroid push-relabel algorithm induce quotients that produce dual certificates of optimality. As observed in [46], this framework is also conducive to fast approximation algorithms.

Fast approximations for base packing and minimum ratio quotients are insufficient for a nearly linear time hierarchical decomposition of the entire matroid. In the worst case, each minimum ratio quotient only reduces the rank by 1, and overall we make r passes over the input where r is the rank of the matroid. To remove this factor of r, we leverage a feature of the matroid push-relabel framework analogous to the use of max-flow push-relabel for parametric flow (e.g., [20]). We first describe the context. At a generic point in the decomposition process, we have fixed parameters $k \in \mathbb{N}$ and $\epsilon \in (0, 1)$, and we want to either pack $(1-\epsilon)k$ bases or find a quotient with ratio at most $(1+\epsilon)k$. The decomposition algorithm repeatedly extracts such quotients until it certifies that the matroid strength is at least $(1-\epsilon)k$ by computing such a packing; then it increases k by a constant factor.

Naively, one reruns the static algorithm from scratch for each quotient. With the push-relabel framework one can instead do the following. For a fixed value of k, after running the push-relabel algorithm once to extract a quotient with ratio at most $(1+\epsilon)k$, one can retain the labels and collection of bases for the remaining elements of the matroid, and continue the matroid push-relabel algorithm from this configuration. Continuing in this fashion, we can repeatedly extract approximately minimum ratio quotients until arriving at an approximate base packing, allowing us to increase k, all in the running time of a single instance of push-relabel. Conceptually, this removes the overhead of r and brings the overall running time (or query complexity) down to nearly linear. We note that to be able to reuse the push-relabel configurations as described above, we need the approximately minimum ratio quotients to correspond to upper level sets of elements with respect to the labels. This leads to an algorithm with invariants that appear "backwards" compared to [17, 46].

There are additional technical hurdles that had to be overcome to transfer the [7] approach to matroids. For example, existing matroid push-relabel approximation algorithms only have truly fast running times for integer weights and small values of k. To adapt this to weighted matroids, we use uniform sparsification to discretize the weights and reduce k to a logarithmic factor of the input size. Here we can apply uniform sparsification because the randomized rounding is only based on the current estimate of the matroid strength k. The uniform sparsification guarantees had to be strengthened beyond preserving quotients to ensure that the output from running the push-relabel algorithm on the sampled weights is still useful for decomposing he matroid with respect to the input weights.

To bring everything back to graphs, the more general algorithm for matroids gives a different algorithm from [7] for sparsifying undirected graphs. Instead of decomposing the graph along greedy tree/forest packings á la [44], we repeatedly compute approximately maximum tree packings and approximately minimum ratio k-cuts via push-relabel. This gives a decomposition in nearly linear time that is directly based on network strength instead of edge connectivity. Randomly sampling based on this decomposition gives a cut-preserving sparsifier with the same number of edges as [7]. Of course the algorithmic end result for graphs is not new, but we are also interested in the explicit conceptual connection between [7] and the Tutte–Nash-Williams theorem and, more broadly, to classical matroid theory.

Related work. Randomized graph sparsification has had a profound impact on randomized algorithm design. Besides its direct applications to graph problems, the general idea of applying random sampling to reduce the input size while approximately preserving key complex structures has been adapted to many other domains. Due to space constraints we limit our literature review to the most directly relevant developments.

Graph sparsification. The well-known randomized contraction algorithm of [32] showed that there is only a polynomial number of minimum cuts, and more generally, $n^{O(\alpha)}$ α-approximate minimum cuts for $\alpha \geq 1$. This allows one to apply random sampling to reduce the graph to $O\big(n\log(n)/\epsilon^2\big)$ edges while preserving the value of the minimum cut up to a $(1+\epsilon)$-factor. This more limited notion of sparsification sufficed to obtain faster algorithms for several flow and cut problems [37, 36, 6, 35, 33, 38]. In 2002, [7] obtained the first $(1+\epsilon)$-cut sparsifiers (preserving all cuts up to a $(1+\epsilon)$-factor) with $O\big(n\log(n)/\epsilon^2\big)$ edges; their algorithm runs in nearly linear time. [19] developed variations of the [7] approach that also sparsify graphs while approximately preserving all cuts. [32, 7] are important inspirations for this paper. Some parts of section 3 generalize ideas in [32, 7].

Cut sparsifiers have been generalized to *spectral sparsifiers* that more generally preserve the Laplacian of a graph [53, 54, 52, 5]. [5] showed that there are $(1+\epsilon)$-spectral sparsifiers with $O\big(n/\epsilon^2\big)$ edges; moreover they can be computed in nearly linear time [42].

Hypergraph sparsification. There has been a wave of recent interest in sparsification for hypergraphs [39, 11, 12, 51, 4, 30, 31, 41, 29]. [12] was the first to obtain a $(1+\epsilon)$-cut sparsifier with $O\big(n\ln(n)/\epsilon^2\big)$ edges, where n is the number of vertices in the graph. Their algorithm was also inspired by [7] and runs in $\tilde{O}\big(p+n^{10}/\epsilon^7\big)$ time, where p is the total input size. Several of the papers above develop sparsifiers that preserve spectral-inspired properties of hypergraphs that generalize hypergraph (2-)cuts. Very recently, [29] obtained a nearly linear time algorithm producing spectral hypergraph sparsifiers with $O\big(n\log(n)\log(\ell)/\epsilon^2\big)$ edges, where ℓ is the maximum number of vertices in any hyperedge. (Note the extra factor of $\log(\ell)$.) Independent follow up work [28] generalizes the spectral direction further to sums of semi-norms.

Matroid and submodular sparsification. Towards faster algorithms for packing disjoint bases in a matroid, [34] developed techniques for quotients of matroids analogous to the techniques developed in [32, 37] for cuts. In particular, [34] obtained polynomial bounds on the number of quotients with minimum or approximately minimum quotient-ratio (defined in section 2), and showed how to sparsify the ground set while preserving the strength of the matroid up to a $(1+\epsilon)$-factor. This lead to faster running times for packing bases. [34] is also an important inspiration for this paper. Theorem 1.2 is to [34] as the results of [7] are to [32]. Some parts of section 3 generalize ideas from [34].

An alternative approach to submodular sparsification is explored in [48], which sparsifies sums of submodular functions. [48] shows, among other results, that (specifically) for the problem of maximizing the union of k sets (out of m total sets), one can reduce the point set to $O\left(mk/\epsilon^2\right)$ while losing only a $(1+\epsilon)$-factor in the approximation guarantee. For sums of symmetric submodular functions over n elements, the very recent work [28] mentioned above produces sparsified sums of $O\left(n\log^{2.5}(n)\log(n/\epsilon)/\epsilon^2\right)$ functions.

Hypergraph k-cut. As mentioned above, our hypergraph sparsifier has the interesting property of approximately preserving all k-cuts in addition to 2-cuts. There are several results, including many recent developments, on minimum k-cut in hypergraphs and related problems [40, 56, 18, 15, 9, 10, 3]. Our k-cut preserving sparsifier can accelerate a minimum k-cut algorithm by reducing the number of edges as a preprocessing step in exchange for a $(1+\epsilon)$-approximation factor.

Organization. Section 2 contains preliminaries including notation and definitions. Section 3 proves the main result, theorem 1.1. Section 4 proves the sparsification result for matroids. Section 5 proves the sparsification result for hypergraphs, theorem 1.3. Section 6 proves the sparsification results for set systems and for CNF-SAT, theorem 1.4 and corollary 1.2. (We do not expand on the other results as they are immediate.) To help separate the structural aspects (i.e., the existence of sparse quotient sparsifiers) from the algorithmic (with emphasis on fast running times), we have placed the most involved algorithmic discussions in the appendix. Appendices A–C describe essential subroutines for decomposing matroids, hypergraphs, and set systems, respectively. In sections 4–6, we only describe the sparsification algorithms for polynomially bounded weights. Appendix D explains how to reduce from general weights to polynomially bounded weights in a black box manner.

Acknowledgements. We thank Chandra Chekuri for his feedback and encouragement. We thank Yuichi Yoshida for highlighting that the hypergraph sparsifier preserves all k-cuts (and not just 2-cuts), and pointing out the reduction from coverage sparsification to hypergraph sparsification. We thank the reviewers for their valuable feedback.

2 Preliminaries

Henceforth, $f : 2^{\mathcal{N}} \to \mathbb{R}_{\geq 0}$ denotes a fixed set function satisfying the assumptions of theorem 1.1. $w \in \mathbb{R}_{\geq 0}^{\mathcal{N}}$ is always a fixed set of nonnegative weights over $\mathcal{N}$. We always have $r = f(\mathcal{N})$.

Notation. For an element $e \in \mathcal{N}$, we let e also denote the singleton set $\{e\}$. We write $S + e$ and $S - e$ to abbreviate $S \cup \{e\}$ and $S \setminus \{e\}$, respectively. $\bar{S}$ denotes the complement of the set S. For a probabilistic event E, $\bar{E}$ denotes the complementary event. For a vector $x : \mathcal{N} \to \mathbb{R}$, and a set S, we write $x(S)$ to denote the sum $\sum_{e\in S} x(e)$. $\tilde{O}(\cdots)$ hides logarithmic factors. A nearly linear running time refers to a running time of the form $\tilde{O}(N) = O\left(N\log^{O(1)} N\right)$ where N is the input size of the problem.

Definitions. The following quantities play a central role in the analysis. For a set S, the *strength-ratio* of S, denoted $\varphi(S)$, is the value

$$\varphi(S) \overset{\text{def}}{=} \frac{w(\mathcal{N}) - w(S)}{f(\mathcal{N}) - f(S)},$$

with the convention that $0/0 = +\infty$. For a quotient $Q = \bar{S}$, the *quotient-ratio* of Q, denoted $\psi(Q)$, is the value

$$\psi(Q) \overset{\text{def}}{=} \frac{w(Q)}{f(\mathcal{N}) - f(S)} = \frac{w(\mathcal{N}) - w(S)}{f(\mathcal{N}) - f(S)} = \varphi(S).$$

The *strength* of f, denoted κ_f, is defined as the minimum strength-ratio over all sets S:

$$\kappa_f \overset{\text{def}}{=} \min_{S\subseteq\mathcal{N}} \varphi(S).$$

Since $\varphi(S) \geq \varphi(\operatorname{span}(S)) = \psi(\mathcal{N} \setminus \operatorname{span}(S))$ for all S, κ_f is also the minimum $\psi(Q)$ over all quotients Q of f. The strength of a matroid is defined as the strength with respect to its rank function.

For a set $S \subseteq \mathcal{N}$, the restriction of f to S is another set function satisfying all the properties assumed of f. We refer to "quotients in S", the "span in S", and so forth, as the corresponding objects for the restriction of f to S. For a subset T of S, we let

$$\varphi(T \mid S) \overset{\text{def}}{=} \frac{w(S) - w(T)}{f(S) - f(T)}$$

denote the strength-ratio of T in S. For a quotient Q in S, we let

$$\psi(Q \mid S) \overset{\text{def}}{=} \frac{w(Q)}{f(S) - f(S \setminus Q)}$$

denote the quotient-ratio of Q as a quotient in S. The *strength of* S is the strength of the restriction of f to S, and denoted κ_S.

For a set $S \subseteq \mathcal{N}$, the *contraction* of S is the function f_S defined by

$$f_S(T) \overset{\text{def}}{=} f(T \mid S) = f(S \cup T) - f(S).$$

If f is monotone, submodular, normalized, and has marginal values at least 1, then so does f_S. We let span_S denote the span function with respect to f_S. For a matroid $\mathcal{M}$ over $\mathcal{N}$, and $S \subseteq \mathcal{N}$, the *matroid contracting* S, denoted $\mathcal{M}/S$, is the matroid over $\mathcal{N} \setminus S$ with rank function $\operatorname{rank}_S(T) = \operatorname{rank}(T \mid S)$ for $T \subseteq \mathcal{N} \setminus S$.

For additional background on submodular functions and matroids, see [50, 16].

Restricting quotients. The following lemma shows that restricting the ground set preserves quotients.

LEMMA 2.1. *Let Q be a quotient, and let $S \subseteq \mathcal{N}$ be a set. Then $Q \cap S$ is a quotient in S.*

Proof. Let $A = S \setminus (Q \cap S) = S \cap \bar{Q}$; we need to show that A is closed in S. We have

$$\operatorname{span}(A) \overset{\text{(a)}}{\subseteq} \operatorname{span}(\bar{Q}) \overset{\text{(b)}}{=} \bar{Q}$$

where (a) is by monotonicity of $\operatorname{span}(\cdots)$ and (b) is because $\bar{Q}$ is closed. Therefore

$$\operatorname{span}(A) \cap S \subseteq \bar{Q} \cap S = A,$$

as desired. ∎

Elements with weight 0. We generally allow the input to contain elements with weight 0. When it comes to preserving the weight of quotients, these elements of course do not contribute any weight, and the sampling algorithms described later never assign positive weight to any such element. That said, we cannot necessarily dismiss these zero weight elements as they still play a role in defining the family of all quotients that have to be preserved. The following lemma addresses this latter concern and shows that one can drop all weight 0 elements as a preprocessing step to sparsification.

LEMMA 2.2. *Let $\epsilon > 0$ and let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$. Let $\mathcal{N}_1 \subseteq \mathcal{N}$ be the support of w. Let $\tilde{w}_1 : \mathcal{N}_1 \to \mathbb{R}_{\geq 0}$ preserve the w-weight of every quotient in $\mathcal{N}_1$ up to a $(1+\epsilon)$-factor. Let $\tilde{w}$ extend $\tilde{w}_1$ to $\mathcal{N}$ be setting $\tilde{w}(e) = w(e) = 0$ for all $e \in \mathcal{N} \setminus \mathcal{N}_1$. Then $\tilde{w}$ preserves the w-weight of all quotients up to a $(1+\epsilon)$-factor.*

Proof. Let Q be a quotient of f. By lemma 2.1, $Q \cap \mathcal{N}_1$ is a quotient in $\mathcal{N}_1$, so $\tilde{w}_1(Q \cap \mathcal{N}_B)$ is within a $(1+\epsilon)$-factor of $w(Q \cap \mathcal{N}_B)$. We also have $\tilde{w}(Q) = \tilde{w}_1(Q \cap \mathcal{N}_B)$ and $w(Q) = w(Q \cap \mathcal{N}_B)$, hence $\tilde{w}(Q)$ is within a $(1+\epsilon)$-factor of $w(Q)$. ∎

3 Quotient sparsification for submodular f: proof of theorem 1.1

Given f and w, the goal is to compute a set of nonnegative weights $w' \in \mathcal{N}^{\mathbb{R}}_{\geq 0}$ with $O\big(n \ln(nr)/\epsilon^2\big)$ nonzero entries and that preserves the weight of every quotient of f up to a $(1+\epsilon)$-factor. To describe the algorithm we need to introduce the following.

Strength decompositions. For $\alpha \geq 1$, an *α-approximate strength decomposition* is a descending sequence of sets $S_0 \supseteq S_1 \supseteq \cdots \supseteq S_k$ such that:

(a) $S_0 = \mathcal{N}$.
(b) $S_k = \emptyset$.
(c) For $i \in [k]$, $\varphi(S_i \,|\, S_{i-1}) \leq \alpha\kappa_{S_{i-1}}$.
(d) The strength ratios $\varphi(S_i \,|\, S_{i-1})$ are nondecreasing in i.

An *exact strength-decomposition* is a 1-approximate strength decomposition.

Given a sequence satisfying the first three properties (a)–(c), it is not difficult to obtain a subsequence satisfying all four properties (a)–(d). Let $S_0 \supseteq \cdots \supseteq S_k$ satisfy the first three properties of a strength decomposition. We want a subsequence that satisfies the fourth property as well. To this end, consider the following process: while there is an index $i \in \{1, \ldots, k-1\}$ such that $\varphi(S_i \,|\, S_{i-1}) > \varphi(S_{i+1} \,|\, S_i)$, remove S_i from the sequence. Clearly this preserves properties (a) and (b) and terminates with property (d) satisfied as well. We need to verify that property (c) is preserved. The following lemma implies that when we remove a set S_{i-1}, we have $\varphi(S_{i+1} \,|\, S_{i-1}) \leq \varphi(S_i \,|\, S_{i-1}) \leq \alpha\kappa_{S_{i-1}}$, as desired.

Lemma 3.1. *Let $S_0, S_1, \ldots, S_k \subseteq \mathcal{N}$ with $S_i \subseteq S_{i-1}$ for all i. Then*

$$\varphi(S_k \,|\, S_0) \leq \max_{i \in [k]} \varphi(S_i \,|\, S_{i-1}).$$

Proof. We have

$$\begin{aligned} w(S_0) - w(S_k) &= \sum_{i=1}^{k} w(S_{i-1}) - w(S_i) = \sum_{i=1}^{k} \varphi(S_i \,|\, S_{i-1})(f(S_{i-1}) - f(S_i)) \\ &\leq \max_i \varphi(S_i \,|\, S_{i-1})(f(S_0) - f(S_k)) \end{aligned}$$

by telescoping series, as desired. ∎

The algorithm. At a high-level, the quotient sparsification algorithm computes an approximate strength decomposition $S_0, S_1, \ldots, S_k$ and then non-uniformly samples the weights based on the decomposition. An exact strength decomposition can be computed in polynomial time as computing each S_i reduces to submodular minimization [24, 27, 49] via the Newton method for fractional combinatorial optimization [13] (cf. [47]). There are faster and more direct algorithms for computing approximate strength decompositions in more concrete settings, some of which are presented later.

We associate each element e with the strength-ratio $\varphi(S_i \,|\, S_{i-1})$ of the unique set S_i such that $e \in S_{i-1} \setminus S_i$. The algorithm randomly rounds each $w(e)$ non-uniformly where elements e are prioritized by their strength-ratios. For example, in the special case of uniform weights, the algorithm randomly samples each element $e \in S_{i-1} \setminus S_i$ with probability inversely proportional to $\varphi(S_i \,|\, S_{i-1})$ and (when e is sampled) assigns weight proportional to $\varphi(S_i \,|\, S_{i-1})$. Intuitively, smaller values of $\varphi(S_i \,|\, S_{i-1})$ implies that S_{i-1} has low strength and that quotients of S_{i-1} are more sensitive to fluctuations in weight. This suggests that the sampling should prioritize (i.e., be more conservative for) these elements. Larger values of $\varphi(S_i \,|\, S_{i-1})$ reflect a stronger set S_{i-1} that can tolerate sparser samples and higher variance.

Pseudocode for the algorithm, called `quotient-sparsification`, is presented in fig. 1 on the following page. For the rest of this section we let $\tilde{w}$ denote the randomized weights returned by `quotient-sparsification`. Observe that $\tilde{w}$ satisfies $\mathbf{E}[\tilde{w}(S)] = w(S)$ for all sets S. We want to show that $\tilde{w}(Q)$ is concentrated at $w(Q)$ for all quotients Q.

Counting quotients. The first step of the analysis observes that there are only polynomially many quotients with weight close to κ_f. Recall that $r = f(\mathcal{N})$, and we assume that all nonzero marginal values of f are at least 1.

Lemma 3.2. *For each integer $t \in \mathbb{N}$, there are at most $n^{t+1} r^t$ quotients of f with weight at most $t\kappa_f$.*

Proof.[1] Let $\ell = \lfloor r \rfloor - t - 1$. Consider the following randomized process that returns a family of quotients of f. Initially, let $R = \emptyset$. While $f(R) < \ell$, we add to R an element $e \in \mathcal{N} \setminus \operatorname{span}(R)$ randomly sampled in proportion to

[1]The special case where the elements are unweighted and f is the rank function of a matroid was proven in [34]. This proof extends the ideas to weighted elements and submodular functions.

```
quotient-sparsification($f$,$w$,$\epsilon$)
1. Compute an $\alpha$-approximate strength decomposition $(S_0 = \mathcal{N}) \supseteq S_1 \supseteq \cdots \supseteq (S_k = \emptyset)$ for a parameter
   $\alpha = O(1)$.
2. For each element $e$:
    A. Let $\tau_e = \frac{c\epsilon^2 \varphi(S_i \mid S_{i-1})}{\alpha \ln(nr)}$, where $e \in S_{i-1} \setminus S_i$, for a sufficiently small constant $c > 0$.
    B. Randomly (and independently) set
       $$\tilde{w}(e) = \begin{cases} \lceil w(e)/\tau_e \rceil \tau_e & \text{with probability } p_e \overset{\text{def}}{=} w(e)/\tau_e - \lfloor w(e)/\tau_e \rfloor, \\ \lfloor w(e)/\tau_e \rfloor \tau_e & \text{with probability } 1 - p_e. \end{cases}$$
3. Return $\tilde{w}$.
```

Figure 1: Computing a sparse set of weights that preserves the weight of all quotients of f.

its weight $w(e)$. This produces a randomized set R with $f(R) \geq \ell$. The random process then outputs the set of all quotients disjoint from R with total weight at most $t\kappa_f$.

To bound the number of quotients that are output, consider any quotient $Q = \bar{S}$ disjoint from R. Then $\operatorname{span}(R) \subseteq \operatorname{span}(S) = S$ because $\operatorname{span}(\cdots)$ is monotonically increasing, $R \subseteq S$, and S is closed. One can repeatedly add elements from $S \setminus \operatorname{span}(R)$ to R as long as $\operatorname{span}(R) \subsetneq S$. Each element increases $f(R)$ by at least 1, and $f(R)$ is bounded above by r, so at most $t+1$ elements are added before $f(R) = f(S)$, $\operatorname{span}(R) = \operatorname{span}(S)$ and $Q = \mathcal{N} \setminus \operatorname{span}(R)$. Conversely, a set S of at most $t+1$ elements induces a quotient $Q = \mathcal{N} \setminus \operatorname{span}(R \cup S)$ disjoint from R. This shows that there at most

$$\sum_{i=0}^{t+1} \binom{n}{i} \leq n^{t+1}$$

quotients disjoint from R.

We claim that any quotient Q with weight at most $t\kappa_f$ is output with probability at least $1/\binom{\lfloor r \rfloor}{t}$. If so, then since the sum of output probabilities over all quotients Q is at most n^{t+1}, there are at most

$$\binom{\lfloor r \rfloor}{t} n^{t+1} \leq n^{t+1} r^t$$

quotients of weight at most $t\kappa_f$, as desired.

To prove the claim, fix a quotient Q with weight at most $t\kappa_f$. It suffices to show that the iteratively sampled set R is disjoint from Q with probability at least $1/\binom{\lfloor r \rfloor}{t}$.

Recall that the process generating R terminates as soon as $f(R) \geq \ell$, which occurs within ℓ iterations because each additional element increase $f(R)$ by at least 1. We reframe this as an ℓ-iteration process in which the iterations where $f(R) \geq \ell$ simply hold R constant. For $i \in \{0, \dots, \ell\}$, let $R^{(\ell)}$ be the value of R after i iterations.

For $i \in [\ell]$, E_i be the event that the ith iteration does not sample an element from Q (including the scenario where there is no sample because $f(R^{(i-1)}) \geq \ell$). Let $X_i \in \{0,1\}$ be the indicator variable for E_i. Consider $\mathbf{E}[X_i \mid R^{(i-1)}] = \mathbf{P}[E_i \mid R^{(i-1)}]$ with $R^{(i-1)}$ fixed. If $f(R^{(i-1)}) \geq \ell$, then we do not sample any element, so

$$\mathbf{E}[X_i \mid f(R^{(i-1)})] = 1.$$

If $f(R^{(i-1)}) < \ell$, then the probability of sampling from Q is bounded above by

$$\begin{aligned} \frac{w(Q)}{w(\mathcal{N}) - w(\operatorname{span}(R^{(i-1)}))} &\leq \frac{t\kappa_f}{w(\mathcal{N}) - w(\operatorname{span}(R^{(i-1)}))} \overset{\text{(a)}}{\leq} \frac{t\kappa_f}{\kappa_f(r - f(R^{(i-1)}))} \\ &= \frac{t}{r - f(R^{(i-1)})} \leq \frac{t}{\lfloor r \rfloor - \lceil f(R^{(i-1)}) \rceil}. \end{aligned}$$

Here (a) is because f has strength κ_f, hence $\varphi(\mathrm{span}(R^{(i-1)})) \geq \kappa_f$. Thus, when $f(R^{(i-1)}) < \ell$, we have

$$\mathbf{E}[X_i \,|\, f(R^{(i-1)})] \geq 1 - \frac{t}{\lfloor r \rfloor - \lceil f(R^{(i-1)}) \rceil} = \frac{\lfloor r \rfloor - t - \lceil f(R^{(i-1)}) \rceil}{\lfloor r \rfloor - \lceil f(R^{(i-1)}) \rceil}. \tag{3.1}$$

We claim that for all $i \in \{1, \ldots, \ell\}$, and all fixed $R^{(i-1)}$,

$$\mathbf{E}\left[\prod_{j=i}^{\ell} X_j \,\middle|\, R^{(i-1)}\right] \geq \begin{cases} 1 & \text{if } f(R^{(i-1)}) > \ell \\ \binom{\lfloor r \rfloor - \lceil f(R^{(i-1)}) \rceil}{t}^{-1} & \text{otherwise.} \end{cases} \tag{3.2}$$

If the claim holds, then we have

$$\mathbf{P}[R^{(\ell)} \cap Q = \emptyset] = \mathbf{E}\left[\prod_{i=1}^{\ell} X_i\right] \geq \binom{\lfloor r \rfloor}{t}^{-1}$$

because $R^{(0)} = \emptyset$, as desired.

We prove the claim by induction (on $\ell - i$). The base case is where $i = \ell$. If $f(R^{(\ell-1)}) \geq \ell$, then

$$\mathbf{E}[X_\ell \,|\, R^{(\ell-1)}] = 1,$$

as desired. Otherwise, we have

$$\mathbf{E}[X_i \,|\, f(R^{(\ell-1)})] \overset{\text{(b)}}{\geq} \frac{\lfloor r \rfloor - \lceil f(R^{(\ell-1)}) \rceil - t}{\lfloor r \rfloor - \lceil f(R^{(\ell-1)}) \rceil} \geq \prod_{j=\lceil f(R^{(\ell-1)}) \rceil}^{\ell} \frac{\lfloor r \rfloor - t - j}{\lfloor r \rfloor - j} = \binom{\lfloor r \rfloor - \lceil f(R^{(\ell-1)}) \rceil}{t}^{-1},$$

as desired, where (b) is by eq. (3.1).

Now suppose $1 \leq i < \ell$. If $f(R^{(i-1)}) \geq \ell$ then we have

$$\mathbf{E}[X_i \,|\, R^{(i-1)}] = 1,$$

as desired. Otherwise, $f(R^{(i-1)}) < \ell$. By induction (on $\ell - i$), we have

$$\mathbf{E}\left[\prod_{j=i}^{\ell} X_j \,\middle|\, R^{(i-1)}\right] = \mathbf{E}\left[X_i \,\mathbf{E}\left[\prod_{j=i+1}^{\ell} X_j \,\middle|\, R^{(i)}\right] \,\middle|\, R^{(i-1)}\right] \geq \mathbf{E}[X_i Y \,|\, R^{(i-1)}],$$

where

$$Y \overset{\text{def}}{=} \begin{cases} 1 & \text{if } f(R^{(i)}) > \ell, \\ \binom{\lfloor r \rfloor - \lceil f(R^{(i)}) \rceil}{t}^{-1} & \text{otherwise.} \end{cases}$$

Since $\lceil f(R^{(i)}) \rceil \geq \lceil f(R^{(i-1)}) + 1 \rceil = \lceil f(R^{(i-1)}) \rceil + 1$, we have

$$\mathbf{E}[X_i Y \,|\, R^{(i-1)}] \geq \mathbf{E}[X_i Z \,|\, R^{(i-1)}] = \mathbf{E}[X_i \,|\, R^{(i-1)}] Z$$

where

$$Z \overset{\text{def}}{=} \begin{cases} 1 & \text{if } f(R^{(i-1)}) > \ell - 1, \\ \binom{\lfloor r \rfloor - (\lceil f(R^{(i-1)}) \rceil + 1)}{t}^{-1} & \text{otherwise.} \end{cases}$$

Finally, by eq. (3.1), we have

$$\mathbf{E}[X_i \,|\, R^{(i-1)}] Z \geq \left(\frac{\lfloor r \rfloor - t - \lceil f(R^{(i-1)}) \rceil}{\lfloor r \rfloor - \lceil f(R^{(i-1)}) \rceil}\right) Z \geq \binom{\lfloor r \rfloor - \lceil R^{(i-1)} \rceil}{t}^{-1}.$$

This establishes eq. (3.2) for all $i \in [\ell]$ and all $R^{(i-1)}$ and completes the proof. ∎

Uniform sampling. Let $\tau_0 \overset{\text{def}}{=} c_0\epsilon^2\kappa_f/\log n$ where $c_0 > 0$ is a sufficiently small constant. We first consider a special case of the quotient sparsification algorithm where $\tau_e \leq \tau_0$ for all e. This would be the case if $k = 1$. In this setting, lemma 3.2 allows us to prove that all quotients are preserved (in the uniform setting) via a straightforward union bound.

LEMMA 3.3. *If $\tau_e \leq \tau_0$ for all $e \in \mathcal{N}$, then with high probability, we have*

$$|\tilde{w}(Q) - w(Q)| \leq \epsilon w(Q) \tag{3.3}$$

for all quotients Q.

Proof.[2] Call a quotient Q *bad* if $\tilde{w}(Q)$ fails (3.3).

For $t \in \mathbb{N}$, let $\mathcal{Q}_t$ be the family of quotients of f with total weight between $[(t-1)\kappa, t\kappa)$. Fix $t \in \mathbb{N}$. By lemma 3.2, we have $|\mathcal{Q}_t| \leq (nr)^{t+1}$. For each quotient $Q \in \mathcal{Q}_t$, by the multiplicative Chernoff bound (scaled by τ_0),

$$\mathbf{P}[Q \text{ is bad}] \leq 2e^{-\epsilon^2 w(Q)/3\tau_0} \leq 2e^{-c_1 t \ln(nr)} = 2(nr)^{-c_1 t}$$

for a sufficiently large constant c_1.[3] Taking the union bound over $\mathcal{Q}_t$, we have

$$\mathbf{P}[\text{some quotient in } \mathcal{Q}_t \text{ is bad}] \leq 2(nr)^{-c_1 t}(nr)^{t+1} \leq 2(nr)^{-c_2 t}$$

for a sufficient large constant c_2. Finally, taking the union bound over all $t \in \mathbb{N}$,

$$\mathbf{P}[\text{some quotient is bad}] \leq \sum_{t\in\mathbb{N}} 2(nr)^{-c_2 t} \leq (nr)^{-c_3}$$

for a sufficiently large constant c_3, as desired. ∎

Nonuniform sampling for the general case. We now consider the general case of the quotient sparsification algorithm where $k > 1$. The high-level idea of the analysis is to apply lemma 3.3 to each S_i in the strength decomposition with the weights appropriately rescaled.

LEMMA 3.4. *With high probability, we have $|\tilde{w}(Q) - w(Q)| \leq \epsilon w(Q)$ for all quotients Q.*

Proof. Let $w_\varphi, \tilde{w}_\varphi : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be the weights defined by

$$w_\varphi(e) = \frac{w(e)}{\varphi(S_i \mid S_{i-1})} \text{ and } \tilde{w}_\varphi(e) = \frac{\tilde{w}(e)}{\varphi(S_i \mid S_{i-1})} \text{ for } e \in S_{i-1} \setminus S_i.$$

We claim that with high probability,

$$|\tilde{w}_\varphi(Q \cap S_i) - w_\varphi(Q \cap S_i)| \leq \epsilon w_\varphi(Q \cap S_i) \tag{3.4}$$

for all quotients Q of f and all sets S_i in the strength decomposition. We prove the claim momentarily. First we show how the claim implies that $|\tilde{w}(Q) - w(Q)| \leq \epsilon w(Q)$ for all quotients Q.

Fix a quotient Q and suppose (3.4) holds for all S_i. For each element $e \in Q$, if $e \in S_{i-1} \setminus S_i$, we have

$$w(e) = \varphi(S_i \mid S_{i-1})w_\varphi(e) = \varphi(S_1)w_\varphi(e) + \sum_{j=2}^{i}(\varphi(S_j \mid S_{j-1}) - \varphi(S_{j-1} \mid S_{j-2}))w_\varphi(e)$$

by telescoping series. Similarly we have

$$\tilde{w}(e) = \varphi(S_1)\tilde{w}_\varphi(e) + \sum_{j=2}^{i}(\varphi(S_j \mid S_{j-1}) - \varphi(S_{j-1} \mid S_{j-2}))\tilde{w}_\varphi(e).$$

[2]For the special case of matroid rank functions, [34] proved the claim for the unweighted setting as well as the weighted setting under a similar but different rounding process based on the Poisson distribution.

[3]By which we mean that c_1 can be made arbitrarily large by making the input constant c_0 arbitrarily small.

Consequently we have

$$w(Q) = \varphi(S_1)w_\varphi(Q) + \sum_{i=2}^{k}(\varphi(S_i \mid S_{i-1}) - \varphi(S_{i-1} \mid S_{i-2}))w_\varphi(Q \cap S_{i-1}),$$

$$\tilde{w}(Q) = \varphi(S_1)\tilde{w}_\varphi(Q) + \sum_{i=2}^{k}(\varphi(S_i \mid S_{i-1}) - \varphi(S_{i-1} \mid S_{i-2}))\tilde{w}_\varphi(Q \cap S_{i-1}).$$

Finally, by (a) the triangle inequality and (b) inequality (3.4), we have

$$\begin{aligned}|\tilde{w}(Q) - w(Q)| &\overset{(a)}{\leq} \varphi(S_1)|w_\varphi(Q) - \tilde{w}_\varphi(Q)| \\ &\quad + \sum_{i=2}^{k}(\varphi(S_i \mid S_{i-1}) - \varphi(S_{i-1} \mid S_{i-2}))|w_\varphi(Q \cap S_{i-1}) - \tilde{w}_\varphi(Q \cap S_{i-1})| \\ &\overset{(b)}{\leq} \epsilon\varphi(S_1)w_\varphi(Q) + \epsilon\sum_{i=2}^{k}(\varphi(S_i \mid S_{i-1}) - \varphi(S_{i-1} \mid S_{i-2}))w_\varphi(Q \cap S_{i-1}) \\ &= \epsilon w(Q),\end{aligned}$$

as desired.

It remains to prove the claim. It suffices to prove that (3.4) holds for fixed S_i and all Q with high probability. The claim then follows from the union bound over all (up to n) choices of S_i.

Fix S_i. We claim that S_i has strength between $1/\alpha$ and 1 with respect to the rescaled weights w_φ. First, we have

$$\frac{w_\varphi(S_i) - w_\varphi(S_{i+1})}{f(S_i) - f(S_{i+1})} = \frac{\varphi(S_{i+1} \mid S_i)}{\varphi(S_{i+1} \mid S_i)} = 1$$

To show that S_i has strength at least $1/\alpha$ with respect to w_φ, let T be any subset of S_i. We need to show that $\varphi(T \mid S_i) \geq 1/\alpha$. We have

$$\begin{aligned}w_\varphi(S_i) - w_\varphi(T) &= \sum_{j=i}^{k-1} w_\varphi(S_j \setminus S_{j+1}) - w_\varphi(T \cap (S_j \setminus S_{j+1})) \\ &= \sum_{j=i}^{k-1} \frac{w(S_j \setminus S_{j+1}) - w(T \cap (S_j \setminus S_{j+1}))}{\varphi(S_{j+1} \mid S_j)}. \qquad (3.5)\end{aligned}$$

For each $j \in \{i, \ldots, k-1\}$, the restriction of f to S_j has strength at least $\varphi(S_{j+1} \mid S_j)/\alpha$. Contracting S_{j+1} does not decrease the strength. Thus

$$w(S_j \setminus S_{j+1}) - w(T \cap (S_j \setminus S_{j+1})) \geq \left(\frac{\varphi(S_{j+1} \mid S_j)}{\alpha}\right)(f(S_j \mid S_{j+1}) - f(T \cap S_j \mid S_{j+1}))$$

for each $j \in \{i, \ldots, k-1\}$. Substituting into (3.5) gives

$$w_\varphi(S_i) - w_\varphi(T) \geq \frac{1}{\alpha}\sum_{j=i}^{k-i} f(S_j \mid S_{j+1}) - f(T \cap S_j \mid S_{j+1}). \qquad (3.6)$$

For each $j \in \{i, \ldots, k-1\}$, we have $f(T \cap S_j \mid S_{j+1}) \leq f(T \cap S_j \mid T \cap S_{j+1})$ by submodularity. This gives

$$w_\varphi(S_i) - w_\varphi(T) \geq \frac{1}{\alpha}\sum_{j=i}^{k} f(S_j \mid S_{j+1}) - f(T \cap S_j \mid T \cap S_{j+1}) = \frac{1}{\alpha}(f(S_i) - f(T))$$

by telescoping series. Thus $\varphi(T \mid S_i) \geq 1/\alpha$, and taken over all subsets T of S_i, we conclude that S_i has strength at least $1/\alpha$.

The randomized weights $\tilde{w}_\varphi$ are randomly rounding w_φ with respect to the thresholds

$$\frac{\tau_e}{\varphi(S_i \mid S_{i-1})} \leq \frac{c\epsilon^2}{\alpha \ln(nr)}$$

for all $e \in S_{i-1} \setminus S_i$. Applying lemma 3.3 to S_{i-1} with weights w_φ and strength (at least) $1/\alpha$, with high probability, $\tilde{w}_\varphi$ preserves the weight $w_\varphi(Q)$ of every quotient Q in S_{i-1} up to a $(1+\epsilon)$-factor. By lemma 2.1, each intersection $Q \cap S_i$, where Q is a quotient of f, is a quotient in S_i. Thus $\tilde{w}_\varphi$ preserves the weight of every intersection $Q \cap S_i$ per the claimed inequality (3.4), over all quotients Q of f, with high probability. This completes the proof. ∎

Completing the proof of theorem 1.1. Let $\tilde{w}$ be the corresponding randomized set of weights output by `quotient-sparsification`. By lemma 3.4, with high probability, $\tilde{w}$ preserves the weight of every quotient up to a $(1+\epsilon)$-factor. It remains to bound the number of nonzeroes in $\tilde{w}$.

We claim that each element e has $\tilde{w}(e) > 0$ with probability at most $w(e)/\tau_e$. Of course this holds if $w(e) \geq \tau_e$. If $w(e) < \tau_e$, then $\tilde{w}(e)$ is nonzero with probability $p_e = w(e)/\tau_e$.

By linearity of expectation, the expected number of nonzeroes is bounded above by

$$\begin{aligned}\sum_e \frac{w(e)}{\tau_e} &= O\big(\ln(nr)/\epsilon^2\big) \sum_{i=1}^{k} \frac{w(S_i) - w(S_{i-1})}{\varphi(S_{i-1} \mid S_i)} \\ &\leq O\big(\ln(nr)/\epsilon^2\big) \sum_{i=1}^{k} f(S_i) - f(S_{i-1}) = O\big(r \ln(nr)/\epsilon^2\big).\end{aligned}$$

By the Chernoff inequality, the actual number is at most a constant factor greater than the expected value with high probability. This completes the proof of theorem 1.1.

4 Sparsifying matroids in the rank oracle model: proof of theorem 1.2

Let $\mathcal{M} = (\mathcal{N}, \mathcal{I})$ be a matroid with n elements and rank r, and let $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be a set of nonnegative weights. Theorem 1.2 claims that $\mathcal{M}$ can be sparsified to $O\big(n \log(n)/\epsilon^2\big)$ elements while preserving the weight of every quotient up to a $(1+\epsilon)$-factor, in nearly linear time and queries to a rank oracle. A polynomial running time and oracle complexity already follows from theorem 1.1.

We follow the general framework from section 3 where we first compute an approximate strength decomposition and assign sampling probabilities based on the decomposition. To compute a decomposition we repeatedly extract sets that are both strong and have low strength-ratio relative to the previous set in the decomposition.

Extracting strong and low strength-ratio sets with small integer weights. The following lemma describes a subroutine for extracting one such set at a time, that is efficient for small integer weights and strength.

Lemma 4.1. *Let w be integral. For $k \in \mathbb{N}$ and $\epsilon \in (0,1)$, in $O(n(k + \log(n)) \log(nk) Q_{\text{RANK}}/\epsilon)$ time, one can compute a closed set $S \subseteq \mathcal{N}$ such that:*
(a) $\kappa(S) \geq k$.
(b) If $S \neq \mathcal{N}$, then $\varphi(S) \leq (1+\epsilon)k$.

The proof of lemma 4.1 is placed in appendix A as it is easier to discuss separately. We mention in passing that the algorithm implementing lemma 4.1 tries to pack k bases to certify that $\kappa(\mathcal{M}) \geq k$. The case where $S \neq \mathcal{N}$ arises in the dual when such a packing is not found.

Lemma 4.1 is only useful when the weights are integral and the strength paramter k is small. We want to extend lemma 4.1 to general weights and strength. The high-level idea is to apply uniform sampling (lemma 3.3) to reduce the weights and strength parameter to small integers, and then apply lemma 4.1 to the sparsified set. The issue is that the output set of elements S is only guaranteed to have the desired strength with respect to the randomized weights generated from the random sample, instead of the input weights w. To ensure that S has the desired strength and strength-ratio with respect to w as well, we need to strengthen lemma 3.3.

Stronger uniform sampling. The following lemma strengthens lemma 3.3 to show that uniform sparsification also preserves the strength of closed sets S with low strength-ratio $\varphi(S)$. The setup is the same as in lemma 3.3 except the hidden constant c_1 should be smaller. The lemma is for all normalized monotone submodular functions f.

LEMMA 4.2. *Let $\tau_0 \overset{\text{def}}{=} c_1\epsilon^2\kappa_f/\log(nr)$ for a sufficiently small constant c_1. Let $\tau : \mathcal{N} \to \mathbb{R}$ be a set of element thresholds values such that $0 \leq \tau_e \leq \tau_0$ for all $e \in \mathcal{N}$. Let $\tilde{w} : \mathcal{N} \to \mathbb{R}_{\geq 0}$ be a randomized set of weights such that for each element $e \in \mathcal{N}$, we randomly and independently set*

$$\tilde{w}(e) = \begin{cases} \lceil w(e)/\tau_e \rceil \tau_e & \text{with probability } p_e \overset{\text{def}}{=} w(e)/\tau_e - \lfloor w(e)/\tau_e \rfloor, \\ \lfloor w(e)/\tau_e \rfloor \tau_e & \text{with probability } 1 - p_e. \end{cases}$$

For a set S, let $\tilde{\kappa}_S$ denote the strength of S with respect to $\tilde{w}$.

With high probability, $\tilde{w}$ satisfies all of the following:

(i) For all quotients Q, $|\tilde{w}(Q) - w(Q)| \leq \epsilon \tilde{w}(Q)$.

(ii) For all closed sets S with $\varphi(S) \leq 2\kappa_f$, $\kappa(S) \geq (1-\epsilon)\tilde{\kappa}(S) - \epsilon\kappa_f$.

Proof. We have already established, in lemma 3.3, that $\tilde{w}$ ϵ-approximates the weight of all quotients with high probability (result (i)). We need to show that result (ii) holds with high probability. Then both results (i) and (ii) hold simultaneously with high probability by the union bound.

For a closed set S, because S is closed, a set $T \subseteq S$ is closed in (the restriction to) S iff it is closed in $\mathcal{N}$. (Equivalently, $\operatorname{span}(T) \subseteq S$ for all $T \subseteq S$ iff S is closed.) Call a set Q a *subquotient* if it is of the form $S \setminus T$, where $S \subseteq \mathcal{N}$ and $T \subseteq S$ are both closed. We identify two types of subquotients of interest:

- *Type 1:* A subquotient $Q = S \setminus T$ is *type 1* if $\varphi(S) \leq 2\kappa_f$ and $\kappa(S) \geq \kappa_f$.
- *Type 2:* A subquotient $Q = S \setminus T$ is *type 2* if $\varphi(S) \leq 2\kappa_f$ and $\varphi(T) \leq 2\kappa_f$.

We claim that with high probability, we approximately preserve the weight of both types of subquotients in the following sense:

- For all quotients $Q = S \setminus T$ of type 1, we have

$$|w(Q) - \tilde{w}(Q)| \leq \epsilon w(Q). \tag{4.7}$$

- For all quotients $Q = S \setminus T$ of type 2, we have

$$|w(Q) - \tilde{w}(Q)| \leq \epsilon w(Q) + \epsilon\kappa_f. \tag{4.8}$$

We defer the proof of the claim that eqs. (4.7) and (4.8) hold with high probability to the end. We first assume they hold and show how they imply result (ii) of the lemma.

Let S be a closed set with $\varphi(S) \leq 2\kappa_f$. We want to show that $\kappa(S) \geq (1-\epsilon)\tilde{\kappa}(S) - \epsilon\kappa_f$. We have two cases.

Case 1: $\kappa(S) \geq \kappa_f$. Then all quotients of S are quotients of type 1. We have

$$\kappa_S = \min_{\substack{T \subseteq S \\ T \text{ closed}}} \frac{w(S) - w(T)}{f(S) - f(T)} \overset{\text{(a)}}{\geq} \min_{\substack{T \subseteq S \\ T \text{ closed}}} (1-\epsilon)\frac{\tilde{w}(S) - \tilde{w}(T)}{f(S) - f(T)} = (1-\epsilon)\tilde{\kappa}_S,$$

where (a) invokes (4.7).

Case 2: $\kappa(S) \leq \kappa_f$. Then there is a closed subset $T \subsetneq S$ such that

$$\varphi(T \mid S) = \frac{w(S) - w(T)}{f(S) - f(T)} \leq \kappa_f.$$

Observe that

$$\varphi(T) \leq \max\{\varphi(S \mid \mathcal{N}), \varphi(T \mid S)\} \leq 2\kappa_f.$$

This makes $S \setminus T$ is a subquotient of type 2.

Now we have

$$\tilde{\kappa}(S) \leq \frac{\tilde{w}(S) - \tilde{w}(T)}{f(S) - f(T)} \overset{\text{(b)}}{\leq} (1+\epsilon)\varphi(T \mid S) + \frac{\epsilon\kappa_f}{f(S) - f(T)} \overset{\text{(c)}}{\leq} (1+\epsilon)\kappa_S + \epsilon\kappa_f,$$

as desired. Here (b) applies (4.8) and (c) is because $f(S) \geq f(T) + 1$ by assumption.

This proves the lemma, assuming the claimed eqs. (4.7) and (4.8) hold with high probability. It remains to prove these claims. We first prove that eq. (4.7) holds with high probability for all quotients of type 1.

Suppose a closed set S has $\kappa_S \geq \kappa_f$. Then τ_0 is also small enough relative to κ_S that, by lemma 3.3, $\tilde{w}$ ϵ-approximations the weight of quotients of S with high probability. That is, we have eq. (4.7) for all subquotients of type 1 with fixed S with high probability. By lemma 3.2, there are poly(rn) closed sets S with $\varphi(S) \leq 2\kappa_f$. Taking the union bound over all c;psed S with $\kappa_f \leq \varphi(S) \leq 2\kappa_f$, we have eq. (4.7) for all subquotients of type 1 with high probability.

Next we consider subquotients of type 2. For any particular subquotient $Q = S \setminus T$, eq. (4.8) holds with high probability by Chernoff bounds.[4] Meanwhile, there are at most poly(rn) subquotients of type 2, because by lemma 3.2, there are at most poly(rn) closed sets S (or T) with $\varphi(S) \leq 2\kappa_f$. Taking the union bound, (4.8) holds for all subquotients of type 2 with high probability. ◻

Strong and low strength-ratio sets with general weights. Now we extend lemma 4.1 to general weights and strengths via uniform sampling, leveraging the stronger guarantees of lemma 4.2.

LEMMA 4.3. *Let $\lambda > 0$ and let $\epsilon > 0$ be sufficiently small. Suppose $\mathcal{M}$ has strength at least $\Omega(\lambda)$. In $O\big(n \log^2(n) Q_{\text{RANK}}/\epsilon^3\big)$ randomized time and with high probability, one can either:*

(i) Declare (correctly) that $\mathcal{M}$ has strength at least λ.

(ii) Compute a set $S \subseteq \mathcal{N}$ such that:

- *(a) S has strength-ratio $\varphi(S) \leq (1+\epsilon)\lambda$.*
- *(b) If S is nonempty, then S has strength $\kappa_S \geq \lambda$.*

Proof. For ease of notation, we relax the guarantee (ii).(a) to have $\varphi(S) \leq (1+\epsilon)^4\lambda$ instead of $\varphi(S) \leq (1+\epsilon)\lambda$. The claimed upper bound of $\varphi(S) \leq (1+\epsilon)\lambda$ then follows from decreasing ϵ by a constant factor as a preprocessing step.

Let $\tau = c\epsilon^2\lambda/\log(n)$ for a sufficiently small constant $c > 0$. Let $k = (1+\epsilon)^3\lambda/\tau = O\big(\log(n)/\epsilon^2\big)$. Decreasing c by a constant factor as needed, we assume that k is an integer.

For each element e, we independently sample a random weight $\tilde{w}(e)$ where

$$\tilde{w}(e) = \begin{cases} \lceil w(e)/\tau \rceil \tau & \text{with probability } p_e = w(e)/\tau - \lfloor w(e)/\tau \rfloor, \\ \lfloor w(e)/\tau \rfloor \tau & \text{with probability } 1 - p_e. \end{cases}$$

All weights in $\tilde{w}$ are integer multiples of τ. We apply lemma 4.1 to $\mathcal{M}$ with the integer weights $\tilde{w}/\tau$ and parameters k and ϵ, returning a closed set S. By lemma 4.1 (for $k = O\big(\log(n)/\epsilon^2\big)$), the running time is $O\big(n \log^2(n) Q_{\text{RANK}}/\epsilon^3\big)$.

Since $(\mathcal{M}, w)$ has strength $\Omega(\lambda)$, $\tilde{w}$ satisfies the conditions of lemma 4.2 for error parameter ϵ. With high probability, $\tilde{w}$ satisfies the properties (i) and (ii) of lemma 4.2 with error parameter ϵ. Henceforth we assume this is the case.

If $S = \mathcal{N}$, then this certifies that $(\mathcal{M}, \tilde{w}/\tau)$ has strength at least k, hence $(\mathcal{M}, \tilde{w})$ has strength at least $k\tau = (1+\epsilon)^3\lambda$. Since $\tilde{w}$ preserves all quotients up to a $(1 \pm \epsilon)$-factor, $\mathcal{M}$ has strength at least $(1-\epsilon)k\tau = (1-\epsilon)(1+\epsilon)^3\lambda \geq \lambda$ with respect to w. So in this case we declare that $\mathcal{M}$ has strength at least λ.

Otherwise $S \subsetneq \mathcal{N}$. By lemma 4.1, S has strength at least $k\tau = (1+\epsilon)^3\lambda$ and strength-ratio at most $(1+\epsilon)k\tau = (1+\epsilon)^4\lambda$ with respect to $\tilde{w}$. Since $\tilde{w}$ ϵ-approximates all quotients, and S is closed, we have

$$\varphi(S) = \frac{w(\mathcal{N}) - w(S)}{f(\mathcal{N}) - f(S)} \leq (1+\epsilon)\frac{\tilde{w}(\mathcal{N}) - \tilde{w}(S)}{f(\mathcal{N}) - f(S)} \leq (1+\epsilon)^5\lambda,$$

as desired. Additionally, by lemma 4.2.(ii), we have

$$\kappa_S \geq (1-\epsilon)(1+\epsilon)^3\lambda - \epsilon\lambda \geq \lambda,$$

as desired. ◻

[4]Here we invoke the following "mixed multiplicative-additive" form which also follows from standard proofs of the Chernoff bound: *Let $X_1, \ldots, X_n \in [0,1]$ be independent random variables, $\epsilon \in (0,1)$, and $\beta > 0$. Let $\mu = \sum_{i=1}^n \mathbf{E}[X_i]$ be the expected sum. Then*
$$\mathbf{P}[|X_1 + \cdots + X_n - \mu| \geq \epsilon(\mu + \beta)] \leq 2e^{-\epsilon^2\beta}.$$

```
matroid-sparsification($\mathcal{M} = (\mathcal{N}, \mathcal{I})$, $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, $\epsilon \in (0,1)$)
/* We assume the total weight is $W$ for some $W = \text{poly}(n)$. Wlog we assume the minimum element weight is
   $r$.                                                                                                   */
1. Let $\epsilon_0 > 0$ be a sufficiently small constant. Let $S_0 = \mathcal{N}$, $\lambda_0 = 1 + \epsilon_0$, and $k = 0$.
2. While $S_k \neq \emptyset$:
   A. Apply lemma 4.3 with groundset $S_k$, strength parameter $\lambda_k$, and error parameter $\epsilon_0$.
      1. If this certifies that $S_k$ has strength at least $\lambda_k$, then set $\lambda_k = (1 + \epsilon_0)\lambda_k$.
      2. Otherwise the subroutine returns a set $S$. Set $S_{k+1} = S$, $\lambda_{k+1} = (1 + \epsilon_0)\lambda_k$, and $k = k + 1$.
3. Let $(T_1 = \mathcal{N}), \ldots, (T_\ell = \emptyset)$ be the subsequence of $S_1, \ldots, S_k$ obtained by repeatedly removing any set $S_i$
   where $\varphi(S_i \mid S_{i-1}) > \varphi(S_{i+1} \mid S_i)$ with respect to $w$.
4. For each element $e \in \mathcal{N}$:
   A. Let $\tau_e = \frac{c\epsilon^2}{\ln(n)} \varphi(T_i \mid T_{i-1})$, where $e \in T_{i-1} \setminus T_i$, for a sufficiently small constant $c > 0$.
   B. Randomly (and independently) sample

      $\tilde{w}(e) = \begin{cases} \lceil w(e)/\tau_e \rceil \tau_e & \text{with probability } p_e \overset{\text{def}}{=} w(e)/\tau_e - \lfloor w(e)/\tau_e \rfloor, \\ \lfloor w(e)/\tau_e \rfloor \tau_e & \text{with probability } 1 - p_e. \end{cases}$

5. Return $\tilde{w}$.
```

Figure 2: Sparsifying a matroid with polynomially bounded weights.

4.1 Proof of theorem 1.2 We are now prepared to prove theorem 1.2. For ease of notation we fix $\epsilon \in (0,1)$ and show how to preserve the weight of all quotients up to a $(1+\epsilon)^3$-factor. The claimed $(1+\epsilon)$-factor follows from decreasing ϵ by a constant factor. We assume that the weights are polynomially bounded. The extension to general weights is described in appendix D.

Without loss of generality we may assume that all elements have strictly positive weight (lemma 2.2). By scaling, we may assume the minimum weight element is r, and the total weight is W for some $W = \text{poly}(n)$. Then $\mathcal{M}$ has strength at least 1, and any strength ratio $\varphi(S \mid T)$ that may arise is bounded between 1 and nW.

Given the `quotient-sparsification` algorithm from section 3, and the guarantees of theorem 1.2, it suffices to compute a $O(1)$-approximate strength decomposition of the rank function of $\mathcal{M}$ in the claimed running time.

To compute the approximate strength decomposition, we apply lemma 4.3 to $\mathcal{N}$ with $\lambda = 1$ and constant error parameter, and then repeatedly apply lemma 4.3 to its own output with λ increasing by a constant factor, until we are left with an empty set. This produces a decreasing chain of closed sets $(S_0 = \mathcal{N}) \supsetneq S_1 \supsetneq \cdots \supsetneq (S_k = \emptyset)$ that, with high probability, satisfies $\varphi(S_{i+1} \mid S_i) \leq O(1)\kappa(S_i)$ for all i. That is, $S_0, \ldots, S_k$ satisfies properties (a)–(c) of a $O(1)$-approximation strength decomposition with high probability. As discuss in section 3, we obtain a $O(1)$-approximate strength decomposition as a subsequence by iteratively removing any S_i such that $\kappa(S_i \mid S_{i-1}) > \kappa(S_{i+1} \mid S_i)$.

We use this subroutine to compute the $O(1)$-approximate strength decomposition in `quotient-sparsification`. (See fig. 2 for pseudocode of the entire algorithm.) By theorem 1.1, with high probability, this gives a randomized set of weights supported by $O\big(r \log(n)/\epsilon^2\big)$ elements and which preserves the weight of every quotient up to a $(1+\epsilon)$-factor. As for the running time, we have three steps to account for: (a) repeatedly invoking lemma 4.3 to compute the chain of closed sets $S_0, \ldots, S_k$; (b) pruning the chain to a proper strength decomposition, and (c) sampling the elements.

For step (a), each time we invoke lemma 4.3 increases λ by a constant factor. λ starts at 1, and is bounded above by nW as that is the maximum possible strength of any closed subset. Thus λ takes on at most $O(\log(nW)) = O(\log n)$ distinct values, and this bounds the number of times we invoke lemma 4.3. Each invocation takes $O\big(n \log^2(n) Q_{\text{RANK}}\big)$ time. This gives $O\big(n \log^3(n) Q_{\text{RANK}}\big)$ total time spend on step (a).

To extract a proper strength decomposition in $O(nQ_{\text{RANK}})$ time, we first compute $w(S_i)$ and $\text{rank}(S_i)$ for all i. We then iterate through the S_i's in increasing order of i starting from $i = 1$. Whenever $\varphi(S_i \mid S_{i-1}) > \varphi(S_{i+1} \mid S_i)$, we drop S_i and, if $i > 1$, decrement i. Each iteration either advances i, or else decreases i by at most 1 while also decreasing the length of the sequence by 1. Thus, the total number of iterations is at most twice the length of the original sequence.

The final step, (c), is easy to implement in $O(n)$ time. This gives the claimed $O\big(n\log^3(n)Q_{\text{RANK}}\big)$ running time overall.

Remark 4.1. *The only point where the matroid structure is leveraged (beyond the fact that the rank function is normalized monotone submodular) is in lemma 4.1. For general normalized monotone submodular f, given a subroutine implementing the interface of lemma 4.1, one can follow the same generic steps as described above and obtain a $O\big(n\log(nr)/\epsilon^2\big)$-size sparsifier with running time a logarithmic factor over the running time of the subroutine with strength parameter $k = O(\log(nr))$ and constant error parameter $\epsilon = \Omega(1)$. This is the case for hypergraphs in section 5.*

5 Hypergraph sparsification: proof of theorem 1.3

Let $H = (V, E)$ be a hypergraph with positive edge weights $w : E \to \mathbb{R}_{>0}$. Let m denote the number of edges, n the number of vertices, and $p = \sum_{e\in E}|e|$ the total size of H. Let $\epsilon \in (0, 1)$ be fixed. The goal is to compute, with high probability and in $\tilde{O}(p)$ time, a reweighted sub-hypergraph with $O\big(n\log(n)/\epsilon^2\big)$ edges that preserves the weight of every k-cut up to a $(1+\epsilon)$-factor.

For each hyperedge $e \in E$, let A_e be any set of $|e| - 1$ pairs of endpoints of e, interpreted as normal edges, that form a spanning tree of the endpoints of e. We call A_e the *auxiliary edges* of e. Let $A \overset{\text{def}}{=} \bigcup_{e\in E} A_e$ denote the multiset of auxiliary edges, where we distinguish auxiliary edges from different hyperedges even if they have the same endpoints. For a set of hyperedges S, we let $A_S \overset{\text{def}}{=} \bigcup_{e\in S} A_e$ denote the combined sets of auxiliary edges from S.

Let $\text{rank}(\cdots)$ be the rank function of the graphic matroid over A. We define a set function f over E by

$$f(S) = \text{rank}(A_S).$$

Equivalently, $f(S)$ is n minus the number of connected components in the hypergraph induced by S. Note that the choice of spanning tree A_e for each hyperedge e does not effect $f(S)$. (In fact, any set of edges spanning the endpoints of e would have the same effect.) It is easy to that f is monotone, submodular, normalized, and integer-valued, because the rank function also has these properties. f has rank $r \leq n-1$. We call f the *hypergraphic polymatroid function* as it defines a natural polymatroid over E.

A convenient property of f is that the marginal values of f have a clean connection to contracting edges in H.[5] For a set of edges S, consider the function $f(\cdot \mid S)$ of marginal values with respect to S. We claim this defines the corresponding function for the hypergraph H/S obtained by contracting all the edges in S. Indeed, for a set of hyperedges T,

$$f(T \mid S) = f(T \cup S) - f(S) = \text{rank}(A_T \cup A_S) - \text{rank}(A_S)$$

is the same as the rank of A_T in the graph obtained by contracting A_S.

We first identify k-cuts of H with quotients of f.

Lemma 5.1. *The quotients in f are the k-cuts of H.*

Proof. For a set of vertices U, let E_U be the set of hyperedges with all endpoints in U.

Consider a k-cut consisting of the edges crossing a partition $(U_1, \ldots, U_k)$ of V. We claim that for $S = E_{U_1} \cup \cdots \cup E_{U_k}$, $Q = E \setminus \text{span}(S)$ defines a quotient equal to k-cut induced by $(U_1, \ldots, U_k)$. First, if an edge $e \in E$ does not cross $(U_1, \ldots, U_k)$, then it is contained in some U_i, hence $e \notin Q$. Conversely, if e crosses $(U_1, \ldots, U_k)$, then there is an auxiliary edge $a \in A_e$ cut by some U_i. This auxiliary edge is not spanned by A_S in the graphic matroid, hence

$$f(S + e) \geq \text{rank}(A_S + a) > \text{rank}(A_S) = f(S).$$

Thus $e \notin \text{span}(S)$, hence $e \in Q$. We conclude that Q equals the k-cut induced by $(U_1, \ldots, U_k)$.

[5]This is the main reason we prefer this polymatroidal model for hypergraph cuts over one based on the hypergraphic matroid.

Conversely let $Q = E \setminus S$ be a quotient, where $S \subseteq E$ is closed. Let $U_1, \dots, U_k$ be the connected components of A_S. We claim that Q equals the k-cut induced by $(U_1, \dots, U_k)$. First we show that Q is a subset of the k-cut. If $e \in Q$, then there is an auxiliary edge $a \in e$ that is not spanned by A_S in the graphic matroid. This implies e properly crosses $(U_1, \dots, U_k)$. Conversely, if a hyperedge e properly crosses $(U_1, \dots, U_k)$, then there is an auxiliary edge $a \in A$ with endpoints in distinct parts U_i. This auxiliary edge a is not spanned by A_S, hence e is not spanned by S, and $e \in Q$. We conclude that Q equals the k-cut induced by $U_1, \dots, U_k$. $\square$

5.1 A refined count on the number of quotients If we apply theorem 1.1 to the hypergraphic polymatroid function, then we obtain a sparsifier with $O\big(n \log(mn)/\epsilon^2\big)$ edges. Our next goal is to replace the $\log(mn)$-factor with $\log(n)$.

The $\log(mn)$-factor originates in lemma 3.2, which (in this context) states that there are at most $(mn)^{O(t)}$ quotients of weight at most $t\kappa_f$, for $t \in \mathbb{N}$. The following lemma refines the argument in lemma 3.2 and replaces the (mn)-term with n.

Lemma 5.2. *For each integer $t \in \mathbb{N}$, there are at most $n^{O(t)}$ quotients of f with weight at most $t\kappa_f$.*

Proof. We recall the ideas of the proof of lemma 3.2, with variables renamed for our setting.

Consider a randomized collection of quotients of f generated as follows. Initially let $R = \emptyset$. While $f(R) < r - t - 1$, we add to R an edge $e \in E \setminus \operatorname{span}(R)$ randomly sampled in proportion to its weight $w(e)$. This produces a randomized set R with $f(R) \geq r - t - 1$. We return all quotients disjoint from R with total weight at most $t\kappa_f$.

The argument in lemma 3.2 has two parts. The first part argues that there are at most m^{t+1} quotients disjoint from R. The second part argues that for any fixed quotient Q with weight at most $t\kappa_f$, Q is disjoint from R with probability at least $1/\binom{r}{t}$. Together this shows that there are at most $m^{t+1}\binom{r}{t}$ quotients of weight at most $t\kappa_f$.

We modify the first part of the argument to show that there are at most $n^{2(t+1)}$ quotients disjoint from R. Again, we recall the argument from lemma 3.2 that obtained the bounded of m^{t+1}. Fix R with $f(R) \geq r - t - 1$. For a quotient $Q = \bar{X}$ disjoint from R, there is a set S of at most $t+1$ edges from X such that $\operatorname{span}(R \cup S) = X$. Conversely for any set S of at most $t+1$ edges disjoint from R, $\operatorname{span}(R \cup S)$ is the complement of a quotient disjoint from R. This shows that there are at most $\sum_{i=0}^{t+1} \binom{m}{i} \leq m^{t+1}$ quotients disjoint from R.

We modify this argument by focusing on the underlying graphic matroid. It is still the case that for a quotient $Q = \bar{X}$ disjoint from R, there is a set S of at most $t+1$ edges from X such that $\operatorname{span}(R \cup S) = X$. We modify the argument for the converse direction.

Fix a spanning forest F_R of A_R. For each set $S \subseteq \mathcal{N} \setminus R$, let F_S be a spanning forest of $A_{R \cup S}$ extending F_R. Call two edge sets $S_1, S_2 \subseteq \bar{R}$ *equivalent* if F_{S_1} and F_{S_2} span the same subset of A. If S_1 and S_2 are equivalent, then $\operatorname{span}(S_1 \cup R) = \operatorname{span}(S_2 \cup R)$, and S_1 and S_2 induce the same quotient.

This shows that the number of quotients disjoint from R is bounded above by the number of equivalence classes of sets $S \subseteq \bar{R}$. The equivalence class of a set S is defined by any spanning forest of $A_S \cup A_R$ extending F_R. This spanning forest extends F_R with at most t edges from A. When choosing these edges, parallel edges in A are equivalent, so each choice is from at most $\binom{n}{2}$ possibilities. This shows that there are at most

$$\sum_{i=0}^{t+1} \binom{n^2}{i} \leq n^{2(t+1)}$$

equivalence classes, hence at most $n^{2(t+1)}$ quotients disjoint from R.

The rest of the proof of lemma 3.2, replacing m^{t+1} with n^{2t+1} as an upper bound for the number of quotients disjoint from R, leads to the conclusion that there are at most $n^{O(t)}$ quotients of weight at most $t\kappa_f$. $\square$

Replacing lemma 3.2 with lemma 5.2 in the proof of theorem 1.1 decreases the number of output edges from $O\big(n \log(mn)/\epsilon^2\big)$ to $O\big(n \log(n)/\epsilon^2\big)$.

5.2 Nearly linear time strength decompositions in hypergraphs It remains to show that an approximate strength decomposition of a hypergraph can be computed in nearly linear time.

While a sparsification based on the hypergraphic polymatroid function f is not exactly a case of matroid sparsification, we still follow the same general framework. Recall that the first ingredient of the matroid sparsification algorithm was a subroutine (described in lemma 4.1) that extract subsets with strength k and strength-ratio close to k for small integer weights and strength parameter k. The following lemma describes the substitute for the hypergraphic polymatroid function.

Lemma 5.3. *Let $H = (V, E)$ be a hypergraph with integer edge weights, and let $k \in \mathbb{N}$ and $\epsilon > 0$ be parameters. In $\tilde{O}(pk/\epsilon)$ time, one can either:*
(i) Certify that f has strength $\kappa_f \geq k$.
(ii) Compute a closed set $S \subsetneq E$ with strength $\kappa_S \geq k$ and strength-ratio $\varphi(S) \leq (1+\epsilon)\lambda$.

The proof of lemma 5.3 is placed in appendix B. Similar to the case for matroids, it is a combinatorial algorithm that is easier to discuss separately.

Given lemma 5.3, we obtain a sparsification algorithm for hypergraphs and polynomially bounded weighted by taking the matroid sparsification algorithm and replacing the use of lemma 4.1 with lemma 5.3. We omit the proof as it is exactly the same beyond updating variable names and terminology in a straightforward manner. (See also remark 4.1.) Similar to the matroid case, the overall running time is a logarithmic factor over the running time of lemma 5.3 with strength parameter $k = O(\log(mn))$ and constant error, hence $\tilde{O}(p)$ overall. General edges weights reduce to polynomial weights in the same way as for matroid sparsification, as discussed in appendix D.

6 Coverage and SAT sparsification: proofs of theorem 1.4 and corollary 1.2

In this section, we prove the sparsification results for set systems and for SAT.

For set systems the setup is as follows. Let $\mathcal{U}$ be a finite set of m points with positive weights $w \in \mathbb{R}^{\mathcal{U}}_{>0}$, and let $\mathcal{F}$ be a collection of n subsets of $\mathcal{U}$. Let $\epsilon \in (0,1)$. The goal is to compute a subset of $O\big(n\log(n)/\epsilon^2\big)$ points $(\tilde{\mathcal{U}}, \tilde{w})$, where $\tilde{\mathcal{U}} \subseteq \mathcal{U}$ and $\tilde{w} \in \mathbb{R}^{\tilde{\mathcal{U}}}_{>0}$, such that

$$\left| w\left(\bigcup_{S\in\mathcal{F}'} S\right) - \tilde{w}\left(\tilde{\mathcal{U}} \cap \bigcup_{S\in\mathcal{F}'} S\right)\right| \leq \epsilon w\left(\bigcup_{S\in\mathcal{F}'} S\right). \tag{6.9}$$

for all subcollections $\mathcal{F}' \subseteq \mathcal{F}$.

For SAT we have the following setup. Let $\varphi(x_1,\dots,x_n)$ be a CNF formula over n variables. Let $\mathcal{C} = \{C_1,\dots,C_m\}$ be the m clauses of φ, and let $w : \mathcal{C} \to \mathbb{R}_{>0}$ be a set of weights on the clauses. The goal is to compute a reweighted subset of $O\big(n\log(n)/\epsilon^2\big)$ clauses $(\tilde{\mathcal{C}}, \tilde{w})$, where $\tilde{\mathcal{C}} \subseteq \mathcal{C}$ and $\tilde{w} \in \mathbb{R}^{\mathcal{C}'}$, such that for any assignment of $x_1,\dots,x_n \in \{\texttt{True},\texttt{False}\}$ satisfying the clauses $S \subseteq \mathcal{C}$,

$$\big|w(S) - \tilde{w}\big(S \cap \tilde{\mathcal{C}}\big)\big| \leq \epsilon w(S). \tag{6.10}$$

We briefly describe a reduction from SAT sparsification to coverage sparsification. At a high level, given a SAT formula $\varphi(x_1,\dots,x_n)$ with weighted clauses $(\mathcal{C}, w)$, we create a set system where each clause is a point, and each literal assignment ($x_j = \texttt{true}$ or $x_j = \texttt{false}$) maps to the set of all clauses satisfied by the assignment. More formally, for each clause C_i, we create a point p_i with the same weight, $w(p_i) = w(C_i)$. For each variable x_j, we have two sets: a set S_j consisting of all points p_i corresponding to clauses C_i satisfied by setting $x_j = \texttt{true}$, and a set T_j consisting of all points p_i corresponding to clauses C_i satisfied by setting $x_j = \texttt{false}$. It is easy to see that a reweighted subset of points satisfying (6.9) for all subcollections of sets $\mathcal{F}'$ gives a reweighted subset of clauses satisfying (6.10) for all possible assignments of $x_1,\dots,x_n$.

Thus corollary 1.2 reduces to theorem 1.4. As mentioned in section 1, there are two ways to prove theorem 1.4. The first way is a direct approach that yields a faster and simpler algorithm. The second way is by a reduction to hypergraph sparsification essentially due to [39]. We present the direct approach first, and sketch the reduction to hypergraph sparsification at the end of the section for the sake of completeness.

The direct proof of theorem 1.4 consists of three parts. The first is the structural aspect regarding the existence of a sparsified set of points $(\tilde{\mathcal{U}}, \tilde{w})$ as described above (up to log-factors). We address this by reducing to submodular quotient sparsification. The second part refines part of the proof of theorem 1.1 to reduce $|\tilde{\mathcal{U}}|$ from $O\big(n\log(nm)/\epsilon^2\big)$ to $O\big(n\log(n)/\epsilon^2\big)$. The third part shows how to compute $(\tilde{\mathcal{U}}, \tilde{w})$ in nearly linear time with high probability.

6.1 From unions to quotients Consider the "hitting set" function f defined by

$$f(X) = |\{S \in \mathcal{F} : X \cap S \neq \emptyset\}| \text{ for } X \subseteq \mathcal{U}.$$

It is easy to see that f is normalized, monotone, submodular, and integer-valued. We also have $f(\mathcal{U}) = n$.

The following lemma connects the quotients of f with unions of subcollections of $\mathcal{F}$.

Lemma 6.1. *A set $Q \subseteq \mathcal{U}$ is a quotient iff it is the union $Q = \bigcup_{S \in \mathcal{F}'}$ of some $\mathcal{F}' \subseteq \mathcal{F}$.*

Proof. Suppose $Q = \bigcup_{S \in \mathcal{F}'}$ for some $\mathcal{F}' \subseteq \mathcal{F}$. Let $X = \bar{Q}$; we need to show that X is closed. By definition, X is the set of points that don't hit any set in $\mathcal{F}'$. If $y \in \text{span}(X)$, then all sets hit by y are hit by X. In particular, y does not hit any set in $\mathcal{F}'$, so $y \in X$. Thus Q is a quotient.

Conversely, suppose $Q = \bar{X}$ is a quotient. Let $\mathcal{F}_X$ be the collection of sets hit by X, and let $\mathcal{F}' = \mathcal{F} \setminus \mathcal{F}_X$. By definition the sets in $\mathcal{F}'$ are contained in Q. On the other hand, for all $p \in Q$, since $p \notin \text{span}(X)$, we have

$$|\{S \in \mathcal{F}' : p \in S\}| = f(p \,|\, X) > 0,$$

hence p is in some set of $\mathcal{F}'$. Thus Q equals the union of $\mathcal{F}'$. ∎

6.2 A refined count on the number of unions By lemma 6.1, the quotients of f are exactly the unions of subcollections of sets. By theorem 1.1, there exists a set of weights $\tilde{w}$ with at most $O\big(n \log(mn)/\epsilon^2\big)$ nonzeroes that preserves the weight of every quotient of f, hence the weight of every union of sets, up to a $(1+\epsilon)$-factor. The next step is to reduce the $\log(mn)$-factor to $\log(n)$.

The $\log(mn)$-factor comes from lemma 3.2, which states that there are at most $(mn)^{t+1}$ unions of weight at most $t\kappa_f$, for $t \in \mathbb{N}$. The following lemma refines the argument in lemma 3.2 and reduces the base of the exponent from mn to $2n$.

Lemma 6.2. *For all $t \in \mathbb{N}$, there are at most $(2n)^t$ quotients of weight at most $t\kappa_f$.*

Proof. Let $R \subseteq \mathcal{U}$ with $f(R) \geq r - t - 1$. The proof of lemma 3.2 argues that there are at most m^{t+1} quotients disjoint from R. We claim that there are at most 2^{t+1} quotients disjoint from R. Plugging this new bound into the rest of the proof of lemma 3.2 shows that there are $2^{t+1}\binom{r}{t} \leq (2r)^{t+1} = (2n)^{t+1}$ quotients of weight at most $t\kappa_f$, as desired.

For a quotient Q to be disjoint from R, all of the sets hit by Q must be disjoint by R. Moreover, by lemma 6.1, every quotient Q is the union of the subfamily of sets hit by Q. Thus every quotient Q disjoint from R is a union of sets disjoint by R. There are at most $t+1$ sets in R, hence at most 2^{t+1} quotients disjoint from R. ∎

Replacing lemma 3.2 with lemma 6.2 in the proof of theorem 1.1 leads to a reweighted subset of $O\big(n \ln(n)/\epsilon^2\big)$ points preserving the weight of every union up to a $(1+\epsilon)$-approximate factor.

6.3 A nearly linear time algorithm Lastly, we consider the algorithmic aspect of theorem 1.4, which claims that a set system can be sparsified in nearly linear time. The bottleneck is in computing an approximate strength decomposition. We let $N = \sum_{S \in \mathcal{F}} |S|$ denote the total size of the set system. In the sequel, the "strength", "strength-ratio", and so forth of a set system refers to that of the associated hitting set function f. Without loss of generality, we may assume all points have nonzero weight (cf. lemma 2.2).

We build an approximate strength decomposition by iteratively computing sets that are strong and have low strength-ratio relative to the previous set in the decomposition. The following lemma describes a subroutine for extracting one such set at a time. It is the most important and involved part of the overall algorithm.

Lemma 6.3. *Let $(\mathcal{U}, \mathcal{F})$ be a set system of total size N. Let $\mathcal{U}$ be weighted with minimum weight 1 and the total weight W. Let $\epsilon \in (0,1)$ and $\lambda > 0$ be parameters. In $O\big(N \log^2(W)/\epsilon^2\big)$ time, one can compute a set of points $\mathcal{U}' \subseteq \mathcal{U}$ such that:*

(a) $\mathcal{U}'$ has strength $\kappa_{\mathcal{U}'} \geq \lambda$.

(b) If $\mathcal{U}' \neq \mathcal{U}$, then $\mathcal{U}'$ has strength ratio $\varphi(\mathcal{U}') < (1+\epsilon)\lambda$.

The proof of lemma 6.3 is placed in appendix C as it is easier to understand separately. We mention in passing that the subroutine is based on the push-relabel flow algorithm applied to a fractional matching problem between sets and points where a set can only be matched to points in the set.

Next we use lemma 6.3 to construct an approximate strength decomposition for polynomially bounded weights.

LEMMA 6.4. *For polynomially-bounded weights, a $O(1)$-approximate strength decomposition can be computed in $O\big(N \log^3(m+n)\big)$ time.*

Proof. By rescaling, we may assume the minimum weight is at least m. Consequently, any strength-ratio is at least 1. Since the maximum weight is $\text{poly}(m,n)$, the maximum possible finite strength-ratio is bounded above by $\text{poly}(m,n)$.

The algorithm is as follows. Let $\mathcal{U}_0 = \mathcal{U}$, $\lambda_0 = 1$, and $k = 0$. We maintain the invariant that $\mathcal{U}_k$ has strength at least λ_k. While $\mathcal{U}_k \neq \emptyset$, we apply lemma 6.3 with strength parameter $2\lambda_k$ and error parameter $\epsilon = 1$ to $\mathcal{U}_k$, returning a set $\mathcal{U}'$. If $\mathcal{U}' = \mathcal{U}_k$ then we replace λ_k with $2\lambda_k$ and repeat with the same index k. Otherwise we set $\mathcal{U}_{k+1} = \mathcal{U}'$, $\lambda_{k+1} = 2\lambda_k$, and $k = k+1$.

This produces a nested sequence of sets $(\mathcal{U}_0 = \mathcal{U}) \supsetneq \mathcal{U}_1 \supsetneq \cdots \supsetneq (\mathcal{U}_k = \emptyset)$. For $i \in \{0,\dots,k\}$, $\mathcal{U}_i$ has strength λ_i. This implies that $k = O(\log(m+n))$ because $2^k \leq \lambda_{k-1} \leq \kappa_{\mathcal{U}_{k-1}} \leq \text{poly}(m,n)$.

For $i \in [k]$, $\mathcal{U}_i$ has strength-ratio $\varphi(\mathcal{U}_i \,|\, \mathcal{U}_{i-1}) \leq 4\lambda_{i-1} \leq 4\kappa_{\mathcal{U}_{i-1}}$. Thus $\mathcal{U}_1,\dots,\mathcal{U}_k$ satisfies the first three properties of a $O(1)$-approximate strength decomposition. We obtain a $O(1)$-approximate strength decomposition as a subsequence by removing any $\mathcal{U}_i$ where $\varphi(\mathcal{U}_i \,|\, \mathcal{U}_{i-1}) > \varphi(\mathcal{U}_{i+1} \,|\, \mathcal{U}_i)$.

As for the running time, we have $O(\log(m+n))$ iterations invoking lemma 6.3, each of which takes $O\big(N \log^2(m+n)\big)$ time. Extracting the strength decomposition as a subsequence can be done in linear time (by the same method as for matroids; see page 17). Overall we obtain a $O\big(N \log^3(m+n)\big)$ running time. ∎

The only nontrivial step in the algorithm `quotient-sparsification` (fig. 1 on on page 10) is to compute an approximate strength decomposition. By lemma 6.4, this step can be implemented in nearly linear time for polynomially bounded weights. This gives a nearly linear time sparsification algorithm for polynomially bounded weights. One can reduce general weights to polynomially bounded weights without increasing the running time, as described in appendix D.

This completes the proof of the algorithmic component of theorem 1.4.

REMARK 6.1. *The running time can be improved by a logarithmic factor by opening up the black box underlying lemma 6.3. We sketch the ideas. As mentioned above, the algorithm underlying lemma 6.3 is a push-relabel flow algorithm. At a high-level, the vertices of this flow problem correspond to the sets $\mathcal{F}$ and the points $\mathcal{U}$, where each set $S \in \mathcal{F}$ is a source of capacity λ and each point $p \in \mathcal{U}$ is a sink of capacity $w(p)$. From one instance of lemma 6.3 to the next, instead of starting a new instance of the push-relabel algorithm, one can keep the configuration from the previous instance, and update it by increasing the source capacity of each set to the next (larger) value of λ. We then run the push-relabel algorithm from this updated configuration to solve the next instance of lemma 6.3. In this manner, the total running time over all instances of lemma 6.3 is that of a single instance, $O\big(N \log^2(m+n)\big)$.*

6.4 Reduction to hypergraphs An alternative proof of theorem 1.4 leverages the hypergraph sparsifier from section 5. [39] described a reduction from SAT sparsification to hypergraph sparsification. The reduction from coverage sparsification to hypergraph sparsification described below is based on essentially the same idea and is included for the sake of completeness.

Given points $\mathcal{U}$, sets $\mathcal{F}$, and weights w, we construct a weighted hypergraph $H = (V,E)$ as follows.

- For each set $S \in \mathcal{F}$, we have a vertex v_S.
- We have an additional auxiliary vertex $v^\star$.
- For each point p, we have a hyperedge e_p consisting of $v^\star$ and v_S for each set S containing p. The weight of e_p is $w(p)$.

We claim that each union of sets in $\mathcal{F}$ corresponds to a cut of hyperedges. Consequently, a cut-preserving sparsifier for H gives a coverage preserving sparsifier for $(\mathcal{F},\mathcal{U})$.

Let $\mathcal{F}' \subseteq \mathcal{F}$ be a collection of sets, and let $Q = \bigcup_{S \in \mathcal{F}'} S$ be their union. Consider the cut in H induced by the set of vertices corresponding to $\mathcal{F}'$, $W = \{v_S : S \in \mathcal{F}'\}$. If an e_p in the cut, then to be incident to a vertex in W, p must lie in some set in $\mathcal{F}'$. Thus $p \in Q$ for every edge e_p in the cut. Conversely, if $p \in Q$, then e_p is incident to some vertex in W. Since e_p is also incident to $v^\star$, and $v^\star \notin W$, the edge e_p is cut by W. Thus the edges cut by W are exactly the edges corresponding to points in Q.

Observe that the number of vertices of H, is one more than the number of sets in $\mathcal{F}$. Thus hypergraph sparsifier yields $O\big(|\mathcal{F}| \log(|\mathcal{F}|)/\epsilon^2\big)$ edges, which implies the same number of points in the corresponding coverage sparsifier, as desired.

Finally, for the sake of running time, we observe that H takes linear time to create, and the total size of H matches the total size of the set system up to constant factors. Thus the nearly linear time algorithm for hypergraph sparsification yields a nearly linear running time for sparsifying set systems. We note that the hypergraph sparsification algorithm is slower than the direct algorithm presented above by logarithmic factors, and more involved sparsifier for H gives a coverage preserving sparsifier for $(\mathcal{F},\mathcal{U})$.

Let $\mathcal{F}' \subseteq \mathcal{F}$ be a collection of sets, and let $Q = \bigcup_{S\in\mathcal{F}'} S$ be their union. Consider the cut in H induced by the set of vertices corresponding to $\mathcal{F}'$, $W = \{v_S : S \in \mathcal{F}'\}$. If an e_p in the cut, then to be incident to a vertex in W, p must lie in some set in $\mathcal{F}'$. Thus $p \in Q$ for every edge e_p in the cut. Conversely, if $p \in Q$, then e_p is incident to some vertex in W. Since e_p is also incident to $v^\star$, and $v^\star \notin W$, the edge e_p is cut by W. Thus the edges cut by W are exactly the edges corresponding to points in Q.

Observe that the number of vertices of H is one more than the number of sets in $\mathcal{F}$. Thus the hypergraph sparsifier yields $O\big(|\mathcal{F}| \log(|\mathcal{F}|)/\epsilon^2\big)$ edges, which implies the same number of points in the corresponding coverage sparsifier, as desired.

Finally, for the sake of running time, we observe that H takes linear time to create, and the total size of H matches the total size of the set system up to constant factors. Thus the nearly linear time algorithm for hypergraph sparsification yields a nearly linear running time for sparsifying set systems. We note that the hypergraph sparsification algorithm is more complicated, and slower by logarithmic factors, than the direct algorithm presented above.

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] Ashwinkumar Badanidiyuru, Shahar Dobzinski, Hu Fu, Robert Kleinberg, Noam Nisan, and Tim Roughgarden. "Sketching valuation functions". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012.* SIAM, 2012, pp. 1025–1035. URL: https://doi.org/10.1137/1.9781611973099.81.

[3] Mourad Baïou and Francisco Barahona. "Packing Hypertrees and the k-cut Problem in Hypergraphs". In: *Learning and Intelligent Optimization - 16th International Conference, LION 2022, Milos Island, Greece, June 5-10, 2022, Revised Selected Papers.* Vol. 13621. Lecture Notes in Computer Science. Springer, 2022, pp. 521–534. URL: https://doi.org/10.1007/978-3-031-24866-5%5C_37.

[4] Nikhil Bansal, Ola Svensson, and Luca Trevisan. "New Notions and Constructions of Sparsification for Graphs and Hypergraphs". In: *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019.* IEEE Computer Society, 2019, pp. 910–928. URL: https://doi.org/10.1109/FOCS.2019.00059.

[5] Joshua D. Batson, Daniel A. Spielman, and Nikhil Srivastava. "Twice-Ramanujan Sparsifiers". In: *SIAM J. Comput.* 41.6 (2012), pp. 1704–1721. Preliminary version in STOC, 2009.

[6] András A. Benczúr and David R. Karger. "Approximating s-t Minimum Cuts in $\tilde{O}(n^2)$ Time". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996.* ACM, 1996, pp. 47–55. URL: https://doi.org/10.1145/237814.237827.

[7] 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.

[8] Jeff A. Bilmes. "Submodularity In Machine Learning and Artificial Intelligence". In: *CoRR* abs/2202.00132 (2022). arXiv: 2202.00132. URL: https://arxiv.org/abs/2202.00132.

[9] Karthekeyan Chandrasekaran and Chandra Chekuri. "Hypergraph k-cut for fixed k in deterministic polynomial time". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020, pp. 810–821. URL: https://doi.org/10.1109/FOCS46700.2020.00080.

[10] Chandra Chekuri and Shi Li. "On the Hardness of Approximating the k-Way Hypergraph Cut problem". In: *Theory Comput.* 16 (2020), pp. 1–8. URL: https://doi.org/10.4086/toc.2020.v016a014.

[11] Chandra Chekuri and Chao Xu. "Minimum Cuts and Sparsification in Hypergraphs". In: *SIAM J. Comput.* 47.6 (2018), pp. 2118–2156. Preliminary version in SODA, 2017.

[12] Yu Chen, Sanjeev Khanna, and Ansh Nagda. "Near-linear Size Hypergraph Cut Sparsifiers". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020, pp. 61–72. URL: https://doi.org/10.1109/FOCS46700.2020.00015.

[13] Werner Dinkelbach. "On Nonlinear Fractional Programming". eng. In: *Management science*. Management Science 13.7 (1967), pp. 492–498.

[14] 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.

[15] Kyle Fox, Debmalya Panigrahi, and Fred Zhang. "Minimum Cut and Minimum k-Cut in Hypergraphs via Branching Contractions". In: *ACM Trans. Algorithms* 19.2 (2023), 13:1–13:22. URL: https://doi.org/10.1145/3570162. Preliminary version in SODA, 2019.

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

[17] 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.

[18] Takuro Fukunaga. "Computing minimum multiway cuts in hypergraphs". In: *Discret. Optim.* 10.4 (2013), pp. 371–382. URL: https://doi.org/10.1016/j.disopt.2013.10.002. Preliminary version in IPCO, 2010.

[19] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. "A General Framework for Graph Sparsification". In: *SIAM J. Comput.* 48.4 (2019), pp. 1196–1223. URL: https://doi.org/10.1137/16M1091666. Preliminary version in STOC 2011.

[20] 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.

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

[22] 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.

[23] 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.

[24] Martin Grötschel, László Lovász, and Alexander Schrijver. "The ellipsoid method and its consequences in combinatorial optimization". In: *Comb.* 1.2 (1981), pp. 169–197. URL: https://doi.org/10.1007/BF02579273.

[25] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity". In: *J. ACM* 48.4 (2001), pp. 723–760. Preliminary version in STOC 1998.

[26] *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 2020. URL: https://doi.org/10.1109/FOCS46700.2020.

[27] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. "A combinatorial strongly polynomial algorithm for minimizing submodular functions". In: *J. ACM* 48.4 (2001), pp. 761–777. Preliminary version in STOC, 2000.

[28] Arun Jambulapati, James R. Lee, Yang P. Liu, and Aaron Sidford. "Sparsifying Sums of Norms". In: *CoRR* abs/2305.09049 (2023). arXiv: 2305.09049. URL: https://doi.org/10.48550/arXiv.2305.09049.

[29] Arun Jambulapati, Yang P. Liu, and Aaron Sidford. "Chaining, Group Leverage Score Overestimates, and Fast Spectral Hypergraph Sparsification". In: *CoRR* abs/2209.10539 (2022). arXiv: 2209.10539. URL: https://doi.org/10.48550/arXiv.2209.10539.

[30] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. "Spectral Hypergraph Sparsifiers of Nearly Linear Size". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 1159–1170. URL: https://doi.org/10.1109/FOCS52979.2021.00114.

[31] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. "Towards tight bounds for spectral sparsification of hypergraphs". In: *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. ACM, 2021, pp. 598–611. URL: https://doi.org/10.1145/3406325.3451061.

[32] David R. Karger. "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm". In: *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*. ACM/SIAM, 1993, pp. 21–30.

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

[34] 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.

[35] David R. Karger. "Random Sampling in Cut, Flow, and Network Design Problems". In: *Math. Oper. Res.* 24.2 (1999), pp. 383–413. URL: https://doi.org/10.1287/moor.24.2.383.

[36] David R. Karger. "Random Sampling in Graph Optimization Problems". PhD thesis. Stanford University, 1995.

[37] David R. Karger. "Using Randomized Sparsification to Approximate Minimum Cuts". In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*. ACM/SIAM, 1994, pp. 424–432.

[38] David R. Karger and Matthew S. Levine. "Fast Augmenting Paths by Random Sampling from Residual Graphs". In: *SIAM J. Comput.* 44.2 (2015), pp. 320–339. Preliminary version in STOC, 2002.

[39] Dmitry Kogan and Robert Krauthgamer. "Sketching Cuts in Graphs and Hypergraphs". In: *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*. ACM, 2015, pp. 367–376. URL: https://doi.org/10.1145/2688073.2688093.

[40] Eugene L. Lawler. "Cutsets and partitions of hypergraphs". In: *Networks* 3.3 (1973), pp. 275–285. URL: https://doi.org/10.1002/net.3230030306.

[41] James R. Lee. "Spectral hypergraph sparsification via chaining". In: *CoRR* abs/2209.04539 (2022). arXiv: 2209.04539. URL: https://doi.org/10.48550/arXiv.2209.04539.

[42] Yin Tat Lee and He Sun. "Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time". In: *SIAM J. Comput.* 47.6 (2018), pp. 2315–2336. URL: https://doi.org/10.1137/16M1061850. Preliminary version in FOCS, 2015.

[43] Michel Lorea. "Hypergraphes et matroides". In: *Cahiers du Centre d'études de recherche opérationnelle* 17 (1975), pp. 289–271.

[44] Hiroshi Nagamochi and Toshihide Ibaraki. "A linear-time algorithm for finding a sparse k-connected subgraph of a k-connected graph." In: *Algorithmica* 7.1 (596 1992), p. 583.

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

[46] Kent Quanrud. "Faster exact and approximation algorithms for packing and covering matroids via push-relabel". In: *CoRR* abs/2303.01478 (2023). arXiv: 2303.01478. URL: https://doi.org/10.48550/arXiv.2303.01478.

[47] Tomasz Radzik. "Fractional Combinatorial Optimization". In: *Encyclopedia of Optimization, Second Edition.* Ed. by Christodoulos A. Floudas and Panos M. Pardalos. Springer, 2009, pp. 1077–1080. URL: https://doi.org/10.1007/978-0-387-74759-0%5C_188.

[48] Akbar Rafiey and Yuichi Yoshida. "Sparsification of Decomposable Submodular Functions". In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022.* AAAI Press, 2022, pp. 10336–10344. URL: https://ojs.aaai.org/index.php/AAAI/article/view/21275.

[49] Alexander Schrijver. "A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time". In: *J. Comb. Theory, Ser. B* 80.2 (2000), pp. 346–355.

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

[51] Tasuku Soma and Yuichi Yoshida. "Spectral Sparsification of Hypergraphs". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019.* SIAM, 2019, pp. 2570–2581. URL: https://doi.org/10.1137/1.9781611975482.159.

[52] Daniel A. Spielman and Nikhil Srivastava. "Graph Sparsification by Effective Resistances". In: *SIAM J. Comput.* 40.6 (2011), pp. 1913–1926. Preliminary version in STOC, 2008.

[53] Daniel A. Spielman and Shang-Hua Teng. "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004.* ACM, 2004, pp. 81–90.

[54] Daniel A. Spielman and Shang-Hua Teng. "Spectral Sparsification of Graphs". In: *SIAM J. Comput.* 40.4 (2011), pp. 981–1025. URL: https://doi.org/10.1137/08074489X.

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

[56] Mingyu Xiao. "An Improved Divide-and-Conquer Algorithm for Finding All Minimum k-Way Cuts". In: *Algorithms and Computation, 19th International Symposium, ISAAC 2008, Gold Coast, Australia, December 15-17, 2008. Proceedings.* Vol. 5369. Lecture Notes in Computer Science. Springer, 2008, pp. 208–219. URL: https://doi.org/10.1007/978-3-540-92182-0%5C_21.

A Computing strong and low strength-ratio sets in the rank oracle model: proof of lemma 4.1

This section describes and analyzes an algorithm fulfilling the requirements of lemma 4.1. The input consists of a matroid $\mathcal{M} = (\mathcal{N}, \mathcal{I})$, integer weights $w : \mathcal{N} \to \mathbb{N}$, and parameters $k \in \mathbb{N}$ and $\epsilon \in (0, 1)$. We assume oracle access to the rank function, and let Q_{RANK} denote the time to query the oracle when discussing running times. The goal is to either (a) certify that $(\mathcal{M}, w)$ has strength k, or (b) output a closed subset S that has strength at least k and strength-ratio at most $(1 + \epsilon)k$.

Preprocessing. For technical reasons we prefer the weights to be bounded above by k. The following lemma shows that we can obtain this by contracting all elements with weight greater than k.

Lemma A.1. *Let H be the set of elements of weight $\geq k$. Let $S \subseteq \mathcal{N} \setminus H$.*
(i) If S has strength at least k in $\mathcal{M}/H$, then $S \cup H$ has strength at least k in $\mathcal{M}$.
(ii) The strength-ratio of S in $\mathcal{M}/H$ equals the strength-ratio of $S \cup H$ in $\mathcal{M}$.
(iii) If S is closed in $\mathcal{M}/H$, then $S \cup H$ is closed in $\mathcal{M}$.

Proof. For the first claim, for any set $A \subseteq S \cup H$, we have

$$\begin{aligned} w(S \cup H) - w(A) &= w(S) - w(A \cap S) + w(H \setminus A) \\ &\overset{(a)}{\geq} k(\operatorname{rank}(S \mid H) - \operatorname{rank}(A \cap S \mid H)) + w(H \setminus A) \\ &= k(\operatorname{rank}(S \cup H) - \operatorname{rank}(A \cup H)) + w(H \setminus A) \\ &= k(\operatorname{rank}(S \cup H) - \operatorname{rank}(A)) + w(H \setminus A) - k \operatorname{rank}(H \mid A) \\ &\overset{(b)}{\geq} k(\operatorname{rank}(S \cup H) - \operatorname{rank}(A)) + w(H \setminus A) - k|H \setminus A| \\ &\overset{(c)}{\geq} k(\operatorname{rank}(S \cup H) - \operatorname{rank}(A)). \end{aligned}$$

Here (a) is by the strength of S in $\mathcal{M}/H$. (b) is because $\operatorname{rank}(H \mid A) \leq |H \setminus A|$. (c) is because each element in H has weight at least k. That is, $\varphi(A \mid S \cup H) \geq k$.

For the second claim, let λ be the strength-ratio of S in $\mathcal{M}/H$. We have

$$w(\mathcal{N}) - w(S \cup H) = w(\mathcal{N} \setminus H) - w(S) = \lambda(\operatorname{rank}(\mathcal{N} \mid H) - \operatorname{rank}(S \mid H)) = \lambda(r - \operatorname{rank}(S \cup H)),$$

so $S \cup H$ also has strength-ratio λ.

For the third claim, let $e \in \operatorname{span}(S \cup H)$. We want to show that $e \in S \cup H$. If $e \notin H$, then have

$$\operatorname{rank}(S + e \mid H) - \operatorname{rank}(S \mid H) = \operatorname{rank}(S \cup H + e) - \operatorname{rank}(S \cup H) \overset{(d)}{=} 0$$

where (d) is because $e \in \operatorname{span}(S \cup H)$. Thus $e \in \operatorname{span}_H(S)$. Since S is closed in $\mathcal{M}/H$, $e \in S$. ∎

We use lemma A.1 to reduce the maximum weight to k as follows. Let H be the set of elements of weight $\geq k$. As a preliminary step, we contract H in $O(nQ_{\text{RANK}})$ time. We run the algorithm (described below) in $\mathcal{M}/H$. If the algorithm certifies that $\mathcal{M}/H$ has strength k, then by lemma A.1, $\mathcal{M}$ has strength k. If the algorithm produces a set $S \subseteq \mathcal{N} \setminus H$ with strength k and strength-ratio at most $(1+\epsilon)k$ in $\mathcal{M}/H$, then by lemma A.1, $S \cup H$ has strength k and strength-ratio at most $(1+\epsilon)k$ in $\mathcal{M}$.

Henceforth we assume that all weights are bounded above by k.

Components of a push-relabel algorithm. The algorithm here is based on the matroid partition push-relabel algorithm of [17]. The algorithm maintains a family of k bases $B_1, \dots, B_k$ and integer labels $\ell : \mathcal{N} \to \mathbb{Z}_{\geq 0}$.

An element e is *active* if it appears in more than $w(e)$ bases. If there are no active elements, then $B_1, \dots, B_k$ is a feasible packing of k bases and certifies that $\mathcal{N}$ has strength k.

The labels are initially set to 0 and are bounded above by a parameter $h = O(\log(nk)/\epsilon)$ called the *height* (with one minor exception explained below). For each label j, and set of elements $X \subseteq \mathcal{N}$, we let $X_j \overset{\text{def}}{=} X \cap \ell^{-1}(j)$ denote the subset of elements with label j. We let $X_{\leq j} \overset{\text{def}}{=} \bigcup_{i \leq j} X_i$ and $X_{\geq j} \overset{\text{def}}{=} \bigcup_{i \geq j} X_i$ be the subsets of elements with labels at most and at least j, respectively.

Invariants. $B_1, \dots, B_k$ and ℓ obey the following invariants.

(I) $\ell(e) = 0$ for any element e appearing in fewer than $w(e)$ bases.

(II) $B_{i,\leq j}$ spans $\mathcal{N}_{\leq j-1}$ for all j.

Basic operations. The high-level goal is to increase the label of all active elements to h. Each iteration the algorithm selects an active element e with label $\ell(e) = j < h$ and executes one of two operations:

1. *Push:* Exchange e out of B_i for some $d \in \mathcal{N}_{j-1}$ such that $B_i - e + d \in \mathcal{I}$.
2. *Relabel:* Increase $\ell(e)$ by 1.

One can push e out of B_i iff $B_i - e$ does not span $\mathcal{N}_{\leq j-1}$. Therefore, to preserve invariant (II), e can be relabeled iff e cannot be pushed.

In addition we have the following operation to remove elements from the system.

3. *Delete:* Remove e from $\mathcal{N}$.

To delete an element e while preserving invariants (I) and (II), we take the following steps. We first set $w(e) = 0$, which excludes e from consideration from invariant (I). Henceforth we focus only on invariant (II). While e is active and has label $\ell(e) < h + 2$, we try to push or (otherwise) relabel e. (We briefly allow $\ell(e)$ to exceed h.) Eventually either e is no longer active (and not in any B_i) or $\ell(e) = h + 2$. If e is not active, then we can remove e from the system without violating invariant (II). If e is active, then it has $\ell(e) = h + 2$, while all other elements have label at most h. By invariant (II), $B_{i,\leq h+1} = B_i - e$ spans $\mathcal{N} - e$ for all B_i containing e. We can now remove e from $\mathcal{N}$ and all B_i while preserving invariant (II).

Counting the basic operations. We first give upper bounds on the total number of push, relabel, and delete operations (including push and relabel operations induced by deletion). There are at most $O(nh)$ relabel operations since each element gets relabeled at most $O(h)$ times.

To account for the push operations, we define a potential Φ by $\Phi = \sum_{i=1}^{k} \sum_{e \in B_i} \ell(e)$. Φ is initially 0, and always nonnegative. Each relabel increases Φ by at most k, and removing an element can only decrease Φ. Each push decreases Φ by 1. As the total increase to Φ is $O(nkh)$, there are at most $O(nkh)$ push operations.

For deletions, of course, each element can be deleted at most once.

Implementation details and running time bounds. We now give more specific implementation details and derive upper bounds on the running time assuming oracle access to the rank function.

Recall that the algorithm repeatedly selects an active element e with label $\ell(e) < h$ and tries to either push e out of some B_i containing e, or otherwise relabels e. It is easy to keep track of the active elements with label less than h so that we can identify the next active element in $O(1)$ time.

Given an active element e with label $\ell(e) = j$, and a base B_i containing e, the first step is to decide if e can be pushed out of B_i. Such a push is possible iff $B_{i,\leq j} - e$ does not span $\mathcal{N}_{\leq j-1}$, which holds iff

$$\operatorname{rank}((B_{i,\leq j} \cup \mathcal{N}_{\leq j-1}) - e) = \operatorname{rank}(B_{i,\leq j} \cup \mathcal{N}_{\leq j-1}) = |B_{i,\leq j}|.$$

Thus a single query to the rank oracle determines whether e can be pushed out of B_i.[6] If e can be pushed out of B_i, then the next step is to identify an element $d \in \mathcal{N}_{j-1}$ to exchange with e. To identify such an element d, we maintain $\mathcal{N}_{j-1}$ in a list $\{d_1, \ldots, d_m\}$, and binary search for the last index ℓ such that

$$\operatorname{rank}((B_{i,\leq j} \cup \{d_1, \ldots, d_\ell\}) - e) = |B_{i,\leq j}|.$$

Then d_ℓ is the desired element d. Here the binary search has $O(\log n)$ iterations and each iteration makes 1 query to the rank oracle.

In summary, it takes 1 query to decide if e can be pushed out of a base B_i, and $O(\log n)$ queries to identify an element d that can be exchanged with d. Given an active element e, we test if e can be exchanged out of at most k bases before deciding to either (a) push e out of some B_i or (b) relabel e. We charge the $O(k)$ oracle queries to the ensuing push or relabel. Each push also costs an additional $O(\log n)$ queries. All put together, we spend $O(nk(k + \log(n))hQ)$ time on push and relabel operations.

Lastly, each deletion reduces to push and relabel operations already accounted for plus $O(1)$ time. An element can only be deleted once, so this adds up to $O(n)$ time which is negligible.

When the push-relabel algorithm halts. Recall that our overall goal is to either certify that $\mathcal{M}$ has strength k, or identify a subset S with strength k and strength-ratio $\varphi(S) \leq (1+\epsilon)k$. Now, the push-relabel algorithm halts when all active elements have label h. If there are no active elements then this certifies that $\mathcal{N}$ has strength k, as desired. The following lemma is about the alternative case where there are still active elements, and shows that there is a set of the form $\mathcal{N}_{\leq j}$ that has the desired strength-ratio $(1+\epsilon)k$.

LEMMA A.2. *Suppose all active elements have label $h = O(\log(nk)/\epsilon)$. Then either:*

(i) There are no active elements, and $B_1, \ldots, B_k$ certifies that the (remaining) ground set has strength k.

(ii) There exists an index $j \in \{1, \ldots, h-1\}$ such that $w(\mathcal{N}_{\geq j}) \leq (1+\epsilon) w(\mathcal{N}_{\geq j+1})$. We have $\operatorname{rank}(\mathcal{N}_{\leq j-1}) < \operatorname{rank}(\mathcal{N})$ and $\varphi(\mathcal{N}_{\leq j-1}) \leq (1+\epsilon)k$.

Proof. Suppose there is at least one active element. Then $\mathcal{N}_h \neq \emptyset$, and $w(\mathcal{N}_h) \geq 1$.

Since $h \geq c \log(nk)/\epsilon$ for a sufficiently large constant c, $w(\mathcal{N}_h) \geq 1$, and $w(\mathcal{N}) \leq nk$, there exists an index $j \in \{1, \ldots, h-1\}$ such that $w(\mathcal{N}_{\geq j}) \leq (1+\epsilon) w(\mathcal{N}_{\geq j+1})$. For this index j, and each B_i, we have $|B_{i,\leq j}| \geq \operatorname{rank}(\mathcal{N}_{\leq j-1})$ because $B_{i,\leq j}$ spans $\mathcal{N}_{\leq j-1}$ (invariant (II)). We also have $w(\mathcal{N}_{\geq j+1}) \leq \sum_{i=1}^{k} |B_{i,\geq j+1}|$ because each $e \in \mathcal{N}_{\geq j+1}$ appears in at least $w(e)$ bases B_i (invariant (I)). Altogether, we have

$$\begin{aligned}(1+\epsilon)k \operatorname{rank}(\mathcal{N}_{\leq j-1}) + w(\mathcal{N}_{\geq j}) &\leq (1+\epsilon)(k \operatorname{rank}(\mathcal{N}_{\leq j-1}) + w(\mathcal{N}_{\geq j+1})) \\ &\leq (1+\epsilon) \sum_{i=1}^{k} |B_i| = (1+\epsilon) k \operatorname{rank}(\mathcal{N}).\end{aligned}$$

[6]Here we count the time spent assembling the query as part of the query.

Rearranging, we have

$$w(\mathcal{N}_{\geq j}) \leq (1+\epsilon)k(\operatorname{rank}(\mathcal{N}) - \operatorname{rank}(\mathcal{N}_{\leq j-1})).$$

Since $w(\mathcal{N}_{\geq j}) > 0$, this implies that $\operatorname{rank}(\mathcal{N}_{\leq j-1}) < \operatorname{rank}(\mathcal{N})$. Dividing both sides by $\operatorname{rank}(\mathcal{N}) - \operatorname{rank}(\mathcal{N}_{\leq j-1})$ gives $\varphi(\mathcal{N}_{\leq j-1}) \leq (1+\epsilon)k$, as desired. $\square$

A.1 Putting it altogether: proof of lemma 4.1 Lemma A.2 shows that after running the push-relabel algorithm, if we do not certify $\mathcal{M}$ to have strength k, we instead have a set $S = \mathcal{N}_{\leq j}$ with strength-ratio $\varphi(S) \leq (1+\epsilon)k$. A natural idea is to recurse on S, either certifying that S has strength k or recursing on a smaller subset with the desired strength ratio. (In the base case, $S = \emptyset$, and $\emptyset$ has strength $+\infty$.)

The only issue with the recursive approach is the efficiency. It is possible that $\operatorname{rank}(S) = \operatorname{rank}(\mathcal{N}) - 1$ in each recursive call, resulting in a depth of recursion of r. To remove the r-factor overhead, we do not re-initialize the push-relabel algorithm on S. Instead we delete all of the elements outside of S, and continue the push-relabel algorithm from there. This way all of the push-relabel computation over all subproblems can be accounted for as parts of a single instance of the push-relabel algorithm. See fig. 3 below for pseudocode.

Let S_{OUT} be the set of elements returned by the algorithm. $B_1, \ldots, B_k$ is a feasible packing of k bases in S_{OUT}, so by lemma A.2, $B_1, \ldots, B_k$ certifies that S_{OUT} has strength at least k. (This includes the case where $S_{\text{OUT}} = \emptyset$ and $B_1, \ldots, B_k = \emptyset$.) Now suppose $S_{\text{OUT}} \neq \mathcal{N}$. Then the algorithm has ℓ iterations for some $\ell > 1$. For $i \in \{0, \ldots, \ell\}$, let S_i be the remaining set of elements after i iterations. (E.g., $S_0 = \mathcal{N}$ and $S_{\text{OUT}} = \operatorname{span}(S_\ell)$.) By lemma A.2, for each index $i \in \ell$, we have $\varphi(S_i \,|\, S_{i-1}) \leq (1+\epsilon)k$. Thus

$$\begin{aligned} w(\mathcal{N}) - w(S_{\text{OUT}}) &= \sum_{i=1}^{\ell} w(S_{i-1}) - w(S_i) \\ &\leq (1+\epsilon)k \sum_{i=1}^{\ell} \operatorname{rank}(S_{i-1}) - \operatorname{rank}(S_i) \\ &= (1+\epsilon)k(\operatorname{rank}(\mathcal{N}) - \operatorname{rank}(S_\ell)) \\ &= (1+\epsilon)k(\operatorname{rank}(\mathcal{N}) - \operatorname{rank}(S_{\text{OUT}})) \end{aligned}$$

by telescoping series, hence $\varphi(S_{\text{OUT}} \,|\, \mathcal{N}) \leq (1+\epsilon)k$.

Lastly we bounded the running time. As discussed above, all the work spent within the push-relabel framework takes $O(nk(k + \log(n)) \log(nk)Q_{\text{RANK}}/\epsilon)$ time. We also need to account for the work identifying the index j in step (2.C). It is easy to maintain $w(\mathcal{N}_{\geq j})$ for all j in $O(1)$ time per relabel and $O(h)$ time per delete. It is also easy to track the set of indices satisfying the conditions of (2.C) with constant overhead. Thus the total time spent on (2.C) is bounded above by $O(nkh) = O(nk \log(nk)/\epsilon)$, and negligible. Lastly, computing the span of the final set S_ℓ takes $O(nQ_{\text{RANK}})$ time.

This concludes the proof of lemma 4.1.

```
1. Initialize the push-relabel algorithm.
2. Repeatedly:
    A. Continue the push-relabel algorithm until all active elements have label $h = O(\log(nk)/\epsilon)$.
    B. If $B_1, \ldots, B_k$ is a feasible packing, then return the span of all the remaining elements.
    C. Otherwise let $j \in \{1, \ldots, h-1\}$ be such $w(\mathcal{N}_{\geq j}) \leq (1+\epsilon)w(\mathcal{N}_{\geq j+1})$. Delete each element in $\mathcal{N}_{\geq j+1}$.
```

Figure 3: Computing a subset with strength k and strength-ratio at most $(1+\epsilon)k$.

B Computing strong and low strength-ratio sets in hypergraphs: proof of lemma 5.3

This section presents an algorithm which fulfills the requirements of lemma 5.3. The input consists of a hypergraph $H = (V, E)$ (with m edges, n vertices, and total size p), integer weights $w \in \mathbb{Z}^E_{\geq 0}$, and parameters $k \in \mathbb{N}$ and $\epsilon > 0$. Let f be the hypergraphic polymatroid function for H (defined in section 5). The goal is to either certify that the hypergraph has strength at least k, or compute a closed set of edges $S \subsetneq E$ with strength-ratio at most $(1+\epsilon)k$ and strength at least k.

Contracting heavy edges. For technical reasons we prefer the weight of each edge e to be bounded above by $kf(e)$. The following lemma allows us to contract edges with weight exceeding this bound.

Lemma B.1. *Let D be a set of hyperedges e with weight $w(e) > kf(e)$. Let $H' = (V', E')$ be the hypergraph obtained by contracting and removing D. Let $S \subseteq E'$.*
(i) If S has strength at least k in H', then $S \cup D$ has strength at least k in H.
(ii) If S has strength-ratio $\varphi_{H'}(S) = \lambda$ in H', then $S \cup D$ has strength-ratio $\varphi(S \cup D) = \lambda$ in H.
(iii) If S is closed in H', then $S \cup D$ is closed in H.

Proof. For the first claim, for any set $A \subseteq S \cup D$, we have

$$\begin{aligned} w(S \cup D) - w(A) &\geq w(S) - w(A \cap S) + w(D \setminus A) \\ &\geq k(f(S \,|\, D) - f(A \cap S \,|\, D)) + w(D \setminus A) \\ &= k(f(S \cup D) - f(A \cup D)) + w(D \setminus A) \\ &= k(f(S \cup D) - f(A)) + w(D \setminus A) - kf(D \,|\, A) \\ &\overset{\text{(a)}}{\geq} k(f(S \cup D) - f(A)), \end{aligned}$$

hence, $\varphi(A \,|\, S \cup D) \geq k$. Here (a) is because

$$kf(D \,|\, A) \overset{\text{(b)}}{\leq} k \sum_{e \in D \setminus A} f(e) \leq w(D \setminus A),$$

where (b) is by submodularity.

For the second claim, we have

$$w(E) - w(S \cup D) = w(E \setminus D) - w(S) \overset{\text{(c)}}{=} \lambda(f(E \,|\, D) - f(S \,|\, D)) = \lambda(f(E) - f(S \cup D)),$$

where (c) is because λ is the strength-ratio of S in H'. Thus $S \cup D$ also has strength-ratio λ.

For the third claim, let e be a hyperedge in H spanned by $S \cup D$ in H. We want to show that $e \in S \cup D$. Suppose $e \notin D$. Recall that the marginal values f_D coincide with the hypergraphic polymatroid function of H'. We have

$$f_D(S + e) - f_D(S) = f(S \cup D + e) - f(S \cup D) \overset{\text{(d)}}{=} 0$$

where (d) is because $e \in \operatorname{span}_H(S \cup D)$. This means that $e \in \operatorname{span}_{H'}(S)$. Since S is closed in H', this means that $e \in S$, as desired. ∎

Henceforth we assume that the total weight is at most $k \sum_e f(e) \leq kp$.

Tree packing certificates Let $T^{(1)}, \ldots, T^{(k)} \subseteq A$. We say that $T^{(1)}, \ldots, T^{(k)}$ is a *packing* if they collectively include at most $w(e)$ auxiliary edges from each hyperedge e. The algorithm is motivated by the following lemma.

Lemma B.2. *Let $T^{(1)}, \ldots, T^{(k)}$ be a packing of k spanning forests of A. Then f has strength at least k.*

Proof. Fix a set of hyperedges S. Since $T^{(1)}, \ldots, T^{(k)}$ is a packing, we have

$$\sum_{i=1}^{k} |T^{(i)} \setminus \operatorname{span}(A_S)| \leq \sum_{i=1}^{k} |T^{(i)} \setminus A_S| \leq w(E) - w(S).$$

Each $T^{(i)}$, as a spanning forest, has $f(E)$ edges, and at most $f(S)$ auxiliary edges spanned by A_S. Thus

$$|T^{(i)} \setminus \operatorname{span}(A_S)| \geq f(E) - f(S)$$

for each i. All put together we have

$$w(E) - w(S) \geq \sum_{i=1}^{k} |T^{(i)} \setminus \operatorname{span}(A_S)| \geq k(f(E) - f(S)).$$

Taken over all S, this shows that f has strength at least k. □

From packing bases to matroid intersection. The algorithm tries to compute a packing of k spanning forests of A to certify the strength of f. However, rather than pack k forests directly, we reframe the problem as a matroid intersection problem.

For a hyperedge e, let $A_e \times k$ denote a set consisting of k copies of each auxiliary edge $a \in A$, denoted $a^{(1)}, \ldots, a^{(k)}$. Let $A \times k \overset{\text{def}}{=} \bigsqcup_{e \in E} A_e \times k$ denote the disjoint union of all these copies. We have $|A \times k| \leq pk$. For $i \in [k]$, let $A^{(i)} \overset{\text{def}}{=} \{a^{(i)} : a \in A\}$ and $A_e^{(i)} \overset{\text{def}}{=} \{a^{(i)} : a \in A\}$. For a set $S \subseteq A \times k$, let $S^{(i)} \overset{\text{def}}{=} S \cap A^{(i)}$ denote the subset of auxiliary edges with index i.

We define one matroid over $A \times k$ that is essentially the k-fold disjoint union of a graphic matroid over each $A^{(i)}$. More formally, the independent sets are the sets $T \subseteq A \times k$ such that for each $i \in [k]$, $T^{(i)} \subseteq A^{(i)}$ forms a forest. The second matroid is the partition matroid over $A \times k$ allowing at most $w(e)$ total auxiliary edges from e (over all i) for all hyperedges e. We let $\operatorname{rank}_{\text{G}}$ and $\operatorname{rank}_{\text{P}}$ denote the rank function and let $\operatorname{span}_{\text{G}}$ and $\operatorname{span}_{\text{P}}$ denote the span function for these graphic and partition matroids, respectively.

Components of a push-relabel algorithm. We develop a push-relabel algorithm for this intersection problem based on the more abstract push-relabel framework described by [17]. The algorithm maintains a base $T \subseteq A \times k$ in the graphic matroid, a base $B \subseteq A \times k$ in the partition matroid, and integer labels $\ell : A \times k \to \mathbb{Z}_{\geq 0}$ for all auxiliary edges. T being a base means that $T^{(i)}$ is a spanning forest of $A^{(i)}$ for all i. B being a base means that for every hyperedge e, B contains exactly $w(e)$ edges from $A_e \times k$. T and B are initialized as arbitrary bases.

We call an auxiliary edge $a^{(i)}$ *active* if $a^{(i)} \in T^{(i)} \setminus B$. If there are no active auxiliary edges, then each hyperedge e has at most $w(e)$ auxiliary edges in T. By lemma B.2, this certifies that f has strength at least k.

We now expand on the integer labels. The integer labels are initially set to 0 and bounded above by a parameter $h = O(\log(pk)/\epsilon)$ called the *height*. For a set of auxiliary edges $X \subseteq A$, and label j, let $X_j \overset{\text{def}}{=} X \cap \ell^{-1}(j)$ denote the subset of auxiliary edges with label j. We let $X_{\leq j} \overset{\text{def}}{=} \bigcup_{i \geq j} X_i$ and $X_{\leq j} \overset{\text{def}}{=} \bigcup_{i \leq j} X_i$ denote the subsets of auxiliary edges with label at most and at least j, respectively. We associate each hyperedge with the maximum label over all its auxiliary edges. For $j \in \mathbb{Z}_{\geq 0}$, let E_j denote the set of hyperedges e where the maximum label over $A_e \times k$ is j. Let $E_{\leq j} \overset{\text{def}}{=} \bigcup_{i \leq j} E_i$ denote the set of hyperedges where all auxiliary edges have label $\leq j$, and let $E_{\geq j} \overset{\text{def}}{=} \bigcup_{i \geq j} E_i$ denote the set of hyperedges with at least one auxiliary edge of label $\geq j$.

Invariants. T, B, and ℓ are bound together by the following invariants.

(I) $\ell(a^{(i)}) = 0$ for any auxiliary edge $a^{(i)} \in B \setminus T^{(i)}$.

(II) $T_{\leq j}^{(i)}$ spans $A_{\leq j-1}^{(i)}$ for all j.

(III) $B_{\geq j}$ spans $A_{\geq j+1}$ in the partition matroid for all j.

Basic operations. The high-level goal is to either eliminate all active auxiliary edges or increase their label to h, for a given parameter $h \in \mathbb{N}$. We do so by iteratively executing one of three operations. Given an active auxiliary edge $a^{(i)} \in A_e \times k$ with label $\ell(a^{(i)}) = j < h$, these operations are:

1. *Push:* Exchange $a^{(i)}$ into B for some $b \in B_{j-1} \cap (A_e \times k)$.
2. *Pull:* Exchange $a^{(i)}$ out of $T^{(i)}$ for some $b^{(i)} \in A_{j-1}^{(i)} \setminus T^{(i)}$ such that $T^{(i)} - a + b$ is a forest.
3. *Relabel:* Increase $\ell(a)$ by 1.

One can push an active $a^{(i)}$ into B iff $a^{(i)} \notin \operatorname{span}_{\text{P}}(B_{\geq j})$. One can pull $a^{(i)}$ out of $T^{(i)}$ iff $T_{\leq j}^{(i)} - a^{(i)}$ does not span $A_{\leq j-1}^{(i)}$. To preserve invariants (II) and (III), one can relabel $a^{(i)}$ iff no push or pull operation is available.

In addition to the operations above, we periodically remove hyperedges from the hypergraph. For this we introduce one more operation.

4. *Delete:* Remove a hyperedge e from the hypergraph and $A_e \times k$ from the push-relabel system.

We now describe how to remove a hyperedge e without violating invariants (I)–(III). We first set $w(e) = 0$, and remove any arcs from $A_e \times k$ from B. This excludes e and its auxiliary edges from consideration for invariants (I) and (III), and we focus on invariant (II). We repeatedly try to pull or (otherwise) relabel any active $a^{(i)} \in A_e \times k$ until $\ell(a^{(i)}) = h + 2$ for all active $a^{(i)} \in A_e \times k$. (We briefly allow labels to exceed h, and only for $A_e \times k$.) The pull and relabel operations preserve invariant (II). The auxiliary edges not in any $T^{(i)}$ can be removed from the system without any violation. Since there are no auxiliary edges with label $h + 1$, the remaining auxiliary edges with label $h + 2$ can now be removed from the system without any violation of invariant (II).

Counting the basic operations. We account for the total number of push, relabel, and pull operations including those prompted by deletion.

There are at most $O(pkh)$ relabel operations since each auxiliary edge $a^{(i)}$ can be relabeled $O(h)$ times. There are at most $O(pkh)$ push operations because each push increases $|B_{\leq j}|$ for some index $j \leq O(h)$, and each $|B_{\leq j}|$ is monotonically increasing from 0 to at most pk. To bound the number of pull operations, we define a potential Φ by

$$\Phi = \sum_{i=1}^{k} \sum_{a^{(i)} \in T^{(i)}} \ell(a^{(i)}).$$

Φ is nonnegative and initially Φ is 0. Each relabel increases Φ by at most 1. Pushing does not effect Φ. Removing an auxiliary edge (at the end of a deletion) does not increase Φ. Pulling decreases Φ by 1. Since the total increase to Φ is $O(pkh)$, the total number of pulls is $O(pkh)$. In summary, we have $O(pkh)$ push, pull, and relabel operations.

As for deletions, of course, each hyperedge can be deleted at most once.

Implementation details and running time bounds. Now we specify more concrete implementation details and obtain real running time bounds. The algorithm repeatedly selects an active auxiliary edge $a^{(i)} \in A_e \times k$ and tries to either push $a^{(i)}$ into B or pull $a^{(i)}$ out of $T^{(i)}$. If $a^{(i)}$ cannot be pushed or pulled, then we relabel $a^{(i)}$. It is important that we try to push an auxiliary edge $a^{(i)}$ before pulling. It does not matter which active $a^{(i)}$ is selected as long as $\ell(a^{(i)}) < h$. The algorithm stops when $\ell(a^{(i)}) = h$ for all active $a^{(i)}$.

It is easy to keep track of the active auxiliary edges $a^{(i)}$ with label $\ell(a^{(i)}) < h$, so that we can either select such an a or decide that none are available in $O(1)$ time. For each hyperedge e, it is easy to track which of its auxiliary edges are in B and at which label, so that given an active $a^{(i)}$, we can check for and execute a push in $O(1)$ time. These are also easy to maintain as $a^{(i)}$'s are relabeled. The remaining challenge is the pull operation. Given $a^{(i)} \in T^{(i)}$, we need to quickly decide if it is possible to pull $a^{(i)}$ out of $T^{(i)}$ for some $b^{(i)}$, and if so, execute the exchange.

We facilitate these operations by maintaining each $T^{(i)}$ in a data structure that maintains the minimum weight spanning forest (MSF) in a dynamic graph. We appeal to the following bounds from [25].

Fact B.1. *A minimum weight spanning forest can be maintained in a dynamically changing graph in $O(\log^4 n)$ amortized time per edge insertion or deletion.*

Fix $i \in [k]$. At a high-level, each $T^{(i)}$ is maintained as an MSF over $A^{(i)}$ where the edge weights are generally set to the labels of the auxiliary edges.[7] The only exception is that edges in the MSF have their weight reduced to -1, which forces the edge to stay in the tree until we decide to remove it at a later point. (An edge weight update translates to a deletion and an insertion.)

Initially all the labels, hence all the weights, are 0. As $T^{(i)}$ acquires edges initially, we set the weight of each added edge to -1. When an auxiliary edge $a^{(i)}$ is relabeled from j to $j + 1$, since $a^{(i)}$ is active, we have $a^{(i)} \in T^{(i)}$, and the edge weight of $a^{(i)}$ does not need to be updated.

Given an active hyperedge $a^{(i)} \in T_j^{(i)}$, to test if $a^{(i)}$ can be pulled out of $T^{(i)}$ for some $b \in T_{j-1}^{(i)}$, we increase the weight of $a^{(i)}$ from -1 to $j - 1/2$. If $a^{(i)}$ is replaced by another edge b, then that edge b is necessarily at label $j - 1$ because $\ell(b) \leq j - 1/2$ and $T_j^{(i)} - a^{(i)}$ spans $A_{\leq j-2}^{(i)}$. We pull $a^{(i)}$ out of $T^{(i)}$ in exchange for b by setting the weight of b to -1 and the weight of a to j. If $a^{(i)}$ is not replaced by another edge, then $a^{(i)}$ cannot be pulled out for any $b \in A_{j-1}^{(i)}$. We restore the weight of $a^{(i)}$ to -1.

[7]These edge weights over $A^{(i)}$, for the data structure maintaining $T^{(i)}$, should not to be confused with the input weights $w(e)$ for each hyperedge e.

Recall that each edge update takes $O(\log^4 n)$ amortized time. It takes $O(pk)$ edge updates to initialize all $T^{(i)}$'s. It takes $O(1)$ edge updates to test if we can pull an $a^{(i)}$ out of $T^{(i)}$, and also to execute the pull. If we end up pulling $a^{(i)}$, we can charge this edge update to the subsequent pull. If not, then the algorithm will promptly relabel $a^{(i)}$, and the edge updates from testing the pull can be charged to the relabel. Altogether, given the upper bounds for pull and relabel operations above, we have a total of $O(pkh)$ edge updates, and spend a total of $O(pkh \log^4 n)$ time, managing the $T^{(i)}$'s over pull and relabel operations.

In summary, the total running time is $O(pkh \log^4 n)$.

Certifying strength k or extracting a low-strength-ratio set. As mentioned above, we run the push-relabel algorithm until all active auxiliary edges have label h. The following lemma states that when the push-relabel algorithm stops, we will have either certified that the hypergraph has strength k, or identified a smaller set of edges of the form $E_{\leq j}$ that we can restrict our attention to.

Lemma B.3. *Suppose all active auxiliary edges have label $h = O(\log(n)/\epsilon)$. Either:*

(i) There are no active auxiliary edges and T certifies that the current hypergraph has strength k.

(ii) There exists an index $j \in \{2, \dots, h-1\}$ such that $w(E_{\geq j+1}) \leq (1+\epsilon)w(E_{\geq j-1})$, in which case $f(E_{\leq j}) < f(E)$ and $\varphi(E_{>j}) \leq (1+\epsilon)k$.

Proof. We have already discussed how the lack of active auxiliary edges implies that T certifies that H has strength k. Suppose there is at least one active auxiliary edge. This implies that E_h is nonempty, and $w(E_h) \geq 1$.

Let $j \geq 1$, and consider any hyperedge $e \in E_{\geq j}$. Then there is some $a^{(i)} \in A_e \times k$ with label $\ell(a^{(i)}) \geq j$. When $a^{(i)}$ was previously relabeled from $j-1$ to j, since no push was available, B_{j-2} had no other auxiliary edges from e, and all $w(e)$ of the auxiliary edges from e in B were in $B_{\geq j-1}$. Since $|B_{\geq j-1} \cap (A_e \times k)|$ never decreases over time, $B_{\geq j-1}$ currently has $w(e)$ auxiliary edges from e as well. Summing over all $e \in E_{\geq j}$, we have $w(E_{\geq j}) \leq |B_{\geq j-1}|$.

We also have, for all indices $j \in \mathbb{Z}_{\geq 0}$ and $i \in [k]$,

$$f(E_{\leq j}) = \text{rank}_{\text{G}}\left(\bigcup_{e \in E_{\leq j}} A_e^{(i)}\right) \overset{\text{(a)}}{\leq} \text{rank}_{\text{G}}\big(A_{\leq j}^{(i)}\big) \overset{\text{(b)}}{\leq} \big|T_{\leq j+1}^{(i)}\big|.$$

Here (a) is because all auxiliary edges of any $e \in E_{\leq j}$ have label at most j (by definition of $E_{\leq j}$). (b) is because $T_{\leq j+1}^{(i)}$ spans $A_{\leq j}^{(i)}$ (by invariant (II)).

Now, since $h \geq c\log(pk)/\epsilon \geq c\log(w(E))/\epsilon$ for a sufficiently large constant c, and $w(E_h) \geq 1$, there exists an index j with $2 \leq j \leq h-1$ such that

$$w(E_{\geq j+1}) \leq (1+\epsilon)w(E_{\leq j-1}).$$

For this choice of index j, we have

$$\begin{aligned}(1+\epsilon)kf(E_{\leq j}) + w(E_{\geq j+1}) &\leq (1+\epsilon)(kf(E_{\leq j}) + w(E_{\geq j-1})) \\ &\leq (1+\epsilon)\left(\sum_{i=1}^{k}\big|T_{\leq j+1}^{(i)}\big| + |B_{\geq j}|\right) \\ &= (1+\epsilon)\left(\sum_{i=1}^{k}\big|T^{(i)} \cap B\big|\right) \\ &\leq (1+\epsilon)kf(E).\end{aligned}$$

Rearranging, we have

$$w(E) - w(E_{\leq j}) = w(E_{\geq j+1}) \leq (1+\epsilon)k(f(E) - f(E_{\leq j})).$$

Since the LHS is nonzero, this first implies that $f(E_{\leq j}) < f(E)$. Dividing both sides by $f(E) - f(E_{\leq j})$ gives $\varphi(E_{\leq j}) \leq (1+\epsilon)k$, as desired. ∎

```
1. Initialize the push-relabel algorithm.
2. Repeatedly:
   A. Continue the push-relabel algorithm until all active auxiliary edges have label $h = O(\log(m)/\epsilon)$.
   B. If there are no active auxiliary edges, then return the span of the set of all remaining hyperedges.
   C. Otherwise let $j \leq h - 2$ be an index such that $w(E_{\geq j+1}) \leq (1+\epsilon)w(E_{\geq j-1})$. Remove all hyperedges in
      $E_{>j}$ from the system.
```

Figure 4: Computing a subset of hyperedges with strength k (from the proof of lemma 5.3).

Completing the proof of lemma 5.3. We are now prepared to complete the proof of lemma 5.3. The algorithm we analyze is given in fig. 4 and we first give a high-level description. Recall that one goal is to find a subset of edges with strength at least k. To this end, we initialize the push-relabel algorithm and run it until all active auxiliary edges have label h. If there are no active edges, then we have certified the hypergraph has strength k, as desired. Otherwise, by lemma B.3 we identify an index j such that $E_{\leq j}$ has $\varphi(E_{\leq j}) \leq (1+\epsilon)k$. We restrict out attention to these edges by deleting $E_{\geq j+1}$. Then we continue the push-relabel algorithm and repeat the above steps, until arriving at a subset of hyperedges with strength k. (This includes the empty set, which has strength $+\infty$.)

Let S_{OUT} be the set output by the algorithm and suppose the algorithm runs for $\ell + 1$ iterations. Let $(S_0 = E) \supsetneq S_1 \supsetneq \cdots \supsetneq S_\ell$ be the descending sequence of sets S at the beginning of each iteration. As mentioned above, the algorithm certifies $S_{\text{OUT}} = \text{span}(S_\ell)$ has strength k before returning it. Now suppose $S_{\text{OUT}} \neq E$. We need to show that S_{OUT} has strength-ratio $\varphi(S_{\text{OUT}} \mid E) \leq (1+\epsilon)k$.

For each $i \in [\ell]$, by lemma B.3, we have

$$\varphi(S_i \mid S_{i-1}) = \frac{w(S_i) - w(S_{i-1})}{f(S_i) - f(S_{i-1})} \leq (1+\epsilon)k.$$

Consequently

$$\begin{aligned} w(E) - w(S_{\text{OUT}}) &= \sum_{i=1}^{\ell} w(S_{i-1}) - w(S_i) \leq (1+\epsilon)k \sum_{i=1}^{\ell} f(S_{i-1}) - f(S_i) \\ &= (1+\epsilon)k(f(E) - f(S_\ell)) \\ &= (1+\epsilon)k(f(E) - f(S_{\text{OUT}})), \end{aligned}$$

hence $\varphi(S_{\text{OUT}} \mid E) \leq (1+\epsilon)\ell$.

Finally we discuss the running time. The total time spent on the push-relabel algorithm is $O(pkh\log^4 n) = O(pk\log(pk)\log^4(n)/\epsilon)$. It is easy to track each difference $w(E_{\geq j+1}) - (1+\epsilon)w(E_{\geq j-1})$ for all j with $O(1)$ overhead per relabel. Consequently we can also identify the index j in step (2.C) in $O(1)$ time.

Lastly, computing the span $S_{\text{OUT}} = \text{span}(S_\ell)$ of the final set of edges S_ℓ can be done in nearly time as follows. First, we build a disjoint union data structure over V labeling the connected components of S_ℓ. For every other edge e, $e \in \text{span}(S_\ell)$ iff all endpoints of e are in the same component. This can be realized by querying the disjoint union data structure $|e| - 1$ pairs of endpoints of e. Altogether, computing the span takes $O(p\alpha(n))$ time, where $\alpha(\cdots)$ is the inverse Ackerman function.

Overall we have a running time of $O(pk\log(pk)\log^4(n)/\epsilon)$. This completes the proof of lemma 5.3.

C Computing strong and low strength-ratio sets for coverage: proof of lemma 6.3

In this section we describe and analyze an algorithm which fulfills the requirements of lemma 6.3. The input consists of a set system $(\mathcal{U}, \mathcal{F})$, positive weights $w : \mathcal{U} \to \mathbb{R}_{>0}$ over the points, and parameters $k \in \mathbb{N}$ and $\epsilon \in (0,1)$. Let

$$f(X) = |\{S \in \mathcal{F} : S \cap X \neq \emptyset\}|,$$

for $X \subseteq \mathcal{U}$, be the hitting set function of $(\mathcal{U}, \mathcal{F})$. We may assume without loss of generality that every set is nonempty. Then f has rank $r = f(\mathcal{U}) = |\mathcal{F}|$.

The goal of lemma 6.3 is to either certify that $\mathcal{U}$ has strength (at least) k (with respect to f), or compute a subset $\mathcal{U}' \subsetneq \mathcal{U}$ with strength k and strength-ratio $\varphi(\mathcal{U}' \,|\, \mathcal{U}) \leq (1+\epsilon)k$. Letting $N = \sum_{S \in \mathcal{F}} |S|$ denote the total size of the set system, and $W = \sum_{p \in \mathcal{U}} w(p)$, we seek a $O\big(N \log^2(W)/\epsilon^2\big)$ running time.

The algorithm is motivated by the following connection between the strength of the hitting set system and matchings between the points and sets.

Lemma C.1. *The hitting set system $(\mathcal{U}, \mathcal{F})$ has strength at least λ iff there is a fractional matching between points $\mathcal{U}$ and sets $S \in \mathcal{F}$ such that:*
- *(a) Points $p \in \mathcal{U}$ are only matched to sets $S \in \mathcal{F}$ where $p \in S$.*
- *(b) Each point $p \in \mathcal{U}$ is matched at most $w(p)$ times.*
- *(c) Each set $S \in \mathcal{F}$ is matched exactly λ times.*

Proof. Hall's matching theorem states that a matching satisfying properties (a)–(c) exists iff for all $\mathcal{F}' \subseteq \mathcal{F}$, letting Q denote the set of points hitting $\mathcal{F}'$, we have

$$\lambda |\mathcal{F}'| \leq w(Q).$$

By lemma 6.1, this condition is equivalent to the inequality above holding for all quotients Q of f. Since $|\mathcal{F}'| = |\mathcal{F}| - f(\bar{Q})$, this is also the definition of $\mathcal{U}$ having strength $\geq \lambda$. □

The high-level approach is to apply push-relabel flow techniques (from [22, 23, 21]) to the matching problem described in lemma C.1. We first describe the general setup.

We have a directed bipartite graph with vertices $V = \mathcal{F} \cup \mathcal{U}$ and edges E directed from each set to the points in the set:

$$E = \{(S, p) : S \in \mathcal{F},\, p \in S\}.$$

Note that $|V|/2 \leq |E| = N$ where N is the total size of the set system. Each edge $e \in E$ has infinite capacity. Each set $S \in \mathcal{F}$ is a source of size λ, and each point $p \in \mathcal{U}$ is a sink of size $w(p)$. We let $x : E \to \mathbb{R}_{\geq 0}$ denote the flow vector; x can also be interpreted as a fractional bipartite matching. We let E_x denote the set of edges (with positive capacity) in the residual graph of x. The *excess* at a set S is defined as λ minus the total flow leaving S. The excess at a point p is defined as the total flow entering p minus $w(p)$. A vertex v is *active* if it has positive excess. We say x is a *feasible matching* if there are no active vertices.

Labels and invariants. In addition to a flow x, the push-relabel framework maintains integer *vertex labels* $\ell : V \to \mathbb{Z}_{\geq 0}$ subject to the following invariants:
- (I) For all $v \in V$, if excess$(v) < 0$, then $\ell(v) = 0$.
- (II) For all $(u, v) \in E_x$, $\ell(v) \geq \ell(u) - 1$.

A typical push-relabel algorithm works to ensure that $\ell(v) \geq |V|$. (Then there is an empty label that induces a certifying cut.) Instead we introduce a parameter $h \in \mathbb{N}$, called the *height*, and stop the algorithm whenever all active vertices have label $\geq h$.

Push-relabel operations. The generic push-relabel algorithm iteratively selects an active vertex v and executes one of the following operations:
1. *Push:* Push flow down from u to v along an edge $(u, v) \in E_x$ such that $\ell(u) = \ell(v) + 1$.
2. *Relabel:* Increase the label $\ell(v)$ by 1.

In order to maintain invariant (II), a vertex v can only be relabeled if it is impossible to push from v.

Removing points. For our setting, we also remove (vertices corresponding to) points $p \in \mathcal{U}$ from the flow network. For this we introduce a third operation:
3. *Remove:* Remove a point $p \in \mathcal{U}$ from the set system and the flow network, along with any set S that becomes empty.

We remove a point $p \in \mathcal{U}$ from the flow network by the following steps. First, we set the flow along any edge incident to p to 0. Then we remove p and all edges incident to p from the graph. Removing a point may increase the excess of sets containing p. This does not violate either invariant. Once p is removed from the flow network, any set S that was previously of the form $S = \{p\}$ is now an isolated vertex that can an also be removed without violating either invariant.

By removing any set that becomes empty, we maintain the property that all sets are nonempty.

The push-relabel algorithm. There are several different variations of the push-relabel algorithm, all following the general format of repeatedly selecting an active vertex v and either pushing from or relabeling v. For our task, the algorithm is allowed to select any active vertex v with label $\ell(v) < h$. When pushing flow from an active vertex v to a vertex w, the algorithm always pushes as much flow as possible, which is the minimum of the excess at v and the residual capacity of the edge (v, w). The push is called a *saturating push* if it uses all the capacity of the edge and removes the edge from the residual graph. Otherwise the push is called a *non-saturating push.* When there are no active vertices v with label $\ell(v) < h$, the algorithm stops.

Counting the basic operations. Observe that the vertex labels never decrease, and the maximum label is h. Therefore there are at most h relabel operations per vertex.

We claim there are at most h saturating pushes on each edge (v, w). Indeed, a saturating push on (v, w) removes (v, w) from the residual graph, and (v, w) can only reappear after a push along the reverse edge (w, v). Pushing along (v, w) requires $\ell(v) = \ell(w) + 1$ while pushing along (w, v) requires $\ell(w) = \ell(v) + \ell(1)$, so for the latter to occur, $\ell(w)$ must have increased by at least 2. Thus a saturating push (v, w) can be charged to the increase in $\ell(w)$ by at least 2.

Thus, there are at most $O(|V|h)$ relabel operations and $O(|E|h)$ saturating pushes. It remains to account for the non-saturating pushes. We define a potential function Φ summing the labels of all active vertices:

$$\Phi \overset{\text{def}}{=} \sum_{\text{active } v} \ell(v).$$

Φ is always nonnegative, and initially $\Phi = 0$ because all vertices have label 0. Each saturating push increase Φ by at most h, and each relabel increases Φ by 1. Removing a point p increases Φ by at most h for each set S containing p. We charge each per-set increase to the deletion of the edge (S, p). Between these three operations, the total increase to Φ is $O\big(|E|h^2\big)$. Meanwhile, each non-saturating push from (say) v to w decreases Φ by at least 1 because the push makes v inactive, and $\ell(w) = \ell(v) - 1$. Thus the total number of non-saturating pushes is bounded by the total increase to Φ, $O\big(|E|h^2\big)$.

Running time bounds. Now we translate the operation bounds to proper running times. It is easy to maintain a list of active vertices with label $< h$, with $O(1)$ overhead as vertices are activated, deactivated, and deleted. For each vertex v, and each label i, we maintain a doubly-linked list of residual edges $(v, w) \in E_x$ with $\ell(w) = i$. With reverse pointers, one can maintain this list in $O(1)$ time whenever an edge is added to or removed from E_x, or as the endpoint w has its label increase. With these data structures in place, we can identify an active vertex v, and identify and execute a basic operation on v, in constant time. Altogether, the total running time is bounded by the total number of non-saturating pushes: $O\big(|E|h^2\big)$.

Certifying strength k or extracting low strength-ratio sets. Suppose we run the push-relabel algorithm until all active vertices have label $h = O(\log(m)/\epsilon)$. The following lemma states that when the algorithm stops, we will have either certified that the set system has strength k, or identified a set of points that we can discard.

For an index $i \in \{0, \dots, h\}$, we let $\mathcal{U}_i \overset{\text{def}}{=} \mathcal{U} \cap \ell^{-1}(i)$ denote the set of (remaining) points with label i. We let $\mathcal{U}_{\leq i} \overset{\text{def}}{=} \bigcup_{j \leq i} \mathcal{U}_j$ denote the set of points with label at most j, and similarly let $\mathcal{U}_{\geq j} \overset{\text{def}}{=} \bigcup_{j \geq i} \mathcal{U}_j$.

LEMMA C.2. *Let $\epsilon \in (0, 1)$, and suppose all active vertices have label h for $h = O(\log(m)/\epsilon)$. Then either:*

(i) x is a feasible matching, certifying that the hitting set system has strength at least λ.

(ii) There exists an index $i \leq h - 2$ such that $w(\mathcal{U}_{\geq i+1}) \leq (1 + \epsilon) w(\mathcal{U}_{\geq i+3})$, in which case $f(\mathcal{U}_{\leq i}) < r$ and $\varphi(\mathcal{U}_{\leq i}) \leq (1 + \epsilon)\lambda$.

Proof. For a set of points $\mathcal{U}' \subseteq \mathcal{U}$, let

$$\hat{x}(\mathcal{U}') \overset{\text{def}}{=} \sum_{p \in \mathcal{U}'} \sum_{\substack{S \in \mathcal{F} \\ p \in S}} x(S, p)$$

denote the total fractional quantity of sets matched to points in $\mathcal{U}'$.

Suppose x is not a feasible matching. We claim there is at least one point p with label $\geq h-1$.

To prove the claim, we first observe that there is at least one active vertex with label h because x is not a feasible matching. If this active vertex is a point then this satisfies the claim. Otherwise the active vertex corresponds to a set S. S is nonempty, and any point $p \in S$ has label $\ell(p) \geq \ell(S) - 1 \geq \ell(h) - 1$, as claimed.

Since $h \geq c \log(W)/\epsilon$ for a sufficiently large constant c, $w(\mathcal{U}_{\geq h-1}) \geq 1$, and $w(\mathcal{U}_{\geq 0}) = w(\mathcal{U}) \leq W$, there must be an index $i \leq h-2$ such that $w(\mathcal{U}_{\geq i+1}) \leq (1+\epsilon) w(\mathcal{U}_{\geq i+3})$ Then

$$w(\mathcal{U}_{\geq i+1}) \leq (1+\epsilon) w(\mathcal{U}_{\geq i+3}) \overset{\text{(a)}}{\leq} (1+\epsilon)\hat{x}(\mathcal{U}_{\geq i+3}),$$

where (a) observes that any point p with nonzero label is fractionally matched by at least $w(p)$ edges (invariant (I)).

We also have $f(\mathcal{U}_{\leq i}) \leq |\mathcal{F}_{\leq i+1}|$ since all sets hit by $\mathcal{U}_{\leq i}$ have label at most $i+1$ (invariant (II)). We then have $\lambda |\mathcal{F}_{\leq i+1}| \leq \hat{x}(\mathcal{U}_{\leq i+2})$ because each set in $\mathcal{F}_{\leq i+1}$ is inactive and fractionally matched λ times to points in $\mathcal{U}_{\leq i+2}$ (invariant (II)). Putting everything together, we have

$$w(\mathcal{U}_{\geq i+1}) + (1+\epsilon)\lambda f(\mathcal{U}_{\leq i}) \leq (1+\epsilon)(\hat{x}(\mathcal{U}_{\geq i+3}) + \hat{x}(\mathcal{U}_{\leq i+2})) = (1+\epsilon)\hat{x}(\mathcal{U}) \leq (1+\epsilon)\lambda r.$$

Rearranging we have

$$w(\mathcal{U}_{\geq i+1}) \leq (1+\epsilon)\lambda(r - f(\mathcal{U}_{\leq i})).$$

Since $\mathcal{U}_{\geq i+1} \neq 0$, this first implies that $f(\mathcal{U}_{\leq i}) < r$. Dividing both sides by $r - f(\mathcal{U}_{\leq i})$ gives $\varphi(\mathcal{U}_{\leq i}) \leq (1+\epsilon)\lambda$, as desired. ∎

Proof of lemma 6.3. We are now prepared to prove lemma 6.3. The high-level idea of the algorithm is as follows. We initialize the push-relabel algorithm as described above and run the push-relabel algorithm until all active vertices have label h. If x is a feasible matching, then this certifies that the hitting set system has strength λ. If not, then by lemma C.2, we can identify an index $i \leq h-2$ such that $\varphi(\mathcal{U}_{\leq i}) \leq (1+\epsilon)\lambda$. We remove all points in $\mathcal{U}_{>i}$, and then repeat, continuing the push-relabel algorithm from the current configuration. The algorithm terminates when it obtains a feasible matching x for the remaining hitting set system. (This includes the empty matching when $\mathcal{U} = \emptyset$.) See fig. 5 on the next page for pseudocode.

Let $\mathcal{U}_{\text{OUT}}$ be the set of points returned by the algorithm. The feasible matching x at termination certifies that $\mathcal{U}_{\text{OUT}}$ has strength at least λ.

Suppose $\mathcal{U}_{\text{OUT}} \neq \mathcal{U}$. Then the algorithm runs for $k+1$ iterations for some $k \in \mathbb{Z}_{\geq 0}$. Let $(\mathcal{U}_0 = \mathcal{U}) \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \cdots \supseteq (\mathcal{U}_k = \mathcal{U}_{\text{OUT}})$ be the descending sequence of point sets that the set system is restricted to in each iteration. For each $i \in [k]$, we have

$$\varphi(\mathcal{U}_i \,|\, \mathcal{U}_{i-1}) = \frac{w(\mathcal{U}_{i-1}) - w(\mathcal{U}_i)}{f(\mathcal{U}_{i-1}) - f(\mathcal{U}_i)} \leq (1+\epsilon)\lambda$$

by lemma C.2. Consequently

$$\begin{aligned} w(\mathcal{U}) - w(\mathcal{U}_{\text{OUT}}) &= \sum_{i=1}^{k} w(\mathcal{U}_{i-1}) - w(\mathcal{U}_i) \leq (1+\epsilon)\lambda \sum_{i=1}^{k} f(\mathcal{U}_{i-1}) - f(\mathcal{U}_i) \\ &= (1+\epsilon)\lambda(f(\mathcal{U}) - f(\mathcal{U}_{\text{OUT}})), \end{aligned}$$

hence $\varphi(\mathcal{U}_{\text{OUT}} \,|\, \mathcal{U}) \leq (1+\epsilon)\lambda$.

It remains to address the running time. As mentioned above, the total time spent on push-relabel operations, and on removing vertices, is $O(|E|h^2) = O(N \log^2(W)/\epsilon^2)$. To identify the index i in step (`2.C`) efficiently, it suffices to keep track of $w(\mathcal{U}_{\leq i})$ as points are relabeled, and maintain a list of the indices i where $w(\mathcal{U}_{\leq i+2}) - w(\mathcal{U}_{\leq i}) \leq \epsilon w(\mathcal{U})$. It is easy to do so with negligible overhead.

This completes the proof of lemma 6.3.

Remark C.1. *The quadratic dependency on* $\log(W)/\epsilon$ *in lemma 6.3 can be reduced by adapting the scaling approach of [1]. For ease of exposition we presented what we believed to be the simplest algorithm for the task. (Besides, for the overall sparsification algorithm, this speedup is not compatible with the alternative speedup described in remark 6.1.)*

```
1. Initialize the push-relabel algorithm.
2. Repeatedly:
   A. Continue the push-relabel algorithm until all active vertices have label $h$.
   B. If $x$ is a feasible matching, then return the set of all remaining points.
   C. Otherwise let $i \in [h-2]$ such that $w(\mathcal{U}_{\leq i+2}) \leq w(\mathcal{U}_{\leq i}) + \epsilon w(\mathcal{U})$. Remove all of $\mathcal{U}_{>i}$ (and all sets
      contained in $\mathcal{U}_{>i}$) from the flow network.
```

Figure 5: Computing a set with strength-ratio at most $(1+\epsilon)\lambda$ and strength at least λ (from the proof of lemma 6.3).

D Extending to general weights

Elsewhere in this article we have described nearly linear time algorithms sparsifying quotients of a function f for polynomially bounded weights. Here we give a nearly black-box reduction from general weights to the polynomially-bounded setting.

For fixed $f : \mathcal{N} \to \mathbb{R}_{\geq 0}$, we assume that we can sparsify, for any disjoint $S, T \subseteq \mathcal{N}$, the function $f_{T|S}$ obtained by contracting T and restricting to S:

$$f_{T|S}(A) = f(A \cup T) - f(T) \text{ for } A \subseteq S.$$

We point out that for each of our concrete classes (matroids, hypergraphs, and coverage), $f_{T|S}$ gives another instance of that class. We assume that, given $f_{T|S}$, where the weights of S lie in a polynomially bounded range, there is a randomized algorithm that produces, with high probability, a randomized set of weights $\tilde{w} : S \to \mathbb{R}_{\geq 0}$ that:

(a) Preserves the weight of every quotient of $f_{T|S}$ up to a $(1+\epsilon)$-factor.
(b) Has $\textsc{Nonzeroes}(f(S \,|\, T))$ nonzeroes for some function $\textsc{Nonzeroes}(x)$.
(c) Takes $\textsc{Time}(S)$ randomized time for some set function $\textsc{Time}(X)$.

We assume that $\textsc{Nonzeroes}$ is superadditive: $\textsc{Nonzeroes}(x) + \textsc{Nonzeroes}(y) \leq \textsc{Nonzeroes}(x+y)$ for $x, y > 0$. We also assume that $\textsc{Time}$ is superadditive over disjoint sets: $\textsc{Time}(A) + \textsc{Time}(B) \leq \textsc{Time}(A \cup B)$ for disjoint $A, B \subseteq \mathcal{N}$. Below we define $\textsc{Nonzeroes}$ and $\textsc{Time}$ for the 4 main examples in this article.

- For normalized monotone submodular f, we have $\textsc{Nonzeroes}(x) = O\big(x \log(nr)/\epsilon^2\big)$ and $\textsc{Time}(S) = O\Big(|S|^{O(1)}\Big)$.
- For matroids, we have $\textsc{Nonzeroes}(x) = O\big(x \log(n)/\epsilon^2\big)$ and $\textsc{Time}(S) = \tilde{O}(|S| Q_{\text{RANK}})$.
- For hypergraphs with n vertices, we have $\textsc{Nonzeroes}(x) = O\big(x \log(n)/\epsilon^2\big)$ and $\textsc{Time}(S) = \tilde{O}(p_S)$, where $p_S = \sum_{e \in S} |e|$.
- For coverage with n sets, we have $\textsc{Nonzeroes}(x) = O\big(x \log(n)/\epsilon^2\big)$ and $\textsc{Time}(S) = \tilde{O}(N_S)$ where $N_S = \sum_{e \in S} f(e)$.

Given this setup, we describe a randomized algorithm that produces randomized weights $\tilde{w} : \mathcal{N} \to \mathbb{R}_{\geq 0}$ that with high probability, (a) preserve the weight of every quotient of f up to a $(1+\epsilon)$-factor, (b) has $\textsc{Nonzeroes}(r)$ nonzeroes, and (c) takes $\textsc{Time}(\mathcal{N})$ randomized time (aside from some simple preprocessing that is unlikely to be a bottleneck for most applications).

The high-level idea is very simple. We partition the elements by weight, so that the weights within each group of elements lie within a polynomial factor, while the ranges of the weights of the groups are disjoint and a polynomial factor apart from each other. We run the polynomially bounded sparsification routine on each group separately and combine the randomized weights for each group.

The algorithm. For ease of notation, we fix $\epsilon > 0$ and describe an algorithm that preserves all quotients up to a $(1+c\epsilon)$-factor for a constant $c > 0$; a proper $(1+\epsilon)$-approximation follows from decreasing ϵ by a constant factor.

For each element e, let $W_e \overset{\text{def}}{=} [w(e), w(e)n/\epsilon]$ be an interval representing an (n/ϵ)-factor window starting at

$w(e)$. Let $W \overset{\text{def}}{=} \bigcup_e W_e$ be the union of these windows, and let $W_1, \ldots, W_h$ be the maximal closed intervals of W listed in increasing order. That is, W is the disjoint union of $W_1, \ldots, W_h$. Each W_i is contained in a $O(n^2/\epsilon)$-factor range.

For each $i \in [h]$, let $\mathcal{N}_i \overset{\text{def}}{=} \{e : w(e) \in W_i\}$ be the set of elements whose weight falls in the window W_i. For $i < j$,

$$\max_{e \in \mathcal{N}_i} w(e) \leq \frac{\epsilon}{n} \min_{e \in \mathcal{N}_j} w(e),$$

so the weights of elements in different windows are an (n/ϵ)-factor apart. We denote $\mathcal{N}_{\leq i} \overset{\text{def}}{=} \bigcup_{j \leq i} \mathcal{N}_j$ for $i \in \mathbb{Z}_{\geq 0}$. Let $n_i \overset{\text{def}}{=} |\mathcal{N}_i|$ for $i \in [h]$.

For $i \in [h]$, let $f_i \overset{\text{def}}{=} f_{\mathcal{N}_{\geq i+1} | \mathcal{N}_i}$ be the function obtained by contracting $\mathcal{N}_{\geq i+1}$ and restricting to $\mathcal{N}_i$. f_i has a ground set of n_i elements and rank $f(\mathcal{N}_i \,|\, \mathcal{N}_{\geq i+1})$.

For each f_i, we run the sparsification algorithm on f_i, producing a randomized set of weights $\tilde{w}_i : \mathcal{N}_i \to \mathbb{R}_{\geq 0}$. Let $\hat{w}$ combine the $\tilde{w}_i$'s into a set of weights over $\mathcal{N}$; i.e.,

$$\hat{w}(e) \overset{\text{def}}{=} \tilde{w}_i(e) \text{ for } e \in \mathcal{N}_i.$$

We return $\hat{w}$.

Correctness. For each i, with high probability, $\tilde{w}_i$ has $\textsc{Nonzeroes}(\mathcal{N}_i)$ nonzeroes, and preserves all quotients of f_i up to a $(1+\epsilon)$-factor. By the union bound (and noting that $h \leq n$), this holds for all $i \in [h]$ with high probability. Henceforth we assume that this is the case.

We first bound the size of the support of $\hat{w}$. We have

$$\sum_{i=1}^{h} \textsc{Nonzeroes}(f(\mathcal{N}_i \,|\, \mathcal{N}_{\geq i+1})) \overset{\text{(a)}}{\leq} \textsc{Nonzeroes}\left(\sum_{i=1}^{h} f(\mathcal{N}_i \,|\, \mathcal{N}_{\geq i+1})\right) \overset{\text{(b)}}{\leq} \textsc{Nonzeroes}(r)$$

by (a) superadditivity and (b) telescoping series.

Next we verify that $\hat{w}$ approximates the weight of all quotients. Consider any quotient Q of f. We want to show that $\hat{w}(Q)$ approximates $w(Q)$. Let e be the maximum weight element in Q, and suppose $e \in \mathcal{N}_i$. Observe that

$$w(\mathcal{N}_{\leq i-1}) \leq \left(\frac{\epsilon w(e)}{n}\right) \cdot n = \epsilon w(e) \leq \epsilon w(Q).$$

In particular,

$$w(Q \cap \mathcal{N}_i) \geq w(Q) - w(\mathcal{N}_{\leq i}) \geq (1-\epsilon) w(Q),$$

so $Q \cap \mathcal{N}_i$ represents almost all of the weight of Q.

By lemma 2.1, $Q \cap \mathcal{N}_{\geq i} = Q \cap \mathcal{N}_i$ is a quotient in (the function restricted to) $\mathcal{N}_{\geq i}$. Since $Q \cap \mathcal{N}_{\geq i}$ is disjoint from $\mathcal{N}_{\geq i+1}$, $Q \cap \mathcal{N}_i$ is a quotient of f_i (i.e., upon contracting $\mathcal{N}_{\geq i}$). Since $\tilde{w}_i$ is an $(1+\epsilon)$-approximation of w for the quotients of f_i, we have

$$(1-\epsilon) w(Q \cap \mathcal{N}_i) \leq (\hat{w}(Q \cap \mathcal{N}_i) = \tilde{w}_i(Q \cap \mathcal{N}_i)) \leq (1+\epsilon) w(Q \cap \mathcal{N}_i).$$

Now we have

$$\hat{w}(Q) \geq \hat{w}(Q \cap \mathcal{N}_i) = \tilde{w}_i(Q \cap \mathcal{N}_i) \geq (1-\epsilon) w(Q \cap \mathcal{N}_i) \geq (1-\epsilon)^2 w(Q).$$

We also have

$$\begin{aligned}\hat{w}(Q) &\leq \tilde{w}_i(Q \cap \mathcal{N}_i) + \sum_{j<i} \tilde{w}_j(\mathcal{N}_j) \overset{\text{(c)}}{\leq} (1+\epsilon) w(Q \cap \mathcal{N}_i) + (1+\epsilon) \sum_{j<i} w(\mathcal{N}_j) \\ &\leq (1+\epsilon) w(Q) + \epsilon(1+\epsilon) w(Q) = (1+\epsilon)^2 w(Q),\end{aligned}$$

where (c) observes that $\mathcal{N}_j$ is a quotient in f_j, hence $\tilde{w}_j(\mathcal{N}_j) \leq (1+\epsilon) w_j(\mathcal{N}_j)$, for all j. Thus $(1-\epsilon)^2 w(Q) \leq \hat{w}(Q) \leq (1+\epsilon)^2 \hat{w}(Q)$, as desired.

Running times. For each f_i, the sparsification routine takes $\textsc{Time}(\mathcal{N}_i)$ randomized time, where we point out that a query to f_i translates to two queries of f in the oracle model. Over all i, this adds up to $\sum_i \textsc{Time}(\mathcal{N}_i) \leq \textsc{Time}(\mathcal{N})$ randomized time by superadditivity, as desired.

For actual instances of f, we must also address the running time of setting up each f_i. We address the three applications discussed in this article.

For matroids, we have $\textsc{Time}(S) = \tilde{O}(|S| Q_{\text{RANK}})$, hence $\textsc{Time}(\mathcal{N}) = \tilde{O}(n Q_{\text{RANK}})$. Each f_i is the rank function of the matroid $\mathcal{M}_i$ obtained by contracting $\mathcal{N}_{\geq i+1}$ and restricting to $\mathcal{N}_i$. It is easy to assemble all the matroids in nearly linear time since their ground sets are disjoint. So the overall running time is again $\tilde{O}(n Q_{\text{RANK}})$.

For hypergraphs, let $p = \sum_{e \in \mathcal{N}} |e|$, and for each i, let $p_i = \sum_{e \in \mathcal{N}_i} |e|$. Each f_i corresponds to the hypergraph H_i obtained by restricting to the edges in $\mathcal{N}_{\geq i}$ and then contracting all edges in $\mathcal{N}_i$. We have $\textsc{Time}(\mathcal{N}_i) = \tilde{O}(p_i)$ for all i, and $\textsc{Time}(\mathcal{N}) = \tilde{O}(p)$. We now bound the time to construct all the hypergraphs. The first hypergraph H_1 is obtained from the input hypergraph H by contracting all the edges in $\mathcal{N}_{\geq 2}$, in linear time. Given H_i, one obtains H_{i+1} in $O(p_i + p_{i+1})$ time by removing all the edges in $\mathcal{N}_i$ and uncontracting all the edges in $\mathcal{N}_{i+1}$. Altogether we spend $O(p)$ time constructing the H_i's, which is negligible.

Lastly we discuss coverage. Recall that $N = \sum_{e \in \mathcal{N}} f(e)$ denotes the total size of the set system. For each i, let $N_i = \sum_{e \in \mathcal{N}_i} f(e)$ denote the total size of the set system restricted to the points in $\mathcal{N}_i$. We have $\textsc{Time}(\mathcal{N}_i) = \tilde{O}(N_i)$ for all i, and $\textsc{Time}(\mathcal{N}) = \tilde{O}(N)$. Each f_i corresponds to the system obtained by first restricting to the points in $\mathcal{N}_{\geq i}$ and then contracting $\mathcal{N}_{\geq i+1}$. This leaves only the sets that are hit by $\mathcal{N}_i$ and not hit by $\mathcal{N}_{\geq i+1}$. The first set system is obtained from the input set system by removing all points in $\mathcal{N}_{\geq i+1}$ and all sets hit by $\mathcal{N}_{\geq i+1}$. The $(i+1)$th set system is obtained from the ith set system by removing the points $\mathcal{N}_i$, reintroducing $\mathcal{N}_{i+1}$, and updating the relevant sets accordingly. It is easy to do this in $O(N_i + N_{i+1})$ time. Altogether we spend $O(N) + \sum_i O(N_i + N_{i+1}) = O(N)$ time constructing the set systems (in succession), which is negligible.