Minimum and Approximate Minimum k-Cuts in Hypergraphs

Chris Bao, Joshua Wang, William Zhao

January 25, 2025

Abstract

The minimum cut problem and its generalizations are important to combinatorial optimization and have numerous applications in network reliability, circuit design, and clustering. Our work considers the minimum k-way cut problem in hypergraphs, which asks for a k-way partition of the vertex set that minimizes the number of crossing hyperedges. We begin by extending the work of Kogan and Krauthgamer (2014), using the randomized contraction technique introduced by Karger and Stein (1995), to bound the number of approximate minimum k-way cuts in low-rank hypergraphs. Next, we consider the branching contraction algorithm of Fox et al. (2019) as applied to the minimum k-way cut problem in unweighted hypergraphs. Under a conjectural bound on the scaled proportions of small hyperedges, we improve the running time to $\tilde{O}(mn^k + n^{4k-3})$. Finally, we generalize the near-linear time $(2+\varepsilon)$-approximation algorithm of Quanrud (2019) for the graph k-way cut problem, achieving an approximation ratio of $r(1+\varepsilon)$ for hypergraphs of rank r. As a component, we provide an algorithm for finding a minimum hypertree with improved runtime compared to the prior result of Baïou and Barahona (2023).

1 Introduction

The *minimum cut problem* for a weighted graph $G = (V, E, w)$ asks us to find a minimum weight *cut*, or subset of edges such that removing these edges disconnects the graph. Some of its numerous applications include network reliability [16], the traveling salesman problem [21], and compiler optimization of parallel computations [6]. In addition to minimum cuts, it is often useful to consider cuts with values close to the minimum, called *approximate minimum cuts*. Approximate minimum cuts have natural applications in network reliability, where one wants to find all likely failure modes instead of only the absolute most probable [25].

Various ways of solving the minimum cut problem in a graph are known. One of the first approaches utilizes the duality between the s-t minimum cut and maximum flow problems; by solving the $s - t$ maximum flow problem for every possible t, a minimum cut can be computed. The celebrated work of Karger [16] introduced the technique of *randomized contraction*, providing a fast algorithm for computing minimum cuts which does not depend on the duality with maximum flow. Karger [16]'s method also provides a significant improvement for combinatorial aspects of the problem (in particular, counting the numbers of minimum or approximate minimum cuts). It has influenced decades of follow-up research using similar and generalized techniques.

Tree packing is an alternative method for computing minimum cuts. Its first prominent use was by Karger [17], who used a duality between minimum cut and maximum spanning tree packing to provide a near-linear time algorithm for the minimum cut problem. Later, linear programming relaxations for tree packing were shown to have bounded integrality gap, resulting in fast linear programming-based approximation algorithms [7].

Another generalization of the minimum cut problem is the minimum k-cut problem. Here, we are asked to find a minimum weight *k-cut*, or subset of edges such that their removal partitions the

graph into at least k connected components. Via a reduction to the maximum clique problem, it can be shown that the minimum k-cut problem with k as input is NP-complete [14]. Therefore, we consider k to be constant. Under this assumption, the first polynomial-time algorithm was given by Goldschmidt and Hochbaum [14], with a running time of $n^{O(k^2)}$. Subsequently, randomized contraction and tree packing were applied to the problem; randomized contraction-based algorithms have achieved nearly optimal running time under certain hardness conjectures, while a tree packing approach leads to fast approximation algorithms [15, 7].

Our work concerns a further generalization: the minimum k-cut problem in hypergraphs. A *hypergraph* is a generalization of a graph in which hyperedges may be arbitrary subsets of the vertices instead of pairs. Hypergraphs can represent many combinatorial problems (for instance, the 3-SAT problem can be represented as a 3-uniform hypergraph). Furthermore, hypergraphs naturally model communication networks where one node influences many others; therefore, minimum and approximate minimum cuts in hypergraphs are relevant to network reliability [4].

1.1 Outline

In Section 2, we first extend the linear-programming approach of Kogan and Krauthgamer [20] by proving a bound on the minimum k-cut size in low-rank hypergraphs, proving Theorem 1.16. Next, we roughly defend the exponential dependence on r in Theorem 1.16. We end by opening discussion on multicriteria minimum cuts and generalizations with Theorem 1.17.

In Section 3, we consider the branching contraction algorithm of Fox et al. [10] for the exact minimum k-cut problem in hypergraphs, deriving a recursive bound for its expected running time. In Theorem 1.18, we then show that Fox et al. [10]'s running time can be improved in the case of unweighted or close-to-unweighted hypergraphs assuming Conjecture 3.2, which bounds the scaled proportions of small hyperedges.

In Section 4, we develop an alternative characterization of hyperforests as well as a faster check of whether a given set of hyperedges forms a hyperforest, proving Theorem 1.20. We show that multiplicative weight updates can be used to obtain a $(1+\epsilon)$-approximate solution to a linear programming relaxation for hypergraph minimum k-cut, and present an algorithm which rounds a $(1+\epsilon)$-approximate solution to the linear programming relaxation to an $r(1+\epsilon)$-approximate minimum k-cut, proving Theorem 1.19.

1.2 Definitions

We begin by defining the setting of our work.

Definition 1.1 (Hypergraph)**.** A *weighted hypergraph* is a triple $H = (V, E, w)$, where $w : E \to \mathbb{R}_+$ and $E \subseteq 2^V$. Throughout, we let $|V| = n$ and $|E| = m$. Each element of V is called a *vertex*, and each element of E is called a *hyperedge*. The *rank* or *size* of a hyperedge e is $|e|$; a hypergraph is *rank-r* if all its edges have rank at most r. A *graph*, denoted by $G = (V, E, w)$, is a rank-2 hypergraph.

Definition 1.2 (k-cut)**.** Given a weighted hypergraph $H = (V, E, w)$, a *k-cut* or *k-way cut* is a tuple $C = (V_1, V_2, \ldots, V_k)$ that partitions V. We say a hyperedge e *crosses* C if e is not contained in some V_i. The *value* of C is the sum of the weights of hyperedges crossing C:

$$\delta_H(C) = \sum\{w(e) \mid e \in E, e \subseteq V_i \forall i \in [k]\}.$$

We refer to 2-cuts simply as *cuts*. Note that the cut and k-cut definition also holds in graphs $G = (V, E, w)$. It is also useful to think of a k-cut as a set of hyperedges whose removal splits H into at least k connected components.

Definition 1.3 (Minimum k-cut)**.** A *minimum k-cut* C is a k-cut in graph G or hypergraph H with minimal value; this value is denoted λ_k. An *α-approximate minimum k-cut* is a k-cut with value at most $\alpha\lambda_k$.

Definition 1.4 (k-cut-set)**.** A *k-cut-set* is the set of hyperedges crossed by a given k-cut; note that one k-cut-set may correspond to multiple k-cuts in a hypergraph (although this correspondence is bijective for graphs). We can define minimum and α-approximate minimum k-cut-sets similarly.

Definition 1.5 (Hyperedge contraction)**.** Given a hypergraph $H = (V, E, w)$ and $e \in E$, the *contracted* hypergraph H/e is obtained by identifying all vertices in e with a single new "metavertex" v_e and removing hyperedges with rank 1. Furthermore, duplicate hyperedges are combined with their weights added. We say that a k-cut C *survives* the contraction of a hyperedge e if e does not cross C. An example of a contraction is shown in Figure 1.

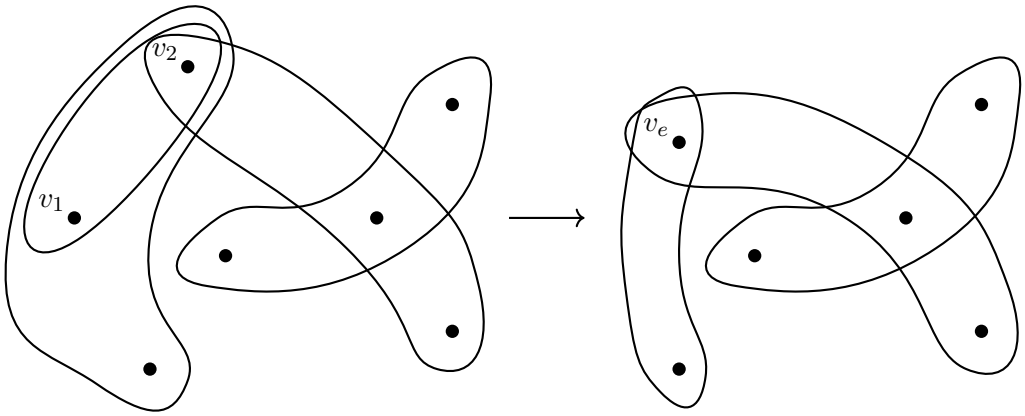

Figure 1: Performing the contraction $H/\{v_1, v_2\}$.

There are many different definitions of acyclity in hypergraphs found in the literature. We use the following definition of a hyperforest.

Definition 1.6 (Hyperforest)**.** Let $H = (V, E, w)$ be a hypergraph. If F is a multiset of hyperedges and $X \subseteq V$, then let $F[X]$ be the multiset of hyperedges in F contained in X. If $|F[X]| \leq |X| - 1$ for all nonempty X, then F is called a *hyperforest*. If additionally $|F|$ is maximal, then F is called a *hypertree*.

Definition 1.7 (Lagrangian Relaxation)**.** Given a linear program (P) of the form

$$\max(\mathbf{c}^\intercal \mathbf{x}) \text{ subject to } \mathbf{A_1 x} \leq \mathbf{b_1}, \mathbf{A_2 x} \leq \mathbf{b_2} \tag{P}$$

and a vector λ with positive components, the following linear program (R) is a *Lagrangian relaxation* of (P):

$$\max(\mathbf{c}^\intercal \mathbf{x} + \lambda^\intercal(\mathbf{b_2} - \mathbf{A_2 x})) \text{ subject to } \mathbf{A_1 x} \leq \mathbf{b_1}. \tag{R}$$

Lagrangian relaxations are useful for bounding and iterating towards optimal solutions. The Multiplicative Weight Update method, discussed in Section 1.5, relies on Lagrangian relaxations to quickly compute approximate solutions to linear programs.

Definition 1.8 (Approximation Algorithm)**.** For a minimization problem, an α-approximation algorithm ($\alpha \geq 1$) returns a solution with value at most α times that of the optimum solution. For a maximization problem, an α-approximation algorithm ($\alpha \leq 1$) returns a solution with value at least α times that of the optimum solution.

1.3 Randomized Contraction

The randomized contraction technique is based on the contraction operation, as defined in Section 1.2. Note that k-cuts which survive a contraction (i.e., do not cross the contracted edge) retain their structure and value. Furthermore, it is highly likely that a given minimum k-cut will survive contraction by a randomly selected hyperedge. Using these facts, Karger [16] devised the following randomized algorithm for computing an exact minimum cut in a graph $G = (V, E, w)$:

```
CONTRACT(G):
    if |V(G)| > 2:
        choose a random edge e ∈ E(G) proportionally to its weight
        return CONTRACT(G/e)
    else:
        return the cut among the two remaining vertices
```

Karger's algorithm and other similar algorithms based on the randomized contraction technique give a lower-bound probability of outputting a given minimum cut which bounds both the runtime and the number of minimum cuts. For instance, Karger [16] showed a lower bound on the probability that CONTRACT outputs any given minimum cut. This immediately results in an upper bound on the number of minimum cuts in a graph.

Theorem 1.1 (Karger [16])**.** *A particular minimum cut in G is produced by the CONTRACT procedure with probability at least $\binom{n}{2}^{-1}$. As a result, the number of minimum cuts in G is at most $\binom{n}{2} = O(n^2)$.*

This method was applied to the enumeration α-approximate minimum cuts and exact minimum k-cuts in graphs by Karger and Stein [19]; they found upper bounds of $O(n^{2\alpha})$ and $O(n^{2k-2})$ respectively. In Section 2, we briefly discuss how these results can be combined to cover α-approximate minimum k-cuts. All of these bounds are tight (or nearly so) for cycle graphs $G = C_n$.

The randomized contraction technique was first extended to weighted hypergraphs by Kogan and Krauthgamer [20]. They were able to bound the number of α-approximate minimum cuts in low-rank hypergraphs:

Theorem 1.2 (Kogan and Krauthgamer [20])**.** *The number of α-approximate minimum cuts in a rank-r weighted hypergraph $H = (V, E, w)$ is $O(2^{\alpha r} n^{2\alpha})$.*

By considering *nonuniform* randomized contraction (in which the probability of selecting a hyperedge depends on its size as well as its weight), Chandrasekaran et al. [5] showed that Karger and Stein [19]'s $O(n^{2k-2})$ bound on the number of exact minimum k-cuts (and thus k-cut-sets) a graph also holds for hypergraphs.

Theorem 1.3 (Chandrasekaran et al. [5])**.** *The number of minimum k-cut-sets in a hypergraph is $O(n^{2k-2})$.*

We now turn our attention to the time complexity of various contraction algorithms. Using the fact that Karger's Algorithm outputs any minimum k-cut with probability $\Omega(1/n^{2k-2})$, we can find a minimum k-cut with high probability by running $O(n^{2k-2} \log(n))$ iterations of CONTRACT and outputting the smallest k-cut found. Because CONTRACT can be implemented with running time $O(n^2)$, this approach gives an $\tilde{O}(n^{2k})$ Monte Carlo algorithm for finding a minimum k-cut [16].

While Karger's Algorithm provides tight bounds on the number of minimum k-cuts, its running time is not optimal. A major improvement was accomplished by Karger and Stein [19], who introduced the *recursive contraction* technique. Their key insight is that later contractions in Karger's Algorithm are much more likely to destroy a given minimum k-cut; therefore, recursively branching at certain points makes it more likely that earlier contractions will not be wasted. The Karger-Stein algorithm for finding minimum k-cuts in a graph $G = (V, E, w)$ is outlined below:

```
RecursiveContract(G):
    if |V(G)| < k·∛2:
        randomly contract G to k vertices
        return the resulting k−cut
    else:
        randomly contract G to |V(G)|/∛2 vertices
        return argmin(RecursiveContract(G), RecursiveContract(G))
```

The factor of $\sqrt[k]{2}$ is chosen so that any given minimum k-cut survives to the next branching step with probability at least $1/2$. Using this fact, Karger and Stein [19] showed the the probability of outputting a minimum k-cut is greatly improved to $\Omega(1/\log(n))$. Therefore, the Karger-Stein algorithm consists of $O(\log^2(n))$ iterations of RecursiveContract. Karger and Stein [19] gave a bound on its running time:

Theorem 1.4 (Karger and Stein [19])**.** *For a graph $G = (V, E, w)$, the Karger-Stein algorithm finds a minimum k-cut with high probability in time $\tilde{O}(n^{2k-2})$.*

Recently, an improved analysis showed that the running time of the Karger-Stein algorithm is optimal under certain hardness conjectures.

Theorem 1.5 (Gupta et al. [15])**.** *For a graph $G = (V, E, w)$, the Karger-Stein algorithm finds a minimum k-cut with high probability in time $\tilde{O}(n^k)$. Via a reduction to the $(k-1)$-Maximum-Clique problem, it is conjectured that the optimal time complexity is $\Omega(n^{(1-o(1))k})$.*

The recursive contraction technique was generalized to hypergraphs by the *branching contraction* algorithm of Fox et al. [10]. Here, branching occurs randomly, rather than at the fixed numbers of vertices of the Karger-Stein algorithm. In particular, there is a chance to create a recursive branch every time a hyperedge is selected for contraction; the branching probability is higher for larger hyperedges, which are more likely to destroy a given minimum k-cut. The branching contraction algorithm is currently the fastest known for the minimum k-cut problem in weighted hypergraphs.

Theorem 1.6 (Gupta et al. [15])**.** *For a hypergraph $H = (V, E, w)$, the branching contraction algorithm finds a minimum k-cut with high probability in running time $\tilde{O}(mn^{2k-2})$.*

We review the branching contraction algorithm in greater detail in Section 3.

1.4 Tree Packing

Another approach to the minimum k-cut problem, entirely distinct from randomized contraction, is the *tree packing* duality. In a weighted graph, the *fractional tree packing number* is the maximum number of (potentially fractional) spanning trees that can be packed into the graph without exceeding the capacity of any edge. It is also the optimum value of the following linear program:

$$\text{maximize} \qquad \sum_T y_T$$

$$\begin{aligned} \text{subject to} \quad & \sum_{T \ni e} y_T \leq w_e, \quad e \in E \\ & 0 \leq y_T, \quad T \quad \text{spanning tree.} \end{aligned}$$

The dual program of the above linear program is a linear programming relaxation for the minimum cut problem, as any cut intersects each spanning tree at least once:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} w_e x_e \\ \text{subject to} \quad & \sum_{e \in T} x_e \geq 1, \quad T \quad \text{spanning tree} \\ & x_e \geq 0, \quad e \in E. \end{aligned}$$

Using a variant of this duality, Karger [17] obtained a near-linear time minimum cut algorithm and better upper bounds for the number of α-approximate minimum cuts.

Theorem 1.7 (Karger [17])**.** *There exists a randomized algorithm that computes a minimum cut of a graph with high probability in $O(m \log^3 n)$ time, as well as a randomized algorithm that finds all minimum cuts of a graph with high probability in $O(n^2 \log n)$ time.*

Theorem 1.8 (Karger [17])**.** *The number of α-approximate minimum cuts in a graph is $O(n^{\lfloor 2\alpha \rfloor})$.*

Naor and Rabani [22] generalized the minimum cut LP relaxation to the minimum k-cut problem and connected it to a variant of fractional tree packing. Chekuri et al. [8] further analyzed the LP relaxation of minimum k-cut, showing it has a duality gap of at most 2. They also bounded the number of α-approximate minimum k-cuts using tree packing.

Theorem 1.9 (Chekuri et al. [8])**.** *The number of α-approximate minimum k-cuts in a graph is $O(n^{\lfloor 2\alpha(k-1) \rfloor})$.*

This bound is sharp, as the cycle graph C_n gives a lower bound of $\Omega(n^{\lfloor 2\alpha(k-1) \rfloor})$; it improves on the randomized contraction bound through the floor function in the exponent.

Hyperforests form a family of independent sets of a matroid, known as the *hypergraphic matroid*. Furthermore, Frank et al. [11] proved that hypertrees are bases of this hypergraphic matroid. Thus, hypertrees are useful for generalizing tree packing to hypergraphs. Baïou and Barahona [2] used hypertree packing to obtain a combinatorial r-approximation algorithm for minimum k-cut in hypergraphs of rank r.

Theorem 1.10 (Baïou and Barahona [2])**.** *There exists a deterministic algorithm that computes an r-approximation to the minimum k-cut in $O(kn^2p^{1+o(1)})$ time for rank-r hypergraphs, where p denotes the sum of the sizes of all hyperedges.*

Hypertree packing also bounds the number of α-approximate minimum k-cuts; this bound can be compared with our result in Section 2.

Theorem 1.11 (Baïou and Barahona [2])**.** *The number of α-approximate minimum k-cuts is $O(n^{\lfloor r\alpha(k-1) \rfloor})$ in a rank-r hypergraph.*

1.5 Multiplicative Weight Update

The following problem is a classic application of multiplicative weight update algorithms, which were first proposed in game theory in the 1950s [1]. On each turn, one attempts to answer a true-false question, aided by n experts. There is immediate feedback given on whether the answer selected is correct. The goal is to perform roughly the same as the best expert. Give each expert a weight, initialized as 1. For each turn, follow the weighted majority (ties are broken arbitrarily), and multiply the weight of each expert that gives an incorrect answer by $(1-\epsilon)$. Then the following theorem bounds the number of mistakes made by our algorithm:

Theorem 1.12 (see [1]). *Let $M(t)$ be the number of mistakes our algorithm makes after t turns, and let $m(t)$ be the least number of mistakes made by any expert. Then $M(t) \leq 2(1+\epsilon)m(t) + \frac{2\log n}{\epsilon}$ at any time t.*

These types of algorithms are called *multiplicative weight update algorithms* (MWU algorithms). Later, Plotkin, Shmoys, and Tardos [23] applied MWU to efficiently compute $(1-\epsilon)$-approximate solutions to *fractional packing problems* of the form

$$\max(\mathbf{c}^\intercal \mathbf{x}) \text{ subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0},$$

where the entries of $\mathbf{A}, \mathbf{b}, \mathbf{c}$ are all nonnegative. Key to their algorithm is the Lagrangian relaxation

$$\max(\mathbf{c}^\intercal \mathbf{y}) \text{ subject to } \mathbf{w}^\intercal \mathbf{A}\mathbf{y} \leq \mathbf{w}^\intercal \mathbf{b}, \mathbf{y} \geq \mathbf{0},$$

where $\mathbf{w}$ represents dual variables that are multiplicatively updated. Their running time depends on the *width* of the problem, or the maximum extent to which any constraint could be violated. Garg and Koenemann [12] later developed width-independent MWU, removing the dependence on the width. Chekuri and Quanrud [7]'s simplified implementation is outlined as follows:

```
WIDTH-INDEPENDENT MWU(A, b, c, ε):
    w^(0) ← 1/c (component−wise)
    x ← 0
    t ← 0
    while t < 1
        j ← arg max c_i/⟨w^(t), Ae_i⟩ //⟨v, w⟩ denotes dot product
        y ← ⟨w^(t), b⟩ · e_j/⟨w^(t), Ae_j⟩ //e_j is indicator vector
        //The optimal solution to the Lagrangian relaxation has only one nonzero value
        δ ← min(min{ε/(ln m/ε) · c_i/⟨e_i, Ay⟩}, 1 − t) //m is the number of components of c
        x ← x + δy
        w_i^(t+δ) ← w_i^(t) exp(δ · log m/ε · ⟨e_i, Ay⟩/c_i) for all i
        //w reflects how close each constraint is to being exceeded
        t ← t + δ
    return x
```

Theorem 1.13 (Chekuri and Quanrud [7]). *If $\epsilon < 1/2$, then this algorithm terminates in $O(m/\epsilon^2 \cdot \log m)$ iterations. Furthermore, $x/(1+O(\epsilon))$ is a $(1-O(\epsilon))$-approximate optimum solution to the initial packing LP.*

They applied the width-independent MWU technique to develop a $(1-\epsilon)$-approximation algorithm for fractional base packing in matroids.

Theorem 1.14 (Chekuri and Quanrud [7])**.** *Let $\mathcal{M}$ be a matroid with n elements and rank k, with an independence oracle with running time Q. Then, there is an algorithm that computes a $(1-\epsilon)$-approximation for fractionally packing disjoint bases of $\mathcal{M}$ with running time $\tilde{O}(nkQ/\epsilon^2)$.*

Quanrud [24] improved the running time for the special case of tree packing in graphs. By combining MWU with a rounding algorithm, they obtained a near-linear $(2+\epsilon)$-approximation algorithm for the minimum k-cut problem in graphs.

Theorem 1.15. *There exists a deterministic algorithm computing an $(2+\epsilon)$-approximation to minimum k-cut in $O(m\log^3 n/\epsilon^2)$ time for graphs.*

1.6 Our Contributions

The remainder of the paper is organized as follows. In Section 2, we generalize Kogan and Krauthgamer [20]'s approach to α-approximate minimum k-cuts, giving a bound which improves on the results of Baïou and Barahona [2].

Theorem 1.16. *In an rank-r hypergraph H with n vertices, there are at most*

$$O\left(\frac{k^{\alpha(k-1)r}}{k!}n^{2\alpha(k-1)}\right)$$

α-approximate k-cuts.

We also consider *multicriteria minimum cuts*, which use a combination of different edge cost criteria; we find that generalizations to hypergraphs and k-cuts are possible.

Theorem 1.17. *In an rank-r hypergraph H with n vertices, there are at most*

$$O\left(\frac{k^{\alpha(k-1)r}}{k!}n^{2\alpha(k-1)+t-1}\right)$$

α-approximate parametric minimum k-cuts.

In Section 3, we examine Fox et al. [10]'s branching contraction algorithm for hypergraph minimum k-cut in the case of unweighted or close-to-unweighted hypergraphs. Given a conjectural bound on the scaled proportions of small hyperedges (i.e., those with sizes at most k), we show that the running time can be improved:

Theorem 1.18. *Assume Conjecture 3.2 and that $k \geq 4$. Then, for a λ-balanced hypergraph $H = (V, E, w)$ (one in which the maximum ratio of weights is at most λ), the branching contraction algorithm of Fox et al. [10] has running time $O(mn^k + \lambda n^{4k-3})$.*

In Section 4, we consider the hypertree packing LP using the MWU framework, generalizing Quanrud's [24] algorithm to derive a faster $r(1+\epsilon)$-approximation algorithm for the minimum k-cut in hypergraphs.

Theorem 1.19. *There exists a deterministic algorithm computing an $r(1+\epsilon)$-approximation to minimum k-cut in*

$$O(mn^2 r\log n/\epsilon^2)$$

time for hypergraphs, where r is the rank of the hypergraph.

As an ingredient, we improve Baïou and Barahona's [3] $O(mrn^{1+o(1)})$ time minimum hypertree algorithm, achieving a faster running time.

Theorem 1.20. *A minimum hypertree can be computed in $O(mnr)$ time.*

2 Randomized Contraction Bounds for Approximate Minimum k-Cuts

2.1 Approximate Minimum k-Cuts in Hypergraphs

Karger [16] first provided an approach to find the number of minimum and α-approximate minimum cuts. We briefly present an extension of their analysis, applied to α-approximate k-cuts in graphs, a simpler version of the hypergraph case in Theorem 2.3. This demonstrates the key idea of Theorem 2.3.

Theorem 2.1. *The number of α-approximate minimum k-cuts in a graph is at most*

$$S(2\alpha(k-1),k)\binom{n}{2\alpha(k-1)} = O((kn)^{2\alpha(k-1)}/k!),$$

where $S(n,k)$ denotes the Stirling number of the second kind.

Proof. Consider the following algorithm, an extension of Karger's Contract algorithm.

```
GraphContract(G, α, k):
    if G has more than 2α(k − 1) vertices (contraction phase):
        choose a random edge e ∈ E(G)
        return GraphContract(G/e, α, k)
    else (enumeration phase):
        randomly partition the remaining vertices into k nonempty groups
        return the corresponding k−cut
```

The algorithm undergoes two stages: first, the *contraction phase*, until G has at most $2\alpha(k-1)$ vertices, followed by the *enumeration phase*, where a single randomly selected k-cut is output. Note that a k-cut C remains an α-approximate minimum k-cut as long as it survives contraction. Now, we claim that a particular α-approximate k-cut C survives the contraction phase with probability at least $\binom{n}{2\alpha(k-1)}^{-1}$, and is chosen by the enumeration phase with probability at least $\Omega(k!\cdot k^{-2\alpha(k-1)})$. Theorem 2.1 would then follow.

When G has n vertices, the probability that C survives a single random edge contraction is

$$\Pr(C \text{ survives contracting } e) = 1 - \frac{|C|}{m} = 1 - \frac{\alpha\lambda}{m},$$

where λ is the minimum cut value. We now show a lower bound on this probability by upper-bounding $\frac{\lambda}{m}$. Consider a k-cut C' as follows. Randomly select $k-1$ vertices $v_1, v_2, \ldots, v_{k-1}$ to form $k-1$ singleton parts of C'. Then, let $V \setminus V_{k-1}$ be the last part of C', where $V_{k-1} = \bigcup_{i=1}^{k-1}\{v_i\}$, as in Figure 2.

For each edge in the graph, the probability that it crosses the k-cut is $1 - \frac{\binom{n-k+1}{2}}{\binom{n}{2}}$. Thus, by linearity of expectation, the expected value of C' is m times the aforementioned probability. Therefore,

$$\frac{\lambda}{m} \le 1 - \frac{\binom{n-k+1}{2}}{\binom{n}{2}}.$$

However,

$$1 - \frac{\binom{n-k+1}{2}}{\binom{n}{2}} = 1 - \frac{(n-k+1)(n-k)}{n(n-1)} = \frac{(k-1)(2n-k)}{n(n-1)} \le \frac{2(k-1)}{n}.$$

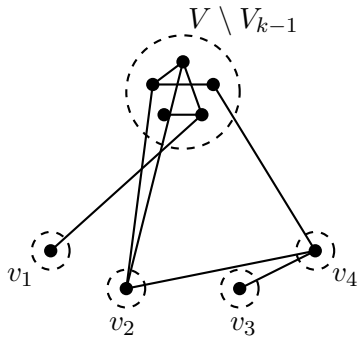

Figure 2: A valid k-cut C' where $k = 5$.

Therefore, it holds that

$$\Pr(C \text{ survives contracting } e) = 1 - \frac{\alpha\lambda}{m} \geq 1 - \frac{2\alpha(k-1)}{n}.$$

Recall that contracting an edge in G reduces the size of the vertex set of G by 1. Thus, the probability that C survives the entire contraction phase is at least

$$\left(1 - \frac{2\alpha(k-1)}{n}\right)\left(1 - \frac{2\alpha(k-1)}{n-1}\right)\cdots\left(1 - \frac{2\alpha(k-1)}{2\alpha(k-1)+1}\right) = \binom{n}{2\alpha(k-1)}^{-1} = \Omega(n^{-2\alpha(k-1)}).$$

Next, we analyze the enumeration phase, which occurs when G has at most $2\alpha(k-1)$ vertices remaining. A simple asymptotic bound on the total number of k-cuts in G follows by assigning each vertex one of k partitions. Thus, we bound the the number of k-cuts by $O(k^{2\alpha(k-1)})$. We improve on this bound by noting that the order of the k partitions does not matter, and thus we may divide by $k!$. Therefore, the total number of k-cuts in G is $O(k^{2\alpha(k-1)}/k!)$. In other words, the number of ways to put most $2\alpha(k-1)$ distinct vertices into k indistinct groups is at most $S(2\alpha(k-1), k) = O(k^{2\alpha(k-1)}/k!)$, where the asymptotic expression is given by [9]. □

For weighted graphs, we choose edges to contract with probability proportional to their weight.

Corollary 2.2. *The number of α-approximate minimum k-cuts in a weighted graph is at most*

$$S(2\alpha(k-1), k)\binom{n}{2\alpha(k-1)} = O((kn)^{2\alpha(k-1)}/k!),$$

where $S(n, k)$ denotes the Stirling number of the second kind.

Recall that Chekuri et al. [8] utilized tree packing to bound the number of α-approximate k-cuts by $O\left(\frac{\lfloor 2\alpha(k-1)\rfloor+1}{1-\{2\alpha(k-1)\}}(kn)^{\lfloor 2\alpha(k-1)\rfloor}\right)$ which has similar, but slightly better polynomial dependence on n, in comparison to Theorem 2.1. However, for particular values of α and k, the constant term could potentially be unbounded.

We move on to considering k-cuts in hypergraphs. Kogan and Krauthgamer [20] obtained an $O(n^{2\alpha})$ polynomial bound on the number of α-approximate minimum cuts for r-rank hypergraphs using linear programming. In what follows, we build on their work by proving an inequality relating degrees of vertices and the minimum k-cut size in the hypergraph, thus extending Kogan and Krauthgamer [20]analysis to k-cuts.

Theorem 2.3. *In a rank-r hypergraph H with n vertices, there are at most*

$$S(\alpha(k-1)r,k)\frac{1+\alpha(k-1)r}{2\alpha(k-1)+1}\binom{n-\alpha(k-1)(r-2)}{2\alpha(k-1)} = O\left(\frac{k^{\alpha(k-1)r}}{k!}n^{2\alpha(k-1)}\right)$$

α-approximate k-cuts, where $S(n,k)$ denotes the Stirling number of the second kind. When $k=2$, the above reduces to the result of Kogan and Krauthgamer [20], as $S(\alpha r,2) = 2^{\alpha r-1}-1$.

Proof. We use an algorithm similar to GraphContract(G,α,k) described in Theorem 2.1, but now, we contract hyperedges uniformly at random. Define λ as the minimum k-cut size in H, and let C be a particular α-approximate minimum k-cut.

```
HypergraphContract(H, α, r, k):
    if G has more than α(k−1)r vertices (contraction phase):
        choose a random hyperedge e ∈ E(H)
        return HypergraphContract(H/e, α, r, k)
    else (enumeration phase):
        for all remaining k−cuts C
            return C if |C| ≤ αλ
```

The algorithm undergoes two stages: first, the *contraction phase*, until H has at most $\alpha(k-1)r$ vertices, followed by the *enumeration phase*, where a remaining α-approximate minimum k-cuts is output.

As before, if an α-approximate minimum k-cut survives an edge contraction, then its size is unchanged, and it remains an α-approximate minimum k-cut. Furthermore, the rank of H is still at most r. That justifies the recursive call HypergraphContract$(H/e,\alpha,r,k)$.

We first consider the probability that C in H survives the contraction phase. Let this probability be $q(C,H)$, and define q_n to be the minimum value of $q(C,H)$ over all α-approximate minimum k-cuts C in hypergraphs H with n vertices. Following Kogan and Krauthgamer [20], we claim that

$$q_n \geq Q_n := \frac{2\alpha(k-1)+1}{1+\alpha(k-1)r}\binom{n-\alpha(k-1)(r-2)}{2\alpha(k-1)}^{-1}.$$

We show this by induction. Note that once we reach a hypergraph with at most $\alpha(k-1)r$ vertices remaining, we are guaranteed to find any of the remaining $\alpha(k-1)r$ cuts, so $q_t = 1$ for $t \leq \alpha(k-1)r$. In particular, $q_t \geq Q_t$ for these values.

Let $p_i = \Pr(|e|=i)$ be the probability of selecting a hyperedge of rank i. Also, let $y_i = \Pr(|e| = i, e \in C)$ be the probability of selecting an edge in C and with rank i. Note that the hypergraph has $t-|e|+1$ vertices after contracting a hyperedge e, so that by induction,

$$q(C,Q) \geq \sum_{i=2}^{r}(p_i - y_i)q_{t-i+1},$$

which implies that

$$q_t \geq \sum_{i=2}^{r}(p_i - y_i)q_{t-i+1}.$$

We show a lower bound on the above expression that holds for any real values of p_i and y_i that satisfy some set of conditions (not just p_i and y_i that correspond to some specific hypergraph). We derive the relations $0 \leq y \leq p$ and $\sum_{i=2}^{r} p_i = 1$. Lastly, we consider a bound based on the size of

the minimum cut. Let w_i be the total sum of edge weights for edges of size i, and let W be the sum of all edge weights in the hypergraph. Then, as the size of C is at most $\alpha\lambda$,

$$\sum_{i=2}^{r} y_i = \Pr(e \in C) \leq \frac{\alpha\lambda}{W}.$$

Next, we show that

$$\lambda \leq \frac{k-1}{t} \sum_{v \in V} \deg(v).$$

As in Figure 2, consider the $k-1$ vertices $V_{k-1} = \{v_1, v_2, \ldots, v_{k-1}\}$ and the k-cut C' defined by the singleton parts $\{\{v_1\}, \{v_2\}, \ldots, \{v_{k-1}\}, V \setminus V_{k-1}\}$. Consider summing the edge weights of all edges passing through v_i for $i = 1, 2, \ldots, k-1$, with multiplicity. By noting that every edge in the cut must be accounted for (at least once) in this sum, we conclude that the size of this cut is at most the sum of degrees over all v_i. Choosing V_{k-1} to be the $k-1$ vertices of least degree, we have that

$$\lambda \leq |C'| \leq \sum_{v \in V_{k-1}} \deg(v) \leq \frac{k-1}{t} \sum_{v \in V} \deg(v).$$

Thus,

$$\sum_{i=2}^{r} y_i \leq \frac{\alpha(k-1)}{tW} \sum_{v \in V} \deg(v) = \frac{\alpha(k-1)}{t} \sum_{i=2}^{r} i\frac{w_i}{W} = \frac{\alpha(k-1)}{t} \sum_{i=2}^{r} i p_i.$$

Thus, we write the following LP over real values p_i and y_i:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=2}^{r} (p_i - y_i) Q_{t-i+1} & \\
\text{subject to} \quad & 0 \leq y_i \leq p_i & \forall i = 2, 3, \ldots, r \\
& \sum_{i=2}^{r} p_i = 1 & \\
& \sum_{i=2}^{r} y_i \leq \frac{\alpha(k-1)}{t} \sum_{i=2}^{r} i p_i, &
\end{aligned}$$

We can improve the last bound to equality for any optimal set of values for p, y. Note that during the contraction phase, we have $t > \alpha(k-1)r$, so that

$$\sum_{i=2}^{r} y_i \leq \frac{\alpha(k-1)}{t} \sum_{i=2}^{r} i p_i \leq \frac{\alpha(k-1)r}{t} \sum_{i=2}^{r} p_i < \sum_{i=2}^{r} p_i.$$

Thus, there must exist some i for which $y_i < p_i$. We may increase the value of y_i, which decreases the value of the objective function without violating any conditions *until* equality is satisfied in the final condition. Thus, it suffices to analyze the following LP:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=2}^{r} (p_i - y_i) Q_{t-i+1} & \\
\text{subject to} \quad & 0 \leq y_i \leq p_i & \forall i = 2, 3, \ldots, r \\
& \sum_{i=2}^{r} p_i = 1 &
\end{aligned}$$

$$\sum_{i=2}^{r} y_i = \frac{\alpha(k-1)}{t} \sum_{i=2}^{r} i p_i.$$

We wish to show that the minimum value of the objective function is at least Q_t. Note that we have $2n-2$ variables, $2n-2$ inequalities, and 2 equalities. Thus, in any extreme point, $2n-4$ of the inequalities are tight. Based on this, we split into four cases.

- Case 1: $0 < y_i = p_i, 0 < y_j = p_j$. This case is impossible since $\sum_{i=2}^{r} y_i = \sum_{i=2}^{r} p_i$.
- Case 2: $0 = y_i < p_i, 0 = y_j < p_j$. This case is impossible since $\sum_{i=2}^{r} y_i = 0 < \frac{\alpha(k-1)}{t} \sum_{i=2}^{r} i p_i$.
- Case 3: $0 = y_i < p_i, 0 < y_j = p_j$. The minimum of the LP is

  $$\min_i \frac{t - \alpha(k-1)r}{t + \alpha(k-1)i - \alpha(k-1)r} Q_{t-i+1}.$$

  first, suppose that $t - i + 1 > \alpha(k-1)r$. It suffices to show that

  $$\frac{t - \alpha(k-1)r}{t + \alpha(k-1)i - \alpha(k-1)r} Q_{t-i+1} \geq Q_t,$$

  or that

  $$\frac{Q_{t-i+1}}{Q_t} \geq 1 + \frac{\alpha(k-1)i}{t - \alpha(k-1)r}.$$

  For shorthand, let $x = t - \alpha(k-1)r$. Then,

  $$\begin{aligned}
  \frac{Q_{t-i+1}}{Q_t} &= \frac{\binom{x+2\alpha(k-1)}{2\alpha(k-1)}}{\binom{x+2\alpha(k-1)-(i-1)}{2\alpha(k-1)}} = \frac{(x+2\alpha(k-1)) \cdots (x+1)}{x \cdots (x-i+2)} \\
  &= \frac{(x+2\alpha(k-1)) \cdots (x+2\alpha(k-1)-i+2)}{x \cdots (x-i+2)} \\
  &= \left(1 + \frac{2\alpha(k-1)}{x}\right) \cdots \left(1 + \frac{2\alpha(k-1)}{x-i+2}\right) \geq \left(1 + \frac{2\alpha(k-1)}{x}\right)^{i-1} \\
  &\geq 1 + \frac{2\alpha(k-1)(i-1)}{x} \geq 1 + \frac{\alpha(k-1)i}{t - \alpha(k-1)r},
  \end{aligned}$$

  as needed. Next, suppose that $t - i + 1 \leq \alpha(k-1)r$, so $Q_{t-i+1} = 1$. Then, we need to show that the optimal value of the linear program satisfies

  $$\frac{t - \alpha(k-1)r}{t + \alpha(k-1)i - \alpha(k-1)r} \geq Q_t = \frac{2\alpha(k-1)+1}{1+\alpha(k-1)r} \binom{t - \alpha(k-1)(r-2)}{2\alpha(k-1)}^{-1}.$$

  This follows from $t - \alpha(k-1)(r-2) > 2\alpha(k-1)$, which implies that

  $$\binom{t - \alpha(k-1)(r-2)}{2\alpha(k-1)}^{-1} \leq \binom{2\alpha(k-1)+1}{2\alpha(k-1)}^{-1} = \frac{1}{2\alpha(k-1)+1}$$

  and from the optimal value of the linear program being

  $$1 - \frac{\alpha(k-1)i}{t + \alpha(k-1)i - \alpha(k-1)r} \geq \frac{1}{1+\alpha(k-1)i} \geq \frac{1}{1+\alpha(k-1)r}.$$

- Case 4: $0 < y_i < p_i$. Note that

$$1 - \frac{\alpha(k-1)}{t} i \geq \frac{t - \alpha(k-1)r}{t + \alpha(k-1)i - \alpha(k-1)r} = 1 - \frac{\alpha(k-1)i}{t + \alpha(k-1)i - \alpha(k-1)r},$$

  which is equivalent to the true statement $\alpha(k-1)r \geq \alpha(k-1)i$, so this case reduces to case 3.

Lastly, as in Theorem 2.1, the enumeration phase produces at most $S(\alpha(k-1)r, k)$ cuts total. Because $S(\alpha(k-1)r, k) = O(k^{\alpha(k-1)r}/k!)$, the theorem follows. □

Baïou and Barahona [2] generalized the tree packing approach of Chekuri et al. [8] and applied it to α-approximate minimum k-cuts in r-rank hypergraphs. Their analysis, when completed, achieves a $O((kn)^{\lfloor r\alpha(k-1)\rfloor})$ upper bound (see Appendix A). Thus, for $r > 2$, the upper bound shown by randomized contraction is an improvement over previous results (we compared the $r = 2$ case, where H is a simple weighted graph, of our contraction method to Chekuri et al. [8]'s tree packing method in a comment below Theorem 2.1). We discuss the algorithm of Baïou and Barahona [2] in greater detail in Section 4.

2.1.1 A demonstration of the exponential k^r dependence

Note that, ignoring the exponential dependence on r, the polynomial dependence on n of this result is very similar to that of the graph case. We roughly show that the exponential dependence on r is necessary by presenting a variation of the argument used by Kogan and Krauthgamer [20]. Consider a hypergraph H, shown in Figure 3, consisting of a central vertex v_c attached to $l + 1$ edges $E_1 = \{e, e_1, e_2, \ldots, e_l\}$ each of rank r and weight 1, where we take $r > k$. We suppose that the only intersection between any two hyperedges in E_1 is exactly $\{v_c\}$. Next, add the l hyperedges $E_2 = \{e_1 \setminus v_c, e_2 \setminus v_c, \ldots, e_l \setminus v_c\} = \{e'_1, e'_2, \ldots, e'_l\}$, each with weight $\alpha - 1$.

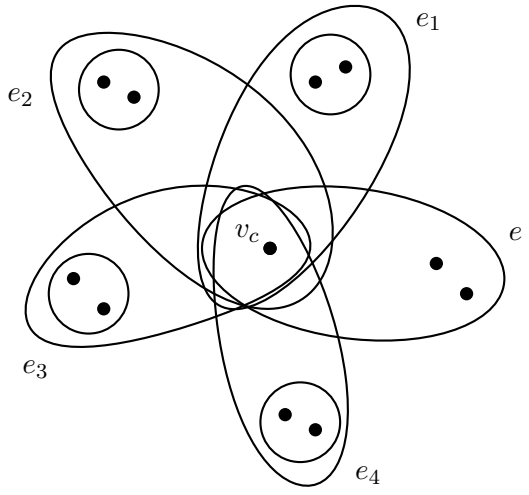

Figure 3: The hypergraph H with $(r, l) = (3, 4)$

Henceforth, we represent a k-cut C by stating its k vertex partitions $\{V_1, \ldots, V_k\}$. The minimum cut value λ must be at least 1, since any k-cut must cross some hyperedge in E_1. In fact, λ is exactly 1, since we can choose $V_2, V_3, \ldots, V_k$ to be singleton parts in e, and V_1 to encompass the remaining vertices. Now, select an edge $e'_i \in E_2$. We may set all the vertices of H outside of e'_i to

be in V_1, and then partition the vertices in e'_i into the groups $V_1, V_2, \ldots, V_k$. Notice that the value of this k-cut is exactly α. For large r, the number of ways to do such a partition will approach $k^r/k!$, as discussed in Theorem 2.1. Thus, the k^r expression indeed is required.

2.2 Enumerating a multicriteria minimum cut

Karger [18] studied parametric minimum cuts for graphs:

Definition 2.1 (Parametric Minimum Cut)**.** Consider a hypergraph $H = (V, E)$ and let $c_1, c_2, \ldots, c_t : E \to \mathbb{Z}_+$ be t hyperedge cost criteria. A cut C is a *parametric minimum cut* if there exist positive multiplicative coefficients $\mu_1, \ldots, \mu_t \in \mathbb{R}_+$ such that C is a minimum cut in the hypergraph $H' = (V, E, w_\mu)$ with the weight function

$$w_\mu(e) = \sum_{i=1}^{t} \mu_i c_i(e).$$

This definition easily extends to α-approximate minimum cuts and minimum k-cuts.

We use the interleaving argument of Karger [18] to show the following theorem, which generalizes their Corollary 4.5.

Theorem 2.4. *The number of α-approximate parametric minimum k-cuts in a rank-r hypergraph is:*

$$O\left(\frac{k^{\alpha(k-1)r}}{k!} n^{2\alpha(k-1)+t-1}\right).$$

Proof. Consider a random *interleaving* of contractions, where we select an $i \in [t]$, and then randomly contract an edge with probability proportional to c_t. As in [18], it suffices to find the number of interleavings. As contractions are commutative, we only need to select the number of times m_i we contract using the weights c_i, for each i. Following the algorithm in Theorem 2.3, we need to contract at most $n - 2\alpha(k-1)$ times, and therefore, the number of interleavings is at most $\binom{n-2\alpha(k-1)+t-1}{t-1} = O(n^{t-1})$ (the well known "stars and bars" theorem [26]). The claimed result follows by multiplying this factor by the result from Theorem 2.3. □

3 Branching Contraction for Unweighted Hypergraph k-Cut

As with many variants of the minimum-cut problem, generalizations of the Karger-Stein recursive contraction algorithm prove to be simple and efficient for the minimum k-cut problem in hypergraphs. Currently, the fastest known algorithm for this problem is the *branching contraction* algorithm of Fox et al. [10]. For a weighted hypergraph $H = (V, E, w)$, it returns a minimum k-cut with high probability in expected time $\tilde{O}(mn^{2k-2})$.

The key difference between the branching contraction algorithm and the classical Karger-Stein algorithm for graph k-cut is the random creation of branches. While Karger-Stein branches at fixed numbers of vertices (decreasing the number of vertices by a factor of $\sqrt[k]{2}$ each time), Fox et al. [10] choose to branch randomly based on the size of the contracted hyperedge. In particular, the branch probability $z_n(e)$ is given by

$$z_n(e) = 1 - \frac{\binom{n-|e|}{k-1}}{\binom{n-1}{k-1}}.$$

In this context, a "branch" consists of copying the hypergraph and making a recursive call before contracting; a recursive call will always be made on the contracted hypergraph. The intuition behind this step is that the number of new minimum k-cuts found by the branch and the number killed by the contraction are equal in expectation. Thus, the algorithm's execution on a hypergraph H may be organized into a "contraction tree", with H as the root and the children of a given node being the hypergraphs obtained by contractions of that node, as shown in Figure 4.

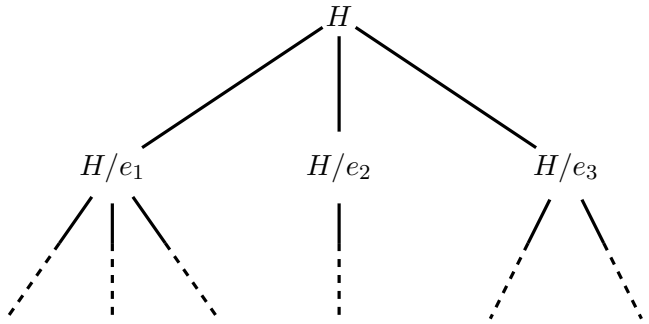

Figure 4: One possible contraction tree for H.

For future reference, we outline the algorithm BranchingContract below. It takes hypergraph $H = (V, E, w)$ as input and returns k-cut S. During recursive calls, S contains the edges which are guaranteed to belong to the final k-cut. BranchingContract depends on subroutine Contract(H, e), which copies H to new hypergraph H' and returns the ordered pair $(H, H'/e)$.

```
BranchingContract(H, S, k):
    move edges of size at least n − k + 2 from E to S
    if E = ∅:
        return S
    select e ∈ E with probability proportional to w(e)
    (H, H'/e) = Contract(H, e)
    with probability z_n(e):
        return arg min(BranchingContract(H, S, k),
                       BranchingContract(H'/e, S, k))
    else:
        return BranchingContract(H'/e, S, k)
```

We call any edge of size at least $n-k+2$ a *k-cover edge*. k-cover edges are removed from E and added to S because they are guaranteed to cross any k-cut in the hypergraph; through repeated contraction, all edges will eventually become singletons or k-cover edges, guaranteeing termination. Thus, at any point in the algorithm we consider the hypergraph H to have edge sizes in the set $G = \{2, \ldots, n-k+1\}$.

Definition 3.1. A hypergraph $H = (V, E, w)$ is λ-balanced for $\lambda \geq 1$ if $\max\limits_{e_1, e_2 \in E} \dfrac{w(e_1)}{w(e_2)} \leq \lambda$. Note that a 1-balanced hypergraph is essentially unweighted.

We will examine the performance of the branching contraction algorithm on λ-balanced hypergraphs. For the remainder of this section, fix some λ-balanced hypergraph $H_0 = (V_0, E_0, w_0)$ with

$|V_0| = n_0$ and $|E_0| = m_0$. We will use $H = (V, E, w)$ with $|V| = n$ and $|E| = m$ to denote a hypergraph obtained by a sequence of contractions from H_0; note that H is not necessarily λ-balanced because duplicate edges are combined after contractions.

Let y_i for $i \in G$ be the proportion by weight of size-i edges in H, and define Y_i similarly for H_0. First, we will show the following lemma about the expected structure of the contraction tree:

Lemma 3.1. *The expected number of children of H in the contraction tree with $n - i + 1$ vertices is $y_i C(H)$, where*

$$C(H) = \binom{n-1}{k-1} \Big/ \sum_{i \in G} y_i \binom{n-i}{k-1}$$

is the expected number of children of H.

Proof. Using the fact the edges are selected by weight, the probability of *not* branching is:

$$\begin{aligned}
\Pr[\text{no branch}] &= \sum_{i \in G} y_i \Pr[\text{no branch} \mid \text{size } i \text{ contracted}] \\
&= \sum_{i \in G} y_i (1 - z_n(i)) \\
&= \frac{1}{\binom{n-1}{k-1}} \sum_{i \in G} y_i \binom{n-i}{k-1}.
\end{aligned}$$

Our expression for $C(H)$ then follows as the expected value of a geometric distribution. We finish by using the fact that each contracted edge is selected at random proportionally to its weight. □

We conjecture the following bound a scaled proportion of small hyperedges (with size at most k) in the intermediate hypergraphs obtained during the algorithm's execution:

Conjecture 3.2. *For a hypergraph H derived from H_0 through a sequence of contractions, we have the following:*

$$\frac{\sum_{i \le k} (k - i + 1) y_i \binom{n-i}{k-1}}{\sum_{i \in G} y_i \binom{n-i}{k-1}} m = O(\lambda n_0^{2k-1}).$$

Let an effective bound be $A\lambda n_0^{2k-1}$.

We note that the conjecture holds for the original λ-balanced hypergraph H_0. Assuming that the minimum edge weight in H_0 is 1, we have the bounds $m y_i \le \lambda \binom{n_0}{i}$ for $i \in G$. Thus:

$$\frac{\sum_{i \le k} (k - i + 1) y_i \binom{n_0-i}{k-1}}{\sum_{i \in G} y_i \binom{n_0-i}{k-1}} m \le \sum_{i \le k} (k - i + 1) \binom{n_0}{i} \binom{n_0 - i}{k-1} = O(\lambda n_0^{2k-1}).$$

Furthermore, the conjecture is supported by computational experiments on certain families of random hypergraphs.

Now, we will derive a recurrence for the running time of BranchingContract. Fox et al. [10] provide an implementation of Contract running in time $O(m(n - |e| + 1))$, where e is the

size of the contracted edge. For the purposes of computing the running time, assume that this time contribution can be uniformly bounded by $m(n-|e|+1)$. Let $T(H)$ be the expected runtime of BranchingContract on hypergraph H, given that H may be obtained from H_0 through a series of contractions. Then, using the result of Lemma 3.1 on the expected number of children with contracted edge size i, we have:

$$T(H) = \sum_{i \in G} y_i C(H) \left(m(n-i+1) + \mathbb{E}\left(T(H/e) \mid |e| = i\right)\right).$$

Let $T(m,n)$ be the maximum of $T(H)$ over the possible H with at most m edges and at most n vertices. Noting that $T(m,n)$ is increasing in m, we have:

$$T(m,n) \leq \sum_{i \in G} y_i C(H)(m(n-i+1) + T(m, n-i+1)).$$

Now we are ready to bound the running time.

Theorem 3.3. *Assuming Conjecture 3.2 and that $k \geq 4$, we have $T(H_0) = O(m_0 n_0^k + \lambda n_0^{4k-3})$.*

Proof. It suffices to show the following claim by inducting on n:

$$T(m,n) \leq \left(mn + B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n\right)\right)\binom{n}{k-1} - mn,$$

where B is a constant. For the base case, take $n \leq k^2 + k$; then, both m and n can be bounded by functions of k, so a sufficiently large B will work.

Now, assume that $n > k^2 + k$ and that the inductive hypothesis holds for all $n' < n$. Then, we have:

$$\begin{aligned} T(m,n) &\leq C(H) \sum_{i \in G} y_i (m(n-i+1) + T(m, n-i+1)) \\ &\leq C(H) \sum_{i \in G} y_i \left(m(n-i+1)\binom{n-i+1}{k-1} + B\lambda n_0^{2k-1}\left(\binom{n-i}{k-1} - n\right)\binom{n-i+1}{k-1}\right). \end{aligned}$$

For the second term in the sum, we have:

$$\begin{aligned} B\lambda n_0^{2k-1}\left(\binom{n-i}{k-1} - n\right)\binom{n-i+1}{k-1} &\leq B\lambda n_0^{2k-1}\left(\binom{n-i}{k-1}\binom{n-i+1}{k-1} - n \cdot \frac{n-i+1}{n-i-k+2}\binom{n-i}{k-1}\right) \\ &\leq B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n \cdot \frac{n-1}{n-k}\right)\binom{n-i}{k-1} \\ &\leq B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n - 1\right)\binom{n-i}{k-1}. \end{aligned}$$

Because $i \geq 2$ and $k \geq 2$. For the first term in the sum, note that:

$$(n-i+1)\binom{n-i+1}{k-1} = \frac{(n-i+1)^2}{n-i-k+2}\binom{n-i}{k-1}.$$

Let $f(i) = \dfrac{(n-i+1)^2}{n-i-k+2}$. Then, we have the following:

- $f(2) = \dfrac{(n-1)^2}{n-k} = \dfrac{(k-1)n}{n-k} + n\left(1 - \dfrac{1}{n}\right) \leq \dfrac{(k-1)n}{n-k} + n\left(1 - \dfrac{1}{\binom{n}{k-1}}\right)$.
- $f(k+1) = \dfrac{(n-k)^2}{n-2k+1} = n\left(1 - \dfrac{n-k^2}{n(n-2k+1)}\right) \leq n\left(1 - \dfrac{k}{n^2}\right) \leq n\left(1 - \dfrac{1}{\binom{n}{k-1}}\right)$.
  Assume $k \geq 4$. Then, the last inequality holds because
  $$\binom{n}{k-1} \geq \frac{n(n-1)(n-2)}{(k-1)(k-2)(k-3)} \geq \frac{n^2}{k}\frac{k^2+k-2}{(k-2)(k-3)} \geq \frac{n^2}{k}.$$
- $f(n-k+1) = k^2 \leq n\left(1 - \dfrac{1}{k+1}\right) \leq n\left(1 - \dfrac{1}{\binom{n}{k-1}}\right)$.
- $f''(i) = \dfrac{2(k-1)^2}{(n-k-i+2)^3}$, which is positive for $i \in G$. Thus, f is convex.

Putting this together, we find $f(i) \leq \dfrac{\max(k-i+1,0)n}{n-k} + n\left(1 - \dfrac{1}{\binom{n}{k-1}}\right)$. Thus, we have:

$$\begin{aligned}
T(m,n) &\leq C(H)\left(\sum_{i>k} my_i\left(1 - \frac{1}{\binom{n}{k-1}}\right)n\binom{n-i}{k-1} + \sum_{i\leq k} my_i \cdot \frac{(k-i+1)n}{n-k}\binom{n-i}{k-1}\right.\\
&\quad \left. + \sum_{i\in G} y_i B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n - 1\right)\binom{n-i}{k-1}\right)\\
&= \binom{n}{k-1}\left(mn\left(1 - \frac{1}{\binom{n}{k-1}}\right)\frac{\sum_{i>k}\binom{n-i}{k-1}}{\sum_{i\in G} y_i\binom{n-i}{k-1}} + m\frac{n}{n-k}\frac{\sum_{i\leq k}(k-i+1)y_i\binom{n-i}{k-1}}{\sum_{i\in G} y_i\binom{n-i}{k-1}}\right.\\
&\quad \left. + B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n - 1\right)\right).
\end{aligned}$$

Using Conjecture 3.2:

$$\begin{aligned}
T(m,n) &\leq \binom{n}{k-1}\left(mn + \frac{n}{n-k}\cdot A\lambda n_0^{2k-1} + B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n - 1\right)\right) - mn\\
&\leq \binom{n}{k-1}\left(mn + A\left(1 + \frac{1}{k}\right)\lambda n_0^{2k-1} + B\lambda n_0^{2k-1}\left(\binom{n-1}{k-1} - n - 1\right)\right) - mn.
\end{aligned}$$

Assuming that $B \geq A(1+1/k)$ completes the induction step. Thus, we find the desired bound for $T(m,n)$. □

Corollary 3.4. *Assuming Conjecture 3.2 and that $k \geq 4$, for unweighted H_0 we have $T(H_0) = O(m_0 n_0^k + n_0^{4k-3})$.*

Remark 3.5. By using the induction hypothesis $T(m,n) \leq m\left(\binom{n-1}{k-1} - n\right)\binom{n}{k-1}$, we may complete the induction without the use of Conjecture 3.2 (or the assumption of λ-balance), recovering the $O(mn^{2k-2})$ time complexity shown by Fox et al. [10].

4 Hypergraph k-cut Approximation

Quanrud [24] obtained a deterministic $(2+\epsilon)$-approximation algorithm for the minimum k-cut problem in a graph, with a near-linear running time independent of k. Our generalization to hypergraphs obtains a similar deterministic running time independent of k and an approximation factor of $r(1+\epsilon)$, where r is the rank of the hypergraph. For low rank hypergraphs, this improves on the $(k-1)$-approximation algorithm of Zhao et al. [28].

Baïou and Barahona [2] studied the following LP relaxation (L) for the hypergraph k-cut problem, where $\mathcal{T}$ is the set of all hypertrees of H:

$$\begin{aligned} \text{minimize} \quad & \sum_{e\in E} w_e x_e \\ \text{subject to} \quad & \sum_{e\in T} x_e \geq |T| + k - n, \quad T \in \mathcal{T} \\ & x_e \in [0,1], \quad e \in E. \end{aligned}$$

Note that they only considered the case of a connected hypergraph (one with no trivial cut), where $|T|$, the size of a hypertree, is $n-1$ for any $T \in \mathcal{T}$. They obtained an r-approximation algorithm for minimum k-cut based on combinatorial methods using max-flow. Their algorithm's running time is $O(kn^2p^{1+o(1)})$, where p is the sum of the sizes of all hyperedges. For comparison, we obtain an $r(1+\epsilon)$-approximation algorithm by using the MWU framework. Our algorithm's running time is $O(mn^2r\log^2 n/\epsilon^2)$. As part of our algorithm, we compute a minimum hypertree in $O(mnr)$ time. This improves on Baïou and Barahona's previous running time of $O(mrn^{1+o(1)})$. We derive their running times in appendix B.

4.1 Setting up a Linear Programming Relaxation

We define the following linear program (C), where $\mathcal{F}$ is the set of all hyperforests of H:

$$\begin{aligned} \text{minimize} \quad & \sum_{e\in E} w_e x_e \\ \text{subject to} \quad & \sum_{e\in F} x_e \geq |F| + k - n, \quad F \in \mathcal{F} \\ & 0 \leq x_e, \quad e \in E. \end{aligned}$$

We show that (C) is equivalent to (L) in the following sense:

Lemma 4.1. *Any feasible solution x to (L) is also a feasible solution to (C). Furthermore, if x is a feasible solution to (C), then $x' = \min(x, 1)$ (taken component-wise) is a feasible solution to (L).*

Proof. Suppose x is a feasible solution to (L). If F is a hyperforest, then extend F to a hypertree, which can always be done by the greedy property of the hypergraphic matroid. Then

$$\sum_{e\in F} x_e = \sum_{e\in T} x_e - \sum_{e\in T\setminus F} x_e \geq |T| + k - n - |T \setminus F| = |F| + k - n,$$

where the inequality follows from the constraints of (L). Conversely, suppose x is a feasible solution to (C). Let T be a hypertree, and let T' be the set of hypjeredges for which $x_e \neq x'_e$. Then

$$\sum_{e\in T} x'_e = \sum_{e\in T'} 1 + \sum_{e\in T\setminus T'} x_e \geq |T'| + |T \setminus T'| + k - n = |T| + k - n,$$

where the inequality follows from the constraints of (C). Hence, x' is feasible in (L). □

Let (P) be the dual of (C), as below:

$$\begin{aligned} \text{maximize} \quad & \sum_F (|F| + k - n) y_F \\ \text{subject to} \quad & \sum_{F \ni e} y_F \leq w_e, \qquad e \in E \\ & 0 \leq y_F, \quad F \in \mathcal{F}. \end{aligned}$$

As (P) has polynomially many constraints, we apply the width-independent MWU framework in the next subsection to compute a $(1-\epsilon)$-approximate solution to (P), and therefore a $(1+\epsilon)$-approximate solution to (C).

4.2 Computing an Approximate Solution

For simplicity, we compute a $(1-O(\epsilon))$-approximate solution to (P). For each hyperedge e, maintain c_e that satisfies

$$\log(c_e w_e) = \frac{\log n}{\epsilon} \frac{\sum_{F \ni e} y_F}{w_e}$$

as well as c'_e that satisfies $c_e \leq (1+\epsilon)c'_e$. We remark that c' can be thought of as the multiplicative weights, as they are only ever updated multiplicatively by factors of $(1+\epsilon)$. Start with a feasible solution $y = 0$, which gives $c'_e = c_e = \frac{1}{w_e}$. At each iteration, compute a $(1+O(\epsilon))$-approximate solution z to the following Lagrangian relaxation (R):

$$\begin{aligned} \text{maximize} \quad & \sum_{F \in \mathcal{F}} (|F| + k - n) z_F \\ \text{subject to} \quad & \sum_{e \in E} c'_e \sum_{F \ni e} z_F \leq \sum_{e \in E} c'_e w_e. \end{aligned}$$

Add δz to y, where δ is chosen such that exactly one element of c' increases by a factor of $1+\epsilon$. Finally, update c and c' accordingly. At the end of the iterations, a $(1-O(\epsilon))$-approximate solution to (P) can be computed given y and w (see [7] for more details). We give the algorithm below:

```
Width−Independent MWU, H = (V, E, w):
    for hyperedges e:
        c_e ← 1/w_e
        c'_e ← 1/w_e
    for hyperforests F:
        y_F ← 0 (implicitly)
    for each iteration:
        compute z, a (1 + O(ε)) solution to (R)
        choose minimum δ so that there is at least one c' increase
        y ← y + δz
        for each hyperedge e update each c_e accordingly
        for each hyperedge e:
            if c'_e(1 + ε) < c_e: c'_e ← c'_e(1 + ε)
```

First, we bound the total number of iterations.

Lemma 4.2. *There are at most* $O\left(\frac{m \log n}{\epsilon^2}\right)$ *iterations.*

Proof. For a fixed e, $\log(c_e w_e) \leq O(\frac{\log n}{\epsilon})$, so $c_e \leq n^{O(1/\epsilon)}$, which implies c'_e can be increased by a $1+\epsilon$ factor at most $O(\frac{\log n}{\epsilon^2})$ times. Since there are m different c' values and each iteration increases at least one of them, the lemma follows. □

The remainder of this section is dedicated to analyzing the running time of each iteration.

4.2.1 Greedy Algorithm and Binary Search

An optimum solution to (R) is easily computed via the greedy algorithm.

Lemma 4.3. *Let $F \in \mathcal{F}$ maximize*

$$\frac{|F| + k - n}{\sum_{e \in F} c'_e}.$$

Then, for some γ, letting $z_F = \gamma$ and all other components 0 maximizes (R). Additionally, F consists of the l hyperedges of the minimum hypertree that have minimum weight, for some l.

Proof. Let

$$s(F) = \frac{|F| + k - n}{\sum_{e \in F} c'_e}.$$

Substitute $z'_F = z_F \sum_{e \in F} c'_e$ into (R), resulting in the following LP:

$$\begin{aligned} \text{maximize} \quad & \sum_{F \in \mathcal{F}} s(F) z'_F \\ \text{subject to} \quad & \sum_{F \in \mathcal{F}} z'_F \leq \sum_{e \in E} c'_e w_e. \end{aligned}$$

The optimum solution has z'_F nonzero for F maximizing $s(F)$, and zero for all other F. Thus, the optimum solution to (R) has z_F nonzero for F maximizing $s(F)$, and zero for all other F. Now suppose that such F contains l hyperedges. Then, by the greedy algorithm for the hypergraphic matroid, these l hyperedges must be the l hyperedges of the minimum hypertree that have minimum weight. □

Lemma 4.4. *A data structure can be maintained in $O(\log m)$ time per iteration such that the sum of the weights of the first l hyperedges of the minimum hypertree can be queried in $O(\log n)$ time.*

Proof. Define

$$L = \left\{ \left(e, \frac{(1+\epsilon)^i}{w_e} \right) : e \in E, i \in \left\{ 0, 1, 2, \ldots, O\left(\frac{\log n}{\epsilon^2} \right) \right\} \right\}.$$

Let B be a segment tree over L, which can be built in $O\left(\frac{m \log n}{\epsilon^2}\right)$ time. When (e, c'_e) is part of the minimum hypertree, mark the corresponding node on B, and unmark the node when (e, c'_e) is no longer part of the minimum spanning hypertree. For each node b, let L_b denote the subtree rooted at b. We keep track of the size of b and the sum of the edge weights. This can be maintained in $O(\log m)$ time per weight update. For a given l, the first l hyperedges of the minimum hypertree correspond to the marked nodes over some interval. This interval can be decomposed into $O(\log n)$ subtrees. Hence, the lemma holds. □

Instead of checking all possible values of l individually, we can use binary search:

Lemma 4.5. *A solution to (R) with respect to c' can be computed in $O(\log k) = O(\log n)$ queries of the sum of the weights of the first l hyperedges of the minimum hypertree.*

Proof. Let $e_1, e_2, \ldots, e_{n-1}$ be the hyperedges of the minimum hypertree, in increasing order of weight. Define

$$f(i) = \frac{i}{\sum_{j=1}^{n-k+i} c'_{e_j}}.$$

It suffices to find $i \in [k-1]$ maximizing $f(i)$. We have

$$\begin{aligned} f(i+1) - f(i) &= \frac{i+1}{\sum_{j=1}^{n-k+i+1} c'_{e_j}} - \frac{i}{\sum_{j=1}^{n-k+i} c'_{e_j}} \\ &= \frac{(i+1)\sum_{j=1}^{n-k+i} c'_{e_j} - i\sum_{j=1}^{n-k+i+1} c'_{e_j}}{\left(\sum_{j=1}^{n-k+i+1} c'_{e_j}\right)\left(\sum_{j=1}^{n-k+i} c'_{e_j}\right)} = \frac{\sum_{j=1}^{n-k+i} c'_{e_j} - ic'_{e_{i+1}}}{\left(\sum_{j=1}^{n-k+i+1} c'_{e_j}\right)\left(\sum_{j=1}^{n-k+i} c'_{e_j}\right)}. \end{aligned}$$

The denominator is positive. Since c'_{e_j} is increasing, if $f(i+1) - f(i) \le 0$ then $f(j+1) - f(j) \le 0$ for all $j \ge i$. Hence, the maximum of f can be found by binary search, evaluating $O(\log k)$ values of f. □

Each value takes $O(\log n)$ time to evaluate (we account for the amortized time separately), so we can compute a solution in $O(\log k \log n) = O(\log^2 n)$ time. Overall, the time per iteration to solve (R), not including maintaining the minimum hypertree with respect to w, is $O(\log m + \log^2 n)$.

4.3 Minimum Hypertree

To compute the minimum hypertree, it suffices to use the greedy algorithm, i.e. repeatedly adding the minimum weight hyperedge that preserves the hyperforest condition. The current state of the art for checking if a given set of hyperedges forms a hyperforest is Baïou [3] $O(rn^{1+o(1)})$ time algorithm based off s-t max flow, resulting in an $O(mrn^{1+o(1)})$ time minimum hypertree algorithm. We give a more detailed running time analysis in appendix B. Because this specific definition of hyperforest is uncommon in the literature, there is no other research the authors are currently aware of. We improve the running times for computing a minimum hypertree to $O(mnr)$.

Lemma 4.6. *Let $e_1, \ldots, e_h$ be hyperedges in $H = (E, V, w)$. Assume that $e_1, \ldots, e_{h-1}$ form a hyperforest. Then $e_1, e_2, \ldots, e_h$ form a hyperforest if and only if it is possible to find distinct vertices $v_1, \ldots, v_{h+1}$ such that $v_i \in e_i$ for $1 \le i \le h$ and $v_{h+1} \in e_h$.*

Proof. First, suppose that there exist distinct $v_1, \ldots, v_{h+1}$ such that $v_i \in e_i$ for $i \in [h]$ and $v_{h+1} \in e_h$. Let $F = \{e_1, \ldots, e_h\}$. We show that $|F[X]| \le |X| - 1$ for every nonempty $X \subseteq V$. Let $X \subseteq V$ be nonempty. We have the following two cases.

- Suppose that $e_h \not\subseteq X$. Since $e_1, \ldots, e_{h-1}$ form a hyperforest of H, it follows that $|F[X]| = |(F - \{e_h\})[X] \le |X| - 1$.
- Suppose that $e_h \subseteq X$. Then $v_k, v_{k+1} \in e_h \subseteq X$. For each $i \in [h-1]$ with $e_i \in F[X]$, we have $v_i \in e_i \subseteq X$. Since $v_1, \ldots, v_{h-1}$ are distinct, the function $(F - \{e_h\})[X] \to X - \{v_h, v_{h+1}\}$ defined by $e_i \to v_i$ is injective. Hence $(F - \{e_h\})[X] \le |X - \{v_h, v_{h+1}\}| = |X| - 2$. Since $e_h \subseteq X$, it follows that $|F[X]| = |(F - \{e_h\})[X]| + 1 \le (|X| - 2) + 1 = |X| - 1$.

Second, suppose that $e_1, \ldots, e_h$ form a hyperforest. Let $e_{h+1} := e_h$. Let $F = \{e_1, \ldots, e_h\}$, and $F' = \{e_1, \ldots, e_{h+1}\}$ where F and F' are multisets. For each $S \subseteq F'$, let $V(S)$ denote the union of all hyperedges in S. By Hall's Marriage Theorem, it suffices to show $|S| \le |V(S)|$ for all $S \subseteq F'$. Let $S \subseteq F'$. Since F is a hyperforest, we have $|F[V(S)]| \le |V(S)| - 1$. Hence, $|F'[V(S)]| \le |F[V(S)]| + 1 \le |V(S)|$. Since $S \subseteq F'$, it follows that $|S| \le |F'[V(S)]| \le |V(S)|$. This completes the proof. □

Theorem 4.7. *The minimum hypertree can be computed in $O(mnr)$ time.*

Proof. Given that $e_1, \ldots, e_{h-1}$ form a hyperforest, as well as a matching $e_i \to v_i$ for $i \in [h-1]$, the condition in the above lemma can be checked with BFS in $O(nr)$ time by reducing it to a matching problem and searching for augmenting paths. This is run $O(m)$ times, for a total running time of $O(mnr)$. We give the algorithm below:

```
MinimumHypertree(H = (V, E, w)) :
    T = {}
    sort all hyperedges by weights w
    For hyperedges e in order of increasing weights:
        Check with BFS if it is possible to assign two distinct vertices to e
        (changing vertices assigned to other hyperedges if necessary)
        if this is possible: T ← T ∪ {e}
            remove one of the two vertices assigned to e
        else: proceed to the next hyperedge
        \\recall a hypertree may contain multiple copies of the same hyperedge
    return T
```

□

It is inefficient to recompute the minimum hypertree every iteration. Instead, as MWU only ever increases the c' values, we use Chekuri and Quanrud's [7] Dynamic Minimum Weight Base algorithm:

Lemma 4.8 (Chekuri and Quanrud [7])**.** *For a matroid $M = (N, I, w)$ of rank k, the minimum weight base can be maintained in $O(kQ)$ amortized time per update to w, with $O(nkQ)$ initialization time, where n is the number of elements of M, and Q is the running time of an independence oracle.*

In the case of a hypergraphic matroid $(E, \mathcal{F}, w)$ over a hypergraph, the rank is $n-1$, and our algorithm above provides $Q = O(nr)$ (as it suffices to check if $F \cup e$ is a hyperforest given F is a hyperforest), so the minimum hypertree can be maintained in $O(n^2 r)$ amortized time per weight update. The initialization of their data structure requires $O(mn^2 r)$ time. Thus, the overall running time to compute a $(1+\epsilon)$-approximate solution to (L) is $O(mn^2 r \log n/\epsilon^2)$.

4.4 Rounding to a k-cut

Given a $(1+\epsilon)$-approximate solution x to (L), we construct an integral solution with objective value at most r times the objective value of x, where r is the rank of the hypergraph, which we can find during the initial minimum hypertree computation. This integral solution represents an $r(1+\epsilon)$-approximate k-cut.

If F is a hypertree in hypergraph H, say a *greedy component* of F is a connected component induced by a prefix of F, and a *greedy cut* is a cut induced by a greedy component of F. Quanrud's greedy-cuts algorithm [24] generalizes to hypergraphs:

```
GreedyCuts(H = (V, E, w), x) :
    let r be the rank (size of maximal hyperedge) of H
    let E' = {e ∈ E : x_e ≥ 1/r}
    if E' is a k−cut: return E'
    remove all hyperedges in E' from H, leaving l components
    let F be a minimum hypertree of H
    return the union of E' and the k − l minimum weight greedy cuts of F
```

A straightforward implementation, directly computing all greedy cuts, achieves a running time of $O(mn)$. Hence, the overall time of our algorithm is $O(mn^2 r \log n/\epsilon^2)$.

We now prove this algorithm's approximation properties. We start with the following lemma, which follows from generalizing the Goemans and Williamson primal-dual algorithm [13] to hypergraphs:

Lemma 4.9 (Takeshita et al. [27])**.** *Let T be a minimum hypertree in a hypergraph $H = (V, E, x)$ of rank r. Let $\mathcal{C}$ be the family of greedy cuts induced by T. Then there exists $y : \mathcal{C} \to \mathbb{R}$ satisfying the following properties:*

- $$\sum_{C \in \mathcal{C}, C \ni e} y_C \leq x_e \text{ for each } e \in E \tag{1}$$
- $$\sum_{C \in \mathcal{C}, C \ni e} y_C = x_e \text{ for each } e \in T \tag{2}$$
- $$r \sum_{C \in \mathcal{C}} y_C \geq \sum_{e \in T} x_e. \tag{3}$$

Lemma 4.10. *GreedyCuts returns a k-cut of cost at most $r \sum_e c_e x_e$.*

Proof. Let y be as in the above lemma. Since $x_e \leq \frac{1}{r}$ for all $e \in E'$, then $y_C \leq \frac{1}{r}$ for all greedy cuts C. Since x is a feasible solution to (L),

$$r \sum y_C \geq \sum_{e \in F} x_e \geq k - 1.$$

Let $C_1, C_2, \cdots, C_{k-1}$ be the $k-1$ minimum greedy cuts. The sum of the values of these greedy cuts is an optimal value of the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{C \in \mathcal{C}} y'_C \cdot \text{cost}(C) \\ \text{subject to} \quad & \sum_{F \ni e} y_F \leq w_e, F \in \mathcal{F} \\ & 0 \leq y'_C \leq 1, C \in \mathcal{C} \\ & \text{support } y' \subseteq \text{support } y. \end{aligned}$$

Since ry is a feasible solution to this problem, it follows that

$$\sum_{i=1}^{k-1} \text{cost}(C_i) \leq r \sum_{C \in \mathcal{C}} y_C \cdot \text{cost}(C) = r \sum_{e \in E'} w_e \sum_{C \ni e} y_C \leq r \sum_{e \in E'} w_e x_e,$$

as desired. □

Our algorithm is most of interest for low-rank hypergraphs.

5 Conclusion and Future Directions

In Section 2, we generalized the work of Kogan and Krauthgamer [20] from the cut case to the k-cut case, when considering α-approximate minimum cuts in low-rank hypergraphs. We also briefly touched on a graph construction achieving the exponential k^r dependence as well as a variation of multicriteria cuts relevant to Theorem 2.3. It is natural to ask whether there exists a low-rank hypergraph construction which better demonstrates a lower bound on the number of α-approximate minimum k-cuts. Furthermore, there exist other variants of multicriteria cuts which have unsatisfactory α-approximate extensions. Another future direction includes addressing these questions.

In Section 3, we showed that the running time of Fox et al. [10]'s branching contraction algorithm can be improved in the case of unweighted or close-to-unweighted hypergraphs under a conjectural bound on the scaled proportions of small hyperedges. In our efforts to show this conjecture, we found that the contraction operation typically causes the distribution of hyperedge weights to approach that of a complete hypergraph (as measured by KL divergence in computational testing); this extends the fact that complete hypergraphs are a fixed point under contraction. Such behavior, if established rigorously, could be helpful in understanding the performance of contraction-based algorithms for hypergraphs.

In Section 4, we provided a faster algorithm for computing a $r(1+\epsilon)$-approximate minimum k cut. It is known that obtaining an approximation ratio of $2-\epsilon$ in graphs is NP-hard. It is natural to ask whether obtaining an approximation ratio better than $r(1-\epsilon)$ is also NP-hard in the case of a hypergraph. Additionally, in the MWU algorithm, maintaining the minimum hypertree is the bottleneck. Thus, it is also natural to ask whether the minimum hypertree can be maintained in less than $O(n^2 r)$ time per weight update, which is the running time of Chekuri and Quanrud [7]'s general dynamic minimum weight base algorithm.

Finally, it is instructive to compare the randomized contraction and tree packing approaches. Randomized contraction typically gives algorithms for finding exact minimum k-cuts, while tree packing tends towards fast approximation algorithms. However, both give bounds on the numbers of approximate minimum k-cuts; comparing our randomized contraction bound of $O(k^{r\alpha} n^{2\alpha(k-1)})$ to Baïou and Barahona [2]'s tree packing bound of $O(n^{\lfloor r\alpha(k-1)\rfloor})$, it can be seen that randomized contraction gives a better polynomial dependence on n.

Tree packing allows one to derive bounds with a floor function in the exponent; considering the number of α-approximate minimum cuts in a graph, Karger [17]'s tree packing approach gave a tight $O(n^{\lfloor 2\alpha\rfloor})$ bound, improving on the $O(n^{2\alpha})$ bound given by randomized contraction. Thus, it would be interesting to investigate if tree packing can be used to similarly improve our randomized contraction-based bound.

Acknowledgements

We kindly thank our mentor, Yuchong Pan, for his leadership and guidance. We are grateful to the PRIMES program for making this research possible.

References

[1] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. doi: 10.4086/toc.2012.v008a006. URL `https://theoryofcomputing.org/articles/v008a006`.

[2] M. Baïou and F. Barahona. Packing hypertrees and the k-cut problem in hypergraphs. In D. E. Simos, V. A. Rasskazova, F. Archetti, I. S. Kotsireas, and P. M. Pardalos, editors, *Learning and Intelligent Optimization*, pages 521–534, Cham, 2022. Springer International Publishing. ISBN 978-3-031-24866-5.

[3] M. Baïou and F. Barahona. On some algorithmic aspects of hypergraphic matroids. *Discrete Math.*, 346(2), Feb 2023. ISSN 0012-365X. doi: 10.1016/j.disc.2022.113222. URL `https://doi.org/10.1016/j.disc.2022.113222`.

[4] R. Cen, J. Li, and D. Panigrahi. Hypergraph unreliability in quasi-polynomial time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1700–1711, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703836. doi: 10.1145/3618260.3649753. URL `https://doi.org/10.1145/3618260.3649753`.

[5] K. Chandrasekaran, C. Xu, and X. Yu. Hypergraph k-cut in randomized polynomial time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 1426–1438, USA, 2018. Society for Industrial and Applied Mathematics. ISBN 9781611975031.

[6] S. Chatterjee, J. R. Gilbert, R. Schreiber, and T. J. Sheffler. Array distribution in data-parallel programs. In K. Pingali, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallel Computing*, pages 76–91, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-49134-7.

[7] C. Chekuri and K. Quanrud. Near-linear time approximation schemes for some implicit fractional packing problems. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–820, 2017. doi: 10.1137/1.9781611974782.51. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611974782.51`.

[8] C. Chekuri, K. Quanrud, and C. Xu. Lp relaxation and tree packing for minimum k-cut. *SIAM Journal on Discrete Mathematics*, 34(2):1334–1353, 2020. doi: 10.1137/19M1299359. URL `https://doi.org/10.1137/19M1299359`.

[9] DLMF 26.8.42. *NIST Digital Library of Mathematical Functions 26.8.42*. `https://dlmf.nist.gov/26.8#E42`, Release 1.2.0 of 2024-03-15. URL `https://dlmf.nist.gov/`. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

[10] K. Fox, D. Panigrahi, and F. Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contractions. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 881–896, 2019. doi: 10.1137/1.9781611975482.54. URL `https://epubs.siam.org/doi/abs/10.1137/1.9781611975482.54`.

[11] A. Frank, T. Király, and M. Kriesell. On decomposing a hypergraph into k connected sub-hypergraphs. *Discrete Applied Mathematics*, 131(2):373–383, 2003. ISSN 0166-218X. doi: 10.1016/S0166-218X(02)00463-8. URL `https://www.sciencedirect.com/science/article/pii/S0166218X02004638`. Submodularity.

[12] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Founda-*

tions of Computer Science, FOCS '98, page 300, USA, 1998. IEEE Computer Society. ISBN 0818691727.

[13] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995. doi: 10.1137/S0097539793242618. URL https://doi.org/10.1137/S0097539793242618.

[14] O. Goldschmidt and D. Hochbaum. Polynomial algorithm for the k-cut problem. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 444–451, 1988. doi: 10.1109/SFCS.1988.21960. URL https://www.jstor.org/stable/3690374.

[15] A. Gupta, E. Lee, and J. Li. The karger-stein algorithm is optimal for k-cut. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 473–484, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450369794. doi: 10.1145/3357713.3384285. URL https://doi.org/10.1145/3357713.3384285.

[16] D. R. Karger. Global min-cuts in *rnc* and other ramifications of a simple mincut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, 01 1993.

[17] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, jan 2000. ISSN 0004-5411. doi: 10.1145/331605.331608. URL https://doi.org/10.1145/331605.331608.

[18] D. R. Karger. Enumerating parametric global minimum cuts by random interleaving. In *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, page 542–555, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450341325. doi: 10.1145/2897518.2897578. URL https://doi.org/10.1145/2897518.2897578.

[19] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4): 601–640, jul 1996. ISSN 0004-5411. doi: 10.1145/234533.234534. URL https://doi.org/10.1145/234533.234534.

[20] D. Kogan and R. Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, page 367–376, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333337. doi: 10.1145/2688073.2688093. URL https://doi.org/10.1145/2688073.2688093.

[21] E. Lawler, J. Lenstra, A. Kan, and D. Shmoys. *The Traveling Salesman Problem*. A Wiley-Interscience publication. John Wiley & Sons, Incorporated, 1985. ISBN 9780471904137. URL https://books.google.com/books?id=guDj0AEACAAJ.

[22] J. Naor and Y. Rabani. Tree packing and approximating k-cuts. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, pages 26–27, 2001. ISBN 0898714907. 2001 Operating Section Proceedings, American Gas Association ; Conference date: 30-04-2001 Through 01-05-2001.

[23] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, pages 495–504, 1991. doi: 10.1109/SFCS.1991.185411.

[24] K. Quanrud. Fast and Deterministic Approximations for k-Cut. In D. Achlioptas and L. A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 23:1–23:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-125-2. doi: 10.4230/LIPIcs.APPROX-RANDOM.2019.23. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.APPROX-RANDOM.2019.23.

[25] A. Ramanathan and C. J. Colbourn. Counting almost minimum cutsets with reliability applications. *Math. Program.*, 39(3):253–261, 1987. doi: 10.1007/BF02592076. URL https://doi.org/10.1007/BF02592076.

[26] R. Stanley. *Enumerative Combinatorics: Volume 1, 2nd edition*. Cambridge University Press, 2011. ISBN 9780511609589. doi: 10.1017/CBO9780511609589.

[27] K. Takeshita, T. Fujito, and T. Watanabe. Primal-dual approximation algorithms for some hypergraph problems. Technical report, Hiroshima University Graduate School of Engineering, 1999.

[28] L. Zhao, H. Nagamochi, and T. Ibaraki. On generalized greedy splitting algorithms for multiway partition problems. *Discrete Appl. Math.*, 143(1–3):130–143, Sept. 2004. ISSN 0166-218X. doi: 10.1016/j.dam.2003.10.007. URL https://doi.org/10.1016/j.dam.2003.10.007.

A Completing the Hypertree Packing Argument of Baïou and Barahona

In what follows, we use the notation of Chekuri et al. [8] and Baïou and Barahona [2]. We consider the α-approximate minimum k-cut problem for a rank-r hypergraph $H = (V, E, w)$.

Theorem A.1 (Baïou and Barahona [2])**.** *Let $(\bar{y}, \bar{z})$ be an optimal solution of the dual LP for k-cut. Let E' be any set of hyperedges such that $w(E') \leq \alpha\lambda$ for some $\alpha \geq 1$, where λ is the minimum k-cut value in H. For each hypertree T let $l_T = |E' \cap E(T)|$. Let $\tau = \sum_T \bar{y}_T$ and $p_T = \bar{y}_T/\tau$. For an integer $h \geq k-1$, let $q_h = \sum_{T: l_T \leq h} p_T$. Then*

$$q_h \geq 1 - \frac{r\alpha(k-1)\left(1 - \frac{1}{n}\right)}{h+1}.$$

The following completion of the hypertree packing argument follows the argument from Chekuri et al. [8] on tree packing in the graph case, but was left out of Baïou and Barahona [2].

Theorem A.2. *The number of α-approximate minimum k-cuts in a rank-r hypergraph is*

$$O((kn)^{\lfloor \alpha r(k-1) \rfloor}).$$

Proof. Set $h = \lfloor \alpha r(k-1) \rfloor$ and define q_h as in Theorem A.1. Note that q_h is the fraction of hypertrees in the packing induced by $(\bar{y}, \bar{z})$ that contain at most h hyperedges in A, the hyperedges induced by some specific α-approximate k-cut. Consider a hypertree T. There are at most n^h subsets of the hyperedges of T with cardinality at most h, and each subset induces at most k^{h+1} distinct k-partitions of V. So, there are at most $k^{h+1}n^h$ distinct k-cuts with at most h hyperedges

from T. It follows, by considering this bound on each hypertree in the packing, that there can be no more than

$$k^{h+1}n^h/q_h = O((kn)^h) = O((kn)^{\lfloor \alpha r(k-1) \rfloor})$$

α-approximate minimum k-cuts in H. □

B The Running Time of Baïou and Barahona's Algorithms

Let the *strength* of a hypergraph be the minimum of

$$\frac{\delta_H(C)}{|C|-1}$$

over all cuts C. Baïou and Barahona provide the algorithm for computing the strength:

Theorem B.1 (Baïou and Barahona [3])**.** *The strength of a hypergraph can be computed with n applications of the push-preflow algorithm on a graph with $O(m+n)$ vertices and $O(p)$ edges, where p denotes the sum of the sizes of all hyperedges.*

In their algorithm each push-preflow application solves n instances of an s-t min cut problem, so the following also holds:

Corollary B.2. *Computing the strength of a hypergraph reduces to solving n^2 instances of s-t max flow on a graph with $O(m+n)$ vertices and $O(p)$ edges.*

s-t max flow can be solved in $O(|E|^{1+o(1)})$ time, so the strength of a hypergraph can be computed in $O(n^2p^{1+o(1)})$ time. We now analyze the runtime of Algorithm 1 of Baïou and Barahona [2]. Let P_j be the final partition returned. The strength of each set in one of the P_is must be computed for each $i < j$, which is $|\cup_{i<j} P_i|$ sets. Note that $|P_{i+1} - P_i| = |P_{i+1}| - |P_i| + 1 \leq 2(|P_{i+1}| - |P_i|)$. Thus,

$$|\cup_i P_i| = 1 + \sum_{i=0}^{j-2} |P_{i+1} - P_i| \leq 1 + 2|P_{j-1}| < 2k.$$

Thus, their runtime is as follows:

Theorem B.3. *There exists an algorithm computing an r-approximation to minimum k-cut in $O(kn^2p^{1+o(1)})$ time for hypergraphs of rank r.*

Baïou and Barahona also provide an algorithm for computing the minimum hypertree:

Theorem B.4 (Baïou and Barahona [3])**.** *In a rank r hypergraph, computing the minimum hypertree reduces to solving m instances of s-t max flow on a graph with $O(n)$ vertices and $O(nr)$ edges.*

Each s-t max flow problem requires $O(rn^{1+o(1)})$ time, so the total running time is $O(mrn^{1+o(1)})$.