Minimum Cut and Minimum k-Cut in Hypergraphs via Branching Contractions

KYLE FOX, The University of Texas at Dallas, USA
DEBMALYA PANIGRAHI, Duke University, USA
FRED ZHANG, UC Berkeley, USA

On hypergraphs with m hyperedges and n vertices, where p denotes the total size of the hyperedges, we provide the following results:

- We give an algorithm that runs in $\widetilde{O}(mn^{2k-2})$ time for finding a minimum k-cut in hypergraphs of arbitrary rank. This algorithm betters the previous best running time for the minimum k-cut problem, for $k > 2$.
- We give an algorithm that runs in $\widetilde{O}(n^{\max\{r,2k-2\}})$ time for finding a minimum k-cut in hypergraphs of constant rank r. This algorithm betters the previous best running times for both the minimum cut and minimum k-cut problems for dense hypergraphs.

Both of our algorithms are Monte Carlo, i.e., they return a minimum k-cut (or minimum cut) with high probability. These algorithms are obtained as instantiations of a generic *branching randomized contraction* technique on hypergraphs, which extends the celebrated work of Karger and Stein on recursive contractions in graphs. Our techniques and results also extend to the problems of minimum hedge-cut and minimum hedge-k-cut on hedgegraphs, which generalize hypergraphs.

CCS Concepts: • **Theory of computation → Network optimization**; **Graph algorithms analysis;**

Additional Key Words and Phrases: Hypergraph cut, minimum cut, hypergraph connectivity, k-cut

ACM Reference format:
Kyle Fox, Debmalya Panigrahi, and Fred Zhang. 2023. Minimum Cut and Minimum k-Cut in Hypergraphs via Branching Contractions. *ACM Trans. Algor.* 19, 2, Article 13 (April 2023), 22 pages.
https://doi.org/10.1145/3570162

1 INTRODUCTION

In this article, we design a randomized branching framework for finding minimum cut and minimum k-cut in hypergraphs, leading to improvements in randomized algorithms for these problems over recent results of Ghaffari et al. [12] and Chandrasekaran et al. [6]. Our algorithms can be

Kyle Fox work was supported in part by NSF Grant No. CCF-1527084.
Debmalya Panigrahi work was supported in part by NSF Grants No. CCF-1527084, No. CCF-1535972, and NSF CAREER Award No. CCF-1750140.
Authors' addresses: K. Fox, The University of Texas at Dallas, Dallas, USA; email: kyle.fox@utdallas.edu; D. Panigrahi, Duke University, Durham, USA; email: debmalya@cs.duke.edu; F. Zhang, UC Berkeley, Berkeley, USA; email: z0@berkeley.edu.

1549-6325/2023/04-ART13 $15.00
https://doi.org/10.1145/3570162

thought of as the natural analog in hypergraphs of the celebrated recursive contraction algorithm of Karger and Stein [20] for finding minimum cut and minimum k-cut in graphs.

1.1 Randomized Contractions in Graphs

Consider an undirected graph with m edges on n vertices. A ***minimum cut*** (or min-cut) of the graph is a minimum weight set of edges whose removal partitions the vertices into two connected components. More generally, a ***minimum k-cut*** (or min-k-cut) is a minimum weight set of edges whose removal partitions the vertices into at least k connected components; a min-cut is simply a min-k-cut for $k = 2$. Min-cut and min-k-cut are fundamental problems in combinatorial optimization and network reliability, and an impressive body of work has been devoted to designing algorithms for these problems. Perhaps the simplest, and one of the most elegant, among them is the randomized contraction algorithm of Karger [18]. For min-cut, this algorithm selects edges at random with probability proportional to their weight (uniformly, if the graph is unweighted) and unifies their endpoints (called ***edge contraction***) until only two vertices remain. The edges that survive at this stage are then output by the algorithm. Karger argues that the algorithm returns a min-cut with probability $\Omega(1/n^2)$, which implies that a min-cut can be found with high probability by running $O(n^2 \log n)$ independent instances of the algorithm.[1] For min-k-cut, the algorithm is exactly identical, except that it terminates when k vertices are left, and outputs all the surviving edges. In this case, the probability of returning a min-cut is $\Omega(1/n^{2k-2})$, which can be similarly boosted by running $O(n^{2k-2} \log n)$ independent instances of the algorithm.

Unfortunately, performing these independent runs can take $\Omega(n^4)$ time for the min-cut algorithm and $\Omega(n^{2k})$ time for min-k-cut, since the number of edges m can be $\Omega(n^2)$. In a later work, Karger and Stein [20] proposed a *branching* approach called *recursive contraction* to significantly speed up this scheme. This algorithm exploits the fact that the probability of contracting a min-cut edge is rather small initially, and only increases in the later stages of the contraction algorithm. So, they proposed to run a single branch of the contraction algorithm until $\lceil n/\sqrt{2} \rceil$ vertices remain, at which stage *two* identical copies of the partially contracted graph are created and independent recursive runs of the branching algorithm are performed on these copies. While it does take time to create and work with these extra copies, the total number of edges in the graph decreases fast enough that the branching algorithm runs in $O(n^2 \log n)$ time and returns a min-cut with probability $\Omega(1/\log n)$. Using $O(\log^2 n)$ independent runs, one can now obtain a total running time of only $O(n^2 \log^3 n)$. With slight modifications, this algorithm can instead return a min-k-cut with high probability in $O(n^{2k-2} \log^3 n)$ time, and this min-k-cut algorithm remains the fastest one known for any constant $k > 2$ when the edges have arbitrary weights.

1.2 Randomized Contractions in Hypergraphs

A ***hypergraph*** H consists of a vertex set V and a set of ***hyperedges*** E. Each hyperedge $e \in E$ is itself an arbitrary subset of vertices. The ***size*** of a hyperedge e is $|e|$, the cardinality of its vertex set. The ***weight*** of a hyperedge e is denoted $w(e)$. For a subset of hyperedges S, let $w(S)$ denote the total weight of hyperedges in the subset. The ***rank*** r of the hypergraph H is the maximum size of a hyperedge, i.e., $r = \max_{e \in E} |e|$. A graph is simply a hypergraph of rank 2. Accordingly, we may say a hyperedge e is incident to a vertex v if $v \in e$. We generally use the notation $p = \sum_{e \in E} |e|$ to denote the total size of all hyperedges in H. Finally, we define the ***order*** of H as the number of vertices $|V|$. As before, a *minimum cut* (or min-cut) is a minimum weight set of hyperedges whose removal partitions the vertices into at least two connected components, and a *minimum k-cut* (or min-k-cut) is a minimum weight set of hyperedges whose removal partitions the vertices into at least k connected components.

[1] We say an event occurs with high probability if it occurs with probability at least $1 - 1/n^{\Omega(1)}$.

A natural approach for computing min-cuts and min-k-cuts in hypergraphs is to try applying the (non-branching) randomized contraction approach of Karger [18] described above. We let H/e denote the ***contraction*** of hyperedge e in hypergraph H, accomplished by unifying e's endpoints and removing hyperedges that now have size 1. Unfortunately, even if hyperedges are unweighted, contracting a hyperedge chosen uniformly at random may destroy a min-cut with fairly high probability when the hyperedges have variable sizes. Indeed, Kogan and Krauthgamer [24] show that this natural strategy finds a min-cut with probability exponentially small in the rank r. Recently, Ghaffari, Karger, and Panigrahi [12] showed how to achieve Karger's $\Omega(1/n^2)$ probability of success by biasing the selection of hyperedges away from those of large size. Chandrasekaran, Xu, and Yu [6] refined this strategy and generalized it to work for k-cuts as well. This leads to randomized algorithms for min-cut and min-k-cut that run in $O(pn^2 \log^2 n)$ and $O(pn^{2k-1} \log n)$ time, respectively. This latter result was the first polynomial time algorithm for minimum k-cut in hypergraphs for arbitrary constant k, and it remains the fastest result to date. In particular, no polynomial time deterministic algorithm is known for any $k > 3$. In contrast, there are deterministic $O(pn+n^2 \log n)$ time algorithms for the min-cut problem in arbitrary hypergraphs [23, 25] and an $O(pmn^3)$ time algorithm for $k = 3$ [32] using other techniques.

Hence, it is reasonable to ask whether we can generalize the *branching* contraction approach of Karger and Stein [20] to hypergraphs as well, to yield faster randomized algorithms for the min-cut and min-k-cut problems. There appear to be two major barriers to doing so.

First, the analysis of Karger and Stein's algorithm depends upon the branching occurring at precisely chosen moments determined by the number of vertices in the partially contracted graph. If one branches too early, then there are too many large graphs in the recursive calls, and the running time increases. If one branches too late, then it decreases the probability of a min-cut surviving beyond the branching point and to the end of a run of the algorithm. When attempting to run the branching algorithm on a hypergraph, there is a risk of contracting a very large hyperedge and skipping right past one or more of these carefully selected branching points. Thus, it appears we cannot select the branching orders in advance of our randomized contractions, and we need a *smoother* procedure that performs branches based on which hyperedges we are contracting.

The second barrier to generalizing Karger and Stein's algorithm concerns the total size p of hyperedges in an arbitrary hypergraph. While we can easily see $p = O(n^r)$ in hypergraphs of rank r, the total size could be much larger when the rank is not fixed. Any generalization of Karger and Stein's algorithm to hypergraphs must be careful to minimize the time spent interacting with hyperedges during contractions, especially in later stages of the algorithm when the number of hypergraph copies is large.

1.3 Our Results

We present a new randomized branching framework for hypergraphs, where instead of branching at fixed graph orders, we *probabilistically* decide whether to branch before each contraction based on the size of the hyperedge we (randomly) select to contract. Our framework allows us to derive new randomized algorithms for computing min-cuts and min-k-cuts in hypergraphs.

For hypergraphs of arbitrarily rank, we show the following result:

Theorem 1.1. *There exists an algorithm that with high probability computes a minimum k-cut of a weighted hypergraph with n vertices and m hyperedges in $O(mn^{2k-2} \log^2 n)$ time for all $k \geq 3$. For the special case of minimum cut ($k = 2$), our algorithm runs in $O(mn^2 \log^3 n)$ time.*

This result improves upon the best randomized running times for both problems due to Chandrasekaran et al. [6] by a factor of $\widetilde{\Omega}(pn/m)$, which is at least $\widetilde{\Omega}(n)$ and at most $\widetilde{O}(n^2)$.[2] For $k > 2$,

[2]We use $\widetilde{O}$ notation to hide polylog factors when they are either too onerous to write or the exact exponents on the polylog are unclear.

this yields the fastest algorithm for the min-k-cut problem to date. (For $k = 2$, i.e., the min-cut problem, the best running time is $O(pn + n^2 \log n)$ [23, 25] due to deterministic algorithms. See Section 1.6 for more details.)

Dense Hypergraphs. Since the number of hyperedges m can be $\Omega(n^r)$, the above result yields running times of $O(n^{r+2} \log^3 n)$ for min-cut and $O(n^{r+2k-2} \log^2 n)$ for min-k-cut ($k \geq 3$) for dense hypergraphs of constant rank r. We improve on these results by the showing that for any constant rank r:

Theorem 1.2. *There exists an algorithm that with high probability computes a minimum k-cut of a weighted hypergraph with n vertices, constant rank r, and total hyperedge size p in $O(p+n^{2k-2} \log^3 n)$ time if $r = 2k-2$ or $O(p + n^{\max\{r,2k-2\}} \log^2 n)$ time otherwise. For the special case of minimum cut ($k = 2$) with $r > 2$, our algorithm runs in $O(p + n^r \log^2 n)$ time.*

For both minimum cut and minimum k-cut, these time bounds improve upon the best ones known for sufficiently dense hypergraphs of rank $r > 2$.

1.4 Extension to Hedgegraphs

Along with studying randomized contractions for hypergraph min-cut, Ghaffari, Karger, and Panigrahi [12] also initiated the study of *minimum hedge-cuts*. A hedgegraph $H = (V, E)$ consists of a collection of vertices V and a collection of disjoint ***hedges*** E, each of which is a subset of edges over the vertex set V. A ***hedge-cut*** is a collection of hedges whose removal separates H into at least two components; a ***hedge-k-cut*** is a hedge-cut that separates H into at least k components. A min-hedge-cut or min-hedge-k-cut is one of minimum total hedge weight. Ghaffari, Karger, and Panigrahi [12] provide a randomized polynomial time approximation scheme and an exact quasi-polynomial time algorithm for computing a min-hedge-cut. The ***span*** of a hedge is the number of connected components in the subgraph induced by its edge set. The randomized contraction result of Chandrasekaran, Xu, and Yu [6] for hypergraph k-cut is actually a special case of a randomized algorithm they give for hedgegraphs, where every hedge has span bounded by a constant s. (Note that a hypergraph is simply a hedgegraph of unit span.) Their general algorithm runs in $O(mpn^{ks+k-s} \log n)$ time where m denotes the number of hedges and p their total size.

Our results for computing min-cuts and min-k-cuts in hypergraphs extend to hedgegraphs of bounded span. With high probability, we can find a min-hedge-cut in $O(m^2 n^{s+1} \log^2 n)$ time and a min-hedge-k-cut in $O(m^2 n^{(s+1)(k-1)} \log^2 n)$ time, improving upon known results for all $k \geq 2$ and $s \geq 2$. Let the ***rank*** of a hedgegraph be the maximum number of vertices incident to edges in any one hedge. For constant rank r, we can find a min-hedge-k-cut with high probability in $O(p + n^{(s+1)(k-1)} \log^3 n)$ time if $r = (s+1)(k-1)$ or $O(p + n^{\max\{r,(s+1)(k-1)\}} \log^2 n)$ time otherwise.

1.5 Techniques and Organization

As mentioned above, our algorithm relies on *probabilistic branching* instead of branching at fixed graph orders like in the original method of Karger and Stein [20]. Instead of explicitly biasing away from larger hyperedges like in previous work [6, 12], we pick hyperedges to contract with probability proportional to their weight. Before contraction, however, we branch with a probability that is dependent on the size of the hyperedge we just selected. These probabilities are carefully selected so that the total probability of branching is at least as high as the probability of contracting an edge in the min-cut. Intuitively, the *expected* number of min-cuts discovered by our algorithm at the bottom of its branches ends up being at least 1. Through a relatively straightforward inductive argument, we prove that any one run of our algorithm returns a min-cut with probability at least $\Omega(1/\log n)$. In fact, the analysis of our algorithm is arguably simpler than that given for Karger

and Stein's [20], and we give a probability bound that is actually *tight* for min-cut in graphs. Our high-level algorithm and the analysis bounding its probability of success appear in Section 2.

To give our specific running time bounds, we need to perform hyperedge contractions efficiently. We first consider arbitrary rank hypergraphs. In earlier work [6, 12], the contraction of a hyperedge e involved going through the incidence list of each vertex in e so that all the incidences could be replaced with a single vertex representing the contracted edge e. This process takes $\Theta(m \cdot |e|)$ time in the worst case, so the running time of contraction increases with $|e|$. Unfortunately, our algorithm is also more likely to branch when $|e|$ is large, increasing our chances of performing another expensive contraction in the future. Therefore, instead of spending $\Theta(m \cdot |e|)$ time checking incidences, we essentially make brand new copies of what remains of each edge after contraction in $O(m(n - |e| + 1))$ time so contractions that are more likely to cause branching are also cheaper to perform. We describe the implementation for arbitrary rank hypergraphs in Section 3.

For hypergraphs of constant rank r, we only make copies of the hypergraph when branching, and we otherwise perform contractions using a similar procedure as that given by Karger and Stein [20] for graphs. Specifically, we explicitly merge new parallel hyperedges after every contraction to guarantee we never store more than $O(n^r)$ hyperedges at a time. We also generalize their edge selection procedure to select and contract a hyperedge in $O(n^{r-1})$ time. Our implementation for constant rank hypergraphs appears in Section 4.

For hedgegraphs, randomly contracting a hedge incident to x vertices induces a branch with the same probability as contracting a hyperedge of size x. The key difference increasing the running time for hedgegraphs is that we might create more branches overall, because a hedgegraph only shrinks by $x - s$ vertices if the contracted hedge has span s. Note that s can be as large as $x/2$ in general, thus guaranteeing a reduction of only $x/2$ vertices, whereas shrinking a hyperedge of size x reduces the number of vertices by $x - 1$. In addition, we need to maintain information about how the components of hedges share vertices to perform fast hedge contractions, and maintaining this information incurs additional running time. For simplicity, we stick to hypergraph min-cut and min-k-cut in most of the article before giving the necessary details for our extensions to hedgegraphs in Section 5.

1.6 Related Work

We give a brief survey of what is known about computing minimum cuts and k-cuts beyond the approaches of Karger and Stein [18, 20]. Nagamochi and Ibaraki [27] gave the first min-cut algorithm for graphs that does not perform repeated computations of minimum s, t-cuts. Their approach was later refined to give $O(mn + n^2 \log n)$ deterministic algorithms for the problem, and this running time remains the best known among deterministic algorithms [26, 29]. This running time can be improved to $O(m+\lambda n^2)$ [26] in unweighted graphs with min-cut value λ and to $O(m \log^{12} n)$ in simple unweighted graphs [22]. The randomized $O(n^2 \log^3 n)$ time algorithm of Karger and Stein [20] improves upon the weighted edge result for graphs that are not too sparse. By applying other techniques, Karger [19] further improved the best randomized running time to $O(m \log^3 n)$.

When k is part of the input, the min-k-cut problem is NP-hard [13]. However, Goldschmidt and Hochbaum [13] were still able to give a deterministic $n^{O(k^2)}$ time algorithm for the problem when k is a constant. Their work was followed by a series of deterministic algorithms [16, 17, 31] that led to an $\widetilde{O}(n^{2k})$ time algorithm due to Thorup [30]. In concurrent work to our own, Chekuri, Quanrud, and Xu [8] improved this running time to $\widetilde{O}(mn^{2k-3})$. This was further improved to $O(k^{O(k)} n^{(2\omega/3+o(1))k})$ by a recent work of Gupta, Lee, and Li [14], where ω is the matrix multiplication constant. For small k, the result also gives a faster randomized algorithm in time $\widetilde{O}(k^{O(k)} n^{k+\lfloor (k-2)/3 \rfloor \omega+1+((k-2) \bmod 3)} W)$, where all weights are bounded by W. For unweighted

graphs and $k \in [8, n^{o(1)}]$, this improves upon the randomized $O(n^{2k-2} \log^3 n)$ time algorithm of Karger and Stein [20], while the latter still remains the state-of-the-art for weighted graphs.

The situation in hypergraphs is a bit different. Queyranne [28] gave the first non-trivial algorithm for min-cut in hypergraphs, and it was later improved by Klimmek and Wagner [23] and Mak and Wong [25] to run in $O(np + n^2 \log n)$ time. Chekuri and Xu [9] improved the running time for unweighted hypergraphs with min-cut value λ to $O(p + \lambda n^2)$. All of these algorithms are deterministic. In fact, all the randomized algorithms mentioned above are actually *slower* at computing min-cuts than these deterministic algorithms, with the aforementioned exception of our $O(p + n^r)$ time algorithm for sufficiently dense hypergraphs of constant rank.

For $k = 3$, there exists a deterministic $O(pmn^3)$ time min-k-cut algorithm due to Xiao [32, Theorem 4.1]. Fukunaga gave a deterministic $O(n^{rk+O(1)})$ time algorithm for any constant k and constant rank r via a hypertree packing approach [11]. In these cases of $k > 2$, we see the randomized algorithm of Chandrasekaran, Xu, and Yu [6] and our own results do provide improvements over the deterministic algorithms, and the randomized approach is the only one known to work with arbitrarily large hyperedges.

Based on the history above, it seems likely that randomized approaches will eventually lead to even faster algorithms for both min-cut and min-k-cut in hypergraphs. We believe our work is a natural and important step toward this end.

1.7 Subsequent Work

Subsequent to the the conference publication of our work [10], the hypergraph minimum cut and related problems were studied in References [1–5, 7, 21]. Karger and Williamson [21] provided a streamlined analysis of our algorithm in the case of graph minimum cut. Gupta, Lee, and Li show that the Karger-Stein algorithm is optimal for graph k-cut [15]. Chandrasekaran and Chekuri [4] gave the first deterministic polynomial-time algorithm for hypergraph k-cut, for fixed k. Their approach significantly differs from ours, as it is based on a divide-and-conquer procedure and does not rely on (randomized) contractions. Beideman et al. [1] provided a faster algorithm for hypergraph minimum cut on low-rank simple hypergraphs. Beideman et al. [2, 3] study counting and enumerating optimal hypergraph cuts. Chandrasekaran and Chekuri [5] considered the problem of min-max hypergraph k-cut and gave a polynomial-time algorithm. Finally, Chekuri and Quanrud [7] devised an $\widetilde{O}(\sqrt{pn}(m+n)^{1.5})$ time algorithm for hypergraph min-cut.

2 BRANCHING CONTRACTIONS

In this section, we provide the high-level details of our branching randomized contraction algorithm. Recall, a non-branching algorithm that simply selects hyperedges to contract proportional to their weight will preserve a min-cut with probability exponentially small in the rank of the given hypergraph [24]. Therefore, previous work biases edge selection away from large hyperedges [6, 12].

For our *branching* contraction algorithm, though, we must also use the size of hyperedges to determine the number of branches we create. The min-cut includes every hyperedge except e. Because all the hyperedges have the same size and nearly the same weight, any randomized contraction procedure will preserve the min-cut after one contraction with probability only $\Theta(1/n)$. Recall that the goal of branching contractions is to preserve the min-cut with probability $\Theta(1/\log n)$; we clearly need $\Omega(n/\log n)$ branches at the very outset of the algorithm. Contrast this with graphs, where the algorithm branches much slower, creating two branches after $n(1 - 1/\sqrt{2})$ edge contractions. Since a hypergraph comprises hyperedges of non-uniform size, the number of branches created therefore needs to vary with the size of the hyperedge chosen for contraction.

Simultaneously biasing selection away from large hyperedges *and* deciding how often to branch based on the size of selected hyperedges makes the analysis very complicated. Instead, we encode the bias against large hyperedges into the probability of branching or not. More concretely, our algorithm selects hyperedges to contract proportionally to their weight just as in graphs [19, 20], but before contracting, it randomly decides to branch with the probability of branching being a function of the selected hyperedge's size. In other words, instead of biasing the hyperedge selected for contraction away from large hyperedges, we branch with a higher probability if a large hyperedge is selected thereby creating a copy of the uncontracted graph and effectively increasing the probability of preserving the min-cut.

Intuitively, a high probability of contracting leads to a large chance of destroying a min-cut. However, a high probability of branching may give a large number of branches, resulting in prohibitive time complexity. Our algorithm is designed to balance these two probabilities such that *in expectation*, one copy of the min-cut survives either in the contracted graph or in the copy of the uncontracted graph we create by branching, and yet the total number of branches can be well controlled.

We begin by determining the probability that a random hyperedge, selected with probability proportional to its weight, belongs to a given min-cut or min-k-cut. Recall, for a given hypergraph $H = (V, E)$ and set of hyperedges $S \subseteq E$, we use $w(S)$ to denote the total weight of S. The following lemma is essentially a restating of one by Chandrasekaran, Xu, and Yu [6, Lemma 3.1] for the case of hypergraph min-k-cut.

Lemma 2.1. *Let C be a minimum k-cut in a hypergraph $H = (V, E)$. We have*

$$\frac{w(C)}{w(E)} \leq \frac{1}{w(E)} \sum_{e \in E} w(e) \left(\frac{\binom{n}{k-1} - \binom{n-|e|}{k-1}}{\binom{n}{k-1}} \right).$$

Proof. Fix a subset $A = \{a_1, \ldots, a_{k-1}\}$ of $k-1$ vertices. Subset A induces a k-cut $C' = (\{a_1\}, \ldots, \{a_{k-1}\}, V \setminus A)$ of weight at least $w(C)$. Each hyperedge $e \in E$ that includes at least one member of A contributes weight $w(e)$ to C'. There are $\binom{n}{k-1}$ such subsets A total, and each hyperedge e contributes weight to the cuts of $\binom{n}{k-1} - \binom{n-|e|}{k-1}$ of them, so $\sum_{e \in E} w(e) \left(\binom{n}{k-1} - \binom{n-|e|}{k-1} \right) \geq \binom{n}{k-1} w(C)$. Multiplying both sides of the inequality by $1/\left(w(E) \binom{n}{k-1} \right)$ proves the lemma. □

Because the expression will arise often throughout the remainder of this article, we define

$$z_n(e) = \frac{\binom{n}{k-1} - \binom{n-|e|}{k-1}}{\binom{n}{k-1}} \quad \text{for any hyperedge } e \in E.$$

We call $z_n(e)$ the ***redo probability*** of contracting hyperedge e in a hypergraph of order n. If our branching algorithm randomly selects hyperedge e for contraction, then it will branch with probability precisely $z_n(e)$. Observe $\frac{w(C)}{w(E)} \leq \frac{1}{w(E)} \sum_{e \in E} w(e) \cdot z_n(e)$. Therefore, as we select a hyperedge to contract proportionally to its weight, our algorithm branches with the same probability that we contract a hyperedge in a min-cut or min-k-cut.

Before giving the full details of our branching algorithm, as an aside, we reinterpret the algorithm of Chandrasekaran, Xu, and Yu [6] in our framework. Suppose we select a hyperedge e to contract proportional to its weight and, instead of *branching* with probability $z_n(e)$, we *redo* the random hyperedge selection with probability $z_n(e)$. In other words, *if we branch*, then we ignore the originally selected hyperedge e and only pay attention to what happens in the new branch. In this case, the probability that we select a hyperedge e and perform its contraction instead of

branching is proportional to $w(e) \cdot (1 - z_n(e)) = \frac{w(e)\binom{n-|e|}{k-1}}{\binom{n}{k-1}}$, which is exactly proportional to the (non-uniform) probability of contraction of hyperedge e in the algorithm of Chandrasekaran, Xu, and Yu [6]. Indeed, this view gives a very short analysis for their algorithm (see Appendix A).

2.1 Algorithm Details

We now give the details for our branching contraction algorithm for min-k-cut in a hypergraph $H = (V, E)$ of order n. The algorithm for min-cut is the special case for $k = 2$. As we show, a single run of our algorithm has (within constant factors) the same probability of finding a min-k-cut as in the branching algorithm of Karger and Stein [20] for graphs.

We say a hyperedge e is ***k-spanning*** if it contains at least $n - k + 2$ vertices, implying it spans every k-cut. We maintain a collection $S \subseteq E$ of hyperedges that we believe belong to the min-k-cut based on earlier randomized contractions. Initially, every k-spanning hyperedge is added to S. Then, we sample a random hyperedge e with probability proportional to its weight (or uniformly at random if the hypergraph is unweighted). We contract the hyperedge e to create the hypergraph H/e, and recursively call our algorithm on H/e. With probability $z_n(e)$, we also *branch*, meaning we recursively call our algorithm on the same hypergraph H. If we branched, then we return the smaller of the cuts returned for H/e and our second run on H. Otherwise, we return the cut found for H/e. The algorithm is given in pseudo-code below as the procedure BranchingContract(H, S). It initially takes the empty set as its second parameter.

BranchingContract(H, S):
For each k-spanning hyperedge e, add e to S and remove e from H
If $E = \emptyset$, then return S

Select a hyperedge e at random with probability proportional to $w(e)$
With probability $z_n(e)$,
    return the smaller of the cuts BranchingContract$(H/e, S)$ and BranchingContract(H, S)
Otherwise, return BranchingContract$(H/e, S)$

2.2 Computation Tree

Consider a single run of BranchingContract on a hypergraph H. We define the ***computation tree*** of the run as follows. The computation tree contains as a node the input hypergraph H and each hypergraph created by a contraction (including multiple nodes for identical hypergraphs created by separate branched contractions). It also contains a special root node not associated with any particular hypergraph. The parent of hypergraph H' is the hypergraph from whose contraction H' was created, and the parent of input hypergraph H is the root node. Last, we remark that the non-contracting branch does not create a new node.

A computation tree can be defined similarly for Karger and Stein's [20] recursive contraction algorithm; here the root node contains *two* children, one for each of the two independent runs on the input graph. However, while this computation tree is *deterministic* in its structure and graph orders of its nodes, the structure and hypergraph orders of our computation tree are themselves *randomized*. In particular, BranchingContract may not yield an isomorphic computation tree to that of the algorithm of Karger and Stein when given a graph as input.

The following lemma bounds the number of hypergraphs of various orders in our computation tree and will prove useful in the analysis of our implementations of BranchingContract. Let $S(n, n_0)$ denote the maximum expected number of nodes of order n_0 created in a computation tree for some hypergraph H of order n, where the maximum is over all hypergraphs of order n.

Lemma 2.2. *Fix integer n_0 such that $k \le n_0 \le n$. We have*

$$S(n, n_0) \le \frac{\binom{n}{k-1}\binom{n-1}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}} = O\left(\left(\frac{n}{n_0}\right)^{2k-2}\right).$$

On graphs, this bound is met with equality.

Proof. We will prove the lemma using induction on n. For $n = n_0$, we have $S(n, n_0) = 1$, because only the root's single-child node has order n. The lemma follows.

Now, suppose $n > n_0$. Fix a hypergraph H of order n. Let $E' \subseteq E$ be the subset of hyperedges such that $|e| \le n - n_0 + 1$. Only contracting a hyperedge of E' can lead to a node of order n_0. Given that such a hyperedge is contracted, any given $e \in E'$ is chosen with probability $\frac{w(e)}{w(E')}$. And conditioned on e being chosen, there will be in expectation at most $S(n-|e|+1, n_0)$ descendants of H of order n_0 formed by calling BranchingContract on H/e. Also, the algorithm will branch with probability $z_n(e)$ and that branch will create at most $S(n, n_0)$ additional descendants in expectation. Thus

$$S(n, n_0) \le \frac{1}{w(E')} \sum_{e \in E'} w(e) \left(z_n(e) \cdot S(n, n_0) + S(n - |e| + 1, n_0)\right).$$

Solving for $S(n, n_0)$, using the definition of $z_n(e)$, and applying the inductive hypothesis, we find

$$\begin{aligned} S(n, n_0) &\le \frac{1}{1 - \frac{1}{w(E')}\sum_{e \in E'} w(e) z_n(e)} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e)\, S(n - |e| + 1, n_0) \\ &= \frac{\binom{n}{k-1}}{\frac{1}{w(E')}\sum_{e \in E'} w(e)\binom{n-|e|}{k-1}} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e)\, S(n - |e| + 1, n_0) \\ &\le \frac{\binom{n}{k-1}}{\frac{1}{w(E')}\sum_{e \in E'} w(e)\binom{n-|e|}{k-1}} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e) \left(\frac{\binom{n-|e|+1}{k-1}\binom{n-|e|}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}}\right). \end{aligned}$$

Finally, we observe $\binom{n-|e|+1}{k-1} \le \binom{n-1}{k-1}$ and simplify to prove the lemma.

$$S(n, n_0) \le \frac{\binom{n}{k-1}}{\frac{1}{w(E')}\sum_{e \in E'} w(e)\binom{n-|e|}{k-1}} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e) \left(\frac{\binom{n-1}{k-1}\binom{n-|e|}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}}\right) = \frac{\binom{n}{k-1}\binom{n-1}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}}. \qquad \square$$

2.3 Probability of Success

We now establish the probability of BranchingContract returning a min-k-cut of hypergraph $H = (V, E)$. For any integer $t \ge k$, let

$$z_t^* = \frac{\binom{t}{k-1} - \binom{t-2}{k-1}}{\binom{t}{k-1}}.$$

Theorem 2.3. *Fix a constant integer $k \ge 2$. Algorithm BranchingContract returns a min-k-cut with probability at least*

$$\frac{1}{\sum_{t=k}^{n} z_t^*} = \Omega\left(\frac{1}{\log n}\right).$$

To see that $\sum_{t=k}^{n} z_t^* = O(\log n)$, notice that each term $z_t^* = \Theta(1/t)$, since the numerator is $O(t^{k-2})$ whereas the denominator is $O(t^{k-1})$. Now to prove the theorem, we need the following observation. In the proof of Theorem 2.3, we only apply it with $w = 1$, but the general statement given below will be useful in the analysis of our hedgegraph cut algorithm.

Observation 1. *Fix integers k, n, w, and x such that $2 \le k \le n-1$, $2 \le x \le n-k+2$, and $1 \le w \le x-1$. We have*

$$\frac{\binom{n}{k-1} - \binom{n-x}{k-1}}{\binom{n}{k-1}} \le \binom{x-1}{x-w} \sum_{t=n-x+w+1}^{n} z_t^*,$$

with equality holding when $w = 1$ and $x = 2$.

Proof (of Observation 1). We express the left-hand side as a telescoping sum and increase the numerators in its terms:

$$\begin{aligned}
\frac{\binom{n}{k-1} - \binom{n-x}{k-1}}{\binom{n}{k-1}} &= \frac{\binom{n}{k-1} - \binom{n-2}{k-1}}{\binom{n}{k-1}} + \sum_{t=n-x+2}^{n-1} \frac{\binom{t-1}{k-1} - \binom{t-2}{k-1}}{\binom{n}{k-1}} \\
&\le \frac{\binom{n}{k-1} - \binom{n-2}{k-1}}{\binom{n}{k-1}} + \sum_{t=n-x+2}^{n-1} \frac{\binom{t}{k-1} - \binom{t-2}{k-1}}{\binom{n}{k-1}} \\
&= \sum_{t=n-x+2}^{n} \frac{\binom{t}{k-1} - \binom{t-2}{k-1}}{\binom{n}{k-1}}.
\end{aligned}$$

Now, observe that $\binom{t}{k-1} - \binom{t-2}{k-1}$ is increasing in t. We replace the smaller terms of the summation with a multiple of the larger terms. More precisely, if the $w-1$ smallest ones among the $x-1$ terms are taken out, the summation is left with $x-w$ terms. Since they are larger than the $w-1$ small terms, multiplying them by a factor of $(x-1)/(x-w)$ suffices to maintain the inequality. Finally, we reduce the denominators to finish the proof:

$$\begin{aligned}
\frac{\binom{n}{k-1} - \binom{n-x}{k-1}}{\binom{n}{k-1}} &\le \binom{x-1}{x-w} \sum_{t=n-x+w+1}^{n} \frac{\binom{t}{k-1} - \binom{t-2}{k-1}}{\binom{n}{k-1}} \\
&\le \binom{x-1}{x-w} \sum_{t=n-x+w+1}^{n} \frac{\binom{t}{k-1} - \binom{t-2}{k-1}}{\binom{t}{k-1}} \\
&= \binom{x-1}{x-w} \sum_{t=n-x+w+1}^{n} z_t^*. \qquad \square
\end{aligned}$$

We are now ready to prove Theorem 2.3.

Proof (of Theorem 2.3). For any n-vertex hypergraphs H, let Z_H be a indicator random variable on whether BranchingContract returns a min-k-cut. Let $q_n = \min_H \Pr(Z_H = 1)$ denote the minimum probability over all n-vertex hypergraphs of BranchingContract returning a min-k-cut. We will use induction over n to prove that $q_n \ge \frac{1}{\sum_{t=k}^{n} z_t^*}$. For the base case, $q_k = 1 = 1/((k-0)/k)$.

We now establish a recurrence for q_n. Fix the hypergraph $H = (V, E)$, where $H \in \arg\min_H \Pr(Z_H = 1)$, and a min-$k$-cut C in H. The algorithm is correct to add all k-spanning hyperedges to S and eventually return them, because every k-spanning hyperedge crosses every k-cut. For simplicity, we assume E has no k-spanning hyperedges from here on. We further assume a min-k-cut is returned for H with probability *at most* $\frac{1}{\sum_{t=k}^{n} z_t^*}$. If this assumption does not hold, then the theorem is already true for hypergraph H.

Recall that when our algorithm selects a hyperedge e to contract, it also branches with probability exactly $z_n(e)$. The edge to contract is chosen with probability proportional to its weight,

so the probability of branching is exactly $\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e)$. Conditioned on the branch being performed, the *branched* call to BranchingContract(H,S) will then succeed in finding a min-k-cut with probability at least q_n. Taken together, we see with probability at least $(\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e))q_n$, the algorithm BranchingContract(H,S) branches and then finds a min-k-cut during the recursive call to BranchingContract(H,S). Let $E' \subseteq (E \setminus C)$ be the subset of hyperedges that do not belong to *some* min-k-cut. By Lemma 2.1, there is at least a $(1-(\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e)))$ probability that the selected hyperedge belongs to E', implying at least one min-k-cut exists in H/e. Conditioned on the event, there is a $w(e)/w(E')$ probability of any *particular* hyperedge $e \in E'$ being chosen. Finally, given that $e \in E'$ is chosen, using our assumption that BranchingContract returns a min-k-cut with probability at most $\frac{1}{\sum_{t=k}^{n} z_t^*}$, we have at least a $(1-z_n(e)(\frac{1}{\sum_{t=k}^{n} z_t^*}))q_{n-|e|+1}$ probability that a min-k-cut is returned by BranchingContract$(H/e,S)$ but no min-k-cut is *also* returned during a branching call to BranchingContract(H,S). Taking everything together, we see

$$q_n \geq \left(\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e)\right)q_n + \left(1-\left(\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e)\right)\right)\frac{1}{w(E')}\sum_{e\in E'} w(e)\left(1-z_n(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\right)q_{n-|e|+1}.$$

Solving for q_n and rearranging some terms, we see

$$\begin{aligned} q_n &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(1-z_n(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\right)q_{n-|e|+1} \\ &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\left(\left(\sum_{t=k}^{n} z_t^*\right)-z_n(e)\right)q_{n-|e|+1}. \end{aligned}$$

For each $e \in E'$, we use the definition of $z_n(e)$ and apply Observation 1 with $w=1$ and $x=|e|$:

$$\begin{aligned} q_n &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\left(\left(\sum_{t=k}^{n} z_t^*\right)-\frac{\binom{n}{k-1}-\binom{n-|e|}{k-1}}{\binom{n}{k-1}}\right)q_{n-|e|+1} \\ &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\left(\left(\sum_{t=k}^{n} z_t^*\right)-\left(\sum_{t=n-|e|+2}^{n} z_t^*\right)\right)q_{n-|e|+1} \\ &= \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\left(\sum_{t=k}^{n-|e|+1} z_t^*\right)q_{n-|e|+1}. \end{aligned}$$

Finally, we apply induction and simplify to finish the proof:

$$\begin{aligned} q_n &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right)\left(\sum_{t=k}^{n-|e|+1} z_t^*\right)\left(\frac{1}{\sum_{t=k}^{n-|e|+1} z_t^*}\right) \\ &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{\sum_{t=k}^{n} z_t^*}\right) \\ &= \frac{1}{\sum_{t=k}^{n} z_t^*}. \end{aligned}$$

□

Suppose H is an unweighted cycle graph and $k = 2$, and let C be a minimum cut in H. Assuming BranchingContract breaks ties in C's favor, it returns C with exactly the probability given in Theorem 2.3.

So far, we have established that the branching randomized contraction algorithm for finding min-cuts and min-k-cuts has an $\Omega(1/\log n)$ probability of success when given a hypergraph of order n. In the next two sections, we give efficient implementations of the algorithm for arbitrary and low-rank hypergraphs. It would be natural to boost this probability of success by repeated runs of the algorithm, but this process needs some care, because the running time of our algorithm is also randomized and the length of individual runs can be correlated with the success of the algorithm. This is in contrast with the algorithm of Karger and Stein [20] whose runtime is deterministic. In this case, they may simply run $\Theta(\log^2 n)$ independent copies of the branching contraction algorithm and returns the smallest cut found.

Our solution is the following. Suppose we implement our branching contraction algorithm to run in T time *in expectation*. We can find a minimum cut or k-cut with high probability in *worst case* $O(T \log^2 n)$ time by subtly modifying Karger and Stein's approach. We say that a ***batch*** is a set of $c \log n$ runs of our algorithm for some sufficiently large constant c. We perform a batch of independent runs by doing the runs sequentially until either all $c \log n$ runs are complete or $4cT \log n$ units of time have passed. (For example, if the first run takes longer than $4cT \log n$ units of time, then no other run in the batch is executed.) The result of the batch is *nothing* if the $4cT \log n$ time units have passed or the smallest cut found otherwise. Our overall minimum cut or k-cut algorithm then returns the smallest cut found over $c' \log n$ independent batches for some constant c'.

We now prove the following lemma.

Lemma 2.4. *Let H be a hypergraph of order n, and suppose* BranchingContract *is implemented to run in T time on H* in expectation. *Then, there is an algorithm that runs in* worst case $O(T \log^2 n)$ *time, and returns a min-k-cut of H with high probability.*

Proof. The running time bound follows immediately, so we concentrate on the probability bound. Consider a single batch of runs, and suppose we let them all run to completion. The probability that all $c \log n$ runs fail to find a min-k-cut is at most

$$1 - \left(1 - \Omega\left(\frac{1}{\log n}\right)\right)^{c \log n} \leq 1/4,$$

when c is sufficiently large. By Markov's inequality, our algorithm stops a batch for spending too much time with probability at most $1/4$. Therefore, by union bound, there is at most a $1/2$ probability that our algorithm fails to return a min-k-cut during a single batch. Finally, $c' \log n$ batches all fail to return a min-k-cut with probability at most

$$1 - (1/2)^{c' \log n} = 1 - 1/n^{c''}.$$

for some constant c''. □

3 ALGORITHM FOR HYPERGRAPHS OF ARBITRARY RANK

We now present our $\widetilde{O}(mn^{2k-2})$ time algorithm for hypergraphs of arbitrary rank. We represent a hypergraph $H = (V, E)$ as a list of hyperedges as well as pointers between vertices and incident hypergedges.

To select a hyperedge for contraction, we keep an explicit list of every hyperedge in the hypergraph. We check their weights and select a hyperedge proportional to its weight in $O(m)$ time.

Now, suppose we select a hyperedge e for contraction. Previous work [6, 12] and our own implementation for constant rank hypergraphs (Section 4) involves finding all hyperedges sharing a vertex with e and modifying those hyperedges to the newly contracted hypergraph. When $|e|$ is large, such a procedure is rather slow. Instead, we essentially create a new copy of the hypergraph during the contraction. However, we ensure that we only spend time proportional to the total size of the contracted hypergraph.

We contract e to create $H/e = (V', E')$ using the following procedure. We initially set $V' := V \setminus e$ and $E' := E \setminus \{e\}$ by creating copies of the individual vertices and hyperedges. However, we *do not* (yet) copy over the incidence lists for the vertices and hyperedges. For each vertex $v \in V'$ and for each hyperedge $e' \in E'$ that is incident to v in H, we add v to the incidence list for the copy of e' in $H \setminus e$ and vice-versa. We remove any hyperedges $e' \in E'$ that have no incidences, because they will all have size 1 after contraction. Next, we add an additional vertex v_e to V' representing the contraction of e. Any hyperedge $e' \in E'$ that is incident to fewer vertices in H/e than in H must have been incident to a vertex of e itself. Therefore, we add v_e to the incidence lists of those hyperedges and vice-versa. We spend constant time per hyperedge and incidence in H/e, so the entire procedure takes $O(m(n - |e| + 1))$ time.

One thing to note about the above procedure is that we make no modifications to H itself. Therefore, we do not need to spend any time copying H when we decide to branch. We can discover k-spanning hyperedges as they are created during contractions, so doing so adds no overhead to the running time. Our overall algorithm consists of running this implementation of BranchingContract $O(\log^2 n)$ times to boost the probability of success as in Lemma 2.4.

Given Lemma 2.2, it it straightforward to analyze our algorithm.

Lemma 3.1. *The expected running time of BranchingContract using our implementation for arbitrary hyperedge sizes is $O(mn^2 \log n)$ if $k = 2$ or $O(mn^{2k-2})$ if $k \geq 3$.*

Proof. The main bottleneck of the algorithm is the time taken to create recursive subproblems. It takes $O(mn_0)$ time to do these operations to create a subproblem of order n_0. By Lemma 2.2, the expected running time is therefore at most

$$\sum_{n_0=k}^{n} O\left(\left(\frac{n}{n_0}\right)^{2k-2}\right) \cdot O(mn_0) = O(mn^{2k-2}) \cdot \sum_{n_0=k}^{n} O\left(\frac{1}{n_0^{2k-3}}\right).$$

If $k = 2$, then the summation comes to $O(\log n)$. Otherwise, it converges to a constant. □

We apply Lemma 2.4 and obtain our main result for hypergraphs of arbitrary rank.

Theorem 3.2 (Restatement of Theorem 1.1). *There exists an algorithm that with high probability computes a minimum k-cut of a weighted hypergraph with n vertices and m hyperedges in $O(mn^2 \log^3 n)$ time if $k = 2$ or $O(mn^{2k-2} \log^2 n)$ time if $k \geq 3$.*

4 ALGORITHM FOR LOW-RANK HYPERGRAPHS

We now present our $\widetilde{O}(n^r)$ time algorithm for finding min-cuts and min-k-cuts in hypergraphs of rank r. Throughout, we assume r is a constant independent of n. We again represent the hypergraph $H = (V, E)$ as a list of hyperedges as well as pointers between vertices and incident hyperedges. By merging parallel hyperedges, we can guarantee that our representation takes $O(n^r)$ space even as contractions are performed.

Our algorithm begins with an $O(m + n^r)$ time preprocessing step that removes duplicate hyperedges by sorting hyperedges lexicographically and replacing identical hyperedges with a single hyperedge of their summed weight. Overloading notation, let $H = (V, E)$ also denote the hypergraph after preprocessing.

After preprocessing, we run BranchingContract $O(\log^2 n)$ times as in Lemma 2.4. When we need to branch, we copy the entire hypergraph we are working with. Hyperedge selection and contraction is performed using the $O(n^{r-1})$ time procedure given in the next section, and we find k-spanning hyperedges as they are created during contractions with no addition to the overall running time.

4.1 Edge Selection and Contraction

Our algorithm needs to perform hyperedge selections and contractions in $O(n^{r-1})$ time each. Similar to Karger and Stein [20], we rely on some simple data structures to aid in our hyperedge selections instead of naively enumerating all edges and randomly selecting from them. We store a value $D(v)$ on each vertex v, which is equal to the sum of the weights of all hyperedges of size exactly r containing v. We also maintain $w(E)$, the total sum of hyperedge weights.

To select a hyperedge for contraction, we enumerate all $O(n^{r-1})$ hyperedges of size at most $r-1$. Let $e_1, \ldots, e_t$ be this list of hyperedges. With probability

$$\frac{\sum_{i=1}^{t} w(e_i)}{w(E)},$$

we pick one of $e_1, \ldots, e_t$ with probability proportional to its weight. If we do not pick one of these hyperedges, then we pick a vertex v with probability proportional to $D(v)$, and then pick one of the $O(n^{r-1})$ hyperedges of size r containing v with probability proportional to its weight. The running time of this procedure is dominated by the time it takes to enumerate the hyperedges involved in a single probabilistic choice. There are at most $O(n^{r-1})$ such edges in each case.

Lemma 4.1. *The above procedure picks a hyperedge of H with probability proportional to its weight.*

Proof. The probability that the procedure picks a fixed hyperedge e_i of size at most $r-1$ is exactly

$$\frac{\sum_{i=1}^{t} w(e_i)}{w(E)} \cdot \frac{w(e_i)}{\sum_{i=1}^{t} w(e_i)} = \frac{w(e_i)}{w(E)}.$$

Now let $e = \{v_1, \ldots, v_r\}$ be a hyperedge of size r. The probability of picking hyperedge e is

$$\frac{w(E) - \sum_{i=1}^{t} w(e_i)}{w(E)} \cdot \sum_{v \in e} \left[\frac{D(v)}{\sum_{v' \in V} D(v')} \cdot \frac{w(e)}{D(v)} \right] = \frac{\frac{1}{r} \sum_{v' \in V} D(v')}{w(E)} \cdot \frac{1}{\sum_{v' \in V} D(v')} \cdot \sum_{v \in e} w(e) = \frac{w(e)}{w(E)}.$$

□

We now turn to contractions. Suppose our algorithm needs to contract hyperedge $e = \{v_1, \ldots, v_{|e|}\}$. We will contract e to one of its vertices, say, $v_1 \in e$. First, we list all $O(n^{r-1})$ hyperedges that are incident to any vertex of e. Then remove members of $e - v_1$ from the hyperedges and add v_1 to each hyperedge it does not already appear in. We remove each hyperedge in the list that consists entirely of v_1. Next, we sort the hyperedges of the list as before, and combine identical hyperedges of the list, summing their weights. Finally, we recompute $D(v_1)$ by summing the weights of all remaining hyperedges in the list. The entire contraction procedure takes $O(n^{r-1})$ time.

4.2 Running Time Analysis

Now, we turn to analyzing our implementation of BranchingContract for hypergraphs of constant rank r. As we will see, the total time spent performing contractions is easily determined using the above discussion and Lemma 2.2's bound on the number of hypergraphs of different orders in the computation tree. The only other operation we have yet to analyze is the time spent

making copies of hypergraphs while branching. In particular, if there is a hypergraph H' in the computation tree with t children, then we need to create $t-1$ copies of H'.

Lemma 4.2. *Suppose the computation tree contains a hypergraph H' of order n_0. Node H' has $1+O(1/n_0)$ children in expectation. In particular, we spend an expected $O(n^{r-1})$ time creating copies of H'.*

Proof. Let t^* be the expected number of children of node H'. Because, $|e| \leq r$ for any hyperedge e in H', we branch with probability at most $\frac{\binom{n}{k-1}-\binom{n-r}{k-1}}{\binom{n}{k-1}}$. Therefore, we can bound the total expected number of children for node H' as a decreasing geometric series:

$$t^* \leq \sum_{t=0}^{\infty} \left(\frac{\binom{n}{k-1}-\binom{n-r}{k-1}}{\binom{n}{k-1}} \right)^t = \frac{1}{1-\frac{\binom{n}{k-1}-\binom{n-r}{k-1}}{\binom{n}{k-1}}} = \frac{\binom{n}{k-1}}{\binom{n-r}{k-1}} = 1 + \frac{\binom{n}{k-1}-\binom{n-r}{k-1}}{\binom{n-r}{k-1}} = 1 + O\left(\frac{1}{n}\right).$$

The lemma now follows from the fact that H' has total hyperedge size at most $O(n^r)$. □

Lemma 4.3. *The expected running time of BranchingContract using our implementation for constant rank r is $O(n^{2k-2}\log n)$ if $r = 2k-2$ or $O(n^{\max\{r,2k-2\}})$ otherwise.*

Proof. Fix any integer n_0 such that $k \leq n_0 \leq n$, and let $\mathcal{H}_{n_0}$ denote the set of hypergraphs of order n_0. Consider an arbitrary hypergraph $H' \in \mathcal{H}_{n_0}$. An ith copy of H' appears in the computation tree with some probability $p_{H',i}$. Conditioned on that copy appearing, we see from Lemma 4.2 that the expected time spent making copies of H' and performing contractions on H' and its copies is $O(n_0^{r-1})$. Recall, $S(n, n_0)$ denotes the maximum expected number of nodes of order n_0 created in a computation tree for some hypergraph H of order n. Lemma 2.2 implies the total expected time spent making copies and doing contractions for nodes from $\mathcal{H}_{n_0}$ is given by

$$\sum_{H' \in \mathcal{H}_{n_0}, i \geq 1} p_{H',i} \cdot O\left(n_0^{r-1}\right) \leq S(n,n_0) \cdot O\left(n_0^{r-1}\right) \leq O\left(\left(\frac{n}{n_0}\right)^{2k-2}\right) \cdot O\left(n_0^{r-1}\right) = O(n^{2k-2}) \cdot O\left(\frac{1}{n_0^{2k-r-1}}\right).$$

We finish the proof by summing the above expression over all possible n_0. If $r < 2k-2$, then $2k-r-1 \geq 2$, and

$$\sum_{n_0=k}^{n} O(n^{2k-2}) \cdot O\left(\frac{1}{n_0^{2k-r-1}}\right) \leq O(n^{2k-2}) \sum_{n_0=k}^{n} \frac{1}{n_0^2} = O(n^{2k-2}).$$

If $r = 2k-2$, then $2k-r-1 = 1$, and

$$\sum_{n_0=k}^{n} O(n^{2k-2}) \cdot O\left(\frac{1}{n_0^{2k-r-1}}\right) = O(n^{2k-2}) \sum_{n_0=k}^{n} \frac{1}{n_0} = O(n^{2k-2}\log n).$$

Finally, if $r > 2k-2$, then $r-2k+1 \geq 0$, and

$$\sum_{n_0=k}^{n} O(n^{2k-2}) \cdot O\left(\frac{1}{n_0^{2k-r-1}}\right) = O(n^{2k-2}) \sum_{n_0=k}^{n} n_0^{r-2k+1} = O(n^r).$$ □

By applying our analysis for boosting the probability of success (Lemma 2.4), we obtain our main results for constant rank hypergraphs.

Theorem 4.4 (Restatement of Theorem 1.2). *There exists an algorithm that with high probability computes a minimum k-cut of a weighted hypergraph with n vertices, constant rank r, and total hyperedge size p in $O(p + n^{2k-2}\log^3 n)$ time if $r = 2k-2$ or $O(p + n^{\max\{r,2k-2\}}\log^2 n)$ time otherwise.*

5 EXTENSION TO HEDGE GRAPHS

Let $H = (V, E)$ be a hedgegraph. To be consistent with Chandrasekaran, Xu, and Yu [6], we let $s(e)$ denote the span of a hedge $e \in E$ and $r(e)$ (for rank) denote the number of vertices incident to the edges in e. Let s be the span of H, defined as the maximum span over all hedges. Throughout this section, we assume that s is a constant.

We modify the branching random contraction algorithm (BranchingContract) for hypergraphs to create a new algorithm HedgeBranchingContract. We ***contract*** a hedge e by unifying the vertices within each connected component of e separately and removing all hedges that then have no non-loop edges. Contracting a hedge e reduces the number of vertices from n to $n - r(e) + s(e)$. Similar to before, we call hedge e a ***k-spanning hedge*** if $r(e) \geq n - k + 2$. Unfortunately, the existence of hedges with span greater than 1 means we may have some k-spanning hedges that do not cross any minimum k-cut. Fortunately, contracting any such hedge leaves a hedgegraph of constant size that can be checked for minimum k-cuts by brute force. Similar to before, let

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

We now define HedgeBranchingContract. As before, the parameter S is empty for the initial call.

HedgeBranchingContract(H, S):
If $n \leq k + s - 1$, then compute a min-k-cut C by brute force and return $C \cup S$

For each k-spanning hedge e with $n - r(e) + s(e) < k$, add e to S and remove e from H
Let $C_h = E$
For each k-spanning hedge e, set C_h to the smaller of C_h and HedgeBranchingContract$(H/e, S)$
Add all k-spanning hedges e to S and remove them from H

Select a hedge e at random with probability proportional to $w(e)$
With probability $z_n(e)$,
    return the smallest of the cuts HedgeBranchingContract$(H/e, S)$,
        HedgeBranchingContract(H, S), and C_h
Otherwise, return the smaller of HedgeBranchingContract$(H/e, S)$ and C_h

We proceed by analyzing the success probability of the branching contraction algorithm. Again, for any integer $t \geq k$, let

$$z_t^* = \frac{\binom{t}{k-1} - \binom{t-2}{k-1}}{\binom{t}{k-1}}.$$

Theorem 5.1. *Algorithm HedgeBranchingContract returns a minimum hedge-k-cut of a (weighted) hedgegraph with probability at least*

$$\frac{1}{2\sum_{t=k}^{n} z_t^*} = \Omega\left(\frac{1}{\log n}\right).$$

Notice that this bound matches the one for hypergraphs (span-1 hedgegraphs) up to constant factors.

Proof. Let q_n denote the minimum probability over all n-vertex hedgegraphs of HedgeBranchingContract returning a fixed min-k-cut. We use induction over n to show $q_n \geq \frac{1}{2\sum_{t=k}^{n} z_t^*}$. For the base case, $q_k = 1 > 1/(2(k-0)/k)$.

If contracting a hedge leaves fewer than k vertices, then it must belong to every k-cut and it is correct to include it in S. We check every other k-spanning hedge e to see if any one of them can be excluded from the min-k-cut. Cut C_h is the best of these checks. If C_h is not a min-k-cut, then it is correct to include every hedge e with $r(e) \geq n - k + 2$ in the cut returned by the algorithm. From here on, we assume there are no hedges e with $r(e) \geq n - k + 2$.

The probability of destroying a min-k-cut with a single contraction remains the same (Lemma 2.1). However, $n - r(e) + s(e)$ vertices remain after contracting a hedge e. Therefore, similar to the proof for hypergraphs, it now remains to solve the following recurrence relation where $E' \subseteq E$ is the subset of hedges that do not belong to *some* min-k-cut:

$$q_n \geq \left(\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e)\right)q_n + \left(1-\left(\frac{1}{w(E)}\sum_{e\in E} w(e)\cdot z_n(e)\right)\right)\frac{1}{w(E')}\sum_{e\in E'} w(e)\left(1-z_n(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\right)q_{n-r(e)+s(e)}.$$

Solving for q_n and rearranging some terms, we see

$$\begin{aligned} q_n &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(1-z_n(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\right)q_{n-r(e)+s(e)} \\ &= \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\left(\left(2\sum_{t=k}^{n} z_t^*\right)-z_n(e)\right)q_{n-r(e)+s(e)}. \end{aligned}$$

For each $e \in E'$, we use the definition of $z_n(e)$ and apply Observation 1 with $w = s(e)$ and $x = r(e)$:

$$\begin{aligned} q_n &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\left(\left(2\sum_{t=k}^{n} z_t^*\right)-\frac{\binom{n}{k-1}-\binom{n-r(e)}{k-1}}{\binom{n}{k-1}}\right)q_{n-r(e)+s(e)} \\ &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\left(\left(2\sum_{t=k}^{n} z_t^*\right)-\left(\left(\frac{r(e)-1}{r(e)-s(e)}\right)\sum_{t=n-r(e)+s(e)+1}^{n} z_t^*\right)\right)q_{n-r(e)+s(e)}. \end{aligned}$$

Observe that $s(e) \leq \lfloor r(e)/2 \rfloor$ for any hedge $e \in E'$. We apply this observation, use induction, and simplify to finish the proof:

$$\begin{aligned} q_n &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\left(\left(2\sum_{t=k}^{n} z_t^*\right)-\left(2\sum_{t=n-r(e)+s(e)+1}^{n} z_t^*\right)\right)q_{n-r(e)+s(e)} \\ &= \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\left(2\sum_{t=k}^{n-r(e)+s(e)} z_t^*\right)q_{n-r(e)+s(e)} \\ &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right)\left(2\sum_{t=k}^{n-r(e)+s(e)} z_t^*\right)\left(\frac{1}{2\sum_{t=k}^{n-r(e)+s(e)} z_t^*}\right) \\ &\geq \frac{1}{w(E')}\sum_{e\in E'} w(e)\left(\frac{1}{2\sum_{t=k}^{n} z_t^*}\right) \\ &= \frac{1}{2\sum_{t=k}^{n} z_t^*}. \end{aligned}$$

□

We now bound the size of the computation tree for hedgegraphs. Let $S(n, n_0)$ denote the maximum expected number of nodes of order n_0 created in a computation tree for some hedgegraph H of order n.

Lemma 5.2. *Fix an integer n_0 such that $k \le n_0 \le n$. We have*

$$S(n, n_0) \le \frac{\binom{n}{k-1}\binom{n-1}{k-1}\cdots\binom{n-s}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}\cdots\binom{n_0-s}{k-1}} = O\left(\left(\frac{n}{n_0}\right)^{(s+1)(k-1)}\right).$$

Proof. We will prove the lemma using induction on n. For $n = n_0$, we have $S(n, n_0) = 1$, because only the root's single-child node has order n.

As before, let $E' \subseteq E$ be the subset of hedges that may be randomly contracted to leave at least n_0 vertices. It now remains to solve the following recurrence relation:

$$S(n, n_0) \le \frac{1}{w(E')} \sum_{e \in E'} w(e)\left(z_n(e) \cdot S(n, n_0) + S(n - r(e) + s(e), n_0)\right).$$

Solving for $S(n, n_0)$ and applying the inductive hypothesis, we find

$$\begin{aligned} S(n, n_0) &\le \frac{\binom{n}{k-1}}{\frac{1}{w(E')}\sum_{e \in E'} w(e)\binom{n-r(e)}{k-1}} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e)\, S(n - r(e) + s(e), n_0) \\ &\le \frac{\binom{n}{k-1}}{\frac{1}{w(E')}\sum_{e \in E'} w(e)\binom{n-r(e)}{k-1}} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e)\left(\frac{\binom{n-r(e)+s(e)}{k-1}\binom{n-r(e)+s(e)-1}{k-1}\cdots\binom{n-r(e)+s(e)-s}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}\cdots\binom{n_0-s}{k-1}}\right). \end{aligned}$$

Since $r(e) \ge s(e) + 1$ and $s \ge s(e)$ for all $e \in E$,

$$\begin{aligned} S(n, n_0) &\le \frac{\binom{n}{k-1}}{\frac{1}{w(E')}\sum_{e \in E'} w(e)\binom{n-r(e)}{k-1}} \cdot \frac{1}{w(E')} \sum_{e \in E'} w(e)\left(\frac{\binom{n-1}{k-1}\binom{n-2}{k-1}\cdots\binom{n-s}{k-1}\cdot\binom{n-r(e)}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}\cdots\binom{n_0-s}{k-1}}\right) \\ &= \frac{\binom{n}{k-1}\binom{n-1}{k-1}\cdots\binom{n-s}{k-1}}{\binom{n_0}{k-1}\binom{n_0-1}{k-1}\cdots\binom{n_0-s}{k-1}}. \end{aligned}$$

□

We now discuss implementations of HedgeBranchingContract to get concrete running times. First, suppose the rank of H is arbitrary. We observe that for a subset of vertices $S \subseteq V$, contracting a hedge e will destroy the cut $(S, V \setminus S)$ if and only if at least one of e's connected components contains vertices from both S and $V \setminus S$. Therefore, we do not worry about the specific edges included in e and instead store e as a collection of disjoint vertex subsets, similar to a hyperedge. We assume there are pointers between vertices and these components and vice versa.

The key change from our implementation for hypergraphs is that we also store an $sm \times sm$ ***hedge component adjacency matrix*** as well where an entry is 1 if the hedge components corresponding to its row and column share a vertex and is 0 otherwise. The adjacency matrix will prove useful for helping us identify which vertices from a contracted hedge e belong to the other hedges.

We now describe how to contract a randomly selected hedge e to form hedgegraph $H' = (V', E')$ in $O(mn_0 + m^2)$ time, where n_0 is the order of H'. Like in hypergraphs, we initially set $V' := V \setminus e$ and $E' := E \setminus \{e\}$ by copying vertices and hedges but not their incidence lists. For each vertex $v \in V'$ and for each hedge $e' \in E'$ that is incident to v in H, we add v to the appropriate incidence list for the copy of e' in H' and vice versa. We now add additional vertices $v_{e,1}, v_{e,2}, \ldots, v_{e,x}$ where $x \le s$ to represent the contraction of e's x components. We add an incidence between $v_{e,i}$ and the jth component of another hedge e' if and only if the ith component of e and the jth component of e'

shared a vertex before contraction. We then go through each hedge to find components that share one of these contraction vertices to merge the component's incidence lists. There are at most s members in common per list, so we can do the merges in time linear in the total size of the lists. We remove any hedges from E' that have a single incident vertex. Finally, we go through each of the contracted vertices to see if there are any new adjacencies between hedge components thanks to these vertices. We spend constant time per hedge-vertex incidence in H' as well as $O(m^2)$ time checking for new hedge component adjacencies for $O(mn_0 + m^2)$ time total.

When contracting a k-spanning hedge, we immediately end up in the base case. Therefore, we do not bother to recompute the hedge component adjacency matrix during these contractions, meaning we perform the contraction and handle the base case in $O(m)$ time. During any one call to HedgeBranchingContract, there are $O(m)$ such contractions and brute force minimum hedge-k-cut computations, so we spend $O(m^2)$ time total performing them. In total, we can charge $O(mn_0 + m^2)$ time to each node of the computation tree, accounting for both the time to create that node and the time spent doing checks for the k-spanning hedges. Summing over the expected number of hedgegraphs of each order n_0 (Lemma 5.2) for the running time, and applying our boosting procedure (Lemma 2.4), we get the following theorem.

Theorem 5.3. *There exists an algorithm that with high probability computes a minimum hedge-k-cut of a weighted hedgegraph with n vertices, m hedges, and constant span $s \geq 2$ in $O(m^2 n^{(s+1)(k-1)} \log^2 n)$ time.*

For hedgegraphs of constant rank r, we need to modify HedgeBranchingContract so that its base case occurs when there are at most $k + r - 2$ vertices remaining. Doing so guarantees there are no k-spanning hedges to worry about, even though the base case can still be handled in constant time. We no longer need the hedge component adjacency matrix. Like our algorithm for hypergraphs, though, we do a prepossessing step of identify hedges with the same induced graph, leading to a total hedgegraph size of $O(n^r)$ (here, the big-Oh is hiding constants exponential in r). Random selection and contraction of a hedge e can still be done in $O(n^{r-1})$ time, because we only need to modify the $O(n^{r-1})$ hedges that share a vertex with e. When branching, we still copy the entire hedgegraph in $O(n^r)$ time, but in expectation we spend $O(n_0^{r-1})$ doing these copies for any node of order n_0 in the computation tree (see Lemma 4.2). We charge $O(n_0^{r-1})$ to each computation tree node of order n_0, sum over the expected number of nodes of each order n_0, and apply our probability boosting procedure to get the following theorem.

Theorem 5.4. *There exists an algorithm that with high probability computes a minimum hedge-k-cut of a weighted hedgegraph with n vertices, constant rank r, constant span s, and total hedge size p in $O(p + n^{(s+1)(k-1)} \log^3 n)$ time if $r = (s+1)(k-1)$ or $O(p + n^{\max\{r,(s+1)(k-1)\}} \log^2 n)$ time otherwise.*

APPENDIX

A SIMPLE ANALYSIS FOR NON-BRANCHING ALGORITHMS

In this Appendix, we present a simple non-branching randomized contraction algorithm for minimum k-cut based on the idea of *redoing* hyperedge selections with a probability dependent on their size. As before, let

$$z_n(e) = \frac{\binom{n}{k-1} - \binom{n-|e|}{k-1}}{\binom{n}{k-1}} \quad \text{for any hyperedge } e \in E.$$

We say a hyperedge e is ***k*-spanning** if it contains at least $n - k + 2$ vertices, implying it spans every k-cut.

We formally describe the algorithm SimpleContract below. Procedure SimpleContract initially takes the empty set as its second parameter.

SimpleContract(H, S):
  For each k-spanning hyperedge e, add e to S and remove e from H
  If $E = \emptyset$, then return S
  Repeat
    Select a hyperedge e at random proportional to its weight
    With probability $1 - z_n(e)$, return SimpleContract$(H/e, S)$

As mentioned in Section 2, this algorithm is almost the same as the one by Chandrasekaran, Xu, and Yu [6] when specialized to hypergraphs. In fact, the probabilities of contracting any given hyperedge e are identical. We can use our interpretation of the algorithm to derive another proof that it returns a fixed min-k-cut with decent probability.

Theorem A.1. *Algorithm SimpleContract returns a fixed minimum k-cut of a (weighted) hypergraph with probability at least* $\frac{k}{\binom{n}{k-1}\binom{n-1}{k-1}} = \Omega(1/n^{2k-2})$.

Proof. Let q_n denote the minimum probability over all n-vertex hypergraphs of SimpleContract returning a fixed min-k-cut. We use induction over n to show $q_n \geq \frac{k}{\binom{n}{k-1}\binom{n-1}{k-1}}$. For the base case, $q_k = 1 = \frac{k}{\binom{k}{k-1}\binom{k-1}{k-1}}$. Fix hypergraph H and minimum cut C. All spanning hyperedges must belong to C, so the algorithm is correct to add them to S and eventually return them. For simplicity, we assume from here on that E and C already have their k-spanning edges removed.

With probability at least $(\frac{1}{w(E)} \sum_{e \in E} w(e) \cdot z_n(e)) q_n$, the algorithm SimpleContract(H, S) randomly selects a hyperedge e for contraction, decides *not to contract* e right away, and then finds a min-k-cut afterward at some point in the future. Let $E' = E \setminus C$. By Lemma 2.1, there is at least a $(1 - (\frac{1}{w(E)} \sum_{e \in E} w(e) \cdot z_n(e)))$ probability that the selected hyperedge belongs to E', implying C still exists in H/e. Conditioned on that event, there is a $\frac{1}{w(E')} \sum_{e \in E'} w(e)$ probability of any *particular* hyperedge $e \in E'$ being chosen. Finally, given that $e \in E'$ is chosen, there is at least a $(1 - z_n(e))\, q_{n-|e|+1}$ probability that SimpleContract does contract e and C is returned by BranchingContract$(H/e, S)$. Note that these two ways of returning C (redoing hyperedge selection and going ahead with the contraction) are mutually exclusive events.

Taking everything together, we see

$$q_n \geq \left(\frac{1}{w(E)} \sum_{e \in E} w(e) \cdot z_n(e)\right) q_n + \left(1 - \frac{1}{w(E)} \sum_{e \in E} w(e) \cdot z_n(e)\right) \frac{1}{w(E')} \sum_{e \in E'} w(e) \cdot (1 - z_n(e)) \cdot q_{n-|e|+1}.$$

Solving for q_n, we see

$$q_n \geq \frac{1}{w(E')} \sum_{e \in E'} w(e) \cdot (1 - z_n(e)) \cdot q_{n-|e|+1}.$$

We now apply induction, use the fact that $|e| \geq 2$, and simplify:

$$q_n \geq \frac{1}{w(E')} \sum_{e \in E'} w(e) \left(\frac{\binom{n-|e|}{k-1}}{\binom{n}{k-1}}\right)\left(\frac{k}{\binom{n-|e|+1}{k-1}\binom{n-|e|}{k-1}}\right)$$

$$\geq \frac{1}{w(E')} \sum_{e \in E'} w(e) \left(\frac{\binom{n-|e|}{k-1}}{\binom{n}{k-1}} \right) \left(\frac{k}{\binom{n-1}{k-1}\binom{n-|e|}{k-1}} \right)$$
$$= \frac{1}{w(E')} \sum_{e \in E'} w(e) \left(\frac{k}{\binom{n}{k-1}\binom{n-1}{k-1}} \right)$$
$$= \frac{k}{\binom{n}{k-1}\binom{n-1}{k-1}}.$$

□

Note that we can give a similarly short analysis for a non-branching algorithm for minimum hedge-k-cuts. However, the algorithm itself is necessarily more complicated due to the hedgegraph analogy of k-spanning hedges not necessarily appearing in all hedge-k-cuts. These issues are discussed in more detail in Section 5.

REFERENCES

[1] Calvin Beideman, Karthekeyan Chandrasekaran, Sagnik Mukhopadhyay, and Danupon Nanongkai. 2022. Faster connectivity in low-rank hypergraphs via expander decomposition. In *Proceedings of the International Conference on Integer Programming and Combinatorial Optimization (IPCO'22)*. 70–83.

[2] Calvin Beideman, Karthekeyan Chandrasekaran, and Weihang Wang. 2022. Counting and enumerating optimum cut sets for hypergraph k-partitioning problems for fixed k. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP'22)*, Vol. 229. 16:1–16:18.

[3] Calvin Beideman, Karthekeyan Chandrasekaran, and Weihang Wang. 2022. Deterministic enumeration of all minimum k-cut-sets in hypergraphs for fixed k. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'22)*. 2208–2228.

[4] Karthekeyan Chandrasekaran and Chandra Chekuri. 2020. Hypergraph k-cut for fixed k in deterministic polynomial time. In *Proceedings of the IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS'20)*. IEEE, 810–821.

[5] Karthekeyan Chandrasekaran and Chandra Chekuri. 2021. Min-max partitioning of hypergraphs and symmetric submodular functions. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*. SIAM, 1026–1038.

[6] Karthekeyan Chandrasekaran, Chao Xu, and Xilin Yu. 2018. Hypergraph k-Cut in randomized polynomial time. In *Proceedings of the 29th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA'18)*. 1426–1438.

[7] Chandra Chekuri and Kent Quanrud. 2021. Isolating cuts, (Bi-)submodularity, and faster algorithms for connectivity. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[8] Chandra Chekuri, Kent Quanrud, and Chao Xu. 2019. LP relaxation and tree packing for minimum k-cuts. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms*.

[9] Chandra Chekuri and Chao Xu. 2017. Computing minimum cuts in hypergraphs. In *Proceedings of the 28th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA'17)*.

[10] Kyle Fox, Debmalya Panigrahi, and Fred Zhang. 2019. Minimum cut and minimum k-cut in hypergraphs via branching contractions. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 881–896.

[11] Takuro Fukunaga. 2013. Computing minimum multiway cuts in hypergraphs. *Discrete Optimiz.* 10, 4 (2013), 371–382. https://doi.org/10.1016/j.disopt.2013.10.002

[12] Mohsen Ghaffari, David Karger, and Debmalya Panigrahi. 2017. Random contractions and sampling for hypergraph and hedge connectivity. In *Proceedings of the 28th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA'17)*.

[13] Olivier Goldschmidt and Dorit S. Hochbaum. 1994. A polynomial algorithm for the k-cut problem for fixed k. *Math. Oper. Res.* 19, 1 (1994), 24–37. Retrieved from http://www.jstor.org/stable/3690374.

[14] Anupam Gupta, Euiwoong Lee, and Jason Li. 2018. Faster exact and approximate algorithms for k-Cut. In *Proceedings of the IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS'18)*. IEEE, 113–123.

[15] Anupam Gupta, Euiwoong Lee, and Jason Li. 2020. The Karger-Stein algorithm is optimal for k-cut. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC'20)*. 473–484.

[16] Yoko Kamidoi, S. Wakabayashi, and Noriyoshi Yoshida. 2002. A divide-and-conquer approach to the minimum k-way cut problem. *Algorithmica* 32, 2 (2002), 262–276.

[17] Yoko Kamidoi, Noriyoshi Yoshida, and Hiroshi Nagamochi. 2006. A deterministic algorithm for finding all minimum k-way cuts. *SIAM J. Comput.* 36, 5 (2006), 1329–1341.

[18] David R. Karger. 1993. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the 4th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA'93)*. 21–30.
[19] David R. Karger. 2000. Minimum cuts in near-linear time. *J. ACM* 47, 1 (2000), 46–76.
[20] David R. Karger and Clifford Stein. 1996. A new approach to the minimum cut problem. *J. ACM* 43, 4 (1996), 601–640.
[21] David R. Karger and David P. Williamson. 2021. Recursive random contraction revisited. In *Proceedings of the Symposium on Simplicity in Algorithms (SOSA)*. 68–73.
[22] Ken-ichi Kawarabayashi and Mikkel Thorup. 2015. Deterministic global minimum cut of a simple graph in near-linear time. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC'15)*. 665–674.
[23] Regina Klimmek and Frank Wagner. 1996. *A Simple Hypergraph Min Cut Algorithm*. Technical Report B 96-02. Bericht FU Berlin Fachbereich Mathematik und Informatik. Retrieved from http://edocs.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000297/1996_02.pdf.
[24] Dmitry Kogan and Robert Krauthgamer. 2015. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 6th Conference on Innovations in Theoretical Computer Science (ITCS'15)*. 367–376.
[25] Wai-Kei Mak and D. F. Wong. 2000. A fast hypergraph min-cut algorithm for circuit partitioning. *Integr. VLSI J.* 30, 1 (Nov. 2000), 1–11.
[26] Hiroshi Nagamochi and Toshihide Ibaraki. 1992. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Disc. Math.* 5, 1 (1992), 54–66.
[27] Hiroshi Nagamochi and Toshihide Ibaraki. 1992. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica* 7, 5&6 (1992), 583–596.
[28] Maurice Queyranne. 1998. Minimizing symmetric submodular functions. *Math. Program.* 82 (1998), 3–12.
[29] Mechthild Stoer and Frank Wagner. 1997. A simple min-cut algorithm. *J. ACM* 44, 4 (1997), 585–591.
[30] Mikkel Thorup. 2008. Minimum K-way cuts via deterministic greedy tree packing. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*. ACM, New York, NY, USA, 159–166. https://doi.org/10.1145/1374376.1374402
[31] Mingyu Xiao. 2008. An improved divide-and-conquer algorithm for finding all minimum k-way cuts. In *Proceedings of the International Symposium on Algorithms and Computation*. Springer, 208–219.
[32] Mingyu Xiao. 2010. Finding minimum 3-way cuts in hypergraphs. *Inf. Process. Lett.* 110, 14-15 (July 2010), 554–558. https://doi.org/10.1016/j.ipl.2010.05.003

Received 5 January 2022; revised 26 October 2022; accepted 26 October 2022