Random contractions and sampling for hypergraph and hedge connectivity

Random Contractions and Sampling for Hypergraph and Hedge Connectivity

Mohsen Ghaffari* David R. Karger† Debmalya Panigrahi‡

Abstract

We initiate the study of hedge connectivity of undirected graphs, motivated by dependent edge failures in real-world networks. In this model, edges are partitioned into groups called hedges that fail together. The hedge connectivity of a graph is the minimum number of hedges whose removal disconnects the graph. We give a polynomial-time approximation scheme and a quasi-polynomial exact algorithm for hedge connectivity. This provides strong evidence that the hedge connectivity problem is tractable, which contrasts with prior work that established the intractability of the corresponding $s-t$ min-cut problem. Our techniques also yield new combinatorial and algorithmic results in hypergraph connectivity. Next, we study the behavior of hedge graphs under uniform random sampling of hedges. We show that unlike graphs, all cuts in the sample do not converge to their expected value in hedge graphs. Nevertheless, the min-cut of the sample does indeed concentrate around the expected value of the original min-cut. This leads to a sharp threshold on hedge survival probabilities for graph disconnection. To the best of our knowledge, this is the first network reliability analysis under dependent edge failures.

1 Introduction

In this paper, we initiate the study of *hedge connectivity* of undirected graphs. Consider an n-vertex graph (or multigraph) $G = (V, E)$ whose m edges have been partitioned into groups we call *hedges*. We say G is *k-hedge-connected* if it is necessary to remove at least k edge groups (hedges) in order to disconnect G. This definition generalizes classical graph connectivity (where each hedge is a single edge) and hypergraph connectivity (where each hedge is a spanning subgraph on the vertices of a hyperedge). It is broader because a hedge can span *multiple unconnected* hyperedges.

The main motivation for our study of hedge connectivity comes from the dependence among edge failures observed in real world networks. Hedges model the simplest form of dependence — that sets of edges fail together. The classical notion of graph connectivity — the minimum number of edges whose removal disconnects a graph — can be a very weak approximation of the robustness of a graph in this scenario. Hypergraphs address part of this weakness but require all the dependent edges to be connected to each other, while hedges do not. In fact, our techniques yield results in hypergraph connectivity as well.

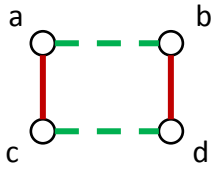

Figure 1: A simple hedge graph showing that the hedge cut function may not be submodular. The dashed edges represent edges of one hedge, and the solid edges represent the edges of the other hedge. Notice that $f(\{a,b\}) - f(\{a\}) = 1 - 2 < 2 - 1 = f(\{a,b,c\}) - f(\{a,c\})$.

We also note that by insisting on the fact that the hedges are disjoint, we are not losing any generality. If hedges overlap on an edge, modeling the fact that the edge can fail as part of multiple groups, we can replace the edge by a path where each edge on the path belongs to a unique hedge. This transformation does not affect the hedge connectivity of the graph since removing any of the hedges containing the original edge disconnects the path in the transformed graph.

In this paper, we use random contraction techniques to study hedge connectivity. A hedge is said to be in a cut if at least one edge from the hedge crosses the cut. To avoid confusion, we use the term *hedge-cut* to denote the set of hedges in a cut, while the term *cut* continues to represent the set of edges (respectively hyperedges) crossing the cut. A min-hedge-cut is a hedge-cut with a minimum number of hedges. An α-minimum hedge-cut is a hedge-cut with at most α times the number of hedges in a min-hedge-cut. These definitions are extended to weighted hedge graphs by defining the *value* of a cut to be the sum of weights of hedges crossing the cut.

Recall that a hedge is more general than a hyperedge:

*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology. Email: ghaffari@mit.edu.

†Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology. Email: karger@mit.edu. This work was supported in part by NSF Award CCF-1117381.

‡Department of Computer Science, Duke University. Email: debmalya@cs.duke.edu. This work was supported in part by NSF Awards CCF-1527084 and CCF-1535972.

a hyperedge is equivalent to a hedge all of whose edges form a single connected component. In general, a single hedge can involve more than one connected component. In fact, we first show that this distinction is fundamental in the following sense: while cuts in graphs and hypergraphs are symmetric submodular functions,[1] hedge-cuts are symmetric but *not necessarily submodular* (see Fig. 1 for an example of a non-submodular hedge cut function). This distinction is crucial from an algorithmic standpoint since the problem of minimizing a symmetric submodular functions is in **P**, which certifies that the connectivity of graphs and hypergraphs is polynomial-time solvable.

But what about hedge connectivity? Previously, it was shown by Zhang *et al.* [23] that the problem of finding a minimum s-t hedge cut — a hedge-cut of minimum value that separates a given pair of vertices s and t — is **NP**-hard.[2] In sharp contrast, we use randomized contraction techniques to give a quasi-polynomial time algorithm for finding a *global* min-hedge-cut, thereby providing strong evidence that this problem is in **P**. We also provide a quasi-polynomial bound on the number of min-hedge-cuts in a graph. Our quasi-polynomial algorithm is obtained by setting parameters appropriately in a more general polynomial time approximation scheme, which finds a $(1+\epsilon)$-minimum hedge-cut in $n^{O(\log(1/\epsilon))}$ time. This is also a sharp departure from the s-t cut problem, which has strong inapproximability results.

We also consider random sampling and reliability in hedge graphs. It is well known [11] that random sampling in graphs yields *cut sparsifiers* — sparser graphs on the same set of vertices where the size of all cuts concentrate around their respective expected values. This relies on the key observation that a graph has a bounded number of *approximately minimum* cuts which means that most cuts are extremely unlikely to diverge from their expectation. We first show that a corresponding result is ruled out for hedge graphs by giving a simple example where there are exponentially many near-minimum hedge cuts and, as a result, random sampling does not preserve the sizes of all hedge-cuts. Nevertheless, we introduce a new notion of *cut representatives* that reflects the connectivity structure of a hedge graph and show that a hedge graph has few approximately minimum cut representatives. We use this to show that while all hedge-cuts need not concentrate around their expected size, the min-hedge-cut in the sample is concentrated around the expected value of the min-hedge-cut in the original graph. In other words, the values of cuts can be smaller than their own expected value in general, but they are not smaller than the expected value of the min-hedge-cut. This property has been useful in graph connectivity in many different applications such as efficient min-cut algorithms, graph sparsification, etc. and we hope that it is so for hedge connectivity as well. As a first application, we derive a sharp threshold on hedge survival probabilities for a hedge graph to remain connected – to the best of our knowledge, this is the first result in network reliability under dependent edge failures.

Our ideas also yield results for traditional hypergraph connectivity. Recently, Kogan and Krauthgamer [18] used random contraction to show that for an r-uniform hypergraph (every hyperedge has r vertices) and for any half-integer $\alpha > 1$, the number of α-minimum cuts[3] is $O(n^{2\alpha} \cdot 2^{\alpha r})$. The first term is tight even for graphs, and they show that the exponential dependence on r in the second term is necessary for $\alpha > 1$, by giving a hypergraph with $\Omega(m \cdot 2^r)$ α-minimum cuts for any $\alpha > 1$. In contrast, we show that every hypergraph (not just r-uniform ones) has at most $\binom{n}{2} = O(n^2)$ min-cuts. We obtain this result by introducing sampling bias in the random contraction process which helps us avoid hyperedges of large cardinality that are intuitively more likely to span the two sides of a cut and therefore should not be contracted. We remark that a quadratic bound on the number of min-cuts in hypergraphs can also be derived from structural characterizations of hypergraphs by Cheng [3], although our proof is much simpler. Furthermore, our randomized contraction strategy also yields a new algorithm for hypergraph connectivity that runs in $\tilde{O}(Mn^2)$ time, where M is the sum of degrees of vertices in the hypergraph. Further optimizations improve the running time to $\tilde{O}((m+nk)n^2)$ for unweighted hypergraphs, which matches the best previous bounds of $O(Mn)$ [17, 19] for small values of k. (Here, k is the number of hyperedges in the min-cut.) Parallel to our work, and appearing in the same conference proceedings, is a new deterministic algorithm for hypergraph min-cut that runs in $O(M + n^2k)$ time[2]. The running time of this algorithm strictly dominates our running time. Nevertheless, our algorithm is quite simple compared to the deterministic algorithms, and only performs a sequence of hyperedge contractions chosen from a carefully defined distribution.

1.1 Related Work. The study of graph min-cuts has a long and rich history, dating back to the work of Gomory and Hu in the 60s [8]. Till the early 90s, the preferred approach was to use a sequence of carefully constructed $s-t$ min-cut instances [9] or dual tree packings [7]. Closer to

[1] A set function is said to be *submodular* if it has decreasing marginal gain, i.e., $A \subset B$ implies $f(A+x) - f(A) \leq f(B+x) - f(B)$. It is *symmetric* if $f(A) = f(\overline{A})$.

[2] They studied the problem under the title of Minimum Label Cut, where each edge of a multigraph has a label, and the cut size is defined as the number of different labels, for which at least one edge with that label crosses the cut.

[3] In all cut counting results for hypergraphs or hedge graphs, two cuts are distinct if they do not contain the same set of hyperedges or hedges. Counting distinct vertex bipartitions would be meaningless, e.g., in a hypergraph where all hyperedges contain all vertices.

this paper is the line of work, starting with the contraction algorithm [11], that focused on randomized techniques for finding min-cuts in undirected graphs [16, 14]. This line of work also seeded the study of random sampling in graph connectivity [12], eventually leading to cut sparsifiers [1, 6] that are random samples of a graph where all cuts concentrate near their expected value. The techniques that we employ in this paper are reminiscent of the first results in both these lines of work – in random contraction, it follows the (non-recursive) contraction approach of Karger [11] and in random sampling, it follows the uniform sampling approach of Karger [12]. However, we require several new ideas to overcome the existence of an exponential number of near-minimum cuts in both random contraction and random sampling. In particular, for the latter, we devise a new theory of cut representatives that might have independent applications.

Random sampling in graphs is useful in the study of network reliability under random edge failures. For independent edge failures, Karger [13] gave an FPTAS for estimating the probability of disconnection of the graph. The running time of this algorithm was improved by Harris and Srinivasan [10], and further by Karger recently [15]. Our work on random sampling in hedge graphs studies network reliability under dependent edge failures. Here, the most basic question is: can we estimate the probability of disconnection of a graph if groups of edges (hedges) fail together? In this work, we show that if the hedge survival probabilities are large enough that the expected value of the min-hedge-cut remains at least $\Omega(\log^2 n)$, then the graph remains connected with high probability. Conversely if the expectation is < 1 , the graph likely becomes disconnected. Indeed, for independent edge failures, a similar result [12] showing a threshold of $\Omega(\log n)$ on the expected size of the min-cut for the graph to stay connected laid the foundation for the network reliability algorithms.

Min-cuts have also been studied in hypergraphs, starting with the work of Queyranne [20] who used the submodularity of the cut function to give an $O(Mn^3)$ time algorithm. (Recall that M denotes the sum of degrees of vertices in the hypergraph.) In subsequent work, the running time has been improved to $O(Mn)$ [17, 19] and very recently, to $O(M + n^2k)$, where k is the number of hyperedges in a min-cut [2]. In a different direction, Rizzi [21] generalized Queyrenne's result to a larger set of symmetric set functions. Unfortunately, the example in Figure 1 shows that hedge cuts do not fit Rizzi's framework. Closer to our work in terms of techniques is the recent work of Kogan and Krauthgamer [18], who used uniform random contraction to bound the number of near-minimum cuts in a r-uniform hypergraph. Their bounds are exponential in r, which they show is required for near-minimum cuts. In contrast, we use non-uniform contraction to show a bound of $\binom{n}{2}$ on the number of *exact* min-cuts in *any* hypergraph.

As noted earlier, we initiate the study of global min-cuts for hedge graphs, and give a quasi-polynomial time exact algorithm and a polynomial time approximation scheme for it. In contrast, previous work by Zhang et al. [23] for the s-t min-cut problem in hedge graphs showed that it is **NP**-hard to obtain an approximation better than $2^{\log^{1-1/\log\log^c n} n}$ for any constant $c < 1/2$, and gave an $O(\sqrt{m})$-approximation algorithm. Moreover, Fellows *et al.* [5] showed that even in graphs with constant pathwidth, the problem is $W[2]$-hard when parameterized by m, and $W[1]$-hard when parameterized by k.

Roadmap. We first give the min-cut algorithm for hedge graphs in Section 2. The analysis of hedge connectivity under random sampling appears next in Section 3. The $\binom{n}{2}$ bound on the number of hypergraph min-cuts appears in Section 4. Finally, we give the new randomized algorithm for the hypergraph min-cut problem in Section 5.

2 Random Contraction for Hedge Connectivity

In this section, we give a randomized contraction algorithm that returns, for any $\epsilon > 0$, a hedge-cut of value at most $(1 + \epsilon)$ times that of the min-hedge-cut with high probability in time $\left(n^{O(\log(1/\epsilon))}\right)$. This gives a polynomial-time approximation scheme for the hedge connectivity problem. For unweighted hedge graphs, choosing $\epsilon < 1/k$ where k is the number of hedges in a min-hedge-cut, gives a quasi-polynomial Monte Carlo algorithm for the exact min-hedge-cut problem that runs in $n^{O(\log k)}$ time.

Let $G = (V, E)$ be an undirected hedge graph with m hedges on n vertices. Each hedge in E is defined by a set of disjoint components in V called *hedge components*, where each component contains at least 2 vertices. Note that this way of defining a hedge is equivalent to the "set of edges" definition used earlier, where the hedge components correspond to the (non-singleton) connected components of the edge set. The total number of vertices in these components is called the *cardinality* of the hedge, and is denoted $|e|$. A hedge is said to cross a cut if at least one component of the hedge has vertices on both sides of the cut.

The main contribution of this section is a random contraction process that returns a hedge-cut of value at most $(1 + \epsilon)k$ with probability at least $1/n^{O(\log(1/\epsilon))}$, for any $\epsilon \in (0, 1)$. Here, k denotes the value of a min-hedge-cut. While our results can be directly derived for weighted hedge graphs, we prefer to demonstrate them for unweighted hedge graphs in this section. For weighted hedge graphs, the only change in our random process is to choose a hedge from a set with probability proportional to its weight instead of uniformly at random.

When a hedge is contracted, the vertices in each component of the hedge are separately unified into single vertices. Hence, the decrease in the number of vertices is equal

to the difference between the cardinality of the hedge and the number of components in it. The main difficulty with hedge contractions vis-à-vis edge contractions (or indeed hyperedge contractions) is the following: both Karger's original analysis for edge contraction and our analysis for hyperedge contraction heavily rely on the fact that the cardinality of a (hyper)edge and the decrease in the number of vertices on its contraction differs by exactly 1. Unfortunately, this is not true in general for a hedge — contracting a hedge of cardinality t only guarantees a decrease of $t/2$ vertices in the graph since each component of the hedge might contain exactly 2 vertices.

Consider, for instance, an unweighted regular hedge graph on n vertices with $k + c$ hedges for some constant c, where each hedge forms a perfect matching on the vertices of the graph. The probability of contracting a hedge outside a fixed min-hedge-cut is only $O(1/k)$. However, the number of vertices does not decrease by $n - 2$, but only by $n/2$, in such a contraction. Nevertheless, our key observation is that in any run, at most $O(\log n)$ such "large" hedges may be contracted, and their cumulative contribution to the success probability is $1/k^{O(\log n)}$. All other contractions must be of "small" hedges, for which we show that the gap between $|e| - 1$ and $|e|/2$ is small enough for the proof to work. Of course, there are both large and small hedges in a hedge graph in general, requiring us to combine the two arguments in a single random contraction process.

2.1 The Algorithm. Formally, we categorize hedges into three groups (here n denotes the number of vertices in the current graph): a hedge e is said to be (a) *large* if $|e| \geq n/2$, (b) *moderate* if $|e| \geq n/4$, but $|e| < n/2$, and (c) *small* if $|e| < n/4$. A hedge graph G is said to be *large* if it contains at least one large hedge; else it is said to be *small*. Note that small hedge graphs may contain both small and moderate hedges.

The randomized contraction algorithm is then defined recursively as given in Figure 2. The algorithm iteratively performs the following steps until there are only two vertices: If the hedge graph is small, i.e., contains only small and moderate hedges, then a hedge chosen uniformly at random is contracted. On the other hand, if the hedge graph is large, i.e., contains at least one large hedge, then the algorithm branches with equal probability to one the following: either all large and moderate hedges are removed from the graph and added to the output set of hedges, or one of these large or moderate hedges is contracted uniformly at random. The intuition behind this branching is that either the min-cut contains a high fraction of the large and moderate hedges or it does not. In the former case, the entire set of large and moderate hedges is a good approximation of the ones that are in the min-cut. Therefore, the branch where all the large and moderate hedges are added to the output cut is the "good" branch. On the other hand, if the min-cut does not contain a high fraction of the large and moderate hedges, then the other branch is "good" since the probability that a randomly chosen hedge from this set is in the min-cut is quite low. Of course, the algorithm does not know which of these cases it is in, and therefore, choses randomly between these options. However, recall that the number of times the algorithm will branch is only logarithmic, and therefore, even though it chooses the right branch only with probability $1/2$, the overall impact of this in the success probability of the algorithm is $1/n^c$ for some constant c depending on the base of the logarithm. Interestingly, the algorithm does not depend on the approximation parameter ϵ, although the probability of the algorithm returning an $(1 + \epsilon)$-approximate min-hedge-cut decreases with the value of ϵ.

2.2 The Analysis. We will give the analysis in two steps: first, we bound the number of branchings in any run of the algorithm, and then derive the desired lower bound on the probability that the algorithm returns a fixed min-hedge-cut.

Branchings. We show that the contraction algorithm encounters $O(\log n)$ large graphs G, and therefore performs $O(\log n)$ branchings, in any run. This follows from two observations: (1) the branch for H_1 only has small edges which can become large only after the number of vertices decreases by a constant factor, and (2) the branch for H_2 contracts a moderate or large edge thereby reducing the number of vertices by a constant factor. A formal inductive proof follows.

LEMMA 2.1. *The total number of large graphs G that the* **Contract** *algorithm encounters, and therefore the total number of branching steps, in any particular run is at most* $\log_{8/7} n$.

Proof. We prove this lemma inductively. Clearly, the lemma holds for $n = 2$, since the algorithm terminates in this case. Now, suppose $n > 2$, and the inductive hypothesis asserts that the lemma holds for all $n' < n$. If G is not large, then the inductive hypothesis implies the lemma since $\log_{8/7} n' < \log_{8/7} n$. Now, suppose G is large. Then, in the two branches, the following happen:

1. H_1 is a graph where every hedge e is small, i.e., $|e| < n/4$. Coupled with the fact that hedge cardinalities do not increase during a run of the **Contract** algorithm, it follows that the next branching step can happen only when the number of vertices has decreased to $n/2$ or smaller. Using the inductive hypothesis at the next branching step, we can claim that the total number of branching steps after the current one is at most $\log_{8/7}(n/2) \leq \log_{8/7} n - 1$. The lemma follows by adding in the current branching step, thereby obtaining an overall bound of $\log_{8/7} n$.

The Contraction Algorithm for Hedge Graphs

Input: Hedge graph G
Output: Set of hedges $S = \mathbf{Contract}(G)$

1. If G has two vertices, then $S = E$.
2. else, if G is a small hedge graph, then
   (a) contract a hedge e chosen uniformly at random from E, and delete all resulting hedge components containing a single vertex. Call this new hedge graph H.
   (b) $S = \mathbf{Contract}(H)$.
3. else, (G is a large hedge graph, let L be the set of large and moderate hedges in G)
   (a) In G, remove all large and moderate hedges (i.e., the set L). Call this new hedge graph H_1.
   (b) In G, contract a hedge e chosen uniformly at random from L, and delete all resulting hedge components of cardinality 1. Call this new hedge graph H_2.
   (c) With probability 1/2 each, (We call this a *branching step*.)
   - $S = L \cup \mathbf{Contract}(H_1)$, or
   - $S = \mathbf{Contract}(H_2)$.

Figure 2: The contraction algorithm for hedge graphs.

2. H_2 is a graph where a randomly chosen hedge of cardinality at least $n/4$ has been contracted. This contraction reduces the number of vertices by at least $n/8$, i.e., H_2 has at most $7n/8$ vertices. Using the inductive hypothesis on H_2 yields a bound of $\log_{8/7}(7n/8) = \log_{8/7} n - 1$ on the remaining number of branching steps. The lemma now follows by adding in the current branching, thereby obtaining an overall bound of $\log_{8/7} n$.

Success Probability. Our goal, in this analysis, is to lower bound the probability that the algorithm outputs a near-minimum hedge-cut of value at most $(1 + \epsilon)k$. We call this the probability of *success*, and denote it $q_{n,\ell}$ for an n-vertex graph with ℓ branching steps (the above lemma asserts that $\ell \leq \log_{8/7} n$). To gain intuition into the analysis, let us consider the special case of $\epsilon < 1/k$, i.e., when we are actually calculating the probability of the algorithm returning an exact min-hedge-cut C. The analysis consists of two parts. First, consider the branching steps. In each branching step, either all the large and moderate hedges are in C – then the first branch succeeds – or at least one large or moderate hedge is not in C – then the second branch succeeds with probability at least $1/k$. Since there are at most $O(\log n)$ branching steps in any run, their cumulative contribution to the success probability is $1/k^{O(\log n)}$.

Now, consider the case where the hedge graph is small. In this case, let us consider a further special case where all hedges have cardinality $t < n/4$, and the number of vertices decreases by $t/2$ on contraction. Note that every *degree hedge-cut* — the set of hedges containing a fixed vertex in their hedge components — contains at least k hedges. Hence, the sum of hedge cardinalities is at least nk, which implies that the number of small hedges must be at least nk/t. Therefore, the probability that the algorithm does not contract a hedge in C is $(1 - t/n)$ and this contraction reduces the number of vertices to $n - t/2$. Inductively, if the probability of success of this random process for γ vertices is denoted $f(\gamma)$, then we get the recurrence

$$f(n) = (1 - t/n) \cdot f(n - t/2).$$

Solving this recurrence gives $f(n) = 1/n^c$ for a large enough constant c.

We now draw upon the above intuition to give a formal analysis of the algorithm. We will restrict the definition of success of the $\mathbf{Contract}$ algorithm to cases where it outputs a set of hedges S containing at most ϵk hedges not in C, where C is any min-hedge-cut that we fix. (This is a stronger notion than approximate min-hedge-cuts, that we will call representatives later in Section 3.) We now state our main lemma.

LEMMA 2.2. *The probability of the contraction algorithm returning a $(1+\epsilon)$-approximate min-hedge-cut is at least $1/n^{O(\log(1/\epsilon))}$.*

The rest of this section is devoted to proving this lemma. First, note that there are two types of contraction steps in the **Contract** algorithm, depending on whether G is small or large. If G is small, the algorithm succeeds if: (1) the contracted hedge is not in C, and (2) the algorithm succeeds on H. Recall that if a hedge e is selected for contraction, then the number of vertices reduces by at least $|e|/2$ in this step. Therefore, we can write the following recurrence:

$$q_{n,\ell} \geq \frac{1}{m} \cdot \sum_{e \notin C} q_{n-|e|/2,\ell}. \tag{2.1}$$

The more involved situation is if the current step is a branching, i.e., G is large. We distinguish between two cases:

- **Case 1:** Of the hedges in L, suppose at least a $\delta = \frac{\epsilon}{1+\epsilon}$ fraction are not in C, i.e., $|L \setminus C| \geq \delta \cdot |L|$. In this case, consider the branch given by graph H_2. The cardinality of the contracted hedge is at least $n/4$, hence the number of vertices in H_2 is at most $7n/8$. Thus, we can write the following recurrence, where the factor of $1/2$ is the probability that the algorithm chooses the right branch among cases 1 and 2:

$$q_{n,\ell} \geq \frac{1}{2} \cdot \delta \cdot q_{7n/8,\ell-1}. \tag{2.2}$$

- **Case 2:** Of the hedges in L, suppose less than a $\delta = \frac{\epsilon}{1+\epsilon}$ fraction are not in C, i.e., $|L \setminus C| < \delta \cdot |L|$. Then, $|L \setminus C| < \epsilon |L \cap C|$. In this case, consider the branch given by graph H_1. We can charge the hedges in $L \setminus C$ to those in $L \cap C$ since we are allowed an approximation factor of $(1+\epsilon)$. Thus, we can write the following recurrence, again having a factor of $1/2$ to represent the probability that the algorithm chooses the right branch:

$$q_{n,\ell} \geq \frac{1}{2} \cdot q_{n,\ell-1}. \tag{2.3}$$

We have now established the recurrence relations that we will use in inductively proving Lemma 2.2. We now want to claim that for some constant $\alpha > 1$ that we will fix later, the following holds:

$$q_{n,\ell} \geq n^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell}. \tag{2.4}$$

To show Eq. (2.4), we use induction on the values of n and ℓ. Clearly, it holds for $n = 2$ and any ℓ, since the probability of success for a graph on 2 vertices is 1. Let us now consider $n > 2$ and any value of ℓ. We need to address the following three scenarios:

1. G **is small:** In this case, we will need lower bounds on the number and cardinality of hedges outside C.

   LEMMA 2.3. *If G is small, then:*

   (a) the sum of cardinalities of hedges not in a min-hedge-cut C is at least $nk/2$, i.e.,

   $$\sum_{e \notin C} |e| \geq nk/2.$$

   (b) the total number of hedges $m \geq 2k$.

   Proof. First, we prove the cardinality lower bound:

   $$\sum_{e \notin C} |e| = \sum_{e \in E} |e| - \sum_{e \in C} |e| \geq nk/2,$$

   Note that we used:

   (a) $\sum_{e \in E} |e| \geq nk$ since the number of hedges containing any vertex is at least the number of hedges in a min-cut k.

   (b) $\sum_{e \notin C} |e| \leq nk/2$ since C contains k hedges and all these hedges are small or moderate, given that G is small.

   Since all hedges are small or moderate, the above bound implies that $m \geq \frac{\sum_{e \in E} |e|}{n/2} \geq \frac{nk}{n/2} \geq 2k$.

   Now, from Eq. (2.1), we have

$$\begin{aligned}
& q_{n,\ell} \\
\geq\ & \frac{1}{m} \cdot \sum_{e \notin C} q_{n-|e|/2,\ell} \\
=\ & \frac{m-k}{m} \cdot \sum_{e \notin C} \frac{q_{n-|e|/2,\ell}}{m-k} \\
\geq\ & \frac{m-k}{m} \cdot \sum_{e \notin C} \frac{1}{m-k} \cdot \left(n - \frac{|e|}{2}\right)^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell} \\
& \text{(using the inductive hypothesis)} \\
\geq\ & \frac{m-k}{m} \cdot \left(\frac{\delta}{2}\right)^{\ell} \cdot \left(n - \frac{\sum_{e \notin C} |e|}{2(m-k)}\right)^{-\alpha} \\
& \text{(by convexity of the function } n^{-\alpha} \text{ for } \alpha > 1) \\
\geq\ & \frac{m-k}{m} \cdot \left(\frac{\delta}{2}\right)^{\ell} \cdot \left(n - \frac{nk/2}{2m}\right)^{-\alpha} \\
& \text{(by property 1 in Lemma 2.3 and } m - k \leq m) \\
=\ & \left(1 - \frac{k}{m}\right) \cdot \left(\frac{\delta}{2}\right)^{\ell} \cdot n^{-\alpha} \cdot \left(1 - \frac{k}{4m}\right)^{-\alpha} \\
=\ & \left(\frac{\delta}{2}\right)^{\ell} \cdot n^{-\alpha} \cdot \frac{1-x}{(1-x/4)^{\alpha}} \\
& \text{(for } x = k/m \in (0, 1/2] \text{ by property 2 in Lemma 2.3)} \\
\geq\ & \left(\frac{\delta}{2}\right)^{\ell} \cdot n^{-\alpha} \quad \text{for any } \alpha \geq 6.
\end{aligned}$$

2. G **is large, Case 1 above:** In this case, from Eq.(2.2), we have

$$\begin{aligned} & q_{n,\ell} \\ \geq & \frac{1}{2} \cdot \delta \cdot q_{7n/8,\ell-1} \\ \geq & \frac{1}{2} \cdot \delta \cdot \left(\frac{7n}{8}\right)^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell-1} \\ & \text{(using the inductive hypothesis)} \\ = & \left(\frac{7n}{8}\right)^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell} \\ \geq & n^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell}. \end{aligned}$$

3. G **is large, Case 2 above:** In this case, from Eq. (2.3), we have

$$\begin{aligned} & q_{n,\ell} \\ \geq & \frac{1}{2} \cdot q_{n,\ell-1} \\ \geq & \frac{1}{2} \cdot n^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell-1} \\ & \text{(using the inductive hypothesis)} \\ \geq & n^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell}, \\ & \left(\text{since } \delta = \frac{\epsilon}{1+\epsilon} < 1\right). \end{aligned}$$

We have now established that the probability of success of the **Contract** algorithm is at least

$$n^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\ell} \geq n^{-\alpha} \cdot \left(\frac{\delta}{2}\right)^{\log_{8/7} n}.$$

If $\epsilon \geq 1$, then $\delta \geq 1/2$, which implies that the probability of success is inverse polynomial in n. So, we will only consider the case when $\epsilon < 1$. In this case, $\delta = \frac{\epsilon}{1+\epsilon} > \epsilon/2$. Therefore, the probability of success is at least

$$n^{-\alpha} \cdot \left(\frac{\epsilon}{4}\right)^{\log_{8/7} n} = n^{O(log(1/\epsilon))}.$$

This completes the proof of Lemma 2.2.

The next theorem now follows by running the **Contract** algorithm $n^{O(\log(1/\epsilon))}$ times and returning the hedge-cut of minimum value returned by any run.

THEOREM 2.1. *There is an $n^{O(\log(1/\epsilon))}$-time randomized approximation scheme for the min-hedge-cut problem in undirected hedge graphs.*

For unweighted hedge graphs, we obtain a quasi-polynomial time exact algorithm by setting $\epsilon < 1/k$ in the above theorem. Correspondingly, we also obtain a bound on the number of min-hedge-cuts.

THEOREM 2.2. *There is an $n^{O(\log k)}$-time randomized algorithm for the min-hedge-cut problem in undirected, unweighted hedge graphs. Furthermore, the number of distinct min-hedge-cuts defined as a set of hedges in an undirected, unweighted hedge graph is $n^{O(\log k)}$.*

We remark that we are not aware of any hedge graph with a superpolynomial number of hedge cuts; in fact, the best lower bound we know of is the $\Omega(n^2)$ lower bound on graphs.

3 Random Sampling in Hedge Graphs

In this section, we will consider uniform random sampling of hedges in a hedge graph. Ideally, one would like to lift the analysis of sampling in graphs (see, e.g., Karger [12]) to hedge graphs and claim that if we independently sample every hedge with probability $O\left(\frac{\log n}{k}\right)$ in a k-connected hedge graph, then with high probability the value of every cut is close to its expectation. The analysis for graphs crucially relies on bounding the number of α-minimum cuts by $n^{O(\alpha)}$, and arguing that each deviates from its expectation with probability $n^{-\alpha}$, thereby allowing a union bound. Unfortunately, there is no corresponding bound on the number of α-minimum hedge-cuts. Consider a graph with $k-1$ hedges each consisting of a spanning tree, and $n-1$ other single-edge hedges that form a single spanning tree. The hedge connectivity is k (the $k-1$ large hedges must be removed, and any singleton hedge can then be removed to disconnect the graph). However, we can create a 2-minimum hedge-cut by removing the $k-1$ large hedges and any k single-edge hedges. And there are $\binom{n}{k}$ ways of choosing the k singleton hedges to remove, each defining a distinct 2-minimum hedge-cut.

As a particular consequence, if we sample each hedge of this graph with probability $p = O\left(\frac{\log n}{k}\right)$, we end up sampling only $O(\log n)$ of the singleton hedges. Therefore, almost all of the 2-minimum hedge-cuts will have none of their singleton hedges sampled, and will therefore diverge significantly (by at least 1/2) from their expectation. On the other hand, a Chernoff bound shows that with high probability, we will include pk of the large hedges, meaning that the hedge connectivity of the sample will be close to its expectation pk with high probability. Thus, in this example, while we cannot hope to prove that all hedge-cuts are near their expectation with high probability, the minimum hedge-cut of the sample is indeed as expected. We will now generalize this argument, and show that in *any* k-hedge-connected graph, the value of the minimum hedge-cut in the sample is near the expected value of the min-hedge-cut with high probability. To do so, we need a new technique, since we have just seen that a union bound over all cuts deviating from their expectation cannot work.

3.1 Representatives. While there can be many α-minimum hedge-cuts, we will argue that they have a small set of *representatives*. A representative for a given hedge-cut C is a hedge-cut C' most of whose edges are also in C. Note that the property is not symmetric, i.e., C might have many hedges that are not in C'. Now, if too few hedges from C are sampled, then it must be the case that too few hedges from C' are sampled as well since C' does not have many hedges outside C. Since the number of representatives is small, we will use the union bound technique to prove that this is unlikely to happen to *any* representative; from this we conclude that it is unlikely to happen to any hedge-cut at all.

We now formalize these ideas. We fix a parameter δ to be set later. We define a *δ-representative* for a hedge-cut C to be a hedge-cut C' such that at most δk hedges of C' are not in C.

LEMMA 3.1. *For any $\alpha \geq 1$ and any $\delta < 1$, there is a set of $n^{O(\alpha/\delta)}$ hedge-cuts containing a δ-representative for every α-minimum hedge-cut of an undirected n-verted hedge graph G.*

Proof. We prove this lemma via a hedge contraction process defined by a sequence of contraction *phases* starting with graph G. In each phase, we contract each hedge in the current contracted graph independently with probability $1/k$. We repeat for $\Theta((\log n)/\delta)$ phases or until there is single vertex left, whichever happens earlier. Each phase produces a set of (at most n) newly contracted vertices. Let the *degree hedge-cut* of each such vertex v be the set of hedges that contain an edge with exactly one endpoint in v; it corresponds to a hedge-cut in G. The total number of contracted vertices, and corresponding degree hedge-cuts in a particular run of the randomized contraction process, is $\tilde{O}(n)$; these are the representatives.

Consider any particular hedge-cut C of G of value at most αk. The probability that in one phase we contract no hedge in C is $(1 - 1/k)^{\alpha k} \leq e^{-\alpha}$. Over the course of the $O((\log n)/\delta)$ phases, the probability that we *never* contract a hedge in C is $n^{-O(\alpha/\delta)}$.

Let us condition the randomized contraction process on the event that no hedge in C is contracted. This conditional randomized process is identical to the unconditional one where all the hedges in C have been removed from G at the outset. This can be visualized as graph G being partitioned into multiple (at least two) disconnected components, on which the randomized contraction process is being run. There is dependence between the components because of hedges spanning multiple components, but this will not matter in our analysis.

If the degree of any vertex at any stage of this conditioned contraction process is at most δk, then the corresponding degree hedge-cut is a δ-representative for C. Thus, we arrive at our crucial observation: in each phase, either C has a δ-representative among the degree hedge-cuts, or else the probability that each vertex of C gets contracted in the next phase is $1-(1-1/k)^{\delta k} \approx \delta$, which means that the number of vertices in each component decreases by a factor $(1-\delta/2)$ in expectation. Thus, after $O((\log n)/\delta)$ such phases, either C has a representative at some stage among the degree hedge-cuts, or the expected number of vertices in each component is $1 + o(1)$. In the latter scenario, using the Markov bound, we can claim that with constant probability, there is some component that has a single vertex. The degree hedge-cut corresponding to this singleton component is then a representative for C. So, we have shown that with constant probability, there is a δ-representative for C among the degree hedge-cuts produced by the randomized contraction process, conditioned on the fact that no hedge from C is contracted at any stage.

We have therefore proven that with probability $n^{-O(\alpha/\delta)}$, one of our phases produces a representative for C. Thus, if we repeat our entire procedure $n^{O(\alpha/\delta)}$ times, the probability that we fail to generate a representative is only a constant, and if we repeat a factor of $O(n)$ times more, the probability that we fail to produce a representative for a given cut drops to 2^{-n}. Since there are at most 2^n distinct hedge-cuts (each corresponding to a vertex bi-partition) we can conclude that with constant probability, the graphs produced will contain a representative for *every* α-minimum hedge-cut. Our $n^{O(\alpha/\delta)}$ repetitions produce, in total, a collection of $n^{O(\alpha/\delta)}$ hedge-cuts that form our set of representatives. Finally, we also need to add the n cuts defined by degree hedges-cuts of the original graph G itself, in case one of those hedge-cuts is already a representative for some cuts.

Ideally, we would like to claim that each representative is close to its expected value after sampling and therefore provides a lower bound on the sizes of all the hedge-cuts it represents. However, this cannot be claimed directly from the lemma above, since the number of representatives is being bounded by the values of the hedge-cuts they represent rather than their own value. For instance, it is possible that the representatives for all values of α are min-hedge-cuts, which being small are more likely to deviate, so that there are too many of them to use a union bound. In the next lemma, we show this is not the case, but in doing so, we get a weaker bound on the number of representatives.

LEMMA 3.2. *For any $\delta < 1$ and $\alpha \geq 1$, there is a set of δ-representatives for all hedge-cuts of value at most $O(nk)$ that contains at most $n^{O(\alpha \log n/\delta)}$ representatives of value at most αk.*

Proof. For a given cut, consider the chain of representatives consisting of the cut, its representative, the representative of that representative, and so on. Call each δ-representative *large* if its value is at least half the value of the cut it is

representing in this chain. Then, every large representative of value at most αk must represent a hedge-cut of value at most $2\alpha k$, and by Lemma 3.1, there are only $n^{O(\alpha/\delta)}$ such representatives. On the other hand, if every representative in the chain is small then, since each must have half the size of the cut its represents, the $O(\log n)$th representative in the chain will have value at most $2k$. The representative of this cut must be large since the representative has value at least k. Each representative in the chain differs from the cut it represents by only δk hedges; thus the final, large representative differs from the initial cut by $O(\delta \log n)$ hedges—i.e., it is an $O(\delta \log n)$-representative. The lemma follows by defining the new δ as the previous $\delta \log n$.

3.2 Sampling. Given our set of representative hedge-cuts, we now proceed to prove our sampling result. The first claim is that the representatives are close to their expected value after sampling. Fix any set of δ-representatives of all hedge-cuts of value at most nk as given by Lemma 3.2. Then, the following holds for this set of representatives.

LEMMA 3.3. *For any $\epsilon \in (0,1)$, if every hedge is sampled with probability $p \geq C \cdot \frac{\log^2 n}{\epsilon^2 \delta k}$ independently, where C is a large enough constant, then the value of each representative after sampling is at least $(1-\epsilon)pk$ with high probability.*

Proof. Using Chernoff bounds on a single representative of value αk yields a failure probability of $n^{-\Omega(\alpha \log n/\delta)}$. Now, applying the union bound over the representatives of value αk, we get a bound of $n^{-\Omega(\alpha \log n/\delta)}$ on the probability of any of these representatives having a value less than $(1-\epsilon)pk$ in the sample. The lemma now follows by using the union bound over all values of α.

This lemma implies that for every hedge-cut of size at most $O(nk)$, the sampled hedge graph contains at least $(1-\epsilon)pk - \delta k$ hedges. From the condition of the lemma, $\delta k \geq C \cdot \frac{\log^2 n}{\epsilon^2 p}$. Clearly, this implies that p must be at least $C \cdot \frac{\log^2 n}{\epsilon^2 k}$. But, what δ should be choose in the analysis for smaller p? Since our additive error is δk, we should set it to the smallest value possible, i.e., $\delta k = C \cdot \frac{\log^2 n}{\epsilon^2 p}$. Then, the additive error is $\epsilon pk + C \cdot \frac{\log^2 n}{\epsilon^2 p}$. The error appears to grow with decrease in the value of p. However, observe that sampling with probability p is identical to repeated sampling with probability $1/2$ over $\lg(1/p)$ phases. The additive errors over the phases form a geometric series with a total error of at most $\epsilon pk + 2C \cdot \frac{\log^2 n}{\epsilon^2}$. To absorb the second error term in the first, we choose $p \geq 2C \cdot \frac{\log^2 n}{\epsilon^3 k}$ to get a total additive error of at most $2\epsilon pk$. For a hedge cut of value $\Omega(nk)$, Chernoff bounds give a probability of $2^{-\Omega n}$ of its value being less than pk in the sample. Using a union bound, we can conclude that with high probability, no cut of value $\Omega(nk)$ decreases to less than pk after sampling.

THEOREM 3.1. *If a graph with hedge connectivity k is sampled with probability $p \geq C' \cdot \frac{\log^2 n}{\epsilon^3 k}$ for a large enough constant C', then with high probability, the value of every hedge-cut in the sample is at least $(1-\epsilon)pk$.*

This theorem ensures that the min-hedge-cut after sampling remains tightly concentrated around the expected value of the min-hedge-cut before sampling, which is analogous to well-known sampling results in graph connectivity [12]. Note that $\Omega\left(\frac{\log n}{k}\right)$ is necessary even for graphs, e.g., when sampling a complete graph. It is not clear whether the additional $\log n$ term in our theorem is an artifact of our proof, or is necessary for hedge graphs.

COROLLARY 3.1. *If in a graph with hedge-connectivity k all edges survive with probability $p \geq C'' \cdot \frac{\log^2 n}{k}$, then the resulting graph is connected with high probability. Conversely, if edges survive with probability $p \leq C''' \cdot \frac{1}{k}$, then the resulting graph is disconnected with constant probability.*

This corollary, showing a sharp threshold on the probability where a hedge graph becomes disconnected, takes a step towards the natural goal of estimating the reliability of a hedge network as was done for regular graphs [13].

4 Counting Hypergraph Min-cuts

In this section, we explore randomized contraction in undirected hypergraphs. In particular, we give a random contraction process that yields any fixed min-cut with probability at least $\frac{1}{\binom{n}{2}}$. This establishes that the number of min-cuts in a hypergraph is at most $\binom{n}{2}$.

Let $G = (V, E)$ be an undirected hypergraph with m hyperedges and n vertices. The *cardinality* or *rank* of a hyperedge e, denoted $|e|$, is the number of vertices it contains. While our results can be directly derived for weighted hypergraphs, we prefer to demonstrate them for unweighted hypergraphs in this section. For weighted hypergraphs, the only change in our random process is that the probability of choosing a hyperedge needs to be adjusted appropriately according to hyperedge weights.

4.1 The Algorithm. We will denote the *value* of a min-cut in G — the minimum number of hyperedges crossing a cut — by k. In this section, we will assume that the value of k is known, since our goal is to obtain a bound on the number of hypergraph min-cuts and not to give a min-cut algorithm. The random contraction process in Figure 4.1 contracts a sequence of hyperedges with the goal of identifying a min-cut at the end. In each step, we let the above notation represent the corresponding parameters in the current contracted hypergraph. The algorithm iteratively does the following as long as there are more than two vertices

and at least one hyperedge in the hypergraph: It removes all spanning hyperedges adding them to the min-cut, and all hyperedges that have been contracted down to a single vertex. Then, it chooses a hyperedge e to contract with probability:

$$p_e = \frac{1}{m-k}\left(1 - k \cdot \frac{|e|}{\sum_{e\in E}|e|}\right).$$

If the algorithm terminates with two vertices, it adds the surviving hyperedges to the min-cut.

Note that the probability distribution over the hyperedges in any contraction step is valid:

$$\begin{aligned}\sum_{e\in E} p_e &= \sum_{e\in E}\frac{1}{m-k}\cdot\left(1 - k\cdot\frac{|e|}{\sum_{e\in E}|e|}\right)\\ &= \frac{1}{m-k}\cdot\left(m - k\cdot\frac{\sum_{e\in E}|e|}{\sum_{e\in E}|e|}\right)\\ &= 1.\end{aligned}$$

4.2 The Analysis. To build intuition, let us first analyze the algorithm for a very special case. Note that if the hypergraph at any stage is r-uniform (every hyperedge has rank r), then the hyperedge being contracted at that stage is chosen uniformly at random. A hypergraph that was uniform to begin with need not remain so under contractions, but for simplicity in this informal analysis, let us assume it does. Then, in each step, a hyperedge is chosen uniformly at random. Now, note that every *degree cut* — the hyperedges containing a fixed vertex — contains at least k hyperedges. This implies that the sum of hyperedge ranks satisfies $\sum_{e\in E}|e| \geq nk$. Since every hyperedge is of rank r, this implies that the number of hyperedges $m \geq nk/r$. Hence, the probability that a hyperedge in a fixed min-cut is *not* chosen in a rank-r step is:

$$1 - k/m \geq 1 - r/n.$$

After the contraction, the number of vertices decreases to $n-r+1$. Cascading these success probabilities over multiple contractions, where the respective ranks are $r_1, r_2, r_3, \ldots$, gives an overall success probability of at least

$$\begin{aligned}&\frac{n-r_1}{n}\cdot\frac{n-r_1-r_2+1}{n-r_1+1}\cdot\ldots\\ &\quad\ldots\frac{(r_k+r_{k-1})-r_{k-1}}{r_k+r_{k-1}}\cdot\frac{(r_k+1)-r_k}{r_k+1}\\ \geq\ &\frac{1}{\binom{n}{2}}.\end{aligned}$$

We now generalize this argument to arbitrary hypergraphs. The main difference is that we can no longer assume the uniformity in hyperedge rank at each step. Let us again consider a special case where the hypergraph has hyperedges of only two ranks, r_1 and r_2. Furthermore, suppose there are exactly $m/2$ hyperedges of each kind. Again, note that every degree cut contains at least k hyperedges. This implies that the sum of hyperedge ranks satisfies

$$\sum_{e\in E}|e| = (m/2)\cdot(r_1+r_2) \geq nk.$$

It follows that $m \geq nk/r_{av}$, where $r_{av} = (r_1+r_2)/2$ is the average rank of hyperedges. Hence, the probability that a hyperedge in a fixed min-cut is *not* chosen in a rank-r step is:

$$1 - k/m \geq 1 - r_{av}/n.$$

If q_n is the probability of success with n vertices, this allows us to write the recurrence:

$$q_n \geq (1 - r_{av}/n)\cdot(1/(m-k))\cdot\sum_{e\notin C} q_{n-|e|+1}, \tag{4.5}$$

where C is any fixed min-cut in the hypergraph. Note that we are aiming to show that $q_n \geq 1/\binom{n}{2}$, which is a convex function. By using convexity inductively, we can then write Eq. (4.5) as

$$q_n \geq (1 - r_{av}/n)\cdot q_{n-r^C_{av}+1},$$

where r^C_{av} is the average rank of hyperedges over the ones that are not in C. If $r^C_{av} \geq r_{av}$, then this product indeed evaluates to at least $1/\binom{n}{2}$, but the problem is that r^C_{av} may be smaller than r_{av}. In other words, it might be the case that conditioning on the fact that a min-cut edge is not contracted actually decreases the progress we make in terms of reducing the number of vertices in the hypergraph.

Note that this problem arises if the min-cut contains hyperedges of large rank. This suggests that we should bias our random contraction toward choose a hyperedge of small rank. Indeed, this is exactly what we do in the algorithm described above.

We now proceed to the formal analysis of the algorithm, where we will need the following fact.

FACT 4.1. *Let $f(t) = 1/\binom{t}{2}$ for positive integers $t \geq 2$. Then, for any $2 \leq d \leq t-1$,*

$$f(t) \leq \left(1 - \frac{d}{t}\right)\cdot f(t-d+1).$$

Proof. We have

$$\begin{aligned}&\left(1-\frac{d}{t}\right)\cdot f(t-d+1)\\ =\ &\frac{t-d}{t}\cdot\frac{2}{(t-d+1)(t-d)}\\ =\ &\frac{2}{t(t-d+1)}\\ \geq\ &\frac{1}{\binom{t}{2}}\quad(\text{since } d\geq 2).\end{aligned}$$

The Contraction Algorithm for Hypergraphs

1. Initialize S, the output set of hyperedges, to an empty set.
2. Repeat while G has more than two vertices and at least one hyperedge:
   - Remove all *spanning* hyperedges ($|e| = n$) from G, add them to S, and decrease the value of k by their number. If k becomes negative, declare failure and terminate.
   - Otherwise, contract exactly one hyperedge e in G selected with probability:
   $$p_e = \frac{1}{m-k}\left(1 - k \cdot \frac{|e|}{\sum_{e \in E}|e|}\right).$$
   - Remove all hyperedges of cardinality 1 from G.
3. If the algorithm terminates with two vertices, then add all the hyperedges in G to S.
4. Return S.

Figure 3: The random contraction algorithm for hypergraphs

We now prove our main lemma.

LEMMA 4.1. *The probability of the contraction algorithm returning any fixed min-cut is at least* $1/\binom{n}{2}$.

Proof. Let q_n denote the probability of the contraction algorithm returning a fixed min-cut C for an n-vertex hypergraph. Note that the algorithm returns C if it never contracts a hyperedge in C. Therefore, we can write the following recurrence:

$$\begin{aligned} q_n &= \sum_{e \notin C} p_e \cdot q_{n-|e|+1} \\ &= \frac{1}{m-k} \cdot \sum_{e \notin C}\left(1 - k \cdot \frac{|e|}{\sum_{e \in E}|e|}\right) \cdot q_{n-|e|+1}. \end{aligned} \tag{4.6}$$

Since the value of every degree cut is at least k, we have $\sum_{e \in E}|e| \geq nk$ at any stage of the algorithm. Using this observation, we can now simplify Eq. (4.6) to

$$q_n \geq \frac{1}{m-k} \cdot \sum_{e \notin C}\left(1 - \frac{|e|}{n}\right) \cdot q_{n-|e|+1}. \tag{4.7}$$

We will now use induction over n to show that $q_n \geq 1/\binom{n}{2}$. For the base case, $q_2 = 1 = 1/\binom{2}{2}$. (Note that since we remove all the spanning hyperedges in every step, we never run the risk of contracting down to a single vertex. Therefore, using $n = 2$ is valid for a base case.) Using the inductive hypothesis on Eq. 4.7, we have

$$\begin{aligned} & q_n \\ \geq\ & \frac{1}{m-k} \cdot \sum_{e \notin C} \frac{1 - |e|/n}{\binom{n-|e|+1}{2}} \\ \geq\ & \frac{1}{m-k} \cdot \sum_{e \notin C} \frac{1}{\binom{n}{2}} \\ & \text{(using Fact 4.1)} \\ =\ & \frac{1}{\binom{n}{2}}. \end{aligned}$$

Since any fixed min-cut is output by this algorithm with probability at least $1/\binom{n}{2}$, and the algorithm outputs at most one cut in each run, the next theorem follows.

THEOREM 4.1. *The number of min-cuts in an undirected hypergraph is at most* $\binom{n}{2}$.

Note that this bound exactly matches corresponding bounds for graphs [4, 11] and is known to be tight, e.g., on an n-vertex cycle. It is interesting to contrast this result with that of Kogan and Krauthgamer [18] who showed that the number of α-minimum cuts in a hypergraph can be exponential in the cardinality of of the hyperedges. It is natural to ask: what if we analyze the probability that our contraction algorithm returns a fixed α-minimum cut? There are two reasons why this analysis does not yield polynomial bounds on the number of α-minimum cuts for $\alpha > 1$. First, the algorithm must be terminated once the number of vertices decreases to αr, where r is the rank, since the probability

of success can decrease to 0 if we continue contracting hyperedges. At this stage, the algorithm can only output a random cut, which has probability of success that is inversely exponential in αr. This difficulty is avoided by us for $\alpha = 1$ by realizing that the hyperedges whose cardinality equals the number of vertices are spanning and hence appear in every cut that has survived till this point. Therefore, we remove these hyperedges from G and add them to the output set, allowing us to continue the contraction process. The second reason is that the set of α-minimum cuts changes during our algorithm. Since spanning hyperedges have to be removed from G (otherwise, contracting such a hyperedge yields a single vertex graph which is a base case with success probability 0), the value of the min-cut in G changes. This does not affect the minimality of an exact min-cut, but a previously α-minimum cut need not be so after this step. Therefore, even if the algorithm preserved the hyperedges of an α-minimum cut till this point, the probability of returning this cut from this point on can be quite low.

5 A Randomized Algorithm for Hypergraph Min-cut

We start with the algorithm defined by the contraction process in Section 4. For weighted hypergraphs where the weight of hyperedge e is w_e, we replace m by $\sum_{e \in E} w_e$ and multiply the contraction probability of a hyperedge by w_e (this simulates a hypergraph with w_e unweighted copies of hyperedge e, for which our results from Section 4 directly apply). First, we address the fact that the value of a min-cut k is unknown to the algorithm, but is used in the probability distribution for hyperedge contraction. The algorithm runs a binary search on the min-cut value. The two important observations are:

- if k is larger than the min-cut value, then at every contraction step, either $\sum_{e \in E} |e| \cdot w_e \geq nk$, in which case the above analysis is valid, or at least one of the singleton vertex cuts is of value less than k. Hence, the random contraction algorithm yields a cut of value at most k with probability at least $\frac{1}{\binom{n}{2}}$.
- if k is smaller than the min-cut value, the random contraction process yields a cut of value at most k with probability 0.

The sum of edge weights provides an upper bound, and the minimum weight edge provides a lower bound, these two bounds differing by at most a polynomial factor if the edge weights are polynomially bounded. Hence, the number of steps of binary search is logarithmic. The rest of this section assumes that the value of k is exactly the min-cut value.

Since the randomized contraction algorithm outputs a fixed min-cut with probability at least $\frac{1}{\binom{n}{2}}$, it follows that repeating the algorithm $\binom{n}{2} \log n$ times and selecting the cut of minimum size among all the runs yields the min-cut with high probability. A single implementation of randomized contraction has at most $n - 2$ rounds of contractions since each hyperedge contraction reduces the number of vertices by at least 1. After contracting a hyperedge, the algorithm must update the set of vertices by unifying the contracted vertices and also update the size of each hyperedge since it dictates the sampling probabilities for the next contraction. A disjoint sets data structure [22] can be used to efficiently track the partition of vertices into contracted sets. However, since this is not our bottleneck, we choose to use a simpler labeling scheme where each vertex is labeled by the contracted set it belongs to and we relabel the smaller of the sets when two of these sets are merged. (On contracting a hyperedge, multiple sets are merged into one, but this can be implemented as a sequence of mergers of two sets each.) Every vertex is relabeled $O(\log n)$ times, and hence the total time for these updates over all the hyperedge contractions is $O(n \log n)$. To update hyperedge sizes, we maintain a boolean matrix of size mn indexed by the m hyperedges and the labels of contracted vertex sets (whose number is at most n). Initially, every vertex has a distinct label, and hence this matrix is simply the adjacency matrix of the hypergraph. When two sets are merged, all entries of this matrix corresponding to the smaller set are set to 0, and the new entries for the larger set are set to 1 iff either of the corresponding entries for two sets being merged was 1. We claim that these updates can be performed in $O(M \log n)$ time overall, where $M = \sum_e |e|$ is the sum of cardinalities of all the hyperedges, i.e., the number of 1's in the adjacency matrix. In each update, an entry of 1 (corresponding to the smaller of the two contracted vertex sets being merged) is replaced with a 0, and a different entry (corresponding to the larger of the two contracted vertex sets) for the same hyperedge is set to 1. If the entry corresponding to the larger set was already 1 before the contraction, then the update can be charged to the decrease in the number of 1's in the matrix. However, if the entry for the larger set was previously 0, then the number of 1's does not decrease. Nevertheless, the number of times this can happen is at most $O(\log n)$ for any original entry of 1, since the size of the contracted vertex set corresponding to the entry of 1 at least doubles every time. Therefore, the total number of updates is $O(M \log n)$. Therefore, the overall running time of the algorithm is $O(Mn^2 \log^2 n)$ if k is known, and $O(Mn^2 \log^3 n)$ if k is unknown.

We now optimize our algorithm to improve the running time *for unweighted hypergraphs*. The modification in our algorithm is as follows: at any stage, if $M \geq 3nk$, we contract a hyperedge chosen uniformly at random; else, we use the random contraction process that we described earlier. First, we show that the probability of producing a min-cut remains $\Omega(1/n^2)$ in this modified algorithm.

We only need to consider the case of $M \geq 3nk$. As earlier, let us fix a min-cut C. Since $M \geq 3nk$, the sum

of cardinalities of hyperedges not in C is at least $2nk$, i.e., $\sum_{e \notin C} |e| \geq 2nk$. Note that the algorithm succeeds if it never contracts a hyperedge in C. As earlier, we can write the following recurrence, where q_n is the probability of success:

$$q_n = \sum_{e \notin C} p_e \cdot q_{n-|e|+1} = \frac{1}{m} \cdot \sum_{e \notin C} q_{n-|e|+1}. \tag{5.8}$$

We now use induction on n. Eq. 5.8 gives:

$$\begin{aligned}
q_n &\geq \frac{m-k}{m} \cdot \left(\frac{1}{m-k} \cdot \sum_{e \notin C} \frac{1}{(n-|e|+1)^2} \right) \\
&\qquad \text{(by the inductive hypothesis)} \\
&\geq \left(1 - \frac{k}{m}\right) \cdot \frac{1}{\left(n - \frac{\sum_{e \notin C} |e|}{m-k} + 1\right)^2} \\
&\qquad \left(\text{by the convexity of the function } \frac{1}{n^2}\right) \\
&\geq \left(1 - \frac{k}{m}\right) \cdot \frac{1}{\left(n - \frac{\sum_{e \notin C} |e|}{2(m-k)}\right)^2} \\
&\qquad (\text{since } |e| \geq 2 \text{ for all hyperedges } e) \\
&\geq \left(1 - \frac{k}{m}\right) \cdot \frac{1}{\left(n - \frac{2nk}{2(m-k)}\right)^2} \\
&\qquad \left(\text{since } \sum_{e \notin C} |e| \geq 2nk\right) \\
&\geq \left(1 - \frac{k}{m}\right) \cdot \frac{1}{n^2} \cdot \frac{1}{\left(1 - \frac{k}{m}\right)^2} \\
&\geq \frac{1}{n^2}.
\end{aligned}$$

Overall, any run of the contraction algorithm has two phases: in the first phase, while $M \geq 3nk$, the algorithm performs uniform random contraction of hyperedges, and in the second phase, when $M < 3nk$, the algorithm switches to the non-uniform random contraction process described previously. To ensure that there is only one transition, the value of n in the transition threshold is fixed to the number of vertices in the *original graph*.

In the first phase, a uniformly random hyperedge can be chosen in amortized $O(1)$ time by creating a uniformly random permutation of hyperedges denoting the contraction sequence as a preprocessing step. Note that the algorithm must discard hyperedges that represent single vertices or span all the vertices, but this can done lazily if such hyperedges are encountered in the contraction sequence. Then, the total running time of all the hyperedge contractions in the first phase is $O(m + n \log n)$. Since the number of entries in the adjacency matrix D is $O(nk)$ in the second phase, the running time of the second phase is $O(nk \log n)$.

But how does the algorithm identify the transition from the first to the second phase? For this purpose, the algorithm runs a binary search over the length of the prefix of the random permutation in the first phase. In particular, the algorithm contracts the first $m/2$ edges in the random permutation in $O(m)$ time and checks the value of M. If it is larger than $3nk$, then the algorithm does a binary search for the threshold in the last $m/2$ edges; else, it does a binary search in the first $m/2$ edges. In either case, we get a recurrence $T(m) = T(m/2) + O(m)$, which evaluates to $O(m)$ for the first phase. Putting together all the pieces, we can now claim that the running time of a single run of this modified contraction algorithm is $O(m + nk \log n)$, which yields an overall running time of $O((m + nk \log n)n^2 \log n)$ for known k and $O((m + nk \log n)n^2 \log^2 n)$ for unknown k.

THEOREM 5.1. *There is a min-cut algorithm for undirected hypergraphs that runs in $O(Mn^2 \log^2 n)$ time. The running time can be improved to $O((m+nk)n^2 \log^3 n)$ if the hypergraph is unweighted.*

References

[1] András A Benczúr and David R Karger. Approximate s-t min-cuts in $\tilde{o}(n^2)$ time. *SIAM Journal on Computing*, 44(2):290–319, 2015.

[2] Chandra Chekuri and Chao Xu. Computing minimum cuts in hypergraphs. *CoRR*, abs/1607.08682, 2016.

[3] Eddie Cheng. Edge-augmentation of hypergraphs. *Math. Program.*, 84(3):443–465, 1999.

[4] Efim A. Dinitz, Alexander V. Karzanov, and Micael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In A. A. Fridman, editor, *Studies in Discrete Optimization*, pages 290–306. Nauka Publishers, Moscow, 1976.

[5] Michael R Fellows, Jiong Guo, and Iyad A Kanj. The parameterized complexity of some minimum label problems. In *Graph-Theoretic Concepts in Computer Science*, pages 88–99. Springer, 2009.

[6] Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 71–80. ACM, 2011.

[7] Harold N Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

[8] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[9] Jianxiu Hao and James B Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 165–174. Society for Industrial and Applied Mathematics, 1992.

[10] David G. Harris and Aravind Srinivasan. Improved bounds and algorithms for graph cuts and network reliability. In *SODA*, pages 259–278, 2014.

[11] David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *SODA*, volume 93, pages 21–30, 1993.

[12] David R Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 648–657. ACM, 1994.

[13] David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, 29(2):492–514, 1999.

[14] David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000.

[15] David R. Karger. Faster and simpler network (un)reliability estimation. In *FOCS*, 2016.

[16] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

[17] Regina Klimmek and Frank Wagner. A simple hypergraph min cut algorithm. 1996.

[18] Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 367–376. ACM, 2015.

[19] Wai-Kei Mak and DF Wong. A fast hypergraph min-cut algorithm for circuit partitioning. *Integration, the VLSI journal*, 30(1):1–11, 2000.

[20] Maurice Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82(1-2):3–12, 1998.

[21] Romeo Rizzi. NOTE - on minimizing symmetric set functions. *Combinatorica*, 20(3):445–450, 2000.

[22] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[23] Peng Zhang, Jin-Yi Cai, Lin-Qing Tang, and Wen-Bo Zhao. Approximation and hardness results for label cut and related problems. *Journal of Combinatorial Optimization*, 21(2):192–208, 2011.