Hypergraph k-Cut in Randomized Polynomial Time*

Karthekeyan Chandrasekaran† Chao Xu† Xilin Yu†

Abstract

In the hypergraph k-cut problem, the input is a hypergraph, and the goal is to find a smallest subset of hyperedges whose removal ensures that the remaining hypergraph has at least k connected components. This problem is known to be at least as hard as the densest k-subgraph problem when k is part of the input (Chekuri-Li, 2015). We present a randomized polynomial time algorithm to solve the hypergraph k-cut problem for constant k. Our algorithm solves the more general hedge k-cut problem when the subgraph induced by every hedge has a constant number of connected components. In the hedge k-cut problem, the input is a *hedgegraph* specified by a vertex set and a disjoint set of *hedges*, where each *hedge* is a subset of edges defined over the vertices. The goal is to find a smallest subset of hedges whose removal ensures that the number of connected components in the remaining underlying (multi-)graph is at least k. Our algorithm is based on random contractions akin to Karger's min cut algorithm. Our main technical contribution is a distribution over the hedges (hyperedges) so that random contraction of hedges (hyperedges) chosen from the distribution succeeds in returning an optimum solution with large probability.

1 Introduction

A hypergraph is specified by a vertex set and a collection of *hyperedges*, where each hyperedge is a nonempty subset of vertices. The hypergraph k-cut problem is the following (abbreviated Hypergraph-k-Cut): Given a hypergraph, find a smallest subset of hyperedges whose removal ensures that the number of connected components in the remaining hypergraph is at least k. Equivalently, the problem asks for a partitioning of the vertex set into k parts with minimum number of hyperedges crossing the partition (a hyperedge is said to cross a partition if it intersects at least two parts). This is an extension of the classic hypergraph min cut problem and has several applications including clustering in VLSI design and network reliability (e.g., see [16, 8, 1, 27, 25]).

A special case of Hypergraph-k-Cut in which the input is in fact a graph (i.e., all hyperedges have cardinality two) is the graph k-cut problem (abbreviated Graph-k-Cut). Graph-k-Cut has a rich history. When k is part of the input, Goldschmidt and Hochbaum showed that the problem is NP-hard [8] while Saran and Vazirani designed a 2-approximation algorithm [22]. When k is a constant, Goldschmidt and Hochbaum gave the first polynomial time algorithm to solve Graph-k-Cut. Their algorithm runs in time $n^{\Theta(k^2)}$, where n is the number of vertices in the input graph [8]. Karger and Stein [13] designed a randomized algorithm that runs in time $O(n^{2(k-1)}\log^3 n)$ which is also the current-best run-time among randomized algorithms. The deterministic algorithms have been improved over a series of works [11, 12, 24] with the current best run-time being $\tilde{O}(n^{2k})$ due to Thorup [23].

The complexity of Hypergraph-k-Cut has remained an intriguing open problem since the works of Goldschmidt-Hochbaum and Saran-Vazirani. The case of $k = 2$, denoted Hypergraph-2-Cut, is well-known to admit deterministic polynomial time algorithms [16, 14, 17]. When k is part of the input, Hypergraph-k-Cut is NP-hard as observed from Graph-k-Cut. Chekuri and Li [3] recently showed that Hypergraph-k-Cut is at least as hard as the *densest k-subgraph* problem from the perspective of approximability. The densest k-subgraph problem is believed to not admit an efficient constant factor approximation assuming $P \neq NP$; it is known to not admit an efficient $n^{1/(\log\log n)^c}$-approximation for some constant $c > 0$ assuming the exponential time hypothesis [18]. Chekuri-Li's result already illustrates that Hypergraph-k-Cut is significantly harder than Graph-k-Cut when k is part of the input.

When k is a constant, several recent works have aimed at designing polynomial time algorithms but have fallen short because they are efficient/return an optimal solution only for either restricted families of hypergraphs or for restricted values of the constant k. We recall these results now. Fukunaga [6] gave a polynomial time algorithm for Hypergraph-k-Cut in constant rank hypergraphs (the *rank* of a hypergraph is the cardinality of the largest hyperedge). A randomized polynomial time algorithm for Hypergraph-k-Cut in

*Chao and Xilin are supported in part by NSF grants CCF-1526799 and CCF-1319376 respectively.

†University of Illinois, Urbana-Champaign, Email: {karthe,chaoxu3,xilinyu2}@illinois.edu.

constant rank hypergraphs for constant k can also be obtained using the *uniform random contraction* technique of Karger and Stein [13] as illustrated by Kogan and Krauthgamer [15]. Moving to arbitrary rank hypergraphs, Xiao[25] generalized the structural results of Goldschmidt-Hochbaum to design a polynomial time algorithm for HYPERGRAPH-3-CUT based on minimum $s - t$ cut computations in hypergraphs. Okumoto, Fukunaga and Nagamochi [20] reduced HYPERGRAPH-k-CUT for constant k to the node-weighted k-way cut problem in graphs [1] and thus obtained a $2(1 - 1/k)$-approximation. They further improved on this approximation factor for $k = 4, 5, 6$. Thus, it has been open to determine the complexity of HYPERGRAPH-k-CUT for constant $k \geq 4$.

In this work, we present a randomized polynomial time algorithm to solve HYPERGRAPH-k-CUT for constant k in arbitrary rank hypergraphs. To the best of our knowledge, this is the first polynomial time algorithm for HYPERGRAPH-k-CUT for constant k.

Our algorithm addresses a more general problem that has garnered much attention recently. We describe this more general problem now. It is often the case with modern networks that a collection of edges in a graph are interdependent and consequently could fail together—e.g., interconnected nodes in an optical network that share/rely on a single resource. Motivated by such scenarios, Ghaffari-Karger-Panigrahi [7] defined the notion of a hedgegraph. A *hedgegraph* $G = (V, E)$ is specified by a vertex set V and a set E of *disjoint* hedges, where each *hedge* is a subset of edges over the vertices in V. We will denote the *graph underlying a hedgegraph* $G = (V, E)$ to be the *multigraph* whose vertex set is V and whose edges are the union of the edges in the hedges of G. Essentially, the graph underlying a hedgegraph is a multigraph whose edges have been partitioned into hedges. In the context of modern networks, a hedgegraph is a multigraph where each hedge corresponds to a resource and a link between two nodes fails if all hedges containing an edge between the two nodes fail, i.e., all resources that the link relies upon become unavailable.

In the $s-t$ hedge cut problem (abbreviated $\{s,t\}$-HEDGE-CUT), the input is a hedgegraph and the goal is to find a smallest subset of hedges whose removal disconnects s and t in the underlying graph. In the global variant of $\{s,t\}$-HEDGE-CUT (abbreviated HEDGE-2-CUT), the input is a hedgegraph and the goal is to find a smallest subset of hedges whose removal leads to at least two connected components in the underlying graph. It is known that $\{s,t\}$-HEDGE-CUT is NP-hard [26] while Ghaffari et al. showed that HEDGE-2-CUT admits a randomized polynomial time approximation scheme [7]. Ghaffari et al. [7] also gave a quasi-polynomial time algorithm to solve HEDGE-2-CUT. It remains open to design a polynomial time algorithm for HEDGE-2-CUT. We make progress towards this question by addressing an interesting and non-trivial family of instances that we describe next. We will later show that this family already encompasses hypergraphs.

The *span of a hedge* is the number of connected components in the subgraph induced by the edges in the hedge. The *span of a hedgegraph* is the largest span among its hedges. HEDGE-2-CUT in hedgegraphs with span one reduces to HYPERGRAPH-2-CUT (by replacing each hedge by a hyperedge over the set of vertices incident to the edges in the hedge) and is hence solvable efficiently. The complexity of HEDGE-2-CUT for constant span hedgegraphs was raised as an open problem by Coudert et al. [5]. We generalize our techniques for HYPERGRAPH-k-CUT to design a polynomial-time algorithm for HEDGE-2-CUT in constant span hedgegraphs. More generally, we consider the hedge k-cut problem (abbreviated HEDGE-k-CUT): The input is a hedgegraph and the goal is to find a smallest subset of hedges whose removal leads to at least k connected components in the underlying graph. Equivalently, the problem asks for a partitioning of the vertex set into k parts with minimum number of hedges crossing the partition (a hedge is said to cross a partition if it has an edge whose two end-vertices are in different parts). We show that HEDGE-k-CUT for hedgegraphs with constant span is tractable for constant k.

As an additional result, we illustrate that the ideas behind the polynomial time approximation scheme for HEDGE-2-CUT by Ghaffari et al. [7] can be generalized to obtain a polynomial time approximation scheme for HEDGE-k-CUT for constant k (for all input hedgegraphs irrespective of their spans).

1.1 Results.

In the rest of the paper, we will assume that $k \geq 2$ is a constant and avoid stating this explicitly. Our main result is that HEDGE-k-CUT in constant span hedgegraphs admits an efficient algorithm. For a hedge e, we use $r(e)$ to denote the number of vertices incident to the edges in e. Throughout, n will denote the number of vertices in the input hedgegraph $G = (V, E)$, $m := |E|$ is the number of hedges and $M := \sum_{e \in E} r(e)$ denotes the input size.

THEOREM 1.1. *For every non-negative constant integer*

[1]The node-weighted k-way cut problem is the following: Given a graph with weights on the nodes and a collection of terminal nodes, remove a smallest weight subset of non-terminal nodes so that the resulting graph has no path between the terminals.

s, there exists a randomized polynomial time algorithm to solve Hedge-k-Cut *in hedgegraphs with span at most s that runs in time $O(mMn^{ks+k-s}\log n)$ and succeeds with probability at least $1-1/n$.*

We obtain a randomized polynomial time algorithm for Hypergraph-k-Cut as a special case of the above result since Hypergraph-k-Cut reduces to Hedge-k-Cut in 1-span hedgegraphs. For an input hypergraph, let n denote the number of vertices and let M denote the sum of the cardinality of the hyperedges.

Corollary 1.1. *There exists a randomized polynomial time algorithm to solve* Hypergraph-k-Cut *that runs in time $O(Mn^{2k-1}\log n)$ and succeeds with probability at least $1-1/n$.*

Corollary 1.1 saves a factor of m in the run-time in comparison to Theorem 1.1 by a careful observation about our algorithm from Theorem 1.1 as applied to 1-span hedgegraphs. We discuss this observation in the proof of Corollary 1.1.

We mention that for the special case of $k = 2$, namely Hypergraph-2-Cut, Ghaffari et al. gave an algorithm based on random contractions [7]. Their algorithm picks a hyperedge to contract according to a distribution that requires knowledge of the value of the optimum 2-cut. They suggest addressing this issue by a standard technique: employ a binary search to find the optimum 2-cut value. In contrast, our contraction algorithm mentioned in Corollary 1.1 *does not* require knowledge of the optimum cut value and is extremely easy to implement. More importantly, it resolves the complexity of the more general problem of Hypergraph-k-Cut.

We recall that a set C of hyperedges in a hypergraph G is said to be a *k-cut-set* if the removal of C from G results in a hypergraph with at least k connected components. A k-cut-set in G is an *optimal k-cut-set* if its cardinality is equal to the minimum number of hyperedges whose removal from G results in a hypergraph with at least k connected components. Our algorithmic technique also leads to the following bound on the number of optimal k-cut-sets:

Corollary 1.2. *The number of optimal k-cut-sets in an n-vertex hypergraph is $O(n^{2(k-1)})$.*

We note that the bound stated in Corollary 1.2 above recovers (i) the bound on the number of optimal k-cut-sets in graphs by Karger and Stein [13] as well as (ii) the bound on the number of optimal 2-cut-sets in hypergraphs [7, 4].

As a final result, we generalize the techniques underlying the polynomial time approximation scheme for Hedge-2-Cut by Ghaffari et al. [7] to obtain a polynomial time approximation scheme for Hedge-k-Cut. In contrast to Theorem 1.1, this result holds for hedgegraphs with arbitrary span. A set C of hedges in a hedgegraph G is said to be a *hedge k-cut-set* if the removal of C leads to at least k connected components in the underlying graph. For $\alpha > 1$, a hedge k-cut-set C is said to be an *α-approximate minimum hedge k-cut-set* if $|C|$ is at most α times the minimum number of hedges whose removal leads to at least k connected components in the underlying graph.

Theorem 1.2. *For any given $\epsilon > 0$, there exists a randomized algorithm to find a $(1+\epsilon)$-approximate minimum hedge k-cut-set in time $Mn^{O(\log(1/\epsilon))}\log n$ that succeeds with probability at least $1-1/n$.*

Setting ϵ to be a value that is strictly smaller than $1/\lambda$, where λ is the value of a minimum hedge k-cut-set in the input hedgegraph, we observe that a $(1+\epsilon)$-approximate minimum hedge k-cut-set would in fact be a minimum hedge k-cut-set. Thus, Theorem 1.2 gives a quasi-polynomial time algorithm to solve Hedge-k-Cut (the value of λ can be found by a binary search). We mention that the run-time dependence on k in the algorithm mentioned in Theorem 1.2 is in the exponent $O(\log(1/\epsilon))$ and hence the algorithm is not a polynomial-time algorithm if k is not a constant.

Our algorithmic technique can also be used to bound the number of optimal k-cut-sets.

Theorem 1.3. *The number of distinct minimum hedge k-cuts-set in an n-vertex hedgegraph with minimum hedge k-cut-set value λ is $n^{O(k+\log\lambda)}$.*

We omit some of the proofs in this extended abstract—in particular the proof of Corollary 1.2, the proofs of some of the helper lemmas associated with proving Theorem 1.2, and the proof of Theorem 1.3. We defer these proofs to the full version of the paper.

Organization. We present the preliminaries in Section 2 and prove Theorem 1.1 and Corollary 1.1 in Section 3. We give an overview of the proof of Theorem 1.2 in Section 4.

1.2 Related work.

For Graph-k-Cut, when k is part of the input, Saran-Vazirani [22] designed a 2-approximation algorithm. Recently, Manurangsi [19] showed that there is no efficient $(2-\epsilon)$-approximation for any constant $\epsilon > 0$ assuming the *Small Set Expansion Hypothesis* [21].

Approximation algorithms for Hypergraph-k-Cut, Hypergraph-k-Partitioning, and more generally, submodular partitioning problems have been well-studied in the literature. Hypergraph-k-Partitioning is similar in flavor to Hypergraph-k-Cut but it counts a different objective. In Hypergraph-k-Partitioning, the input is a hypergraph and the goal is to find a partitioning of the vertex set into k non-empty parts $V_1, \ldots, V_k$ so that $\sum_{i=1}^{k} |\delta(V_i)|$ is minimum (where $\delta(V_i)$ is the set of hyperedges that cross the part V_i). Hypergraph-k-Partitioning and Hypergraph-k-Cut coincide to Graph-k-Cut when the input hypergraph is a graph. Hypergraph-k-Partitioning is a special case of the submodular k-partitioning problem since the hypergraph cut function is submodular. In the submodular k-partitioning problem, the input is a non-negative submodular set function $f : 2^V \to \mathbb{R}_+$ (given by the evaluation oracle) and the goal is to partition the ground set V into k non-empty sets $V_1, \ldots, V_k$ in order to minimize $\sum_{i=1}^{k} f(V_i)$. Submodular k-partitioning for the case of $k = 3$ is known to admit efficient algorithms [20] while approximation algorithms have been designed for larger constants k [28, 20]. Submodular k-partitioning for $k = 4$ admits efficient algorithms if f is symmetric, i.e., $f(X) = f(V \setminus X)$ for all $X \subseteq V$ [9].

We also mention that approximation algorithms are known for the fixed-terminal variant [28, 2]. In submodular k-way partitioning, the input is a non-negative submodular set function $f : 2^V \to \mathbb{R}_+$ (given by the evaluation oracle) and distinct elements $v_1, \ldots, v_k \in V$, and the goal is to find a partitioning of the ground set V into k non-empty sets $V_1, \ldots, V_k$ such that $v_i \in V_i$ $\forall\ i \in \{1, \ldots, k\}$ in order to minimize $\sum_{i=1}^{k} f(V_i)$. This generalizes hypergraph k-way cut (where the goal is to delete the smallest number of hyperedges in order to disconnect a given collection of k nodes). The current-best known approximation for submodular k-way partitioning is 2 for general submodular functions and $3/2 - 1/k$ for symmetric submodular functions.

The main motivation behind the definition of hedgegraphs is to understand the connectivity properties of modern networks in which the reliability of links have certain dependencies. In particular, the links could depend on a common resource. Two natural models have been considered depending on whether a link fails if either all or at least one of the resources that the link depends upon fails [5]. In this work, our definition of hedgegraphs considers the former model where a link fails only if all resources that the link depends upon fail. The term *hedgegraph* for this model was given by Ghaffari et al. [7] who also showed that Hedge-2-Cut has a polynomial time approximation scheme.

2 Preliminaries

For positive integers a and b with $a < b$, we will follow the convention that the inverse binomial expression $\binom{a}{b}^{-1}$ is 1. The set of positive integers less than or equal to ℓ is denoted as $[\ell]$. Let $G = (V, E)$ be a hedgegraph. We will denote an edge between two vertices a and b by an unordered tuple $\{a, b\}$ and a hedge as a set of edges. We emphasize that the hedges in a hedgegraph are disjoint—if an edge appears in ℓ different hedges, then it contributes ℓ edges to the underlying graph. For a hedge $e \in E$, let $G[e]$ denote the subgraph induced by the edges in e. We emphasize that there are no isolated vertices in $G[e]$. Let $V(e)$ denote the vertices in $G[e]$. We recall that $r(e) = |V(e)|$. Let $s(e)$ denote the number of connected components in $G[e]$, i.e., the span of e. Let $s := \max\{s(e) : e \in E\}$, i.e., s denotes the span of the hedgegraph G. The hedgegraphs of interest in this work satisfy $s \geq 1$.

Our algorithm is based on repeated contractions. Our notion of the contraction operation is identical to the well-known notion that appears in the literature. We define this operation formally for the sake of completeness. Let $U \subset V$ be a subset of vertices in G. We define G *contract* U, denoted G/U, to be a graph on vertex set $V' := (V - U) \cup \{u\}$, where u is a newly introduced vertex, and on hedge set E', where E' is obtained as follows: for each hedge $e \in E$, we define the hedge e' to be

$$\begin{aligned} e' :=& \{(\{a,b\} - U) \cup \{u\} : |\{a,b\} \cap U| = 1, \{a,b\} \in e\} \\ &\cup \{\{a,b\} : \{a,b\} \cap U = \emptyset, \{a,b\} \in e\} \end{aligned}$$

and obtain $E' := \{e' : e' \neq \emptyset, e \in E\}$. For a hedge $e \in E$, let $C_1, \ldots, C_s$ denote the vertex sets of connected components in $G[e]$. The hedgegraph obtained by contracting the hedge e, denoted G/e, is the hedgegraph obtained by contracting the vertex set of each component in $G[e]$ individually, i.e., $G/e := G/C_1/C_2/\ldots/C_s$. We observe that contracting a hedge does not increase the span.

We need the following technical lemma.

Lemma 2.1. (Majorization inequality) *(see Theorem 108 in [10]) Let $y_1, \ldots, y_\ell$ and $x_1, \ldots, x_\ell$ be two finite non-increasing sequence of real numbers in $[a, b]$ with the same sum. Let $f : [a, b] \to \mathbb{R}$ be a convex function. If $\sum_{i=1}^{j} y_i \leq \sum_{i=1}^{j} x_i$ for all $1 \leq j \leq \ell$, then*

$$\sum_{i=1}^{\ell} f(y_i) \leq \sum_{i=1}^{\ell} f(x_i).$$

3 Hedge k-Cut in Constant Span Graphs

In this section, we design an algorithm to solve Hedge-k-Cut in constant span hedgegraphs. For ease of description and analysis, we will focus on the minimum cardinality variant. We will specify how to adapt it to solve the minimum cost variant at the end of the section.

Overview. We recall Karger's random contraction algorithm for graphs (more generally, multigraphs): pick an edge uniformly at random, contract it and repeat until there are 2 vertices left at which point output the edges between the two vertices. In order to analyze the correctness, one can fix a min-cut C and argue that most of the edges will not be in C. Indeed, suppose the value of the min-cut is λ, then every isolating cut (i.e., a cut induced by a single vertex) has value at least λ, and hence the number of edges is at least $n\lambda/2$. Consequently, the probability of picking an edge in C is at most $2/n$. If we use the same algorithm as above to solve Hypergraph-2-Cut, then it is unclear how to analyze the resulting algorithm. This is due to the existence of n-vertex hypergraphs for which a hyperedge from a min-cut could be chosen with probability as large as a constant and not at most $2/n$. We avoid this issue by choosing a hyperedge (or hedge) to contract from a different probability distribution that is not uniform.

The contraction algorithm. We will present an algorithm that outputs a particular minimum hedge k-cut-set with inverse polynomial probability. Hence, returning a hedge k-cut-set with minimum value among the ones output by polynomially many executions of the contraction algorithm will indeed find a minimum hedge k-cut-set with constant probability. For the purposes of Hypergraph-k-Cut, we recommend the reader to consider $s = 1$ in the following algorithm and analysis (with the standard notion of hyperedge contraction).

Let n be the number of vertices in the input hedgegraph $G = (V, E)$. For a hedge $e \in E$, we recall that $r(e)$ is the number of vertices incident to the edges in e and define

$$\delta_e := \begin{cases} \binom{n-r(e)}{k-1} / \binom{n}{k-1} & \text{if } n - r(e) \geq k-1, \text{ and} \\ 0 & \text{if } n - r(e) < k-1. \end{cases}$$

Our contraction algorithm will pick a hedge e with probability proportional to δ_e, contract it, update the values of δ_e based on the new number of vertices and $r(e)$ for every $e \in E$ and repeat until the number of vertices is small. When the number of vertices is at most a constant, we do a brute-force search. We emphasize that our brute-force search outputs all minimum hedge k-cut-sets in the hedgegraph with constant number of vertices. We do this for the purposes of convenience in the correctness analysis.

We note that a hedge e is present in every hedge k-cut-set of G if and only if $|V(G/e)| < k$. We recall that $|V(G/e)| = n - r(e) + s(e)$. Hence, if a hedge e is present in every hedge k-cut-set, then $n - r(e) + 1 \leq n - r(e) + s(e) < k$ and consequently, $\delta_e = 0$. Thus, our algorithm will never contract hedges that are present in every hedge k-cut-set. The algorithm is described in Figure 1.

We now analyze the correctness probability of the contraction algorithm. The following lemma shows a lower bound on the number of hedges in G as a function of the minimum hedge k-cut-set value.

Lemma 3.1. *Let $G = (V, E)$ be a hedgegraph with $m := |E|$ and λ being the minimum hedge k-cut-set value. Then,*

$$m - \lambda \geq \sum_{e \in E} \delta_e.$$

Proof. We will prove the lemma by exhibiting an upper bound on λ by the probabilistic method. Let W be a subset of $k-1$ vertices chosen uniformly at random among all subset of vertices of size $k-1$. Now consider the k-partition of the vertex set given by $\mathcal{P} := \{\{v\} | v \in W\} \cup \{V \setminus W\}$. We claim that the expected value of the hedge k-cut-set given by $\mathcal{P}$ is $m - \sum_{e \in E} \delta_e$ and hence $\lambda \leq m - \sum_{e \in E} \delta_e$.

We now prove the claim. Let e be a hedge in G. The probability that e does not cross $\mathcal{P}$ is $\binom{n-r(e)}{k-1} / \binom{n}{k-1} = \delta_e$. Thus, the probability that e contributes to the hedge k-cut-set $\mathcal{P}$ is $1 - \delta_e$. The claim follows by linearity of expectation.

□

We need the following combinatorial statement.

Lemma 3.2. *Suppose $n > 2(k-1)(s+1)$. Then, for every hedge e with $r(e) \in \{2, \ldots, n-k+1\}$, we have*

$$\delta_e \binom{n - r(e) + s(e)}{(k-1)(s+1)}^{-1} \geq \binom{n}{(k-1)(s+1)}^{-1}.$$

Proof. If $n - r(e) + s(e) < (k-1)(s+1)$, then $\binom{n-r(e)+s(e)}{(k-1)(s+1)}^{-1} = 1$ using the convention fixed at the beginning of Section 2. Since $r(e) \leq n-k+1$, we have $\binom{n-r(e)}{k-1} \geq 1$. Thus,

$$\begin{aligned} \delta_e &= \binom{n-r(e)}{k-1} \binom{n}{k-1}^{-1} \\ &\geq \binom{n}{k-1}^{-1} \\ &\geq \binom{n}{(k-1)(s+1)}^{-1} \end{aligned}$$

since $n > 2(k-1)(s+1)$ and $s \geq 1$.

```
HEDGE-k-CUT(G)
Input: Hedgegraph $G = (V, E)$ with $n := |V|$ and span $s$

  1. Initialize a list of hedge $k$-cut-set candidates with $E$ as an initial candidate.
  2. Repeat:
     (a) If $n \le 2(k-1)(s+1)$, then compute all minimum hedge $k$-cut-sets in $G$ by a brute-force search, add
         them to the list of candidates and go to Step 3.
     (b) For every $e \in E$ such that $\delta_e = 0$ and $|V(G/e)| \ge k$:
           i. compute all minimum hedge $k$-cut-sets in $G/e$ by a brute-force search and add them to the list of
              candidates.
     (c) If $\sum_{e\in E} \delta_e = 0$ go to Step 3.
     (d) Choose a hedge $e$ in $G$ with probability proportional to $\delta_e$.
     (e) Contract and update: $G \leftarrow G/e$, $n \leftarrow |V(G)|$ and update $\delta_e$ for every hedge $e$ in the contracted graph
         $G$.
  3. Output all hedge $k$-cut-sets with minimum value among the candidates.
```

Figure 1: Contraction algorithm for constant span hedgegraphs.

For the rest of the proof, we will assume that $n - r(e) + s(e) \ge (k-1)(s+1)$. We now note that the binomial in the LHS of the lemma is well-defined and non-zero. For notational convenience, let $t = (k-1)(s+1)$. Then we need to show that

$$\delta_e \binom{n - r(e) + s(e)}{t}^{-1} \ge \binom{n}{t}^{-1}. \tag{3.1}$$

We distinguish two cases based on whether $s + 1 \le r(e)$ or $r(e) \le s$.

Case 1: Suppose $s + 1 \le r(e)$. We recall that $s \ge 1$. Since $s(e) \le s$, we have

$$\delta_e \binom{n - r(e) + s(e)}{t}^{-1} \ge \delta_e \binom{n - r(e) + s}{t}^{-1}.$$

Let $x = r(e)$. Then it suffices to show that

$$\binom{n-x}{k-1}\binom{n}{k-1}^{-1}\binom{n-x+s}{t}^{-1} \ge \binom{n}{t}^{-1}. \tag{3.2}$$

Consider the LHS of (3.2) as a function of x. There exists a constant $C_{n,k,s}$ (that depends on n, k and s) using which the LHS can be written as

$$\begin{aligned}
&\mathrm{LHS}(x)\\
&= C_{n,k,s}\frac{(n-x)!(n-x+s-t)!}{(n-x-k+1)!(n-x+s)!}\\
&= C_{n,k,s}\frac{(n-x+s-t)!}{(n-x-k+1)!\prod_{i=1}^{s}(n-x+i)}\\
&= C_{n,k,s}\left(\frac{1}{\prod_{i=k-1}^{t-s-1}(n-x-i)}\right)\left(\frac{1}{\prod_{i=1}^{s}(n-x+i)}\right).
\end{aligned}$$

The last equation follows since $t = (k-1)(s+1)$, and hence $k-1 \le t-s$. From the above expression, we have that LHS(x) is an increasing function of x. Thus we only need to show inequality (3.2) when x is the minimum value in the domain of interest, i.e., $x = r(e) = s+1$. Hence, it suffices to show that

$$\binom{n-s-1}{k-1}\binom{n}{k-1}^{-1}\binom{n-1}{t}^{-1}\binom{n}{t} \ge 1. \tag{3.3}$$

To show the above, we write out the LHS of (3.3):

$$\begin{aligned}
\text{LHS of (3.3)} &= \frac{(n-s-1)!(n-k+1)!}{(n-s-k)!(n-1)!(n-t)}\\
&= \frac{\prod_{i=s+1}^{s+k-1}(n-i)}{(n-t)\prod_{i=1}^{k-2}(n-i)}.
\end{aligned}$$

In order to show that LHS of (3.3) ≥ 1, we need to show that the denominator is no greater than the numerator. Taking negative logarithm of both the denominator and the numerator, we only need to show that

$$\sum_{i=s+1}^{s+k-1}(-\log(n-i)) \le \sum_{i=1}^{k-2}(-\log(n-i)) - \log(n-t). \tag{3.4}$$

We recall that $t = (k-1)(s+1)$. We know that $\sum_{i=s+1}^{s+k-1}(n-i) = (2n-2s-k)(k-1)/2 = (n-t) + \sum_{i=1}^{k-2}(n-i)$ and $\sum_{i=s+1}^{s+j}(n-i) < \sum_{i=1}^{j}(n-i)\ \forall j \in [k-2]$. Since negative logarithm is a convex function, inequality (3.4) follows by applying Lemma 2.1 using the choice $\ell := k-1$, $y_i := n-s-i$, $x_i := n-i$ for every $i \in [k-2]$ and $y_{k-1} := n-s-(k-1), x_{k-1} := n-t$.

Case 2: Suppose $r(e) \le s$. By the assumptions of the lemma, we have $r(e) \ge 2$ and hence $s \ge 2$. We recall that $r(e)$ is the number of vertices incident to the edges in e while $s(e)$ is the number of connected components in the subgraph induced by the edges in e. Hence, $r(e) - s(e) \ge r(e)/2$. Consequently, we have

$$\delta_e \binom{n-r(e)+s(e)}{t}^{-1} \ge \delta_e \binom{n-r(e)/2}{t}^{-1}.$$

Let $x = r(e)$. Then it suffices to show that

$$(3.5)\qquad \binom{n-x}{k-1}\binom{n}{k-1}^{-1}\binom{n-x/2}{t}^{-1} \ge \binom{n}{t}^{-1}.$$

Proceeding similarly to the analysis in Case 1 above, we can show that the LHS of (3.5) is an increasing function of x. Then we only need to show inequality (3.5) for $x = 2$, i.e., we need to show that

$$(3.6)\qquad \binom{n-2}{k-1}\binom{n}{k-1}^{-1}\binom{n-1}{t}^{-1}\binom{n}{t} \ge 1.$$

Inequality (3.6) is a special case of inequality (3.3), which we have already proven in Case 1. This concludes our proof for the combinatorial statement. □

We now show a lower bound on the success probability of the algorithm.

THEOREM 3.1. *For an n-vertex m-hedge input hedgegraph with span s, the contraction algorithm given above outputs any fixed minimum hedge k-cut-set with probability at least*

$$\binom{n}{(k-1)(s+1)}^{-1}.$$

Moreover, for constant k and s, it can be implemented to run in time $O(nmM)$.

Proof. For a hedgegraph H, let $\mathcal{O}(H)$ denote the set of optimal k-cut-sets in H. For $C \in \mathcal{O}(H)$, let $q(H, C)$ denote the probability that the algorithm executed on H outputs C. Let $\mathcal{G}_{n,s}$ be the set of n-vertex hedgegraphs with span at most s. We define

$$q_n := \inf_{H\in\mathcal{G}_{n,s}} \min_{C\in\mathcal{O}(H)} q(H, C).$$

We will prove by induction on n that $q_n \ge \binom{n}{(k-1)(s+1)}^{-1}$. Let $G = (V, E) \in \mathcal{G}_{n,s}$ with $C \in \mathcal{O}(G)$. Let us define $m := |E|$ and $\lambda := |C|$.

We first note that the algorithm will terminate in finite time. This is because either the number of vertices is strictly decreasing in each iteration and the algorithm reaches the base case in Step 2(a) or the condition is met in Step 2(c). If C is in the list of candidates, it will be part of the output because it is a minimum hedge k-cut-set. Therefore we just have to prove that C is in the list of candidates.

To base the induction, we consider $n \le 2(k-1)(s+1)$. For such n, we have $q(G, C) = 1$ since the algorithm solves such instances exactly by a brute-force search and returns all minimum hedge k-cut-sets, hence $q_n = 1$.

We now show the induction step. We begin by addressing two easy cases: (i) Suppose $\delta_e = 0$ for some hedge $e \in E \setminus C$. Since $e \in E \setminus C$, we know that contracting e does not destroy C, so C is still a minimum hedge k-cut-set in G/e. We also know that $|V(G/e)| \ge k$. This is because contracting any hedge f with $|V(G/f)| < k$ would destroy all hedge k-cut-sets but C survives the contraction of e. Since $\delta_e = 0$ and $|V(G/e)| \ge k$, Step 2(b)i will add all minimum hedge k-cut-sets in G/e including C to the list of candidates, so $q(G, C) = 1$. (ii) Suppose $C = E$. Then, all hedges are present in every hedge k-cut-set. Therefore, $\delta_e = 0$ for every hedge $e \in E$. So the algorithm executes only one iteration of Step 2 and will go to Step 3 after executing Step 2(c). Since all hedges are present in every hedge k-cut-set, contracting any hedge $e \in E$ will destroy all hedge k-cut-sets. Consequently, Step 2(b) of the algorithm will not find any candidate and Step 3 will correctly return all hedges in G since the initialized list contains E as a candidate. Hence, $q(G, C) = 1$.

Thus, we may assume that (i) $n > 2(k-1)(s+1)$, (ii) $\delta_e > 0$ for all $e \in E \setminus C$, and (iii) $E \setminus C \ne \emptyset$. In particular, (ii) and (iii) imply that $\sum_{e\in E}\delta_e > 0$. Let $p_e := \delta_e / \sum_{e\in E}\delta_e$ for every $e \in E$. We note that $(p_e)_{e\in E}$ is a probability distribution supported on the hedges because $p_e \ge 0\ \forall e \in E$ and $\sum_{e\in E} p_e = 1$. The algorithm picks a hedge e to contract according to the distribution defined by $(p_e)_{e\in E}$. We note that since $\sum_{e\in E}\delta_e > 0$, we have $\delta_e > 0$ for some $e \in E$ and thus we will contract some hedge.

The algorithm executed on G outputs C if the hedge e that it contracts is not in C and the algorithm executed on the contracted hedgegraph G/e outputs C. Let $e \in E \setminus C$. The hedgegraph G/e has $n - r(e) + s(e) < n$ vertices and moreover, the span of G/e is at most s and hence $G/e \in \mathcal{G}_{n-r(e)+s(e),s}$. Furthermore, the k-cut-set C is still a minimum hedge k-cut-set in G/e and hence $C \in \mathcal{O}(G/e)$. Thus, $q(G/e, C) \ge q_{n-r(e)+s(e)}$ by definition. Thus, we have

$$\begin{aligned}
q(G,C) \geq& \sum_{e\in E\setminus C} p_e \cdot q(G/e,C)\\
\geq& \sum_{e\in E\setminus C} p_e \cdot q_{n-r(e)+s(e)}\\
=& \frac{1}{\sum_{e\in E}\delta_e}\sum_{e\in E\setminus C}\delta_e\cdot q_{n-r(e)+s(e)}\\
&(\text{since } p_e=\delta_e/\sum_{e\in E}\delta_e)\\
\geq& \frac{1}{\sum_{e\in E}\delta_e}\sum_{e\in E\setminus C}\delta_e\cdot\binom{n-r(e)+s(e)}{(k-1)(s+1)}^{-1}.\\
&(\text{by induction hypothesis})
\end{aligned}$$

For every $e\in E\setminus C$, we know that $\delta_e>0$, which implies that $n-r(e)\geq k-1$ by definition of δ_e. Moreover, by the assumption on G, we have that $n>2(k-1)(s+1)$. Hence, by Lemma 3.2, for every hedge $e\in E\setminus C$, we have

$$\delta_e\cdot\binom{n-r(e)+s(e)}{(k-1)(s+1)}^{-1}\geq\binom{n}{(k-1)(s+1)}^{-1}.$$

Substituting this in the previously derived lower bound for $q(G,C)$, we have

$$\begin{aligned}
q(G,C) \geq& \frac{1}{\sum_{e\in E}\delta_e}\sum_{e\in E\setminus C}\binom{n}{(k-1)(s+1)}^{-1}\\
=& \frac{m-\lambda}{\sum_{e\in E}\delta_e}\binom{n}{(k-1)(s+1)}^{-1}\\
&(\text{since } |C|=\lambda \text{ and } |E|=m)\\
\geq& \binom{n}{(k-1)(s+1)}^{-1}.\\
&(\text{by Lemma 3.1})
\end{aligned}$$

In all cases, we have shown that $q(G,C)\geq\binom{n}{(k-1)(s+1)}^{-1}$ for an arbitrary $G\in\mathcal{G}_{n,s}$ and an arbitrary $C\in\mathcal{O}(G)$. Therefore, we have

$$q_n=\inf_{H\in\mathcal{G}_{n,s}}\min_{C\in\mathcal{O}(H)}q(H,C)\geq\binom{n}{(k-1)(s+1)}^{-1}.$$

This concludes our proof of the correctness probability by induction.

We now analyze the running time of the contraction algorithm. A hedge contraction operation takes $O(M)$ time: To contract a hedge e, we construct a hash table of the vertices in the hedge, which also stores which component each vertex is in. The second step is contraction. We process every hedge and mark a vertex if it needs to be contracted. If so, we also mark which vertex it contracts to. Marking vertices takes $O(1)$ time per vertex encountered in a hedge as we only need to check if it is in the hash table that we constructed. Therefore, marking vertices in all hedges takes $O(M)$ time in total. Then, we replace all the marked vertices with the new vertices and update the hedges accordingly in $O(M)$ time. Hence the contraction operation can be implemented to run in $O(M)$ time.

We analyze the run-time for one iteration of Step 2. The brute-force operation in Step 2(a) takes $O(Mk^{2(k-1)(s+1)})$ time. The for-loop in Step 2(b) applies at most $O(m)$ contractions. Each contraction takes $O(M)$ time and each brute-force search for minimum hedge-k-cut-set takes $O(Mk^{k+s})$ time. Hence Step 2(b) runs in time $O(mM)$. Step 2(c) verifies if $\sum_{e\in E}\delta_e=0$ which can be done in $O(m)$ time. Step 2(d) picks a random hedge given a probability distribution on the hedges which again takes $O(m)$ time. Step 2(e) contracts and updates the δ_e values. Contraction takes $O(M)$ time. In order to update the δ_e values, we can precompute $\binom{a}{k-1}$ for all $k-1\leq a\leq n$ in $O(n(k-1))$ arithmetic operations. After every contraction, we can thus update δ_e for each e in constant time using the table. Now, we can compute $\sum_{e\in E}\delta_e$ in $O(|E|)=O(m)$ time. With these values, the probability p_e for all $e\in E$ can be found in $O(m)$ time. Hence, the total run-time of one iteration of Step 2 is $O(mM)$.

Since the number of vertices strictly decreases after each contraction, the total number of iterations of Step 2 is at most n. By the above discussion, the contraction algorithm can be implemented to run in $O(nmM)$ time. We mention that the bottleneck of the algorithm is Step 2(b)i. If Step 2(b)i is never executed, then the running time is $O(nM)$.

□

The contraction algorithm can be adapted to solve the min-weight variant, where each hedge e has weight $w(e)$, and the goal is to find a subset of hedges of minimum total weight to remove so that the underlying graph has at least k connected components. In this case, we set $\delta_e:=w(e)\binom{n-r(e)}{k-1}/\binom{n}{k-1}$ if $n-r(e)\geq k-1$ and $\delta_e:=0$ if $n-r(e)<k-1$, and run the same contraction algorithm as above. The correctness and run-time arguments are analogous to the one in Theorem 3.1 and we avoid repeating in the interests of brevity.

Theorem 1.1 follows from Theorem 3.1 by executing the contraction algorithm $\binom{n}{(k-1)(s+1)}\log n$ times and outputting a hedge k-cut-set with the minimum value among all executions. We next focus on the special case of HYPERGRAPH-k-CUT. We restate and prove

Corollary 1.1.

Corollary 1.1. *There exists a randomized polynomial time algorithm to solve* Hypergraph-k-Cut *that runs in time $O(Mn^{2k-1}\log n)$ and succeeds with probability at least $1-1/n$.*

Proof. We will show that a hypergraph can be transformed to a hedgegraph with span one without changing the value of any k-cut-set. By Theorem 1.1, such a transformation immediately gives a randomized polynomial time algorithm to solve Hypergraph-k-Cut that runs in time $O(nmMn^{2(k-1)}\log n)$ and succeeds with probability at least $1-1/n$. We discuss the run-time improvement after showing the transformation.

Let $G=(V,E)$ be an input hypergraph. We construct a hedgegraph $H=(V,E')$, where E' is obtained as follows: for every hyperedge $e\in E$, fix an arbitrary vertex $v\in e$ and introduce a hedge $e'\in E'$ consisting of edges $\{v,u\}$ for all $u\in e-\{v\}$. Thus, the subgraph induced by the edges in e', i.e., $G[e']$, is a star centered at v that is adjacent to all the vertices in e and hence has span one. We emphasize that the constructed hedges are disjoint, i.e., if an edge appears in ℓ constructed hedges, then the underlying graph has ℓ copies of the edge with each copy being present in one of the hedges.

We now show that the value of any k-cut-set is preserved by this transformation. Let $\{V_1,\ldots,V_k\}$ denote a partitioning of the vertex set V into k non-empty parts. We claim that a hyperedge e crosses the partition $\{V_1,\ldots,V_k\}$ in the hypergraph G if and only if the corresponding hedge e' crosses the partition $\{V_1,\ldots,V_k\}$ in the hedgegraph H. Suppose e' crosses the partition $\{V_1,\ldots,V_k\}$ in the hedgegraph H. Consider the center vertex v of e'. Without loss of generality, let $v\in V_1$. Then there exists a vertex $u\in e'\cap V_j$ for some $j\in[k]\setminus\{1\}$. Now $u,v\in e$, and hence e crosses $\{V_1,\ldots,V_k\}$ in the hypergraph G. On the other hand, suppose e crosses the partition $\{V_1,\ldots,V_k\}$ in the hypergraph G. Consider the center vertex v of the star corresponding to e'. Without loss of generality, let $v\in V_1$ and suppose e intersects V_1 and V_j for some $j\in[k]\setminus\{1\}$. Let $u\in V_j\cap e$. Then $v\in V_1$ while $u\in V_j$ and hence e' crosses $\{V_1,\ldots,V_k\}$ in the hedgegraph H.

We now argue that the algorithm will never execute Step 2(b)i for hedgegraphs with span one. For every hedge e with $\delta_e=0$, we have that $n-r(e)<k-1$ and consequently $|V(G/e)|=n-r(e)+s(e)=n-r(e)+1<k$. Hence, we would have no hedges e in the hedgegraph with $\delta_e=0$ and $|V(G/e)|\geq k$.

This observation shows that the running time is $O(nM)$ as analyzed in the proof of Theorem 3.1. Now, the corollary follows by running the modified contraction algorithm $\binom{n}{2(k-1)}\log n$ times and returning the hedge k-cut-set with minimum value among all executions.

□

4 PTAS for Hedge-k-Cut

In this section, we provide a polynomial time approximation scheme and a quasi-polynomial time exact algorithm for Hedge-k-Cut for constant k. We generalize the contraction approach for Hedge-2-Cut given by Ghaffari et al. [7]. In their contraction algorithm, Ghaffari et al. distinguish large and small hedgegraphs based on the existence of small, medium, and large hedges. We generalize these definitions for the purposes of Hedge-k-Cut and handle the cases similarly.

Let $G=(V,E)$ be a hedgegraph with $n:=|V|$. We define a hedge e to be *small* if $r(e)<n/(4(k-1))$, *moderate* if $n/(4(k-1))\leq r(e)<n/(2(k-1))$, and *large* if $r(e)\geq n/(2(k-1))$. We define a hedgegraph to be *large* if it contains at least one large hedge, and to be *small* otherwise. We use the algorithm in Figure 2.

We next give an overview of the analysis of the correctness probability of the algorithm. We omit some of the proofs here and defer them to the full version of the paper. The following lemma bounds the number of branching steps performed by the algorithm.

Lemma 4.1. *The total number of branching steps in one execution of the contraction algorithm on an n-vertex hedgegraph is at most*

$$\log_{\frac{8(k-1)}{8(k-1)-1}} n.$$

The next two lemmas will be used to lower bound the success probability of the algorithm in returning a $(1+\epsilon)$-approximate minimum hedge k-cut-set. We recall that for a hedge e, the number of vertices incident to the edges in e is denoted by $r(e)$.

Lemma 4.2. *If $G=(V,E)$ is an n-vertex small hedgegraph with C being a minimum hedge k-cut-set in G with value λ, then*

(i) $\sum_{e\in E\setminus C} r(e)\geq n\lambda/(2(k-1))$ *and*

(ii) $m\geq 2\lambda$.

Lemma 4.3. *For every $x\in(0,1/2)$ and $c\geq 4$, we have $(1-x)\cdot(1-x/c)^{-3c/2}\geq 1$.*

We now lower bound the success probability of the algorithm.

Lemma 4.4. *For an n-vertex input hedgegraph, the contraction algorithm given above returns a $(1+\epsilon)$-approximate minimum hedge k-cut-set with probability*

Input: Hedgegraph $G = (V, E)$
Contract(G):

1. If G has k vertices, then return $F = E$.
2. Else if the graph underlying G has at least k components, then return $F = \emptyset$.
3. Else, remove all hedges e such that $n - r(e) + s(e) \leq k - 1$ from G and add them to F.
4. If G is a small hedgegraph, then contract a hedge e chosen uniformly at random from E. Let the resulting hedgegraph be H and return $F \cup$ **Contract**(H).
5. Else, let L be the set of large and moderate hedges in G. Perform a *branching step*: go to one of the two following branches with equal probability.
   (a) Remove all large and moderate hedges from G to obtain a hedgegraph H_1 and return $F \cup L \cup$ **Contract**(H_1).
   (b) Contract a hedge e chosen uniformly at random from L to obtain a hedgegraph H_2 and return $F \cup$ **Contract**(H_2).

Figure 2: Contraction algorithm for arbitrary span hedgegraphs

$n^{-O(\log(1/\epsilon))}$. *Moreover, it can be implemented to run in time* $O(Mn)$.

Proof. We first note that any hedge e with $n - r(e) + s(e) \leq k - 1$ must be in every hedge k-cut-set. By deleting such hedges from the input hedgegraph and adding them to the output set F, the algorithm ensures that it never contracts hedges such that the resulting hedgegraph has at most $k - 1$ components (vertices). So, the algorithm always outputs a hedge k-cut-set. In the rest of the proof, we will lower bound the probability that the output is a $(1+\epsilon)$-approximate minimum hedge k-cut-set for a fixed $\epsilon > 0$.

Let $\mathcal{H}(n, \ell)$ be the family of hedgegraphs on n vertices for which the contraction algorithm will always terminate using at most ℓ branchings. We say that the algorithm *succeeds* on an input hedgegraph H if it outputs a $(1 + \epsilon)$-approximate minimum hedge k-cut-set of H. Let $q(H)$ denote the probability that the algorithm succeeds on H. We define

$$q_{n,\ell} := \inf_{H \in \mathcal{H}(n,\ell)} q(H).$$

For notational simplicity, let $\gamma := \epsilon/(1 + \epsilon)$. We will prove by induction on n that

$$q_{n,\ell} \geq n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell} \quad \forall\, n \geq k.$$

Let $G \in \mathcal{H}(n, \ell)$, with vertex set $V = [n]$ and hedge set E.

To base the induction, we consider $n = k$. For such n, the algorithm returns the only hedge k-cut-set of G, so $q(G) = 1$ and hence $q_{n,\ell} = 1$.

We next show the induction step. If G has at least k components, the algorithm returns the only minimum hedge k-cut-set, which is the empty set, and hence $q(G) = 1$. Thus, we may assume that $n > k$ and the graph underlying G has fewer than k components. Let us fix a minimum hedge k-cut-set C of G and suppose that its value is λ. We will first show that $q(G) \geq n^{-6(k-1)} \cdot (\gamma/2)^{\ell}$ for all $G \in \mathcal{H}(n, l)$. We distinguish three cases and handle them differently.

1. Suppose G is small. The algorithm succeeds if it contracts a hedge e that is not in C and the algorithm succeeds on the resulting hedgegraph G/e which has $n - r(e) + s(e)$ vertices. Hence, $q(G) \geq 1/m \cdot \sum_{e \in E \setminus C} q(G/e)$. We note that $G/e \in \mathcal{H}(n - r(e) + s(e), \ell)$ and that $n - r(e) + s(e) \leq n - r(e)/2$ for any hedge e. Therefore,

$$\begin{aligned} & q(G) \\ \geq & \frac{1}{m} \sum_{e \in E \setminus C} q(G/e) \\ \geq & \frac{1}{m} \sum_{e \in E \setminus C} q_{n-r(e)+s(e),\ell} \end{aligned}$$
(by definition of $q_{n,l}$)

$$\geq \frac{1}{m}\sum_{e\in E\setminus C}(n-r(e)+s(e))^{-6(k-1)}\cdot\left(\frac{\gamma}{2}\right)^{\ell}$$

(by inductive hypothesis)

$$\geq \frac{1}{m}\sum_{e\in E\setminus C}\left(n-\frac{r(e)}{2}\right)^{-6(k-1)}\cdot\left(\frac{\gamma}{2}\right)^{\ell}$$

(by $n-r(e)+s(e)\leq n-r(e)/2$)

$$=\frac{m-\lambda}{m}\left(\frac{\gamma}{2}\right)^{\ell}\frac{1}{m-\lambda}\sum_{e\in E\setminus C}\left(n-\frac{r(e)}{2}\right)^{-6(k-1)}.$$

Since $k\geq 2$, and $n-r(e)/2\geq 1$ for all hedges e, the function $f(r(e)):=(n-r(e)/2)^{-6(k-1)}$ is convex as a function of $r(e)$ for every hedge $e\in E$. By Jensen's inequality, we obtain

$$\frac{1}{m-\lambda}\sum_{e\in E\setminus C}\left(n-\frac{r(e)}{2}\right)^{-6(k-1)} \geq \left(n-\frac{\sum_{e\in E\setminus C}r(e)}{2(m-\lambda)}\right)^{-6(k-1)}.$$

Therefore, we have

$$q(G)$$
$$\geq\frac{m-\lambda}{m}\cdot\left(\frac{\gamma}{2}\right)^{\ell}\cdot\left(n-\frac{\sum_{e\in E\setminus C}r(e)}{2(m-\lambda)}\right)^{-6(k-1)}$$
$$\geq\frac{m-\lambda}{m}\cdot\left(\frac{\gamma}{2}\right)^{\ell}\cdot\left(n-\frac{n\lambda/(2(k-1))}{2m}\right)^{-6(k-1)}$$

(by Lemma 4.2)

$$=(1-\frac{\lambda}{m})\left(\frac{\gamma}{2}\right)^{\ell}n^{-6(k-1)}\left(1-\frac{\lambda}{4(k-1)m}\right)^{-6(k-1)}$$
$$=\left(\frac{\gamma}{2}\right)^{\ell}n^{-6(k-1)}\cdot(1-x)\left(1-\frac{x}{4(k-1)}\right)^{-6(k-1)}.$$

(for $x:=\lambda/m\in(0,1/2)$)

The last equality follows by setting $x:=\lambda/m$. We have $x\in(0,1/2)$ since $m\geq 2\lambda$ by Lemma 4.2. We recall that we would like to prove that $q(G)\geq n^{-6(k-1)}\cdot(\gamma/2)^{\ell}$, so we only need to prove that $(1-x)\left(1-x/(4(k-1))\right)^{-6(k-1)}\geq 1$ for $x\in(0,1/2)$. Let $c=4(k-1)$, then $(1-x)\left(1-x/(4(k-1))\right)^{-6(k-1)}=(1-x)(1-x/c)^{-3c/2}$. Since $k\geq 2$, we have $c\geq 4$. Therefore, by Lemma 4.3, we have $(1-x)(1-x/c)^{-3c/2}\geq 1$ for $x\in(0,1/2)$. This concludes our proof that $q(G)\geq n^{-6(k-1)}\cdot(\gamma/2)^{\ell}$ in case 1.

2. Suppose G is large and $|L\setminus C|\geq\gamma\cdot|L|$. The algorithm succeeds if it goes to branch (b), contracts a hedge $e\in L\setminus C$, and succeeds on the resulting graph $H_2=G/e$. By the condition that $|L\setminus C|\geq\gamma\cdot|L|$, the probability of picking a hedge $e\in L\setminus C$ is γ. Hence, $q(G)\geq 1/2\cdot\gamma\cdot q(H_2)$. The algorithm contracts either a moderate or a large hedge e for which $r(e)\geq n/(4(k-1))$ and H_2 has $n-r(e)+s(e)$ vertices where $n-r(e)+s(e)\leq n-r(e)/2$. Hence, H_2 has at most $n-r(e)/2\leq n-n/(8(k-1))=((8(k-1)-1)/(8(k-1))\cdot n$ vertices. We also note that $H_2\in\mathcal{H}(|V(H_2)|,\ell-1)$. Therefore,

$$q(G)\geq\frac{1}{2}\cdot\gamma\cdot q(H_2)$$
$$\geq\frac{1}{2}\cdot\gamma\cdot q_{|V(H_2)|,\ell-1}$$

(by definition of $q_{n,l}$)

$$\geq\frac{1}{2}\cdot\gamma\cdot(|V(H_2)|)^{-6(k-1)}\cdot\left(\frac{\gamma}{2}\right)^{\ell-1}$$

(by inductive hypothesis)

$$\geq\frac{1}{2}\cdot\gamma\cdot\left(\frac{(8(k-1)-1)}{8(k-1)}\cdot n\right)^{-6(k-1)}\left(\frac{\gamma}{2}\right)^{\ell-1}$$

(by $|V(H_2)|\leq\frac{(8(k-1)-1)}{8(k-1)}\cdot n$)

$$=\left(\frac{(8(k-1)-1)}{8(k-1)}\cdot n\right)^{-6(k-1)}\left(\frac{\gamma}{2}\right)^{\ell}$$
$$\geq n^{-6(k-1)}\cdot\left(\frac{\gamma}{2}\right)^{\ell}.$$

3. Suppose G is large and $|L\setminus C|<\gamma\cdot|L|$. Since $|L\setminus C|<\gamma\cdot|L|$, we have that $|L\cap C|=|L|-|L\setminus C|>(1-\gamma)\cdot|L|$. We will show that the algorithm succeeds if it goes to branch (a), and succeeds on the resulting graph $H_1=G-L$.

Suppose the algorithm follows branch (a). The value of a minimum hedge k-cut-set in H_1 is at most $|C-L|=|C|-|L\cap C|<|C|-(1-\gamma)|L|$. If the algorithm returns a $(1+\epsilon)$-approximate minimum hedge k-cut-set of H_1, then the algorithm would return a set of size at most $(1+\epsilon)(|C|-|L|(1-\gamma))=|C|(1+\epsilon)-|L|$. Consequently, the algorithm returns a hedge k-cut-set of size at most $(|C|(1+\epsilon)-|L|)+|L|=|C|(1+\epsilon)$ for G. This shows that if the algorithm returns a $(1+\epsilon)$-approximate minimum hedge k-cut-set in the hedgegraph H_1 obtained in branch (a), then the algorithm returns a $(1+\epsilon)$-approximate minimum hedge k-cut-set in the hedgegraph G. Also, by definition, $H_1\in\mathcal{H}(n,\ell-1)$. Therefore,

$$q(G)\geq\frac{1}{2}\cdot q(H_1)\geq\frac{1}{2}\cdot q_{n,\ell-1}.$$

We will now prove that $q(G) \geq n^{-6(k-1)} \cdot (\gamma/2)^{\ell}$ by induction on ℓ. The following claim shows the base case of the statement, i.e., for $\ell = 0$:

Claim 4.1. *$q(G) \geq n^{-6(k-1)}$ for all $n \geq k$ and $G \in \mathcal{H}(n, 0)$.*

Proof. We prove by induction on n. For the base case where $n = k$, we have $q(G) = 1 \geq n^{-6(k-1)}$ for all $G \in \mathcal{H}(n, 0)$. For the inductive step, consider $G \in \mathcal{H}(n, 0)$ to be a hedgegraph on $n > k$ vertices and m hedges with a fixed minimum hedge k-cut-set C in G. We note that G is small since $G \in \mathcal{H}(n, 0)$, therefore we are in case 1. Hence, we have

$$q(G) \geq n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell} = n^{-6(k-1)}.$$

by the same proof as that of case 1. $\square$

By Claim 4.1, we have the base case $q(G) \geq n^{-6(k-1)}$ for all $G \in \mathcal{H}(n, 0)$. For the induction step, we use the lower bound and the inductive hypothesis to obtain

$$\begin{aligned} q(G) \geq& \frac{1}{2} \cdot q_{n,\ell-1} \\ \geq& \frac{1}{2} \cdot \left(\frac{\gamma}{2}\right)^{\ell-1} \cdot n^{-6(k-1)} \\ \geq& \left(\frac{\gamma}{2}\right)^{\ell} \cdot n^{-6(k-1)}. \end{aligned}$$

In all cases, we have shown that $q(G) \geq (\gamma/2)^{\ell} \cdot n^{-6(k-1)}$ for an arbitrary $G \in \mathcal{H}(n, \ell)$. Therefore, for all $n \geq k$, we have

$$q_{n,\ell} = \inf_{H \in \mathcal{H}(n,\ell)} q(H) \geq n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell}.$$

We know that $\gamma < 1$. Substituting the upper bound on ℓ from Lemma 4.1, we obtain that

$$\begin{aligned} q_{n,\ell} &\geq n^{-6(k-1)} \left(\frac{\gamma}{2}\right)^{\ell} \\ &\geq n^{-6(k-1)} \left(\frac{\gamma}{2}\right)^{\log_{\frac{8(k-1)}{8(k-1)-1}} n}. \end{aligned}$$

If $\epsilon \geq 1$, then $\gamma \geq 1/2$, so $q_{n,\ell}$ is at least inverse polynomial in n. If $\epsilon < 1$, then $\gamma = \epsilon/(1+\epsilon) > \epsilon/2$. So the success probability is at least

$$n^{-6(k-1)} \left(\frac{\epsilon}{4}\right)^{\log_{\frac{8(k-1)}{8(k-1)-1}} n} = n^{-O(\log \frac{1}{\epsilon})}.$$

We now argue that the contraction algorithm can be implemented to run in $O(Mn)$ time. In the contraction algorithm, finding the set of hedges e with $n - r(e) + s(e) \leq k - 1$ and finding the set of large and moderate hedges can each be done in $O(m)$ time. Similar to the proof of Theorem 3.1, a contraction step can be implemented to run in $O(M)$ time by processing hedges one by one to mark contracted vertices and replacing them with a new vertex. Since in one execution of the contraction algorithm there can be at most n contractions, the contraction algorithm can be implemented to run in $O(Mn)$ time. $\square$

Theorem 1.2 follows from Lemma 4.4 by executing the contraction algorithm $n^{O(\log 1/\epsilon)} \log n$ times and returning a hedge k-cut-set with the minimum value among all executions.

Setting ϵ to be a value that is strictly smaller than $1/\lambda$, where λ is the value of a minimum hedge k-cut-set in the input hedgegraph, we observe that a $(1+\epsilon)$-approximate minimum hedge k-cut-set would in fact be a minimum hedge k-cut-set. Hence, we have the following corollary (the value λ can be found by a binary search).

Corollary 4.1. *There exists a randomized algorithm to solve* Hedge-k-Cut *that runs in time $Mn^{O(\log \lambda)} \log n$, where λ is the value of a minimum hedge k-cut-set in the input hedgegraph.*

References

[1] C. Alpert and A. Kahng, *Recent developments in netlist partitioning: A survey*, Integration: the VLSI Journal **19** (1995), no. 1-2, 1–81.

[2] C. Chekuri and A. Ene, *Approximation Algorithms for Submodular Multiway Partition*, Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science, FOCS '11, 2011, pp. 807–816.

[3] C. Chekuri and S. Li, *A note on the hardness of approximating the k-way Hypergraph Cut problem*, Manuscript, `http://chekuri.cs.illinois.edu/papers/hypergraph-kcut.pdf`, 2015.

[4] C. Chekuri and C. Xu, *Computing minimum cuts in hypergraphs*, Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '17, 2017, pp. 1085–1100.

[5] D. Coudert, P. Datta, S. Perennes, H. Rivano, and M.-E. Voge, *Shared Risk Resource Group: Complexity and Approximability Issues*, Research Report RR-5859, INRIA, 2006.

[6] T. Fukunaga, *Computing Minimum Multiway Cuts in Hypergraphs from Hypertree Packings*, Proceedings of the 14th International Conference on Integer Programming and Combinatorial Optimization, IPCO '10, 2010, pp. 15–28.

[7] M. Ghaffari, D. Karger, and D. Panigrahi, *Random Contractions and Sampling for Hypergraph and Hedge Connectivity*, Proceedings of the 28th Annual ACM-

SIAM Symposium on Discrete Algorithms, SODA '17, 2017, pp. 1101–1114.

[8] O. Goldschmidt and D. Hochbaum, *A Polynomial Algorithm for the k-cut Problem for Fixed k*, Mathematics of Operations Research **19** (1994), no. 1, 24–37.

[9] F. Guiñez and M. Queyranne, *The size-constrained submodular k-partition problem*, Manuscript, https://docs.google.com/viewer?a=v&pid=sites&srcid =ZGVmYXVsdGRvbWFpbnxmbGF2aW9ndWluZX pob21lcGFnZXxneDo0NDVlMThkMDg4ZWRlOGI1, 2012.

[10] G. Hardy, J. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, 2nd ed., 1952.

[11] Y. Kamidoi, S. Wakabayashi, and N. Yoshida, *A Divide-and-Conquer Approach to the Minimum k-Way Cut Problem*, Algorithmica **32** (2002), no. 2, 262–276.

[12] Y. Kamidoi, N. Yoshida, and H. Nagamochi, *A Deterministic Algorithm for Finding All Minimum k-Way Cuts*, SIAM Journal on Computing **36** (2007), no. 5, 1329–1341.

[13] D. Karger and C. Stein, *A new approach to the minimum cut problem*, Journal of ACM **43** (1996), no. 4, 601–640.

[14] R. Klimmek and F. Wagner, *A simple hypergraph min cut algorithm*, Internal Report B 96-02 Bericht FU Berlin Fachbereich Mathematik und Informatik, 1995.

[15] D. Kogan and R. Krauthgamer, *Sketching cuts in graphs and hypergraphs*, Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS '15, 2015, pp. 367–376.

[16] E. Lawler, *Cutsets and Partitions of Hypergraphs*, Networks **3** (1973), 275–285.

[17] W.-K. Mak and D. Wong, *A fast hypergraph min-cut algorithm for circuit partitioning*, Integration: the VLSI Journal **30** (2000), no. 1, 1–11.

[18] P. Manurangsi, *Almost-polynomial Ratio ETH-hardness of Approximating Densest k-subgraph*, Proceedings of the 49th Annual ACM Symposium on Theory of Computing, STOC '17, 2017, pp. 954–961.

[19] ———, *Inapproximability of Maximum Biclique Problems, Minimum k-Cut and Densest At-Least-k-Subgraph from the Small Set Expansion Hypothesis*, Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP '17), ICALP '17, 2017, pp. 79:1–79:14.

[20] K. Okumoto, T. Fukunaga, and H. Nagamochi, *Divide-and-conquer algorithms for partitioning hypergraphs and submodular systems*, Algorithmica **62** (2012), no. 3, 787–806.

[21] P. Raghavendra and D. Steurer, *Graph Expansion and the Unique Games Conjecture*, Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC '10, 2010, pp. 755–764.

[22] H. Saran and V. Vazirani, *Finding k Cuts within Twice the Optimal*, SIAM Journal on Computing **24** (1995), no. 1, 101–108.

[23] M. Thorup, *Minimum k-way Cuts via Deterministic Greedy Tree Packing*, Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC '08, 2008, pp. 159–166.

[24] M. Xiao, *An Improved Divide-and-Conquer Algorithm for Finding All Minimum k-Way Cuts*, Proceedings of 19th International Symposium on Algorithms and Computation, ISAAC '08, 2008, pp. 208–219.

[25] ———, *Finding minimum 3-way cuts in hypergraphs*, Information Processing Letters (Preliminary version in TAMC 2008) **110** (2010), no. 14, 554–558.

[26] P. Zhang, J-Y. Cai, L-Q. Tang, and W-B. Zhao, *Approximation and hardness results for label cut and related problems*, Journal of Combinatorial Optimization **21** (2011), no. 2, 192–208.

[27] L. Zhao, *Approximation algorithms for partition and design problems in networks*, Ph.D. thesis, Graduate School of Informatics, Kyoto University, Japan, 2002.

[28] L. Zhao, H. Nagamochi, and T. Ibaraki, *Greedy splitting algorithms for approximating multiway partition problems*, Mathematical Programming **102** (2005), no. 1, 167–183.