CUTS AND CONNECTIVITY IN GRAPHS AND HYPERGRAPHS

BY

CHAO XU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Doctoral Committee:

Professor Chandra Chekuri, Chair
Assistant Professor Karthekeyan Chandrasekaran, Director of Research
Professor Jeff Erickson
Associate Professor Tamás Király, Eötvös Loránd University

Abstract

In this thesis, we consider cut and connectivity problems on graphs, digraphs, hypergraphs and hedgegraphs.

The main results are the following:

- We introduce a faster algorithm for finding the reduced graph in element-connectivity computations. We also show its application to node separation.
- We present several results on hypergraph cuts, including (a) a near linear time algorithm for finding a $(2+\epsilon)$-approximate min-cut, (b) an algorithm to find a representation of all min-cuts in the same time as finding a single min-cut, (c) a sparse subgraph that preserves connectivity for hypergraphs and (d) a near linear-time hypergraph cut sparsifier.
- We design the first randomized polynomial time algorithm for the hypergraph k-cut problem whose complexity has been open for over 20 years. The algorithm generalizes to hedgegraphs with constant span.
- We address the complexity gap between global vs. fixed-terminal cuts problems in digraphs by presenting a $2-\frac{1}{448}$ approximation algorithm for the global bicut problem.

To my parents and my wife.

Acknowledgments

This thesis would not have been possible without the help and encouragement of many people.

First and foremost, I would like to thank my advisors Karthik Chandrasekaran and Chandra Chekuri. I am fortunate to have the opportunity to work with both of them. Karthik's enthusiasm and diligence for research has been a great inspiration. Chandra's help for establishing my academic connections has been invaluable. Both helped me greatly with research, writing, presentation skills and academic aspects in general. They were also able to understand my situations and gave me important advices in various aspects of my life.

I am very fortunate to be a student of Jeff Erickson for the first two years. Jeff helped me greatly with my first search for a summer internship and was always there to give me useful advice.

Two other UIUC faculties I'm particularly grateful to are Sariel Har-Peled and Ruta Mehta. Sariel worked with me in my early graduate years, helping me to view everything in a randomized fashion. Ruta was always an encouragement, attending many of my talks and giving me useful feedback.

I would also like to thank Boris Aronov for hosting me in NYU Tandon and Ken-ichi Kawarabayashi for hosting me at the National Institute of Informatics. I spent a productive and memorable summer with each.

I would also like to thank the long list of people who have given advice or collaborated with me – Michael Bender, Kristóf Bérczi, Takuro Fukunaga, Tamás Király, Yusuke Kobayashi, Euiwoong Lee, Joseph Mitchell, Thapanapong Rukkanchanunt, Steven Skiena, Yutaro Yamaguchi, and Qian Zhang.

I am exceedingly grateful to the entire theory group. The group was a supportive community the year I arrived, and it still maintained a tight and caring culture when the group grew so large that a table in Mandarin Wok was insufficient. Especially, I would like to thank the theory students who had serious research discussions with me, even if no paper came out of it (yet) — Hsien-Chih Chang, Shalmoli Gupta, Konstantinos Koiliaris, Vivek Madan, Sahand Mozaffari, Kent Quanrud, Ben Raichel, and Xilin Yu.

Lastly, thank you to all my friends and family. Their unrelenting support has lifted my spirits even when faced with difficult times. My parents have always believed and been proud of me. Finally, I would like to thank my wife, Cheng, whose love and support have been indispensable toward my growth as a person.

Table of Contents

Chapter 1: Introduction

One of the classic problems in combinatorial optimization is the min-st-cut problem. In the min-st-cut problem, the input is a graph and the output is a minimum cardinality set of edges such that their removal disconnects the nodes s and t. A similar problem, the min-cut problem, asks for a set of edges such that their removal disconnects some pair of nodes. In particular, the min-cut is the minimum among min-st-cuts over all nodes s and t.

One of the themes of this thesis is to study how min-cuts behave in hypergraphs. Hypergraphs generalize graphs by allowing each edge to contain more than two nodes. Hypergraphs are useful for modeling systems in a variety of areas. For example, they have applications in clustering and VLSI design [1, 2]. Richer graph models present challenges in designing algorithms due to their additional complexity. The *size* of a hypergraph is the sum of its edge sizes, which can be much larger than the number of edges. This thesis features algorithms that match the state of the art graph algorithms in terms of the running time, and they are conceptually simple. We show that several structural properties and algorithmic results for min-cut in graphs also hold for hypergraphs.

We also show that the min-k-cut problem, a generalization of the min-cut problem, can be solved in randomized polynomial time for hypergraphs. In fact, the algorithmic ideas extend to solve the min-k-cut-problem in an even richer graph model, namely hedgegraphs of constant span. Ghaffari, Karger, and Panigrahi identified the notion of hedgegraphs as a convenient graph model for the following scenarios [3]: It is often the case in 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.

The other theme of the thesis is the phenomenon of complexity gap between global and fixed-terminal cut problems in graphs and digraphs. A typical fixed-terminal problem is the following: Consider a constant k and some property P that captures the idea of "disconnected". Given a graph (digraph) G, and a k-tuple of nodes T, we are interested in removing a minimum number of the edges F, such that T satisfies property P in $G - F$. The nodes in T are called terminals. In the global variant, we are given G. We want to find a minimum set of edges F such that there exists *some* k-tuple of terminals T that satisfies P in $G - F$. The solution to the global variant is the minimum over the solutions over the fixed-terminal problem. If the fixed-terminal problem can be solved in polynomial time, then the global problem can be solved in polynomial time by trying all possible k-tuple of terminals. Interestingly, there are examples where the global problem is strictly easier: the

global problem is in **P** and the fixed-terminal problem is **NP-Hard**, or the global problem allows a strictly better approximation ratio. This thesis makes progress in this area by demonstrating problems where their global and fixed-terminal variants exhibit a complexity gap.

1.1 NOTATIONS

The set of positive integers less than or equal to ℓ is denoted as $[\ell]$. The $\tilde{O}$ notation suppresses poly-logarithmic factors. A problem is *tractable* if every instance of it can be solved in polynomial time with respect to the instance size.

1.1.1 Set functions

Let $f : 2^V \to \mathbb{R}$ be a real-valued set function defined over a finite set V. The function f is *symmetric* if $f(A) = f(V \setminus A)$ for all $A \subseteq V$. The function f is *submodular* if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$. The function f is non-negative if $f(A) \geq 0$ for all $A \subseteq V$. For a set function f on V and a set family $\mathcal{F} \subseteq 2^V$, we define $f(\mathcal{F}) = \sum_{S \in \mathcal{F}} f(S)$. A *$k$-partition* of a set V is a family of k disjoint non-empty subsets of V, such that their union is V. A *k-subpartition* of V is a k-partition of some set $U \subseteq V$. For a given set function f over V, a *non-trivial minimizer* is a set S such that $f(S) = \min\{f(T) \mid \emptyset \subsetneq T \subsetneq V\}$. The *domain* of f, denoted $\mathrm{dom}(f)$, is V. We define $\lambda_f(x, y) = \min\{f(S) : S \subseteq V, x \in S, y \notin S\}$, and $\lambda(f) = \min_{x,y \in V} \lambda_f(x, y)$. Hence $\lambda(f)$ is the value of a non-trivial minimizer of f.

1.1.2 Graphs and cuts

We introduce some common notions. We refer the reader to a standard textbook, e.g. [4], for concepts related to graphs and digraphs. Let V be a finite set of nodes.

Graphs. A graph $G = (V, E)$ on node set V has a set of edges E, where an edge is a set of at most two nodes. A *self-loop* is a singleton edge. An edge e *crosses* X if $e \cap X$ and $e \cap (V \setminus X)$ are both non-empty. The size of a graph is the number of edges in the graph.

Digraphs. A digraph $D = (V, E)$ has a set of (directed) edges E, where an edge is an ordered pair of nodes. For an edge (u, v), we sometimes write uv. The *head* of an edge uv is u, and the *tail* is v. The size of a digraph is the number of edges in the digraph.

Hypergraphs. A hypergraph $H = (V, E)$ has a set of hyperedges E, where each hyperedge e is a subset of nodes. A hyperedge e *crosses* X if $e \cap X$ and $e \cap (V \setminus X)$ are both non-empty. The *size* of a hypergraph is $\sum_{e \in E} |e|$. The *rank* of a hypergraph, denoted by r, is $\max_{e \in E} |e|$. We note that graphs are hypergraphs of rank 2. A hypergraph $H' = (V', E')$ is a *subhypergraph* of $H = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Thus a subhypergraph of H is obtained by deleting nodes and hyperedges. For $U \subseteq V$, we denote the *induced subhypergraph* of H on U as $H[U] = (U, \{e \mid e \subseteq U, e \in E\})$.

Hedgegraphs. Consider a multigraph $G = (V, E')$. Let E be a partition of the edges in E'. We say $H = (V, E)$ is a hedgegraph, and G is the graph underlying the hedgegraph H. Each partition class in E is called a *hedge*. A hedge e *crosses* X if some edge in e crosses X. The *size* of a hedgegraph is the size of its underlying multigraph.

For a graph (hypergraph/hedgegraph) $H = (V, E)$, we define the function $\delta_H : 2^V \to E$ with $\delta_H(S)$ being the set of all edges (hyperedges/hedges) in H that cross S. We will drop H from the subscript if the graph is clear from the context. The number of crossing edges (hyperedges/hedges) is $d_H(S) = |\delta_H(S)|$. A graph (digraph/hypergraph/hedgegraph) is capacitated if there is a non-negative capacity function $c : E \to \mathbb{R}_+$ associated with it. If all capacities are 1 we call it uncapacitated; we allow multiple copies of an edge (hyperedges/hedges) in the uncapacitated case. We define $c_H : 2^V \to \mathbb{R}_+$ as the *cut function*, the total capacity of all crossing edges (hyperedges/hedges), namely $c_H(S) = \sum_{e \in \delta_H(S)} c(e)$. In the uncapacitated case, $c_H(S) = |\delta_H(S)| = d_H(S)$. A *isolated set* in a graph (hypergraph/hedgegraph) G is a non-empty set X of nodes such that $d_H(X) = 0$. A *component* is a inclusion-wise minimal isolated set. Components in graphs (hypergraph/hedgegraph) form a partition of the nodes.

For pairwise disjoint node subsets $A_1, \ldots, A_k$, we let $E_H(A_1, \ldots, A_k) = \{e \mid e \cap A_i \neq \emptyset, i \in [k]\}$ to be the set of edges (hyperedges/hedges) that crosses each of $A_1, \ldots, A_k$. The function $d_H(A_1, \ldots, A_k) = \sum_{e \in E_H(A_1, \ldots, A_k)} c(e)$ denotes the total capacity of the edges (hyperedges/hedges) in $E_H(A_1, \ldots, A_k)$.

In a directed graph D, $E_D(X, Y)$ is the set of edges with tail in X and head in Y. We use $\delta^{in}_D(X) := E_D(Y, X)$, $\delta^{out}_D(X) := \delta^{in}_D(V \setminus X)$, $d^{in}_D(X) := |\delta^{in}_D(X)|$ and $d^{out}_D(X) := |\delta^{out}(X)|$.

For a set family $\mathcal{F}$, we say an edge (hedge) e *crosses* $\mathcal{F}$ if it crosses any set in $\mathcal{F}$. For a subset S of nodes, we define $E[S]$ to be the set of edges in the induced subgraph $G[S]$.

1.1.3 Connectivity

We discuss some connectivity concepts in *undirected graphs*.

Edge-connectivity. In undirected graphs, the edge-connectivity between two nodes s and t is the maximum number of edge disjoint st-paths. It is denoted $\lambda_G(s,t)$. By Menger's theorem, denote $\lambda_G(s,t)$ is same as the min-st-cut $(S, V \setminus S)$. The edge-connectivity of G, denoted $\lambda(G)$, is the minimum st-edge-connectivity over all distinct $s,t \in V$. The edge-connectivity of G is equivalent to $\min_{\emptyset \subsetneq S \subsetneq V} c_G(S)$. This cut based definition immediately generalizes to hypergraphs. Under this definition, a cut S is a min-cut of H if $c(S) = \lambda(H)$. A graph (hypergraph) is *k-edge-connected* if $\lambda(H) \geq k$.

Node-connectivity. The node-connectivity between u and v, denoted by $\kappa_G(u,v)$ is the maximum number of internally node-disjoint paths between u and v. The node-connectivity of G, denoted $\kappa(G)$, is the minimum st-node-connectivity over all distinct $s,t \in V$.

Element-connectivity. Let $T \subseteq V$ be a set of terminals; nodes in $V \setminus T$ are referred to as non-terminals. For any two distinct terminals $u,v \in T$, the element-connectivity between u and v is the maximum number of uv-paths in G that are pairwise "element"-disjoint where elements consist of edges and non-terminals. We note that the element-disjoint paths need not be disjoint in terminals. We use $\kappa'_G(u,v)$ to denote the element-connectivity between u and v. The element-connectivity of G, denoted $\kappa'(G)$, is the minimum st-element-connectivity over all distinct $s,t \in V$.

1.1.4 k-cut and k-partition

The statements in this section hold for graphs, hypergraphs and hedgegraphs. We only give the description for graphs for simplicity. Let $\mathcal{P}$ be a k-partition and H a graph. The k-partition $\mathcal{P}$ is a *k-way-partition* for a set of k terminals T if each set in the family contains exactly one terminal. More formally, $\mathcal{P}$ is a k-way-partition for terminals T if $|P \cap T| = 1$ for all $P \in \mathcal{P}$. The *value* of $\mathcal{P}$ in H, denoted $c_H(\mathcal{P})$, is the total capacity of the edges crossing it. That is, $c_H(\mathcal{P}) := \sum_{e \in \delta_H(P), P \in \mathcal{P}} c(e)$. The problems k-WAY-PART and k-PART are the problems of finding a minimum value k-way-partition and a minimum value k-partition respectively, for a given graph.

A *cut* $(S, V-S)$ is a 2-partition of the nodes. We will abuse the notation and call a set S as a cut to refer to cut $(S, V-S)$ in graphs. An st-cut is a cut S such that $|S \cap \{s,t\}| = 1$. A *k-cut* is another name for a k-partition of the nodes. A *k-cut-set* (or cut-set if $k=2$) is

an inclusion-wise minimal set of edges such that its removal disconnects the graph into at least k components. For a set of k terminals T, a k-cut-set is a *k-way-cut-set* if there is no path between any pair of nodes in T. The Graph-k-Cut (Graph-k-Way-Cut) problem takes a graph (and k terminals) and asks for a minimum capacity k-cut-set (k-way-cut-set).

The k-way-partitions and k-way-cut-sets are closely related. The edges crossing a k-way-partition is a k-way-cut-set, and every k-way-cut-set arises this way. Consequently, k-way-Part (k-Part, respectively) and Graph-k-Way-Cut (Graph-k-Cut, respectively) are equivalent problems.

When we discuss a graph (digraph/hypergraph/hedgegraph), we always use n to denote number of nodes, m the number of edges (hyperedges/hedges) and p the size.

1.1.5 List of problems

Here we give a master index of properties that we consider. These properties correspond to the list of problems that we consider in this thesis (See Table 1.1).

Definition 1.1. *A pair of terminals s and t is*

- disconnected *in a hypergraph/hedgegraph if s and t are in different connected components,*
- disconnected *in a graph/digraph if there is no path from s to t and there is no path from t to s.*

For a set of k terminals T, it is disconnected if it is pairwise disconnected.

Definition 1.2. *In a digraph, a k-tuple of terminals $T = (t_1, \ldots, t_k)$ is* linearly-disconnected *if there is no path from t_i to t_j for all $i < j$.*

Definition 1.3. *In a graph, consider a k-tuple of terminals $T = (t_1, \ldots, t_k)$ and a k-tuple of positive integers $s = (s_1, \ldots, s_k)$. The terminals T is s-*size-disconnected *if there exist a k-partition $(S_1, \ldots, S_k)$, such that $t_i \in S_i$, $|S_i| \geq s_i$ and $|\delta(S_i)| = 0$ for all i.*

1.2 THESIS CONTRIBUTION AND ORGANIZATION

The thesis consists of four body chapters with each chapter being self-contained. The algorithmic results of chapters 2, 4 and 5 are summarized in Table 1.2.

1.2.1 Element connectivity

Chapter 2 is concerned with element-connectivity, a concept in between edge-connectivity and node-connectivity. The chapter is an adaptation of the paper [5]. For a graph $G = (V, E)$, let $T \subseteq V$ be a set of terminals that form an independent set. A well-known result, known as the reduction lemma, shows that every edge between two nodes in $V \setminus T$, can be either deleted or contracted while maintaining element-connectivity between all pairs of terminals. Hence applying it sequentially, we obtain a bipartite graph with terminals as one part of the bipartition. The resulting graph is called a reduced graph. Very basic questions concerning element connectivity have not been explored in contrast to the substantial literature on edge and node connectivity. For instance, how fast can element-connectivity be computed? How fast can the graph H promised by the reduction lemma be computed?

In particular, we obtain the following theorem.

Theorem 1.1. *Let G be a graph on n nodes and m edges and T a set of terminals. There is an $O(|T|nm)$ time algorithm that given G and T outputs the reduced graph of G.*

The above theorem improves the naive running time of $O(|T|^2nm^2)$.

The key observation that underlies the algorithms is that a symmetric submodular function can be defined over the terminal set T that corresponds to the element-connectivity between then. This in turn allows us to compute and exploit a Gomory-Hu tree for this function.

In another result we show that a result of Hassin and Levin [6] on flow-trees for separation with node capacities can be easily understood via element-connectivity and the existence of Gomory-Hu tree for it.

1.2.2 Hypergraph cuts

Chapter 3 is concerned with hypergraph cuts. We generalize structural and algorithmic properties of graph cuts to hypergraphs. The chapter is an adaptation of the papers [7, 8].

Finding a k-certificate in near-linear time Every connected graph contains a spanning tree that *certifies* the connectivity of the graph. A k-edge-connected graph has a similar subgraph with $O(kn)$ edges, known as a k-certificate, that certifies that the graph is k-edge-connected. A k-certificate is powerful in algorithm design because it is a sparse graph that demonstrates a lower bound on the connectivity of the given graph. Most importantly, finding a k-certificate takes only *linear time* [9]. Therefore, finding a k-certificate is a preprocessing step in various graph algorithms [10]. A k-certificate also exists for a k-edge-connected hypergraph [11]. To find a k-certificate of a hypergraph, one repeatedly

strips off 1-certificates. Unfortunately, it is too slow for large k. Moreover, the size of the resulting certificate could still be large, i.e. $\Omega(kn^2)$.

We show that a k-certificate for a hypergraph can also be found in *linear time*. Moreover, the total *size* of the k-certificate that we find is $O(kn)$.

Theorem 1.2. *Let H be a hypergraph on n nodes and m edges with size p. There is an $O(p)$ time algorithm that given H and k outputs a k-certificate H' of H such that the size of H' is $O(kn)$.*

In uncapacitated settings, the result gives us faster algorithms for finding min-cuts, approximate min-cuts and max flows. In capacitated settings, k-certificates are essential in finding cut sparsifiers.

Finding all min-cuts quickly The cactus representation of a graph is a graph with $O(n)$ edges that captures all min-cuts of a graph [12]. Finding all min-cuts is equivalent to computing the cactus representation. It took many years of continuous progress to reach the fastest time and smallest space algorithm [13–17]. Finding a cactus representation takes the same amount of time as finding a *single min-cut*, and the space complexity is *linear*. A hypercactus representation captures all min-cut information of a hypergraph [18,19]. Finding the hypercactus representation takes polynomial time, but it is much slower than finding a min-cut.

We show that finding a cactus representation is no harder than finding a single min-cut. Our algorithm is much simpler than the previous algorithms use the conceptually clean Cunningham's decomposition framework [20]. Finally, the algorithm takes *linear* space. The framework depends on finding *splits*, min-cuts that separate at least 2 nodes on each side. The approach alone is already prohibitive: finding a split is no easier than finding a min-cut. The main algorithmic insight is a near-linear time split oracle. The oracle either finds a split or gives us two nodes whose contraction does not destroy any min-cut. We obtain the following theorem.

Theorem 1.3. *Let H be a hypergraph on n nodes with size p. A compact $O(n)$ sized data structure that encodes all min-cuts can be found in $O(np + n^2 \log n)$ time and $O(p)$ space.*

As a consequence of the proof, we obtain a $\binom{n}{2}$ upper bound on the number of distinct min-cut-sets.

$(2+\epsilon)$ min-cut approximation for hypergraphs Matula showed that a $(2+\epsilon)$ approximation for the min-cut of uncapacitated graphs can be computed in deterministic

$O(m/\epsilon)$ time [21]. The algorithm generalizes to capacitated graphs and runs in $O(\frac{1}{\epsilon}(m \log n + n \log^2 n))$ time (as mentioned by Karger [22]). We show that Matula's algorithm extends to hypergraphs.

Theorem 1.4. *Let H be a hypergraph on n nodes with size p. There is an $O(\epsilon^{-1}(p + n \log n) \log n)$ time algorithm that given H and ϵ outputs a cut with value at most $(2 + \epsilon)$ times a min-cut.*

In fact, the algorithm generalizes to a special class of submodular functions which behaves similar to hypergraph cut functions.

Cut-sparsifiers A sparse subgraph that preserves all cut values to within a $(1 \pm \epsilon)$ factor is a cut-sparsifier. Benczúr and Karger, in their seminal work [23], showed a $(1 \pm \varepsilon)$-cut-sparsifier exists using a random sampling approach. Moreover, the sparsifier can be computed in near-linear time by a randomized algorithm. A cut sparsifier for a hypergraph also exists using the same sampling algorithm as that of Benczúr and Karger [11,24]? However, finding the probability distribution is the bottleneck. The sampling probability is inversely proportional to the strength of an edge. The strength measures the importance of an edge to the cuts that it crosses. We provide a near-linear time algorithm to approximate the strength in hypergraphs. As a consequence, we get a near-linear time algorithm for a hypergraph cut sparsifier.

Theorem 1.5. *Let H be a rank r hypergraph on n nodes with size p. There is an $O(p \log^2 n \log p)$ time algorithm that given H and ϵ outputs a $(1 \pm \epsilon)$-cut sparsifier of H of $O(nr(r + \log n)/\epsilon^2)$ edges with high probability.*

1.2.3 Hypergraph k-cut

In Chapter 4, we present a randomized polynomial time algorithm for hypergraph k-cut. It is an adaptation of the paper [25].

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 [26]. We focus on this problem for constant k. Goldschmidt and Hochbaum showed that a min-k-cut in graphs can be found in polynomial time [27]. Subsequent works improved the running time using techniques including divide and conquer, tree packing, and randomized contractions [28–31]. Finding a min k-cut of a

hypergraph has applications in network reliability and clustering in VLSI design [1, 2]. We present a randomized polynomial time algorithm to solve the hypergraph k-cut problem.

Theorem 1.6. *Let H be a hypergraph on n nodes with size p. There is a randomized $O(pn^{2k-1}\log n)$ time algorithm that given H outputs a min k-cut and succeeds with probability at least $1 - 1/n$.*

The algorithmic technique extends to solve the more general hedge k-cut problem when the subgraph induced by every hedge has a constant number of connected components. The algorithm is based on random contractions akin to Karger's min cut algorithm. The main technical contribution is a non-uniform 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. In addition, we present an alternative contraction based randomized polynomial time approximation scheme for hedge k-cut in arbitrary span hedgegraphs. The algorithm and analysis also lead to bounds on the number of optimal solutions to the respective problems.

1.2.4 Global vs. Fixed-terminal cut

Chapter 5, we consider global and fixed-terminal cut problems in both graphs and digraphs. The section is based on [32, 33].

***st*-Sep-k-Cut** Given s and t, we are interested in deleting minimum number of edges so that there are at least k components with s and t being in different components. The problem is denoted as *st*-Sep-k-Cut. The complexity of *st*-Sep-k-Cut was raised as an open problem by Queyranne [34]. We show that the problem can be solved in polynomial time.

Theorem 1.7. *There is a polynomial-time algorithm to solve st-Sep-k-Cut.*

***s*-Size-k-Cut** Consider a fixed tuple $s = (s_1, \ldots, s_k) \in \mathbb{N}^k$. In the s-Size-k-Cut problem, we want to remove edges so the remaining graph can be partitioned such that the ith set contains at least s_i nodes. When $k = 2$, consider ordering the cuts by value. A *randomized* algorithm is known for arbitrary k with running time $\tilde{O}(n^{2\sigma})$, where σ is the sum of the elements in s [35]. We improve the running time and also give a deterministic algorithm.

Theorem 1.8. *Let $s = (s_1, \ldots, s_k)$ and $s_i \geq s_{i+1}$ for all $i \leq k-1$. s-Size-k-Cut for graph G can be solved in $O(n^{2(\sigma - s_1 + 2)})$ time, where $\sigma = \sum_{i=1}^{k} s_i$ and n is the number of nodes of G.*

The algorithm uses Thorup's tree packing algorithm to enumerate possible candidates for the optimal solution. The fixed-terminal variant is **NP-Hard** for $k \geq 3$ since Graph-k-Cut reduces to s-Size-k-Way-Cut.

$(s, *, t)$-Linear-3-Cut We discuss Linear-k-Cut, another generalization of Graph-k-Cut to directed graphs. The fixed-terminal variant was introduced by Erbacher et al. in [36] and it is **NP-Hard**. We study the case when $k = 3$ and we fix 2 of the nodes. Formally, given a digraph and two nodes s and t, we want to remove a minimum number of edges, such that there exists some node $r \notin \{s, t\}$, such that there is no path from s to t, r to t and r to s. This is the $(s, *, t)$-Linear-3-Cut problem. We devise an approximation algorithm for $(s, *, t)$-Linear-3-Cut.

Theorem 1.9. *There is a polynomial-time* 3/2*-approximation algorithm for* $(s, *, t)$-Linear--3-Cut.

The 3/2 approximation is crucial for an approximation algorithm for BiCut, which we introduce next.

BiCut The bicut problem is another natural generalization of min-cut to digraphs. In the fixed-terminal bicut problem, the input is a digraph with two specified nodes and the goal is to find a smallest subset of edges whose removal ensures that the two specified nodes cannot reach each other. In the global bicut problem, the input is a digraph and the goal is to find a smallest subset of edges whose removal ensures that *there exist* two nodes that cannot reach each other. Fixed-terminal bicut is **NP-Hard**, admits a simple 2-approximation, and does not admit a $(2 - \epsilon)$-approximation for any constant $\epsilon > 0$ assuming the unique games conjecture [37]. We improve on the approximability of the global variant in comparison to the fixed-terminal variant.

Theorem 1.10. *There is a polynomial-time* $(2 - 1/448)$*-approximation algorithm for* BiCut.

Global	Fixed-Terminal	Structure	Removal	Property
Graph-k-Cut	Graph-k-Way-Cut	graph	edges	disconnected
Hypergraph-k-Cut	Hypergraph-k-Way-Cut	hypergraph	edges	disconnected
Hedge-k-Cut	Hedge-k-Way-Cut	hedgegraph	hedge	disconnected
BiCut	st-BiCut	digraph	edges	disconnected
Linear-k-Cut	Linear-k-Way-Cut	digraph	edges	linear-disconnected
s-Size-k-Cut	s-Size-k-Way-Cut	graph	edges	s-size-disconnected
st-Sep-k-Cut		graph	edges	$\{s,t\} \subseteq T$, disconnected

Table 1.1: List of problems and property.

Problem	Complexity	Reference
Graph-k-Cut	**P**	[27]
Graph-k-Way-Cut	**NP-Hard** iff $k \geq 3$	[38]
Hypergraph-k-Cut	**RP**	Theorem 4.1
Hypergraph-k-Way-Cut	**NP-Hard** iff $k \geq 3$	Graph-k-Way-Cut
Hedge-k-Cut	**RP** if constant span, RPTAS if arbitrary span	Theorem 4.1
Hedge-k-Way-Cut	**NP-Hard** for all k when span ≥ 2 and rank ≥ 4	[39, 40]
BiCut	unknown if **NP-Hard**, $(2 - 1/448)$-approximable	Theorem 5.6
st-BiCut	**NP-Hard**, $(2 - \epsilon)$-inapproximable	[37]
Linear-k-Cut	**P** if $k = 2$, otherwise unknown	
Linear-k-Way-Cut	**P** iff $k = 2$	[41]
$(s, *, t)$-Linear-3-Cut	$\frac{3}{2}$-approximation, no hardness known	Theorem 5.5
st-Sep-k-Cut	**P**	Theorem 5.1
s-Size-k-Cut	**P**	Theorem 5.4
s-Size-k-Way-Cut	**NP-Hard** iff $k \geq 3$	Graph-k-Way-Cut

Table 1.2: Complexity of the global and fixed-terminal problems. The inapproximability results are under the assumption of the unique games conjecture.

Chapter 2: Element connectivity

Let $G = (V, E)$ be an undirected simple graph. Let $T \subseteq V$ to be a set of terminals that forms a independent set. The edges E and nodes in $V \setminus T$ are called elements. Recall the element connectivity between $s \in T$ and $t \in T$ is the maximum number of element disjoint paths between s and t. Via Menger's theorem one can characterize element connectivity in an equivalent way via cuts. That is, the element connectivity of s and t equals the minimum number of elements such that their removal disconnect s and t. See Figure 2.1 for example.

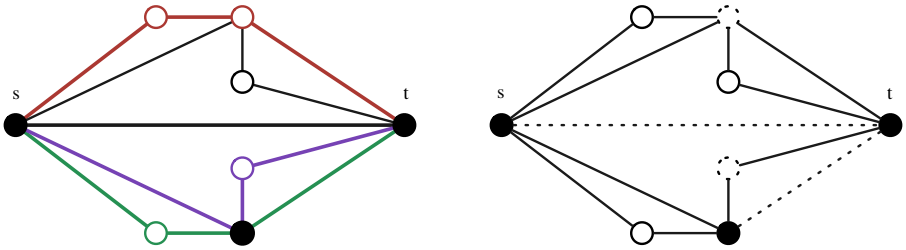


Figure 2.1: The black nodes are the terminals. The left image shows 4 element-disjoint st-paths. The right image shows removing 4 elements disconnects s and t. $\kappa'(s,t) = 4$

Element-connectivity can be seen to generalize edge-connectivity by letting $T = V$. At the same time, element-connectivity is also closely related to node-connectivity. If T is an independent set then $\kappa'_G(u,v)$ is the maximum number of paths from u to v that are disjoint in non-terminals. In particular, if T contains exactly two nodes s and t, then $\kappa_G(s,t) = \kappa'_G(s,t)$. Element-connectivity has found several applications in network design, routing and related problems, some of which we will discuss later. Several of these applications rely on an interesting graph reduction operation shown first by Hind and Oellermann [42]. To describe their result we use the notation G/pq to denote the graph obtained from G by contracting the edge pq, and $G - pq$ to denote the graph with edge pq deleted.

Theorem 2.1 (Hind & Oellermann [42]). *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a terminal-set such that $\kappa'_G(T) \geq k$. Let pq be* any *edge where $p, q \in V \setminus T$. Then $\kappa'_{G_1}(T) \geq k$ or $\kappa'_{G_2}(T) \geq k$ where $G_1 = G - pq$ and $G_2 = G/pq$.*

Chekuri and Korula generalized the theorem to show that the same reduction operation also preserves the *local* element-connectivity of every pair of terminals.

Theorem 2.2 (Chekuri & Korula [43]). *Let $G = (V, E)$ be an undirected graph and $T \subseteq V$ be a terminal-set. Let pq be* any *edge where $p, q \in V \setminus T$ and let $G_1 = G - pq$ and $G_2 = G/pq$. Then one of the following holds: (i) $\forall u, v \in T$, $\kappa'_{G_1}(u,v) = \kappa'_G(u,v)$ (ii) $\forall u, v \in T$, $\kappa'_{G_2}(u,v) = \kappa'_G(u,v)$.*

	Min Cut	Min Cut(WHP)	All-pair	All-pair (WHP)	Reduce
$\boldsymbol{\lambda}$	$\tilde{O}(m)$ [44,45]	$\tilde{O}(m)$ [46]	$\tilde{O}(n^{27/8})$ [47]	$\tilde{O}(nm)$ [48]	-
$\boldsymbol{\kappa'}$	$O(\|T\|\ \mathbf{MF}(n,m))$	same as all-pair	$O(\|T\|\ \mathbf{MF}(n,m))$	$\tilde{O}(\|T\|n^{\omega})$ [49],$O(m^{\omega})$ [50]	$O(\|T\|nm)$
$\boldsymbol{\kappa}$	$O(n^{7/4}m)$ [51]	$\tilde{O}(nm)$ [52]	$O(n^{9/2})$ [53]	$\tilde{O}(n^{2+\omega})$ [49]	-

Figure 2.2: The running time for various algorithms for a graph with n nodes and m edges and terminal nodes $|T|$. The row for κ' is our result. $\mathbf{MF}(n,m)$ is the running time for a maximum flow on unit capacity directed graph with n nodes and m edges, which is known to be $O(\sqrt{n}m)$ [53]. WHP indicates with high probability bounds for randomized algorithms. ω is the matrix multiplication constant.

We refer to the preceding theorem as the reduction lemma following the usage from [43]. By repeatedly applying the reduction lemma, as observed in prior work, we obtain the following corollary.

Corollary 2.1. *Given a graph $G = (V, E)$ and a terminal set $T \subseteq V$ there is a minor $H = (V', E')$ of G such that (i) $T \subseteq V'$ and (ii) $V' \setminus T$ is an independent set in H and (iii) $\kappa'_H(u,v) = \kappa'_G(u,v)$ for all $u, v \in T$. In particular, if T is an independent set in G then H is a bipartite graph with bipartition $(T, V' \setminus T)$.*

The minor H in the previous corollary is called a *reduced graph* of G. A graph is *reduced* if there are no edges between non-terminals.

Remark A reduced graph $G = (V, E)$ where the terminals T form an independent set can be interpreted as a hypergraph $H = (T, E')$. H contains an edge e_v for every non-terminal v in G, where e_v is the set of neighbors of v in G. Element-connectivity of terminals T in G is equivalent to hypergraph edge-connectivity in H.

We obtain algorithmic results summarized in the second row of Figure 2.2. In another result we show that a result of Hassin and Levin [6] on flow-trees for separation with node capacities can be easily understood via element-connectivity and the existence of Gomory-Hu tree for it.

Applications: Element-connectivity has found important applications in three areas: network design, packing element-disjoint Steiner trees and forests, and more recently in routing for node-disjoint paths and related applications. Our algorithmic improvements most directly affect the second application, namely the problem of packing element-disjoint Steiner trees and forests. We briefly describe the simpler case of packing element-disjoint Steiner trees which was the original motivation for the graph reduction step [42]. Here we are given

a graph $G = (V, E)$ and terminal set T and the goal is to find the maximum number of Steiner trees for T that are pairwise element-disjoint. It is known that in general graphs one can find $\Omega(k/\log |T|)$ trees where $k = \kappa'_G(T)$ and there are examples where this bound is tight [54]. In planar graphs one can find $\Omega(k)$ trees [43, 55]. Algorithms for these first need to compute k, and then work with the reduced graph. Computing the reduced graph is the bottleneck and our result thus implies an $O(|T|nm)$-time algorithm for these packing problems; the previous bounds is $O(k|T|^2m^2)$.

There is a vast amount of literature on algorithms for computing edge and node connectivity in graphs, and related problems on flows and cuts. As the table in Fig 2.2 shows, the edge connectivity versions have faster algorithms and are much better understood. This is not surprising since edge-connectivity has additional structure that can be exploited, including the existence of a Gomory-Hu tree. In contrast, node-connectivity does not admit even a weaker notion of flow-trees [56]. This chapter can be seen as a first step in exploiting the basic properties of element-connectivity to obtain faster algorithms. In this context we mention the the splitting-off operation to preserve edge-connectivity introduced by Lovász [57] and strengthened by Mader [58]. The algorithmic aspects of splitting-off have been exploited in several papers on edge-connectivity including some recent ones [48, 59] and we hope similar ideas may bear fruit for element-connectivity.

2.1 PRELIMINARIES

A capacitated spanning tree (R, c) on V is called a *Gomory-Hu tree* (*cut tree*) of f if for all $st \in E(R)$, $f(A) = \lambda_f(s, t) = c(st)$, where A is a component of $R - st$. $\lambda_f(s, t)$ for all $s, t \in V$ can be read off from the Gomory-Hu tree as the smallest capacity on the unique path between s and t. A Gomory-Hu tree always exists when f is non-negative, symmetric, and submodular (see [60]).

Equivalent digraph: One view of element-connectivity that greatly helps with computation is to define a flow problem. One can see that $\kappa'_G(s, t)$ is the maximum s-t-flow in G with unit capacities on the edges *and* non-terminal nodes (terminal nodes have no capacity constraint). This prompts us to define a equivalent directed graph, which we get from applying the standard node split operation for a graph when there are node capacities.

Let $N = V \setminus T$ be the set of non-terminals. Let $N^- = \{v^- | v \in N\}$ and $N^+ = \{v^+ | v \in N\}$. The *equivalent digraph* of G, denoted by $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{V} = N^- \cup N^+ \cup T$ and the arc set $\tilde{E}$ is obtained from G as follows:

1. For every $v \in N$, $(v^-, v^+) \in \tilde{E}$.

2. For every $uv \in E$ where $u, v \in N$, $(u^+, v^-), (v^+, u^-) \in \tilde{E}$.

3. For every $uv \in E$ where $u \in T, v \in N$, $(u, v^-), (v^+, u) \in \tilde{E}$.

4. For every $uv \in E$ where $u, v \in T$, $(u, v), (v, u) \in \tilde{E}$.

All the arcs in $\tilde{G}$ implicitly have unit capacity. Any maximum integral acyclic s-t-flow f_{st} in $\tilde{G}$ corresponds to a set of maximum element-disjoint s-t-paths in G. Hence we do not distinguish between maximum flows in $\tilde{G}$ and maximum element-disjoint paths in G. A flow in $\tilde{G}$ contains a node v (edge e) in G to mean the corresponding element-disjoint path contains node v (edge e). It is easy to see the following lemma holds.

Lemma 2.1. *$\lambda_{\tilde{G}}(s,t) = \kappa'_G(s,t)$ for all $s, t \in T$.*

2.2 ELEMENT-CONNECTIVITY AND CONNECTIONS TO SUBMODULARITY

It is natural to work with cut-functions that capture element-connectivity. We define a cut function $c_G : 2^T \to \mathbb{R}_+$ over the terminals as follows: for $U \subseteq T$, $c_G(U)$ is the minimum number of elements whose removal disconnects U from $T \setminus U$ in G. To formally define $c_G(U)$ we consider node tri-partitions (A, Z, B) where $U \subseteq A$ and $(T \setminus U) \subseteq B$; among all such tri-partitions the one with minimum $|Z| + |E(A,B)|$ defines $c_G(U)$. $Z \cup E(A,B)$ is called a *cut-set*.

Theorem 2.3 (Menger's theorem for element-connectivity)**.** *For a graph G with terminal nodes T, for all $s, t \in T$, $\kappa'_G(s,t) = \min\{c_G(U) : U \subseteq T, |U \cap \{s,t\}| = 1\}$.*

Proof.

$$\min_{U \subseteq T, |U \cap \{s,t\}|=1} c_G(U) = \min_{U \subseteq T, |U \cap \{s,t\}|=1} \left(\min_{(X,Z,Y), U \subseteq X, T \setminus U \subseteq Y, Z \subseteq V \setminus T} |Z| + |E(X,Y)| \right) \tag{2.1}$$

$$= \min_{(X,Z,Y), |\{s,t\} \cap X|=1, Z \subseteq V \setminus T} |Z| + |E(X,Y)| \tag{2.2}$$

$$= \kappa'_G(s,t) \tag{2.3}$$

All (X, Z, Y) are tri-partitions of V. The last line comes from [61]. □

If T is a independent set in the previous theorem, then the cut-set can be taken to contain no edges.

For our purposes a crucial observation is that c_G is a non-negative symmetric submodular function. First, we state a lemma.

Lemma 2.2. *Let $f : 2^V \to \mathbb{R}$ be a submodular function and $P : 2^T \to 2^{2^V}$ is a function with the property that if $X \in P(A)$ and $Y \in P(B)$, then $X \cup Y \in P(A \cup B)$ and $X \cap Y \in P(A \cap B)$ then $f_P : 2^T \to \mathbb{R}$ defined as*

$$f_P(X) = \min_{Y \in P(X)} f(Y) \tag{2.4}$$

is submodular.

Proof. Let $X^* = \arg\min_{Y \in P(X)} f(Y)$, note since $X^* \in P(X)$ and $Y^* \in P(Y)$, we have $X^* \cup Y^* \in P(X \cup Y)$ and $X^* \cap Y^* \in P(X \cap Y)$.

$$\begin{aligned} f_P(X) + f_P(Y) &= f(X^*) + f(Y^*) && (2.5)\\ &\geq f(X^* \cup Y^*) + f(X^* \cap Y^*) && (2.6)\\ &\geq f((X \cup Y)^*) + f((X \cap Y)^*) && (2.7)\\ &= f_P(X \cup Y) + f_P(X \cap Y) && (2.8) \end{aligned}$$

□

Theorem 2.4. *Let G be a graph with terminal nodes T. c_G is a non-negative symmetric submodular function over T.*

Proof. Clearly c_G is symmetric and non-negative. For all $X \subseteq T$, we have $c_G(X) = \min\{|\delta_{\bar{G}}(Y)||Y \in P(X)\}$, where $P(X) = \{X \subseteq Y, T \setminus X \subseteq V \setminus Y\}$ and $|\delta_{\bar{G}}()|$ is a submodular function. Lemma 2.2 shows c_G is submodular. □

The preceding theorem implies that c_G admits a Gomory-Hu tree. Since every symmetric submodular function have one [60].

2.3 ALGORITHMIC ASPECTS OF ELEMENT-CONNECTIVITY

In this section we describe our algorithmic contributions to element-connectivity. In particular we describe how the running times in the second row of the table in Fig. 2.2 can be realized. Our main contribution is a faster algorithm for graph reduction. In the entire section, we are always working with a graph $G = (V, E)$ with n nodes, m edges and terminal nodes T.

2.3.1 Computing element-connectivity

Single pair element-connectivity The equivalent directed graph allows us to compute local element-connectivity by running a single maximum flow on a unit capacity directed

graph.

Lemma 2.3. *$\kappa'_G(s,t)$ can be computed in $O(\mathbf{MF}(n,m))$ time.*

Note that if $T = \{s,t\}$ then $\kappa'_G(s,t) = \kappa_G(s,t)$ and moreover maximum bipartite matching can be reduced to $\kappa_G(s,t)$. Thus, improving the time to compute $\kappa'_G(s,t)$ is not feasible without corresponding improvements to other classical problems.

All-pair element connectivity To compute $\kappa'_G(s,t)$ for all pairs $s,t \in T$, we can compute the Gomory-Hu tree that we know exists from Theorem 2.4. Unlike the single-pair case where element-connectivity behaves like node-connectivity, in the all-pair case it is closer to edge-connectivity, and in particular there are at most $|T|-1$ distinct element-connectivity values. A Gomory-Hu tree (R,c) representing the all-pair element-connectivities can be built recursively by solving $|T|-1$ minimum element cut computations, which correspond to maximum flow computations in the equivalent directed graph. Hence all-pair element-connectivity takes $|T|-1$ maximum flows. For dense graphs, if we allow randomization, maximum flow can be solved in $\tilde{O}(n^\omega)$ time [49], where ω is the matrix multiplication constant.

There is an alternative approach for sparse graphs using network coding. Cheung *et al.* describe a randomized algorithm that computes the edge-connectivity in a *directed* graph between *every* pair of nodes in $O(m^\omega)$ time with high probability [50]. Since $\kappa'_G(s,t) = \lambda_{\tilde{G}}(s,t)$ for all $s,t \in T$, all-pair element-connectivity can also be computed in $O(m^\omega)$ time with high probability.

Global element connectivity The global element-connectivity $\kappa'_G(T)$ can be easily obtained from the all-pair problem.

Theorem 2.5. *$\kappa'_G(T)$ can be computed in $O(|T|\ \mathbf{MF}(n,m))$ time.*

It is a very interesting open problem to improve the running time for computing $\kappa'_G(T)$. Global edge-connectivity admits a near-linear time algorithm [22, 44], but global node connectivity algorithms are much slower.

2.3.2 Computing a reduced graph

This section highlights the main result of the chapter, an $O(|T|nm)$ time algorithm to find a reduced graph. For G a graph with terminals T, H is called a *reduction* of G if it can be obtained from G by a sequence of reduction operations.

The reduction lemma suggests a simple algorithm: pick any edge incident to two nonterminal nodes, check which one of the two operations preserves element-connectivity, reduce and repeat. For a graph on n nodes and m edges, compute all-pair element-connectivity from scratch. The naive scheme when combined with the non-obvious $O(|T|\ \mathbf{MF}(n,m))$ algorithm for all-pair case would take $O(|T|m\ \mathbf{MF}(n,m))$ time. Cheriyan and Salavatipour [54] described exactly this algorithm where they in fact computed all-pair connectivity using $|T|^2$ max-flow computations; their focus was not in improving the running time. We obtain the following

Theorem 2.6. *For a graph G with n nodes, m edges and terminals T, a reduced graph of G can be computed in $O(|T|nm)$ time.*

The two high-level ideas to obtain an improved run-time are the following.

1. The algorithm initially computes and then maintains a Gomory-Hu tree (R, c) for the element-connectivity of T. For each edge $st \in E(R)$ it maintains a corresponding maximum flow between s and t in $\tilde{G}$ as it evolves with reduction operations.

2. Instead of considering reduction operations on an edge by edge basis in an ad-hoc manner we consider all edges incident to a non-terminal node v and process them as a batch.

The first idea alone would give us a run-time of $O(|T|m^2)$. The second idea gives a further improvement to $O(|T|nm)$.

Reduction by node elimination: We call an edge pq between two non-terminals a *reducible* edge. For a given non-terminal v let $D(v)$ be the set of all reducible edges incident to v. We say v is *active* if $D(v) \neq \emptyset$. An *elimination* operation on an active node v either contracts an edge in $D(v)$ or removes *all* edges in $D(v)$. If the graph is not reduced, there is always an elimination operation that preserves element-connectivity. Indeed, if $D(v)$ cannot be removed, then consider the edges in $D(v)$ in an arbitrary but fixed sequence and apply the reduction operation. At some point there is an edge e such that removing it reduces the element-connectivity. By reduction lemma, we can contract e. An elimination reduces the number of active nodes by at least 1. There can only be $O(n)$ eliminations. Moreover, crucially, we can implement a node elimination operation in the same amount of time as an edge reduction operation.

Our goal is to decide quickly whether an active node v can be eliminated (that is, $D(v)$ can be removed) and if not which of the edges in $D(v)$ can be contracted. For this purpose we define a capacitying of edges of E as follows. First we order the edges in $D(v)$ arbitrarily

as $e_1, e_2, \ldots, e_h$ where $h = |D(v)|$. We define a capacity function $w : E \to \{1, 2, \ldots, h+1\}$ where $w(e_i) = i$ and $w(e) = h + 1$ for all $e \in E \setminus D(v)$.

Given a set of capacities ρ on the edges, for each pair (s, t) of terminals we define $\beta_\rho(s, t)$ as the maximum capacity a such that the element-connectivity between s and t remains the same even if we remove all edges with capacity less than a. We call $\beta_\rho(s, t)$ as the *bottleneck capacity* for (s, t). Suppose we used the capacity function as defined above based on the numbering for $D(v)$ then v can be eliminated iff $\beta_w(s, t) > h$ for all pairs of terminals (s, t). In fact we can also obtain information on which of the edges in $D(v)$ can be contracted if v cannot be eliminated. Even further, we need to check only the terminal pairs that correspond to edges of a Gomory-Hu tree for the element-connectivity of T. This is captured in the following theorem which forms the crux of our algorithm.

Theorem 2.7. *Let (R, c) be a Gomory-Hu tree for the element-connectivity of terminal set T in G. Consider an active non-terminal v and the capacity function w and let $\ell = \max_{st \in E(R)} \beta_w(s, t)$. Define G' as G/e_ℓ if $\ell < |D(v)| + 1$ and $G - D(v)$ otherwise. Then $\kappa'_{G'}(u, v) = \kappa'_G(u, v)$ for all terminal pairs (u, v).*

Proof. Recall that $D(v) = \{e_1, e_2, \ldots, e_h\}$ where $h = |D(v)|$. Let $S = \{e_1, \ldots, e_{\ell-1}\}$. Since $\beta_w(s, t) \geq \ell$ for all $st \in E(R)$ it follows that the element-connectivity does not change for any pair $st \in E(R)$ if we delete the edges in S from G. From Lemma 2.6 it follows in fact that the element-connectivity of all pairs remains the same in $G - S$ as in G. Thus all edges in S are deletable. If $\ell = h + 1$ then $S = D(v)$ and $G' = G - S$ and we have the desired property. If $\ell \leq h$ then there is at least one $st \in E(R)$ such that $\kappa'_{G-S}(s, t) = \kappa'_G(s, t)$ but $\kappa'_{G-S-e_\ell}(s, t) < \kappa'_G(s, t)$. From the reduction lemma applied to $G - S$ we see that e_ℓ is not deletable, and hence by the reduction lemma $(G - S)/e_\ell$ preserves all the element-connectivities of the terminals. This also implies that G/e_ℓ preserves all element-connectivities. □

Computing $\beta_w(s, t)$ is relatively fast if we already have an existing maximum flow from s to t. This is captured by the lemma below.

Lemma 2.4. *Given a maximum s-t-flow f_{st} in $\tilde{G}$, an active non-terminal v and a capacitying w, we can find $\beta_w(s, t)$ and a corresponding flow in $O(m)$ time.*

Proof. Consider a maximum s-t flow f_{st} in $\tilde{G}$. In a flow decomposition of f_{st} there is at most one flow path that uses the non-terminal v. We can find such a flow path in $O(m)$ time and reduce the flow by at most one unit to obtain a new flow f'_{st} which does not have any flow through v. Not that f'_{st} is non-zero only on edges e with $w(e) = |D(v)| + 1$. If the value of f'_{st} is the same as that of f_{st} then $\beta_w(s, t) = |D(v)| + 1$ and we are done. Otherwise, we

claim that $\beta_w(s,t) = \ell$ iff the maximum bottleneck capacity for a path from s to t in the residual graph of f'_{st} in $\tilde{G}$ is ℓ. Assuming that the claim is true we can find $\beta_w(s,t)$ by a maximum bottleneck path computation in the residual graph in $O(m)$ time since the edges are sorted by capacity (the algorithm is quite simple but we refer the reader to [62]).

Now we prove the claim. If there is a path of maximum bottleneck capacity ℓ in the residual graph we can augment f'_{st} by one unit to obtain a maximum flow f''_{st} that uses only edges with capacity ℓ or greater and hence $\beta_w(s,t) \geq \ell$. Suppose $\beta_w(s,t) = \ell'$. Remove the edges $\{e_1, \ldots, e_{\ell'-1}\}$ from G and their corresponding arcs from $\tilde{G}$. There is a maximum s-t flow of value $\kappa'_G(s,t)$ in this new graph H. f'_{st} is a flow of value $\kappa'_G(s,t) - 1$ in H and hence there must be an augmenting path in the residual graph of f'_{st} in H and this augmenting path has bottleneck capacity at least ℓ' and is also a valid path in residual graph of f'_{st} in $\tilde{G}$. Thus $\ell \geq \ell'$. Thus $\beta_w(s,t) = \ell$ as desired. □

Theorem 2.7 and Lemma 2.4 lead to an algorithm as follows. We initially compute and then maintain a Gomory-Hu tree (R,c) for C_G on the terminals. For each edge $st \in E(R)$ we also maintain a maximum flow f_{st} in the current graph $\tilde{G}$. In each iteration we do an elimination procedure on an active node using Theorem 2.7. Each iteration either reduces the number of active nodes by one or contracts an edge. Thus the number of iterations is $O(n)$. The algorithm in Figure 2.3 gives a formal description. Note that the tree (R,c) does not change throughout the algorithm but the flows f_{st} for each $st \in E(R)$ get updated in each iteration. We need to clarify how we update these flows, analyze the overall running time, and argue about the correctness of the algorithm.

Updating the flows: Each elimination changes the graph. The algorithm updates the maximum flows f_{st} for each $st \in E(R)$ to reflect this change. If the new graph is $G - D(v)$, then no flow need updating since the computation of $\beta_w(s,t)$ already finds a new flow that avoids all edges in $D(v)$. We address the case when we contract an edge from $D(v)$.

Lemma 2.5. *Let G be a graph with m edges with terminals T. Let $H = G/e$ for some reducible edge e. If f_{st} is a maximum s-t flow in $\tilde{G}$, then we can find f'_{st}, a maximum s-t-flow in $\tilde{H}$ in $O(m)$ time.*

Proof. Let $e = pq$. We delete flow paths in f_{st} that use p or q. This removes at most 2 unit of flow (since each non-terminal has unit capacity) and the reduced flow is a valid flow in $\tilde{H}$. In two augmentations we can find a maximum flow in $\tilde{H}$. Each step can be easily implemented in $O(m)$ time. □

Analysis of running time: The time spent to build the Gomory-Hu tree (R,c) and the

```
Input: undirected graph G, terminals T
(R, c) ← Gomory-Hu tree of C_G
for st ∈ E(R)
   f_st ← maximum st-flow in G̃
while there exists an active non-terminal node v:
   w ← assign capacities to D(v)
   for st ∈ E(R):
         compute β_w(s, t) in O(m) time using f_st
   ℓ ← min{β_w(s, t) | st ∈ E(R)}
   if ℓ > |D(v)|:
         G ← G − D(v)
   else:
         G ← G/e_ℓ
   for st ∈ E(R)
         update f_st
return G
```

Figure 2.3: Reduce a graph G with terminal nodes T

initial maximum flows for each edge $st \in E(R)$ take $|T| - 1$ maximum flow computations. Thus the time for this is $O(|T|\ \mathbf{MF}(n, m))$

As we argued there are $O(n)$ iterations of the while loop. In each iteration we need to compute $\beta_w(st)$ for each $st \in E(R)$. Lemma 2.4 shows that each such computation can be done in $O(m)$ time using the stored maximum flow; thus the total time is $O(|T|m)$. Updating the maximum flows also takes $O(|T|m)$ time using Lemma 2.5. Thus the overall running time for all the iterations of the while loop is $O(|T|nm)$ which dominates the time to compute the initial Gomory-Hu tree.

Correctness: Theorem 2.7 shows the correctness of the elimination procedure. It remains to argue that the Gomory-Hu tree (R, c) computed in the preprocessing step remains valid throughout the algorithm as the graph G changes and gets simplified. A similar idea is used implicitly by Gabow for preserving local edge-connectivity while applying split-off operations [63]. The following simple lemma gives a justification.

Lemma 2.6. *Let (R, c) be a Gomory-Hu tree for c_G with terminal nodes T in G. Let H be a reduction of G. If $\kappa'_G(s, t) = \kappa'_H(s, t)$ for all $st \in E(R)$ then (R, c) is also a Gomory-Hu tree for c_H.*

Proof. Since removing and contracting edges can't increase cut size, we have $c_G \geq c_H$. Fix any $st \in E(R)$. Let A be a component of $R - st$.

$$c_H(A) \geq \lambda_{c_H}(s, t) = \lambda_{c_G}(s, t) = c(st) = c_G(A) \geq c_H(A). \tag{2.9}$$

Hence this shows (R, c) is a Gomory-Hu tree for c_G, then it is also an Gomory-Hu tree for c_H. □

2.4 FLOW TREE FOR SEPARATION

A capacitated spanning tree (T, w) on V is a *flow tree* for a symmetric function $f : V^2 \to \mathbb{R}_+$ if for every u, v, $f(u, v) = w(e)$, where e is the minimum capacity edge on the unique path between u and v in T. For a function $f : V^2 \to \mathbb{R}$, a function $A_f : 2^V \to \mathbb{R}$ is called a f-cut if

$$f(s, t) = \min_{U \subseteq V, |U \cap \{s,t\}|=1} A_f(U) \tag{2.10}$$

for all $s \neq t \in V$. If A_f has a Gomory-Hu tree, then the same tree is a flow tree for f. Note that a function f may have a flow tree but not a Gomory-Hu tree.

Flow tree is a tree that is similar to the Gomory-Hu Tree in the sense that it gives us the minimum cut value between all pair of nodes. However, the flow tree does not necessarily give us the structure of cut. Although its definition is somewhat weaker than the Gomory-Hu trees, it does not require strict structure of the cut function.

Hassin and Levin [6] considered a variants of node cut, the separation, and present an explicit ancestor tree construction algorithm to build a flow tree for this variant. An older paper by Granot and Hassin also provided an explicit construction for the node capacitated case [64].

A set of nodes S is a *node cut* of u and v if removing S increase the number of components. Given a positive capacity function $c : V \to \mathbb{N}$. A set of node is a *separation* between two nodes u and v is either $\{u\}$, $\{v\}$ or a node cut of u and v. The capacity of a st-separation S is $\sum_{v \in S} c(v)$. The minimum st-separation is the st-separation of minimum capacity. The capacity of such separation is denoted as $\overline{\kappa}(s, t)$.

A common characterization of the separation of a graph $G = (V, E)$ with capacity function c is to produce an directed equivalent graph the edges. We give a variant of the standard formulation of the directed equivalent graph [64].

Let $V' = V \cup \{v^+ | v \in V\} \cup \{v^- | v \in V\}$, $E = \{v^- v | v \in V\} \cup \{v v^+ | v \in V\} \cup \{v^+ u^- | v, u \in V\}$. The capacities are $c'(v v^+) = c'(v^- v) = c(v)$, $c'(v^+ u^-) = \infty$.

Let $G' = (V', E')$, then $\overline{\kappa}_G(u, v)$ for all $u \neq v \in V$ is the value of the maximum flow from u to v in G'.

We provide two alternative ways to demonstrates there is a flow tree for separation.

Theorem 2.8. *For undirected graph $G = (V, E)$ with node capacities c. There exist a symmetric submodular $\overline{\kappa}$-cut function $A_{\overline{\kappa}}$.*

Proof. Consider the function $A_{\overline{\kappa}} : 2^V \to \mathbb{R}^+$ defined as $A_{\overline{\kappa}}(U) = \min_{U \subseteq U'} |\delta_{G'}(U')|$ for all $U \subseteq V$. $A_{\overline{\kappa}}$ is clearly symmetric, and it is submodular by Lemma 2.2. It characterizes all the minimum uv-cuts for $u, v \in V$ in G'. Hence $A_{\overline{\kappa}}$ is a $\overline{\kappa}_G$-cut. □

Element-connectivity can also capture separation.

For a $G = (V, E)$ and capacity function $c : V \to \mathbb{N}$. Let $V'_v = \{v'_1, v'_2, \ldots, v'_{c(v)}\}$. $\overline{V} = \{\overline{v} | v \in V\}$

$$V' = \overline{V} \cup \bigcup_{v \in V} V'_v \tag{2.11}$$

$$E' = \{\overline{v}v | v \in V\} \cup \{v'u' | vu \in E, v' \in V'_v, u' \in V'_u\} \tag{2.12}$$

Theorem 2.9. *Let $G = (V, E)$ with capacity function $c : V \to \mathbb{N}$. If $G' = (V', E')$, then for all $s \neq t \in V$, $\kappa'_{G'}(\overline{s}, \overline{t}) = \overline{\kappa}_G(s, t)$.*

Hence, if we build a Gomory-Hu tree for element-connectivity on G' with terminals $\overline{V}$, then it directly correspond to a flow tree for separation.

2.5 OPEN PROBLEMS

We described an $O(|T|mn)$ time algorithm for finding the reduced graph of a n vertex m edge graph with terminal nodes T. The main open question is whether we can find a reduced graph faster. The fastest algorithm for finding the all-pair local element-connectivity is the same as the algorithm for finding the global element-connectivity. Is there a faster algorithm for finding the global element-connectivity? We refer the reader to [65] for further related open problems.

Chapter 3: Hypergraph cuts

Algorithms for min-cuts and min-st-cuts in graphs have been extensively studied. Traditional algorithms for min-cut were based on computing a sequence of $(n-1)$ min-st-cuts; min-st-cuts are most efficiently computed via network flow although one can also compute them via submodular function minimization. The first algorithm for finding a min-cut in an *undirected* graph that avoided the use of flows was due to Nagamochi and Ibaraki [9]. They devised a surprising and influential algorithm based on maximum-adjacency orderings (MA-ordering) which is an ordering of the nodes based on a simple greedy rule. An MA-ordering can be computed in $O(m)$ time for uncapacitated graphs and in $O(m + n \log n)$ time for capacitated graphs. It has the following interesting property: if s and t are the last two nodes in the ordering then $\{t\}$ is an min-st-cut. This yields a simple $O(mn + n^2 \log n)$ time algorithm [66] for computing a min-cut in a capacitated graph and is currently the asymptotically fastest *deterministic* algorithm. MA-orderings have other important structural properties which lead to several algorithmic and structural results — many of these are outlined in [67]. Karger devised another highly influential technique based on random contractions [22] which led to a randomized $O(n^2 \log^3 n)$-time Monte Carlo algorithm for computing a min-cut in capacitated graphs [29]. Subsequently, using sampling techniques for cut-sparsification and tree packings, Karger devised a randomized $O(m \log^3 n)$ time Monte Carlo algorithm [46]. More recently Kawarabayashi and Thorup [44] devised a deterministic $O(m \log^{12} n)$ time algorithm for *simple* uncapacitated graphs. Henzinger, Rao and Wang [45] improved the deterministic running time in simple uncapacitated graphs to $O(m \log^2 n \log \log n)$ using flow partitioning.

What about connectivity in hypergraphs? A simple and well-known reduction shows that $\lambda_H(s,t)$ can be computed via st network flow in the node capacitated bipartite graph G_H associated with H. Thus, using $(n-1)$ network flows one can compute $\lambda(H)$. However, Queyranne [68] showed that the Nagamochi-Ibaraki ordering approach generalizes to find the min-cut of an arbitrary symmetric submodular function[1]. A specialization of the approach of Queyranne with a careful implementation leads to a deterministic $O(np + n^2 \log n)$-time algorithm for capacitated hypergraphs and an $O(np)$-time algorithm for uncapacitated hypergraphs. Two other algorithms achieving the same run-time were obtained by Klimmek and Wagner [69] and Mak and Wong [70]. Both these algorithms are based on the Nagamochi and Ibaraki ordering approach. Surprisingly, the orderings used by these three algorithms can be different for the same hypergraph even though they are identical for graphs[2]! We

[1] For a submodular function $f : 2^V \to \mathbb{R}$ the min-cut is defined naturally as $\min_{\emptyset \subsetneq S \subsetneq V} f(S)$.

[2] This observation does not appear to have been explicitly noted in the literature.

will later show that we can exploit their different properties in the algorithms.

Apart from the above mentioned results, very little else is known in the algorithms literature on min-cut and related problems in hypergraphs despite several applications, connections, and theoretical interest. Recent work has addressed streaming and sketching algorithms when the rank is small [11, 24]. In this chapter the three main questions we address are the following.

- Are there faster (approximation) algorithms for min-cut computation in hypergraphs?
- How many distinct min-cuts can there be? Can a compact representation called the hypercactus that is known to exist [18, 19] be computed fast? For graphs it is well-known that there are at most $\binom{n}{2}$ min-cuts and that there exists a compact $O(n)$-sized data structure called the *cactus* to represent all of them.
- Are there fast algorithm to *sparsify* a hypergraph to (approximately) preserve cuts? In graphs such results have found powerful applications in addition to mathematical elegance.

3.1 OVERVIEW

In this chapter we address the preceding questions and obtain several results that we outline below.

Sparse certificate and fast algorithm for small min-cuts: A k-certificate of a graph $G = (V, E)$ is a subgraph $G' = (V, E')$ of G that preserves all local connectivities in G up to k; that is $\lambda_{G'}(s, t) \geq \min\{k, \lambda_G(s, t)\}$ for all $s, t \in V$. Nagamochi and Ibaraki [9] showed, via MA-ordering, that a *k-certificate* with $O(kn)$ edges can be found in linear time. In the hypergraph setting, a k-certificate is a hypergraph preserving local connectivity up to k. A k-certificate with at most $k(n-1)$ edges is called a *k-sparse certificate*, and it exists by greedy spanning hypergraph packing [11]. However, the sum of degrees in the certificate can be $O(kn^2)$. Indeed, there are examples where any k-sparse certificate cannot avoid the n^2 factor. We consider a more general operation where we allow *trimming* of hyperedges; that is, a node $v \in e$ can be removed from e without e itself being deleted. Trimming has been used for various connectivity results on hypergraphs. For example, in studying k-partition-connected hypergraphs, or in extending Edmonds' arborescence packing theorem to directed hypergraphs [71] (see [72, Section 7.4.1, Section 9.4] for a exposition of the results using the trimming terminology).

We refer to a subhypergraph obtained through edge deletion and trimming and which preserves all cuts of value up to k as a *k-trimmed certificate*. We show that for any hypergraph H on n nodes there is a k-trimmed certificate H' that preserves all the local connectivities up to k such that the size of H' in terms of the sum of degrees is at most $2k(n-1)$. In fact the certificate has the stronger property that all cuts are preserved up to k: formally, for any $A \subseteq V$, $|\delta_{H'}(A)| \geq \min\{k, |\delta_H(A)|\}$. Moreover such a certificate can be constructed in $O(p)$ time. This leads to an $O(p + \lambda n^2)$ time for computing the min-cut in an uncapacitated hypergraph, substantially improving the $O(np)$ time when λ is small and p is large compared to n. Sparsification is of independent interest and can be used to speed up algorithms for other cut problems. We show an application to cut-sparsification in capacitated hypergraphs.

$(2+\epsilon)$ approximation for min-cut: Matula [21], building on the properties of MA-ordering, showed that a $(2+\epsilon)$ approximation for the min-cut of an uncapacitated graph can be computed in deterministic $O(m/\epsilon)$ time. The algorithm generalizes to capacitated graphs and runs in $O(\frac{1}{\epsilon}(m \log n + n \log^2 n))$ time (as mentioned by Karger [22]). Although the approximation is less interesting in light of the randomized $\tilde{O}(m)$ algorithm of Karger [46], it is a useful building block that allows one to deterministically estimate the value of a min-cut. For hypergraphs there was no such approximation known. In fact, the survey [65] observed that a near-linear time randomized $O(\log n)$-approximation follows from tree packing results, and raised the question of whether Matula's algorithm can be generalized to hypergraphs[3].

In this chapter we answer the question in the affirmative and obtain a $(2+\epsilon)$-approximation algorithm for the min-cut of a capacitated hypergraph that runs in near-linear time — more formally in $O(\frac{1}{\epsilon}(p \log n + n \log^2 n))$ time. For a uncapacitated hypergraph, the algorithm runs in $O(p/\epsilon)$ time. In fact we show that the same algorithm generalizes to a class of non-negative symmetric submodular functions that satisfy subadditive connectivity.

All min-cuts and hypercactus: We consider the problem of finding all the min-cuts in a hypergraph. For any capacitated graph G on n nodes, it is well-known, originally from the work of Dinitz, Karzanov and Lomonosov [12], that there is a compact $O(n)$ sized data structure, namely a cactus graph, that represents all the min-cuts of G. A cactus is a connected graph in which each edge is in at most one cycle (can be interpreted as a tree of cycles). As a byproduct one also obtains the fact that there are at most $\binom{n}{2}$ distinct min-cuts in a graph; Karger's contraction algorithm gives a very different proof. After a sequence of improvements, there exist deterministic algorithms to compute a cactus representation of the min-cuts of a graph in $O(mn + n^2 \log n)$ time [17] or in $O(nm \log(n^2/m))$-time [15, 16]. For uncapacitated graphs, there is an $O(m + \lambda^2 n \log(n/\lambda))$-time algorithm [15]. There is also

[3]We use near-linear-time to refer to algorithms that run in time $O(p \log^c n)$ for some fixed constant c.

a Monte Carlo algorithm that runs in $\tilde{O}(m)$ time [73] building on the randomized near-linear time algorithm of Karger [46]. In effect, the time to compute the cactus representation is the same as the time to compute the min-cut! We note, however, that all the algorithms are fairly complicated, in particular the deterministic algorithms.

The situation for hypergraphs is not as straight forward. First, how many distinct min-cuts can there be? Consider the example of a hypergraph $H = (V, E)$ with n nodes and a single hyperedge containing all the nodes. Then it is clear that every $S \subseteq V$ with $1 \leq |S| < |V|$ defines a min-cut and hence there are exponentially many. However, all of them correspond to the same edge-set. A natural question that arises is whether the number of distinct min-cuts in terms of the edge-sets is small. Indeed, one can show that it is at most $\binom{n}{2}$. It is a relatively simple consequence of fundamental decomposition theorems of Cunningham and Edmonds [74], Fujishige [75], and Cunningham [20] on submodular functions from the early 1980s. Recently Ghaffari et al. also proved this fact through a random contraction algorithm [3]. Cheng, building on Cunningham's work [20], explicitly showed that the min-cuts of a hypergraph admit a compact hypercactus structure. Later Fleiner and Jordan [18] showed that such a structure exists for any symmetric submodular function defined over crossing families. However, these papers were not concerned with algorithmic considerations.

In this chapter we show that the hypercactus representation of the min-cuts of a hypergraph, a compact $O(n)$ sized data structure, can be computed in $O(np + n^2 \log n)$ time and $O(p)$ space. This matches the time to compute a single min-cut. The known algorithms for cactus construction on graphs are quite involved and directly construct the cactus. We take a different approach. We use the structural theory developed in [19,20] to build the canonical decomposition of a hypergraph which then allows one to build a hypercactus easily. The algorithmic step needed for constructing the canonical decomposition is conceptually simpler and relies on an efficient algorithm for finding a non-trivial min-cut (one in which both sides have at least two nodes) in a hypergraph H if there is one. The technical contribution is to show that there is an algorithm for finding a slight weakening of this problem that runs in $O(p + n \log n)$ time. Interestingly, we establish this via the ordering from the paper of [70]. The algorithm yields a conceptually simple algorithm for graphs as well and serves to highlight the power of the decomposition theory for graphs and submodular functions [20,74,75].

Approximating strengths and $(1+\epsilon)$-cut sparsifiers: Benczúr and Karger, in their seminal work [23], showed that all cuts of a capacitated graph $G = (V, E)$ on n nodes can be approximated to within a $(1 \pm \varepsilon)$-factor by a sparse capacitated graph $G' = (V, E')$ where $|E'| = O(n \log n/\varepsilon^2)$. Moreover, G' can be computed in near-linear time by a randomized algorithm. The algorithm has two steps. In the first step, it computes for each edge e a

number p_e which is proportional to the inverse of the approximate *strength* of edge e (see Section 3.6 for a formal definition). The second step is a simple importance sampling scheme where each edge is independently sampled with probability p_e and if it is chosen, the edge e is assigned a capacity u_e/p_e where u_e is the original capacity/capacity of $e \in E$. It is shown in [23] that the probabilities $p_e, e \in E$ can be computed by a deterministic algorithm in $O(m \log^3 n)$ time where $m = |E|$ if G is capacitated and in $O(m \log^2 n)$ time if G is uncapacitated or has polynomially bounded capacities. More recent work [76] has shown a general framework for cut sparsification where the sampling probability p_e can also be chosen to be approximate connectivity between the end points of e. In another seminal work, Batson, Spielman and Srivastava [77] showed that spectral-sparsifiers, which are stronger than cut-sparsifiers, with $O(n/\varepsilon^2)$ edges exist, and can be computed in polynomial time. Lee and Sun recently showed such sparsifiers can be computed in randomized $\tilde{O}(m)$ time, where $\tilde{O}$ hide polylog factors [78].

Guha et al. [11] devised a simple method for cut-sparsifier of uncapacitated hypergraphs in the streaming setting. The approach follows [23]. They showed that there is a sketch of $O(\epsilon^{-2} n \operatorname{polylog} n)$ space where a $(1 \pm \epsilon)$-cut-sparsifier can be constructed. The running time is $\tilde{O}(np)$. However, their results cannot be generalized to capacitated hypergraphs. Kogan and Krauthgamer [24] extended Benczúr and Karger's sampling scheme and analysis to show the following: for any capacitated hypergraph $H = (V, E)$ with rank r there is a $(1 \pm \varepsilon)$-approximate cut-sparsifier with $O(n(r + \log n)/\varepsilon^2)$ edges. They show that sampling with (approximate) strengths will yield the desired sparsifier. Finding the strengths of edges in hypergraphs can be easily done in polynomial time. In this chapter, we develop a near-linear time algorithm for computing approximate strengths of edges in a capacitated hypergraph. For this purpose we rely on the sparsification for uncapacitated hypergraphs just as Benczúr and Karger relied on the sparsification of Nagamochi and Ibaraki. Fast sparsification leads to improved algorithms for several cut problems and we outline a few simple and direct applications in Section 3.6.

3.1.1 Other Related Work

Kogan and Krauthgamer's [24] sparsification relied on a more careful analysis of the random contraction algorithm of Karger when applied to hypergraphs. They showed that if the rank of the hypergraph is r then the number of α-min-cuts for *half-integer* $\alpha \geq 1$ is at most $O(2^{\alpha r} n^{2\alpha})$ which is a substantial improvement over a naive analysis that would give a bound of $O(n^{r\alpha})$. The exponential dependence on r is necessary. Asssi et al. later generalized it to all real $\alpha \geq 1$ [79]. Kogan and Krauthgamer's cut-sparsification results rely on the num-

ber of approximate minimum cuts. In particular, given a n-node capacitated hypergraph $H = (V, E)$ of rank r they show that there is a capacitated hypergraph $H' = (V, E')$ with $O(\frac{n}{\epsilon^2}(r + \log n))$ edges such that every cut capacity in H is preserved to within a $(1 \pm \epsilon)$ factor in H'. Aissi et al. [79] considered parametric min-cuts in graphs and hypergraphs of fixed rank and obtained polynomial bounds on the number of distinct min-cuts.

Hypergraph cuts have also been studied in the context of k-way cuts. We defer the discussion to chapter 4.

Organization: Section 3.2 sets up requisite notation, and also describes different node orderings and their properties which underpin most of the results in the chapter. Section 3.3 describes sparsification for uncapacitated hypergraphs. Section 3.5 describes the algorithm for computing the cactus representation of all the minimum cuts. Section 3.4 describes the extension of Matula's algorithm and analysis to obtain a $(2+\varepsilon)$-approximation for the global minimum cut of hypergraphs and a larger class of symmetric submodular functions. Section 3.6 describes our algorithm to compute approximate strengths of edges in a capacitated hypergraph that yields a near linear-time cut sparsification algorithm.

3.2 PRELIMINARIES

For disjoint A and B, $d'_H(A, B) = \sum_{e \in E_H(A,B), e \subseteq A \cup B} c(e)$ where only edges completely contained in $A \cup B$ are considered. As before, if H is clear from the context we drop it from the notation. $size(H)$ is defined as $\sum_{e \in E} |e|$ in the uncapacitated case and as $\sum_{e \in E} |e| c(e)$ in the capacitated case.

Removing a node $v \in e$ from e is called *trimming* e [72]. H' is a *trimmed subhypergraph* of H if there is a injection $\phi : E \to E'$ where $\phi(e) \subseteq e$. Namely, H' is obtained by deleting nodes and trimming edges.

Contracting an edge e in a hypergraph $H = (V, E)$ results in a new hypergraph $H' = (V', E')$ where all nodes in e are identified into a single new node v_e. Any edge $e' \subseteq e$ is removed. Any edge e' that properly intersects with e is adjusted by replacing $e' \cap e$ by the new node v_e. We note H' by H/e.

Equivalent digraph: min-st-cut in a hypergraph H can be computed via an maximum st-flow in an associated capacitated digraph (see [80]) $\vec{H} = (\vec{V}, \vec{E})$ that we call the equivalent digraph. $\vec{H} = (\vec{V}, \vec{E})$ is defined as follows:

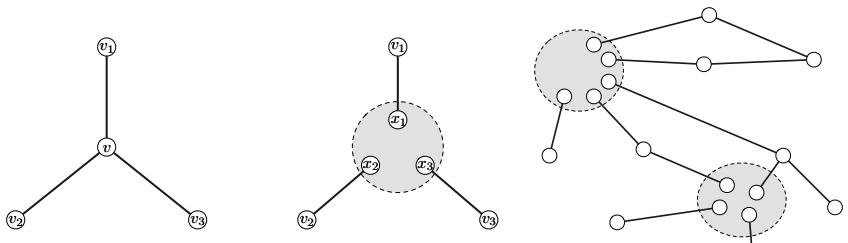
1. $\vec{V} = V \cup E^+ \cup E^-$, where $E^+ = \{e^+ | e \in E\}$ and $E^- = \{e^- | e \in E\}$.

2. If $v \in e$ for $v \in V$ and $e \in E$ then (v, e^-) and (e^+, v) are in $\vec{E}$ with infinite capacity.

3. For each $e \in E$, $(e^-, e^+) \in \vec{E}$ has capacity equal to $c(e)$.

Careful reader would note this is basically the same as the equivalent digraph in section 2.2. For any pair $s, t \in V(H)$, there is bijection between the finite capacity st-cuts in $\vec{H}$ and st-cuts in H. We omit further details of this simple fact. The number of nodes in the equivalent digraph is $O(m)$, and number of edges is $O(p)$. Suppose we wish to compute an minimum st-cut in the equivalent digraph where $s, t \in V$. If we use Orlin's algorithm [81] naively we obtain a running time of $O(mp)$ for max st-flow. However, the longest simple path in the equivalent digraph has length at most n. For any blocking flow based algorithm, there are at most n blocking flow iterations. Using dynamic trees, the running time of max st-flow on the equivalent digraph using Dinic's algorithm is $O(np \log n)$ [82, 83].

Cactus and hypercactus: A *cactus* is a graph in which every edge is in at most one cycle. A *hypercactus* is a hypergraph obtained by a sequence of *hyperedge insertions* starting from a cactus. A hyperedge insertion is defined as follows. A node v in a hypergraph with degree at least 3 and only incident to edges of rank 2, say $vv_1, \ldots, vv_k$ is called a v-star. A hyperedge insertion replaces a v-star by deleting v, adding new nodes $x_1, x_2, \ldots, x_k$, adding new edges $\{x_1, v_1\}, \{x_2, v_2\}, \ldots, \{x_k, v_k\}$ and a new hyperedge $\{x_1, x_2, \ldots, x_k\}$. See Figure 3.1 for examples.



(a) Example of hyperedge insertion operation on node v

(b) A hypercactus

Figure 3.1: Examples of hypercactus. Grey regions are hyperedges.

3.2.1 Node orderings

Nagamochi and Ibaraki's work on node orderings for graphs has been generalized to symmetric submodular functions by Queyranne [68] and further extended by Rizzi [84]. We set up the requisite notation in the abstract context before addressing the case of hypergraphs.

An ordered pair of nodes (s,t) is called a *pendant pair* for a function f if $f(\{t\}) = \lambda_f(s,t)$. A pendant pair for a hypergraph defined naturally as a pendant pair of its cut function. There are three algorithms for computing a hypergraph min-cut following the Nagamochi-Ibaraki approach of finding a pendant pair, contracting the pair, and recursing. It was generalized to symmetric submodular functions by Queyranne [68], and later generalized even further by Rizzi [84], which we describe below.

Monotone and consistent maps: A function g defined over pairs of disjoint subsets of V is called *order inducing* if it has the following properties.

- Symmetric: $g(A,B) = g(B,A)$ for all disjoint sets $A, B \subseteq V$.
- Monotone: $g(A,B) \leq g(A',B)$ if $A \subseteq A'$ and A' and B are disjoint subsets of V.
- Consistent: $g(A, B \cup C) \geq g(B, A \cup C)$ if A, B and C are disjoint and $g(A,C) \geq g(B,C)$.

The reason for calling such functions order inducing will be clear later. A node ordering is an ordering of the nodes $v_1, \ldots, v_n$. $V_i = \{v_1, \ldots, v_i\}$ are the first i nodes in the ordering. If $f(S) = g(S, \overline{S})$ for all $S \subseteq V$, we say *g realizes f*. Rizzi considered a *max-back ordering* of g. That is, an ordering of the nodes $v_1, \ldots, v_n$ such that

$$g(V_{i-1}, v_i) \geq g(V_{i-1}, v_j) \tag{3.1}$$

for all $j \geq i$.

Theorem 3.1 ([84]). *Let $v_1, \ldots, v_n$ be a max-back ordering of an order inducing function g over V and suppose g realizes f. Then $f(\{v_n\}) = \lambda_f(v_{n-1}, v_n) = g(V_{n-1}, v_n)$.*

In particular, if g realizes f, then a pendant pair of f can be found by computing the max-back ordering of g. Brinkmeier strengthened the preceding theorem as follows.

Theorem 3.2 ([85]). *Let $v_1, \ldots, v_n$ be a max-back ordering of an order inducing function g over V and suppose g realizes f. Then $\lambda_f(v_{i-1}, v_i) \geq g(V_{i-1}, v_i)$.*

It's easy to prove the following theorem via induction.

Theorem 3.3. *Let $g_1, \ldots, g_k$ be a collection of order inducing functions each of which realizes f, then any convex combination of $g_1, \ldots, g_k$ is an order inducing function that realizes f.*

Connectivity function of a submodular function: Let f be a symmetric submodular function. The *connectivity function d_f* of f is defined on all pairs of disjoint subsets of V as

follows: $d_f(A,B) = \frac{1}{2}(f(A) + f(B) - f(A \cup B))$. It can be easily seen that d_f realizes f. In particular, we define a *Queyranne ordering* of a symmetric submodular function f as a max-back ordering with respect to d_f.

Lemma 3.1. *Let c be the cut function of a hypergraph, then $d_c(A,B) = \frac{1}{2}(d(A,B)+d'(A,B))$ for all disjoint $A, B \subseteq V$.*

Proof. We just have to consider the case where c is the cut function of a hypergraph with a single edge e. By definition $d_c(A,B) = \frac{1}{2}(c(A) + c(B) - c(A \cup B))$.

If e neither crosses A nor B, then we have $d_c(A,B) = 0$ and $d(A,B) = d'(A,B) = 0$. It's clear $d_c(A,B) = \frac{1}{2}(d(A,B) + d'(A,B))$. Hence, without loss of generality, we assume e crosses A. We have $c(A) = 1$. If e does not cross B, and e has an element not in B, so $e \cap B = \emptyset$. This shows $d_c(A,B) = 0$ and $c(B) = 1$. We have e must cross $A \cup B$, because e contains an element not in A, and $e \cap B = \emptyset$. Therefore $c(A \cup B) = 1$. Hence $d_c(A,B) = \frac{1}{2}(c(A) + c(B) - c(A \cup B)) = 0 = \frac{1}{2}(d(A,B) + d'(A,B))$.

In the remaining case e must cross both A and B, therefore $c(A) = c(B) = 1$ and $d(A,B) = 1$. If $e \subseteq A \cup B$, then $d'(A,B) = 1$ and $c(A \cup B) = 0$, we obtain $d_c(A,B) = 1 = \frac{1}{2}(d(A,B)+d'(A,B))$. On the other hand, if $e \not\subseteq A \cup B$, then $d'(A,B) = 0$ and $c(A \cup B) = 1$, we also obtain $d_c(A,B) = \frac{1}{2}(1+1-1) = \frac{1}{2}(d(A,B)+d'(A,B)) = 1/2$, the desired result. □

One can observe that if f is non-negative, then d_f is also non-negative. By submodularity, for any disjoint $A, B \subseteq V$, $d_f(A,B) = \frac{1}{2}(f(A)+f(B)-f(A \cup B)) \geq \frac{1}{2}(f(A)+f(B)-f(A \cup B) - f(A \cap B)) \geq 0$.

Node orderings for hypergraphs We work with several node orderings defined for hypergraphs. Given a hypergraph $H = (V, E)$ and an ordering of the nodes $v_1, \ldots, v_n$, several other orderings and quantities are induced by such an ordering. The *head* of an edge $h(e)$, defined as $v_{\min\{j|v_j \in e\}}$, is the first node of e in the ordering. An ordering of the edges $e_1, \ldots, e_m$ is called *head ordering*, if $\min\{j|v_j \in e_i\} \leq \min\{j|v_j \in e_{i+1}\}$. An edge e is a *backward edge* of v if $v \in e$ and $h(e) \neq v$. The head of a backward edge incident to v comes before v in the node order.

Definition 3.1. *An ordering of nodes $v_1, \ldots, v_n$ is called an* α-ordering *for $\alpha \in [0,1]$ if for all $1 \leq i < j \leq n$, $\alpha d(V_{i-1}, v_i) + (1-\alpha)d'(V_{i-1}, v_i) \geq \alpha d(V_{i-1}, v_j) + (1-\alpha)d'(V_{i-1}, v_j)$. An ordering is called*

1. *a* maximum adjacency ordering *or* MA-ordering *if it is a* 1*-ordering.*
2. *a* tight ordering *if it is a* 0*-ordering.*

3. a Queyranne ordering *if it is a* $\frac{1}{2}$*-ordering.*

In graphs the three orderings coincide if the starting node is the same and ties are broken in the same way. However, they can be different in hypergraphs. As an example, consider a hypergraph with nodes a, x, y, z and four edges with capacities as follows: $c(\{a, x\}) = 4, c(\{a, y\}) = 3, c(\{a, x, z\}) = 4$ and $c(\{a, y, z\}) = 8$. Capacities can be avoided by creating multiple copies of an edge. Consider orderings starting with a. It can be verified that the second node has to be x, y and z for tight, Queyranne, and MA-ordering respectively which shows that they have to be distinct.

Klimmek and Wagner used the MA-ordering [69]. Mak and Wong used the tight ordering [70]. Queyranne defined an ordering for symmetric submodular functions [68] which when specialized to cut functions of hypergraphs is the one we define, see Lemma 3.1. All three orderings can be computed in $O(p + n \log n)$ time for capacitated hypergraphs, and in $O(p)$ time for uncapacitated hypergraphs.

In Sections 3.3, 3.4 and 3.5, we use MA-ordering, tight ordering and Queyranne ordering, respectively. We state a lemma that summarizes the crucial property that all α-orderings share. These three different ordering can be used to find the min-cut because of a more general phenomenon; every α-ordering is an order inducing functions that realizes the hypergraph cut function.

Lemma 3.2. *d and d' are order inducing functions that realize the cut function of a hypergraph.*

Proof. The proof is routine. We just have to show d and d' are order inducing for hypergraph with only one edge. So we consider the hypergraph with a single edge e. It's clear that d is symmetric. $d(A, B) = 1$ implies the edge crosses both A and B, then certainly e also crosses $A \subseteq A'$, therefore d is monotone. Similarly we can show d' is monotone.

To prove d is consistent, we have to prove that $d(A, B \cup C) \geq d(B, A \cup C)$ for all disjoint sets A, B, C such that $d(A, C) \geq d(B, C)$. If $d(B, A \cup C) = 0$ then we are done because d is non-negative. Hence we can assume $d(B, A \cup C) = 1$. In other words, the edge has at least 1 node in B, and at least one node in $A \cup C$. This shows $d(A, B \cap C) = 1$ if $A \cap e$ is non-empty, and we are done. Assume $A \cap e$ is empty, then this shows $e \cap C$ is non-empty, hence $1 = d(B, C) \leq d(A, C) = 0$, a contradiction.

Similarly, we can prove d' is consistent. Again, we can assume $d'(B, A \cup C) = 1$. So we know $e \subseteq A \cup B \cup C$, and $e \cap B$, $e \cap (A \cup C)$ are non-empty. Note if $A \cap e$ is non-empty, then $d'(A, B \cap C) = 1$, and we are done. Assume $A \cap e$ is empty, then this shows $e \cap C$ is non-empty. Because $e \cap A = \emptyset$ and $e \subseteq A \cup B \cup C$, hence $e \subseteq B \cup C$ and $1 = d'(B, C) \leq d(A, C) = 0$, a contradiction. □

Therefore, we obtain the following lemma for all α-orderings of the hypergraph as a direct corollary of Theorem 3.2.

Lemma 3.3. *Let $v_1, \ldots, v_n$ be a α-ordering of a hypergraph, then $\{v_n\}$ is a min v_{n-1}-v_n-cut.*

3.3 K-TRIMMED CERTIFICATE AND FASTER MIN-CUT ALGORITHM FOR SMALL λ

This section shows that a well-known sparsification result for graphs can be generalized to hypergraphs. The hypergraphs in this section are *uncapacitated* although multiple parallel edges are allowed.

Given an uncapacitated hypergraph H and a non-negative integer k, the goal of sparsification is to find a sparse trimmed subhypergraph H' of H such that $|\delta_{H'}(A)| \geq \min(k, |\delta_H(A)|)$ for all $A \subseteq V$. Namely, it preserves all cuts up to value k. Such trimmed subhypergraphs are called *k-certificates*. A non-trimmed subhypergraph that is a k-certificate with at most $k(n-1)$ edges is called a *k-sparse certificate*. It is known that there exists a k-sparse certificate for each hypergraph [11]. The fact that such a certificate exists is not hard to prove. One can generalize the forest decomposition technique for graphs [9] in a straight-forward way. However, the size of the resulting certificate could be large. Indeed, there might not exist any k-sparse certificate with *size* $O(kn)$. Consider the following example. Let $H = (V, E)$ be a hypergraph on n nodes and $n/2-1$ edges (assume n is even) where $E = \{e_1, \ldots, e_{n/2-1}\}$ and $e_i = \{v_i, v_{n/2}, \ldots, v_n\}$ for $1 \leq i < n/2$. Any connected subhypergraph of H has to contain all the edges and thus, even for $k = 1$, the sum of degrees is $\Omega(n^2)$.

However, we prove a stronger result if trimming is allowed. H' is a k-certificate of H and $size(H') \leq 2k(n-1)$, then it is called a *k-trimmed certificate*. One can see a k-trimmed certificate has at most $k(n-1)$ edges. Hence k-trimmed certificates and k-sparse certificates coincide in graphs.

Theorem 3.4. *Let $H = (V, E)$ be a hypergraph on n nodes and m edges with $size(H) = p$. There is an algorithm that given H creates a data structure in $O(p)$ time such that the following holds: for any given non-negative integer k, the data structure returns a k-trimmed certificate H' of H in $O(size(H'))$ time.*

The above theorem is tight for graphs. The proof is an adaptation of that of Frank, Ibaraki and Nagamochi [61] for the graph version of the sparsification theorem.

Given a hypergraph $H = (V, E)$ consider an MA-ordering $v_1, \ldots, v_n$ and let $e_1, \ldots, e_m$ be the induced head ordering of the edges. Let $D_k(v)$ to be the first k backward edges of v in

the head ordering, or all the backward edges of v if there are fewer than k backward edges. For each node v and backward edge e of v, we remove v from e if $e \notin D_k(v)$. The new hypergraph from this operation is H_k. Formally, given H and k, $H_k = (V, E_k)$ is defined as follows. For an edge $e \in E$ let e' denote the edge $\{v \mid v \in e \in D_k(v) \text{ or } v = h(e)\}$; E_k is defined to be the edge set $\{e' \mid e \in E, |e'| \geq 2\}$. It is easy to see that if $j \leq k$, H_j is a trimmed subhypergraph of H_k.

We observe that $size(H_k) \leq 2k(n-1)$. Each node v that is not v_1 is in at most k backward edges in H_k for a total contribution of at most $k(n-1)$ to the sum of degrees, and the remaining contribution of at most $k(n-1)$ comes from head of each edge which can be charged to the backward edges.

We sketch a data structure that can be created from hypergraph H in $O(p)$ time, such that for all k, the data structure can retrieve H_k in $O(kn)$ time. First, we compute the MA-ordering, which takes $O(p)$ time. Using the MA-ordering, we obtain the induced head ordering of the edges, and the head for each edge, again in $O(p)$ time; we omit the simple details for this step. For each node v, we can sort all the backedges of v use the head ordering in $O(p)$ time as follows: we maintain a queue Q_v for each node v, and inspect the edges one by one in the head ordering. When e is inspected, we push e into queue Q_v if $v \in e$ and v is not the head of e. This completes the preprocessing phase for the data structure. To retrieve H_k, we maintain a set of queues that eventually represent the set of edges E_k. For each node v, find the edges in $D_k(v)$, which is exactly the first k edges (or all edges if there are fewer than k edges) in Q_v. For each edge $e \in D_k(v)$, we push v into a queue Q_e (if Q_e was not already created, we first create Q_e and push the head node of e into Q_e). At the end of the process, each queue Q_e contains the nodes of an edge in E_k. The running time is $O(size(H_k)) = O(kn)$ since we only process $D_k(v)$ for each v.

It remains to show that H_k is a k-certificate of H.

Lemma 3.4. *If $v_1, \ldots, v_n$ is an MA-ordering of H, and H_k is a hypergraph obtained from H via the ordering, then $v_1, \ldots, v_n$ is an MA-ordering of H_k.*

Proof. By construction, $d_{H_k}(V_i, v_j) \leq \min(k, d_H(V_i, v_j))$ for all $i < j$. Consider the first $\min\{k, d_H(V_i, v_j)\}$ edges incident to v_j in the head ordering; v_j is not trimmed from them. Hence $d_{H_k}(V_i, v_j) \geq \min\{k, d_H(V_i, v_j)\}$.

For all $i \leq j$,

$$d_{H_k}(V_{i-1}, v_i) \geq \min\{k, d_H(V_{i-1}, v_i)\} \geq \min\{k, d_H(V_{i-1}, v_j)\} \geq d_{H_k}(V_{i-1}, v_j). \tag{3.2}$$

This establishes that $v_1, \ldots, v_n$ is an MA-ordering for H_k. □

For $X \subseteq V$ we define $\gamma(X) = \{e \mid e \cap X \neq \emptyset\}$ to be the set of edges that contain at least one node from X. We need a helper lemma below.

Lemma 3.5. *Let $H = (V, E)$ be a hypergraph and $A, B \subseteq V$. For $u \in B$ and $v \in V$, if $E(u, v) \subseteq \delta(A) \cap \gamma(B)$, then*

$$|\gamma(B) \cap \delta(A)| \geq |\gamma(B - u) \cap \delta(A)| + |E(u, v) \setminus E(B - u, u, v)|. \tag{3.3}$$

Proof. Consider an edge $e \in E(u, v) \setminus E(B-u, u, v)$. We claim that $e \notin \gamma(B-u)$. Indeed, if $e \in \gamma(B-u)$, then e is an edge that intersects $B-u, \{u\}$ and $\{v\}$, but then $e \in E(B-u, u, v)$. This shows $E(u, v) \setminus E(B - u, u, v)$ is disjoint from $\gamma(B - u)$, and therefore disjoint from $\gamma(B - u) \cap \delta(A)$.

We have (i) $\gamma(B - u) \cap \delta(A) \subseteq \gamma(B) \cap \delta(A)$ since $\gamma(B - u) \subseteq \gamma(B)$, and (ii) $E(u, v) \setminus E(B - u, u, v) \subseteq \gamma(B) \cap \delta(A)$ by assumption. Since we have argued that $\gamma(B - u)$ and $E(u, v) \setminus E(B - u, u, v)$ are disjoint, we have the desired inequality

$$|\gamma(B) \cap \delta(A)| \geq |\gamma(B - u) \cap \delta(A)| + |E(u, v) \setminus E(B - u, u, v)|. \tag{3.4}$$

□

Lemma 3.6. *Let $v_1, \ldots, v_n$ be an MA-ordering for $H = (V, E)$. Then, for all $i < j$ and $A \subseteq V$ such that $v_i \in A$ and $v_j \notin A$. $|\gamma(V_{i-1}) \cap \delta(A)| \geq d(V_{i-1}, v_j)$.*

Proof. Proof by induction on (i, j) ordered lexicographically. For the base case consider $i = 1$ and $j > 1$. Indeed, in this case both sides of the inequality are 0 and the desired inequality holds trivially. Assume lemma is true for all (i', j') where $1 \leq i' < j'$, such that $j' < j$ or $j' = j$ and $i' < i$. We consider two cases.

Case 1: $v_{i-1} \in A$. Then because $v_j \notin A$, $E(v_{i-1}, v_j) \subseteq \delta(A) \cap \gamma(V_{i-1})$. We apply Lemma 3.5 with $B = V_{i-1}$ and $u = v_{i-1}$ and $v = v_j$ to obtain:

$$\begin{aligned}
|\gamma(V_{i-1}) \cap \delta(A)| &\geq |\gamma(V_{i-2}) \cap \delta(A)| + |E(v_{i-1}, v_j) \setminus E(V_{i-2}, v_{i-1}, v_j)| && (3.5)\\
&\geq d(V_{i-2}, v_j) + |E(v_{i-1}, v_j) \setminus E(V_{i-2}, v_{i-1}, v_j)| \quad \text{(induction on } (i-1, j), A) && (3.6)\\
&= d(V_{i-1}, v_j). && (3.7)
\end{aligned}$$

Case 2: $v_{i-1} \notin A$. Note that $i \geq 2$. Consider $A' = V \setminus A$. We have $v_{i-1} \in A'$ and $v_i \notin A'$; therefore, $E(v_{i-1}, v_i) \subseteq \delta(A') \cap \gamma(V_{i-1})$. Applying Lemma 3.5 with $B = V_{i-1}$, $u = v_{i-1}$ and

$v = v_i$,

$$|\gamma(V_{i-1}) \cap \delta(A')| \geq |\gamma(V_{i-2}) \cap \delta(A')| + |E(v_{i-1}, v_i) \setminus E(V_{i-2}, v_{i-1}, v_i)| \tag{3.8}$$

$$\geq d(V_{i-2}, v_i) + |E(v_{i-1}, v_i) \setminus E(V_{i-2}, v_{i-1}, v_i)| \quad \text{(induction on } (i-1, i), A') \tag{3.9}$$

$$= d(V_{i-1}, v_i) \tag{3.10}$$

$$\geq d(V_{i-1}, v_j) \quad \text{(from MA-ordering)}. \tag{3.11}$$

Since $\delta(A') = \delta(V \setminus A) = \delta(A)$, $|\gamma(V_{i-1}) \cap \delta(A')| = |\gamma(V_{i-1}) \cap \delta(A)|$.

This finishes the proof. □

Using the preceding lemma we finish the proof that H_k is a k-trimmed certificate.

Theorem 3.5. *For every $A \subseteq V$, $|\delta_{H_k}(A)| \geq \min(k, |\delta_H(A)|)$.*

Proof. By induction on k. The statement is clearly true for $k = 0$. $|\delta_{H_i}(A)| \leq |\delta_{H_k}(A)|$ for all $i \leq k$, because H_i is a trimmed subhypergraph of H_k.

Consider any $k > 0$. If $|\delta_H(A)| = k' < k$, then by induction,

$$k' = |\delta_{H_{k'}}(A)| \leq |\delta_{H_k}(A)| \leq |\delta_H(A)| = k'. \tag{3.12}$$

Therefore, it suffices to only consider A such that $|\delta_H(A)| \geq k$. We will derive a contradiction assuming that $|\delta_{H_k}(A)| < k$. Since $|\delta_H(A)| \geq k$, there exists an edge $e \in E(H)$ such that $e \in \delta_H(A)$ but was either trimmed to $e' \in E(H_k)$ such that $e' \notin \delta_{H_k}(A)$ or e is removed completely because it is trimmed to be a singleton $\{h(e)\}$. Let $v_i = h(e)$ and without loss of generality we can assume $v_i \in A$ (otherwise we can consider $\overline{A}$). Since e' does not cross A in H_k, there is a $v_j \in e \cap \overline{A}$ with $j > i$ (since v_i is the head of e) and v_j was trimmed from e during the sparsification.

$D_k(v_j)$ has exactly k edges because backward edge e of v_j is not in $D_k(v_j)$. For each $f \in D_k(v_j)$, the trimmed $f' \in E(H_k)$ contains both $h(f) = v_\ell$ and v_j; we claim that for each such f, $\ell \leq i$ for otherwise e would be ahead of f in the head order and v_j would be trimmed from f before it is trimmed from e. From this we obtain that $E_{H_k}(V_{j-1}, v_j) = E_{H_k}(V_i, v_j)$ and hence $d_{H_k}(V_{j-1}, v_j) = d_{H_k}(V_i, v_j) = k$.

For the remainder of the the proof, we only work with H_k and the quantities d, δ, E, γ are with respect to this hypergraph and not H. We have

$$k = d(V_{j-1}, v_j) = d(V_i, v_j) = d(V_{i-1}, v_j) + d(v_i, v_j) - d(V_{i-1}, v_i, v_j). \tag{3.13}$$

Hence $d(V_{i-1}, v_j) = k - d(v_i, v_j) + d(V_{i-1}, v_i, v_j)$.

From Lemma 3.4 $v_1, \ldots, v_n$ is an MA-ordering of H_k as well. Applying Lemma 3.6 to H_k, v_i and v_j and A, $|\gamma(V_{i-1}) \cap \delta(A)| \geq d(V_{i-1}, v_j)$. Combining this inequality with the preceding one,

$$|\gamma(V_{i-1}) \cap \delta(A)| \geq d(V_{i-1}, v_j) = k - d(v_i, v_j) + d(V_{i-1}, v_i, v_j). \tag{3.14}$$

We also observe that $E(V_{i-1}, v_i, v_j) \subseteq E(v_i, v_j) \subseteq \delta(A)$ because $v_i \in A$ and $v_j \notin A$. We obtain a contradiction by the following set of inequalities:

$$\begin{aligned}
k &> |\delta(A)| \quad \text{(by assumption)} && (3.15)\\
&\geq |\gamma(V_i) \cap \delta(A)| && (3.16)\\
&= |\gamma(V_{i-1}) \cap \delta(A)| + |E(v_i, v_j) \setminus E(V_{i-1}, v_i, v_j)| && (3.17)\\
&= |\gamma(V_{i-1}) \cap \delta(A)| + d(v_i, v_j) - d(V_{i-1}, v_i, v_j) && (3.18)\\
&\geq (k - d(v_i, v_j) + d(V_{i-1}, v_i, v_j)) + d(v_i, v_j) - d(V_{i-1}, v_i, v_j) \quad \text{(from 3.14)} && (3.19)\\
&= k && (3.20)
\end{aligned}$$

This finishes the proof. □

One can ask whether tight ordering or Queyranne ordering would also lead to k-trimmed certificates. We observe that they do not work if the only modification is the input ordering, and H_k is constructed the same way through the head ordering of the edges.

For tight ordering, consider $H = (\{0, 1, 2, 3\}, E)$, where $E = \{\{0, 1, 2\}, \{0, 2, 3\}, \{1, 2\}\}$. $0, 1, 2, 3$ is a tight ordering. H_2 is H with edge $\{1, 2\}$ removed. $\lambda_{H_2}(1, 2) = 1 < 2 = \lambda_H(1, 2)$.

For Queyranne ordering, consider $H = (\{0, \ldots, 4\}, E)$, where

$$E = \{\{0, 1, 2\}, \{0, 1, 2, 3\}, \{0, 1, 3, 4\}, \{1, 3, 4\}, \{2, 3\}\}. \tag{3.21}$$

$0, 1, 2, 3, 4$ is a Queyranne ordering. H_3 is all the edges except the edge $\{2, 3\}$. We have $\lambda_{H_3}(2, 3) = 2 < 3 = \lambda_H(2, 3)$.

There may be other ways to obtain a k-trimmed certificate using these orderings but we have not explored this.

Algorithmic applications: Computing connectivity in uncapacitated hypergraphs can be sped up by first finding a trimmed certificate of the given hypergraph and then running a standard algorithm on the certificate. This is especially useful when one is interested in

small values of connectivity. For min-cut we obtain the following theorem.

Theorem 3.6. *The min-cut of a uncapacitated hypergraph H with n nodes and $size(H) = p$ can be computed in $O(p + \lambda n^2)$ time, where λ is the value of the min-cut of H.*

Proof. Using Theorem 3.4, we first compute a data structure in $O(p)$ time that allows us to retrieve H_k in $O(kn)$ time for any given k. Suppose we knew a number k such that $k \geq \lambda$ and $k = O(\lambda)$. We can compute the k-trimmed certificate H_k in $O(kn)$ time and compute $\lambda(H_k) = \lambda(H)$ in $O(\lambda n^2)$ time using one of the known algorithms since $size(H_k) = O(\lambda n)$. To find k we apply exponential search for the smallest i such that $2^i > \lambda$. Each search computes hypergraph min-cut on H_{2^i}, which takes $O(2^i n^2)$ time. For any $k > \lambda$, the value of the min-cut on the k-trimmed certificate equals to λ. Therefore, the search stops when the value of min-cut of H_{2^i} is strictly smaller than 2^i. The total time for the computation is $O(p + \sum_{i=1}^{1+\lceil \log \lambda \rceil} 2^i n^2) = O(p + \lambda n^2)$. □

A similar idea can be used to compute the edge-connectivity in H between some given pair s, t. An algorithm for st connectivity that runs in time $T(n, m, p)$ on a hypergraph with n nodes, m edges and sum of degrees p can be sped up to $T(n, m, \lambda(s, t)n)$. k-trimmed certificate can also help in computing α-approximate min-cuts for $\alpha > 1$.

k-sparse certificate: In some applications trimming is not meaningful and we actually want a k-sparse certificate instead of a k-trimmed certificate. We have already mentioned that one can find a k-sparse certificate with $k(n-1)$ edges via the forest peeling technique of Nagamochi and Ibaraki; this was done in [11]. Even in this setting the ordering based algorithm has an advantage over the naive forest peeling in terms of the run time. Suppose we want a k-sparse certificate. This can be done easily by first computing H_k as before, and replacing each edge in H_k by its original untrimmed counterpart in H. Note that the total number of edges is at most $k(n-1)$ and the running time is $O(p)$.

We need a capacitated version of the k-sparse certificate for Section 3.6. Given a hypergraph $H = (V, E)$ with capacities $c : E \to \mathbb{R}_+$, we say that a subhypergraph $H' = (V, E')$ with $E' \subseteq E$ and capacities $c' : E' \to \mathbb{R}_+$ is a *k-sparse certificate* if (i) for all $A \subseteq V$, $c'(\delta_{H'}(A)) \geq \min(c(\delta_H(A)), k)$, (ii) $c'(e) \leq c(e)$ for all $e \in E'$ and (iii) $\sum_{e \in E'} c'(e) \leq k(n-1)$.

If c is integer valued then we can run the uncapacitated algorithm by creating $c(e)$ copies of an edge e and obtain the desired k-sparse certificate; the running time will not be strongly polynomial. The general capacitated case is handled by simulating the uncapacitated algorithm in an implicit fashion as follows. Given a capacitated hypergraph $H = (V, E)$ consider an MA-ordering of nodes $v_1, \ldots, v_n$. Let $e_1, \ldots, e_m$ be the induced head ordering of the edges, just as before. The algorithm is similar for the uncapacitated version. For a node v, assume

via renumbering that its backward edges are $e_1, \ldots, e_\ell$ with capacities $c_1, \ldots, c_\ell$, and if $i < j$ then e_i comes before e_j in the head order of the edges. For v we define $c_v : E \to \mathbb{R}_+$ as follows: for $1 \leq i \leq \ell$, $c_v(e_i) = \max(\min(k - \sum_{j<i} c_j, c_i), 0)$ and $c_v(e) = 0$ for any edge $e \in E \setminus \{e_1, \ldots, e_\ell\}$. We now define $c' : E \to \mathbb{R}_+$ as follows: $c'(e) = \max_{v \in V} c_v(e)$. Let $E' = \{e \in E \mid c'(e) > 0\}$. The capacitated hypergraph $H' = (V, E')$ with capacity function c' is a k-sparse certificate of H. It is easy to see that the algorithm is implicitly simulating the uncapacitated algorithm with edges duplicated. With some basic but careful bookkeeping, c' can be computed in $O(p)$ time. The running time is dominated by finding the MA-ordering, which is $O(p + n \log n)$ time.

This leads to the following theorem.

Theorem 3.7. *There is an algorithm that given uncapacitated hypergraph H and integer k, finds a k-sparse certificate E' of H in $O(p)$ time. If the hypergraph is capacitated, it finds a k-sparse certificate in $O(p + n \log n)$ time.*

3.4 CANONICAL DECOMPOSITION AND HYPERCACTUS REPRESENTATION

In this section, we are interested in finding a canonical decomposition of a capacitated hypergraph which captures, in a compact way, information on all the min-cuts. Cunningham [20] proved that such decomposition exists for an arbitrary non-negative submodular function, following previous work by Cunningham and Edmonds [74] and Fujishige [75]. Cheng [19] showed that the canonical decomposition can be used to efficiently, and relatively easily, build a hypercactus representation. Subsequently Fleiner and Jordan [18] obtained a similar result for arbitrary symmetric submodular functions. One can view the canonical decomposition as a more fundamental object since it is unique while the cactus and hypercactus representations are not necessarily unique.

As noted already by Cunningham, the key tool needed to build a canonical decomposition is an algorithm to find a non-trivial min-cut. Here we show an efficient algorithm for finding such a min-cut in a hypergraph, and then use it to build a canonical decomposition. We can then construct a hypercactus from the canonical decomposition as shown in [19]. We believe that this approach is easier to understand and conceptually simpler than the existing deterministic cactus construction algorithms for graphs that build the cactus directly.

A cut is *trivial* if one side of the cut has exactly one node. A *split* is a non-trivial min-cut. An *st* split is a split that separates s and t.

3.4.1 An efficient split oracle for hypergraphs

Given a hypergraph H we would like to find a split if one exists. It is not hard to come up with a polynomial-time algorithm for this task but here we wish to design a faster algorithm. We accomplish this by considering a weaker guarantee which suffices for our purposes. The algorithm, given H and the min-cut value λ, outputs either a split in H or a pair of nodes $\{s, t\}$ such that there is no st split in H. We call such an algorithm a split oracle. We describe a near-linear-time split oracle.

We first show how to use a maximum st flow in $\vec{H}$ to help decide whether there is an st split, and output one if it exists.

Lemma 3.7. *Given a maximum st flow in the equivalent digraph of H, and the value of min-cut λ in H, there is an algorithm that in $O(p)$ time either finds a st split, or certifies that no st split exists in H.*

Proof. If the value of the maximum st flow is greater than λ, there is no st split. Otherwise, there is an st split iff there is a non-trivial min-st-cut in H.

Suppose a digraph G has k minimum u-v-cuts for some node pair (u, v). Given a maximum u-v flow in G and an integer ℓ, there is an enumeration algorithm [86] that outputs $\min(\ell, k)$ distinct min-u-v-cuts in $O(\ell m)$ time where m is the number of edges in G.

We run the enumeration algorithm with $\ell = 3$ on $\vec{H}$ for the pair (s, t). Every min-st-cut in $\vec{H}$ corresponds to a min-st-cut in H. If the algorithm returns at most two cuts and both are trivial then there is no st split. Otherwise one of the output cuts is an st split. The running time is $O(p)$ since the number of edges in $\vec{H}$ is $O(p)$. □

One can find a maximum st flow in $\vec{H}$ using standard flow algorithms but that would not lead to a near-linear time algorithm. In graphs, Arikati and Mehlhorn [87] devised a linear-time algorithm that computes the maximum flow between the last two nodes of an MA-ordering. Thus, we have a near-linear-time split oracle for graphs. Recall that in hypergraphs there are three orderings which all yield a pendant pair. We generalized Arikati and Mehlhorn's algorithm to a linear-time algorithm that tries to find a maximum flow between the last two nodes of an MA-ordering of a hypergraph (the flow is in the equivalent digraph). Even though it appears to correctly compute a maximum flow in all the experiments we ran, we could not prove its correctness. Instead we found a different method based on the tight ordering, that we describe below.

Let $v_1, v_2, \ldots, v_n$ be a tight ordering for a hypergraph $H = (V, E)$. We define a tight graph $G = (V, E')$ with respect to H and the given tight ordering as follows. For each edge

$e \in E$, we add an edge e' to E', where e' consists of the last 2 nodes of e under the tight ordering. The key observation is the following.

Lemma 3.8. *Suppose $H = (V, E)$ is a hypergraph and $v_1, \ldots, v_n$ is a tight ordering for H, and $G = (V, E')$ is the corresponding tight graph. Then, for $1 \leq i < j \leq n$, $d'_G(V_i, v_j) = d'_H(V_i, v_j)$.*

Proof. Consider any edge e counted in $d'_H(V_i, v_j)$. $e \subseteq V_i \cup \{v_j\}$, and e contains v_j. e' contains v_j, and the other end of e' is in V_i. Therefore e' is counted in $d'_G(V_i, v_j)$. This shows that $d'_H(V_i, v_j) \leq d'_G(V_i, v_j)$.

To see the other direction, consider an $e' \in E$ corresponding to an edge $e \in E$. If e' is counted in $d'_G(V_i, v_j)$, it must be that v_j is the last node in e and the second last node of e is in V_i. This implies that $e \subseteq V_i \cup \{v_j\}$, and therefore counted in $d'_H(V_i, v_j)$, and completes the direction $d'_H(V_i, v_j) \geq d'_G(V_i, v_j)$. □

The preceding lemma implies that the tight ordering for H is a tight ordering for G. From Lemma 3.3,

$$\lambda_G(v_{n-1}, v_n) = d'_G(V_{n-1}, v_n) = d'_H(V_{n-1}, v_n) = \lambda_H(v_{n-1}, v_n) \tag{3.22}$$

Letting $s = v_{n-1}$ and $t = v_n$, we see that $\lambda_G(s, t) = \lambda_H(s, t)$. Moreover, an st flow in G can be easily lifted to an st flow in $\vec{H}$. Thus, we can compute an st max flow in G in linear-time using the algorithm of [87] and this can be converted, in linear time, into an st max-flow in $\vec{H}$.

This gives the following theorem.

Theorem 3.8. *The split oracle can be implemented in $O(p + n \log n)$ time for capacitated hypergraphs, and in $O(p)$ time for uncapacitated hypergraphs.*

3.4.2 Decompositions, Canonical and Prime

We define the notion of decompositions to state the relevant theorem on the existence of a canonical decomposition. In later subsections we describe the computational aspects.

A hypergraph H is *prime* if it does not contain any split; in other words all min-cuts of H are trivial. A capacitated hypergraph is called a *solid polygon* if it consists of a cycle where each edge has the same capacity a and a hyperedge containing all nodes with capacity b. If $a = 0$, it is called *brittle*, otherwise it is called *semi-brittle*. A solid polygon is *not* prime if it has at least 4 nodes. For a semi-brittle hypergraph with at least 4 nodes, every split consists of two edges on the cycle and the hyperedge covering all nodes. For a brittle hypergraph with at least 4 nodes, any non-trivial cut is a split.

Given a hypergraph $H = (V, E)$ and a set U, a function $\phi : V \to U$ defines a new hypergraph through a sequence of contraction operations as follows: for each element $u \in U$, contract $\phi^{-1}(u)$ into u. The resulting hypergraph is the ϕ-contraction of H. A hypergraph obtained from $H = (V, E)$ by contracting $S \subseteq V$ into a single node is denoted by H/S.

$\{H_1, H_2\}$ is a *simple refinement* of H if H_1 and H_2 are hypergraphs obtained through a split (V_1, V_2) of H and a new *marker* node x as follows.

1. H_1 is H/V_2, such that V_2 gets contracted to x.

2. H_2 is H/V_1, such that V_1 gets contracted to x.

If $\{H_1, H_2\}$ is a simple refinement of H, then min-cut value of H_1, H_2 and H are all equal.

A set of hypergraphs $\mathcal{D} = \{H_1, H_2, \ldots, H_k\}$ is called a *decomposition* of a hypergraph H if it is obtained from $\{H\}$ by a sequence of operations each of which consists of replacing one of the hypergraphs in the set by its simple refinement; here we assume that each operation uses new marker nodes. A decomposition $\mathcal{D}$ is a simple refinement of decomposition $\mathcal{D}'$ if $\mathcal{D}$ is obtained through replacing one of the hypergraph in $\mathcal{D}'$ by its simple refinement. A decomposition $\mathcal{D}'$ is a *refinement* of $\mathcal{D}$ if $\mathcal{D}'$ is obtained through a sequence of simple refinement operations from $\mathcal{D}$. If the sequence is non-empty, $\mathcal{D}'$ is called a *strict refinement*. Two decompositions are *equivalent* if they are the same up to relabeling of the marker nodes. A decomposition is *minimal* with property $\mathcal{P}$ if it is not a strict refinement of some other decomposition with the same property $\mathcal{P}$. A *prime* decomposition is one in which all members are prime. A decomposition is *standard* if every element is either prime or a solid polygon.

Every element in the decomposition is obtained from a sequence of contractions from H. Hence we can associate each element H_i in the decomposition with a function $\phi_{H_i} : V \to V(H_i)$, such that H_i is the ϕ_{H_i}-contraction of H. Every decomposition $\mathcal{D}$ has an associated *decomposition tree* obtained by having a node for each hypergraph in the decomposition and an edge connecting two hypergraphs if they share a marker node.

The important theorem below is due to [20], and stated again in [19] specifically for hypergraphs.

Theorem 3.9 ([20])**.** *Every hypergraph H has a unique (up to equivalence) minimal standard decomposition. That is, any two minimal standard decompositions of H differ only in the labels of the marker nodes.*

The unique minimal standard decomposition is called the *canonical decomposition*. As a consequence, *every* standard decomposition is a refinement of the canonical decomposition.

We remark that minimality is important here. It captures all the min-cut information in H as stated below.

Theorem 3.10 ([19,20]). *Let $\mathcal{D} = \{H_1, \ldots, H_k\}$ be a canonical decomposition of H.*

1. *For each min-cut S of H, there is a unique i, such that $\phi_{H_i}(S)$ is a min-cut of H_i.*
2. *For each min-cut S of H_i, $\phi^{-1}_{H_i}(S)$ is a min-cut of H.*

Note that each hypergraph in a canonical decomposition is either prime or a solid polygon and hence it is easy to find all the min-cuts in each of them. We observe that any decomposition $\mathcal{D}$ of H can be compactly represented in $O(n)$ space by simply storing the node sets of the hypergraph in $\mathcal{D}$.

Recall that a set of edges $E' \subseteq E$ is called a min cut-set if $E' = \delta(S)$ for some min-cut S. As a corollary of the preceding theorem, one can easily prove that there are at most $\binom{n}{2}$ distinct min cut-sets in a hypergraph. This fact is not explicitly stated in [19,20].

Corollary 3.1. *A hypergraph with n nodes has at most $\binom{n}{2}$ distinct min cut-sets.*

Proof. Let H be a hypergraph on n nodes. If H is prime, then there are at most n min cut-sets. If H is a solid-polygon, then there are at most $\binom{n}{2}$ min cut-sets. Let $\mathcal{D}$ be a canonical decomposition of H. $\mathcal{D}$ is obtained via a simple refinement $\{H_1, H_2\}$ of H with size a and b, followed by further refinement. Then, by induction, there are at most $\binom{a}{2} + \binom{b}{2}$ min cut-sets in H_1 and H_2. Here $a + b = n + 2$ and $a, b \leq n - 1$. Therefore $\binom{a}{2} + \binom{b}{2} \leq \binom{3}{2} + \binom{n-1}{2} \leq \binom{n}{2}$ when $n \geq 4$. □

As we already mentioned, the preceding corollary is also derived via a random contraction algorithm in [3].

3.4.3 Computing a canonical decomposition

In this section we describe an efficient algorithm for computing the canonical decomposition of a hypergraph H.

We say that two distinct splits $(A, \overline{A})$ and $(B, \overline{B})$ cross if A and B cross, otherwise they do not cross. One can easily show that every decomposition is equivalently characterized by the set of non-crossing splits induced by the marker nodes. Viewing a decomposition as a collection of non-crossing splits is convenient since it does not impose an order in which the splits are processed to arrive at the decomposition — any ordering of processing the non-crossing splits will generate the same decomposition.

Call a split *good* if it is a split that is not crossed by any other split; otherwise the split is called *bad*. A canonical decomposition corresponds to the collection of all good splits. The canonical decomposition can be obtained through the set of of good splits [74, Theorem 3] via the following simple algorithm. If H is prime or solid polygon return $\{H\}$ itself. Otherwise find a good split $(A, \overline{A})$ and the simple refinement $\{H_1, H_2\}$ of H induced by the split and return the union of the canonical decompositions of H_1 and H_2 computed recursively. Unfortunately, finding a good split directly is computationally intensive.

On the other hand finding a prime decomposition can be done via a split oracle by a simple recursive algorithm, as we shall see in Section 3.4.4. Note that a prime decomposition is not necessarily unique. We will build a canonical decomposition through a prime decomposition. This was hinted in [20], but without details and analysis. Here we formally describe such an algorithm.

One can go from a prime decomposition to a canonical decomposition by removing some splits. Removing a split corresponds to gluing two hypergraphs with the same marker node into another hypergraph resulting in a new decomposition. We formally define the operation as follows. Suppose $\mathcal{D}$ is a decomposition of H with a marker node x contained in H_1 and H_2. We define a new contraction of H obtained by *gluing* H_1 and H_2. Let ϕ_{H_1} and ϕ_{H_2} be the contractions of H, respectively. Define function $\phi' : V \to (V(H_1) \cup V(H_2)) - x$ as follows

$$\phi'(v) = \begin{cases} \phi_{H_1}(v) & \text{if } \phi_{H_2}(v) = x \\ \phi_{H_2}(v) & \text{if } \phi_{H_1}(v) = x \end{cases} \tag{3.23}$$

H_x is the contraction of H defined by ϕ'. The gluing of $\mathcal{D}$ through x is the set $\mathcal{D}_x = \mathcal{D} - \{H_1, H_2\} \cup \{H_x\}$. The operation reflects removing the split induced by x from the splits induced by $\mathcal{D}$, therefore it immediately implies the following lemma.

Lemma 3.9. *$\mathcal{D}_x$ is a decomposition of H. Moreover, $\mathcal{D}_x$ can be computed from $\mathcal{D}$ and H in $O(p)$ time.*

In order to compute D_x implicitly, we only have to obtain ϕ_{H_1}, ϕ_{H_2} and compute a single contraction. Therefore $O(p)$ space is sufficient if we can obtain ϕ_{H_1} and ϕ_{H_2} in $O(p)$ time and space.

We need the following simple lemma.

Lemma 3.10. *Let H be a solid polygon. Any decomposition of H is a standard decomposition. Thus, if $\mathcal{D}$ is a decomposition of H, for any marker node, gluing it results in a standard decomposition of H.*

Proof. We first prove that any decomposition of H is a standard decomposition. This is by induction. If the solid polygon consists of a cycle with positive capacity, then exactly two edges in the cycle and the edge that contains all nodes crosses a split. One can verify that contraction of either side of the split results in a solid polygon or a prime hypergraph. Otherwise, the solid polygon is a single hyperedge covering all nodes. Any contraction of this hypergraph is a solid polygon.

The second part of the lemma follows from the first and the fact that gluing results in a decomposition. □

The following lemma is easy to see.

Lemma 3.11. *Given a hypergraph H there is an algorithm to check if H is a solid polygon in $O(p)$ time.*

Adding a split corresponds to a simple refinement. Therefore a decomposition $\mathcal{D}'$ is a refinement of $\mathcal{D}$ then the set of induced splits of $\mathcal{D}$ is a subset of induced splits of $\mathcal{D}'$.

Consider the following algorithm that starts with a prime decomposition $\mathcal{D}$. For each marker x, inspect if gluing through x results in a standard decomposition; one can easily check via the preceding lemma whether the gluing results in a solid polygon which is the only thing to verify. If it is, apply the gluing, if not, move on to the next marker. Every marker will be inspected at most once, therefore the algorithm stops after $O(n)$ gluing operations and takes time $O(np)$. The goal is to show the correctness of this simple algorithm.

The algorithm starts with a prime decomposition $\mathcal{D}$ which is a standard decomposition. If it is minimal then it is canonical and no gluing can be done by the algorithm (otherwise it would violate minimality) and we will output a canonical decomposition as required. If $\mathcal{D}$ is not minimal then there is a canonical decomposition $\mathcal{D}^*$ such that $\mathcal{D}$ is a strict refinement of $\mathcal{D}^*$. Let $\mathcal{D}^* = \{H_1, H_2, \ldots, H_k\}$ where each H_i is prime or a solid polygon. Therefore $\mathcal{D} = \cup_{i=1}^{k} \mathcal{D}_i$ where $\mathcal{D}_i$ is a refinement of H_i. If H_i is prime than $\mathcal{D}_i = \{H_i\}$. If H_i is a solid polygon then $\mathcal{D}_i$ is a standard decomposition of H_i. The goal is to show that irrespective of the order in which we process the markers in the algorithm, the output will be $\mathcal{D}^*$. Let the marker set for $\mathcal{D}^*$ be M^* and that for $\mathcal{D}$ be $M \supset M^*$. Suppose the first marker considered by the algorithm is x. There are two cases.

The first case is when $x \in M - M^*$. In this case the marker x belongs to two hypergraphs G_1 and G_2 both belonging to some $\mathcal{D}_i$ where H_i is a solid polygon. The algorithm will glue G_1 and G_2 and from Lemma 3.10, this gives a smaller standard decomposition $\mathcal{D}'_i$ of H_i.

The second case is when the marker $x \in M^*$. Let x belong to two hypergraphs G_1 and G_2 where $G_1 \in \mathcal{D}_i$ and $G_2 \in \mathcal{D}_j$ where $i \neq j$. In this case we claim that the algorithm will

not glue G_1 and G_2 since gluing them would not result in a standard decomposition. To see this, let $\mathcal{D}'$ be obtained through gluing of G_1 and G_2. The split induced by x is in $\mathcal{D}^*$ but not $\mathcal{D}'$. Because the splits induced by $\mathcal{D}^*$ is not a subset of splits induced by $\mathcal{D}'$, $\mathcal{D}'$ is not a refinement of $\mathcal{D}^*$. However, as we noted earlier, every standard decomposition is a refinement of $\mathcal{D}^*$. Hence $\mathcal{D}'$ is not a standard decomposition.

From the two cases we see that no marker in M^* results in a gluing and every marker in $M - M^*$ results in a gluing. Thus the algorithm after processing $\mathcal{D}$ outputs $\mathcal{D}^*$. This yields the following theorem.

Theorem 3.11. *A canonical decomposition can be computed in $O(np)$ time given a prime decomposition.*

Next we describe an $O(np + n^2 \log n)$ time algorithm to compute a prime decomposition.

3.4.4 Computing a prime decomposition

We assume there exists an efficient *split oracle*. Given a hypergraph H and the value of the min-cut λ, the split oracle finds a split in H or returns a pair $\{s, t\}$, such that there is no *st* split in H. In the latter case we would like to recurse on the hypergraph obtained by contracting $\{s, t\}$ into a single node. In order to recover the solution, we define how we can uncontract the contracted nodes.

Definition 3.2. *Consider a hypergraph H. Let $H' = H/\{s, t\}$, where $\{s, t\}$ is contracted to node $v_{\{s,t\}}$. Let G' be a ϕ'-contraction of H' such that $\phi'(v_{\{s,t\}}) = v_{\{s,t\}}$. We define uncontracting $v_{\{s,t\}}$ in G' with respect to H as a graph G obtained from a ϕ-contraction of H, where ϕ is defined as*

$$\phi(v) = \begin{cases} \phi'(v) & \text{if } v \notin \{s, t\} \\ v & \text{otherwise} \end{cases} \tag{3.24}$$

See Figure 3.2 for a simple recursive algorithm that computes a prime decomposition based on the split oracle. The following lemma justifies the soundness of recursing on the contracted hypergraph when there is no *st* split.

Lemma 3.12. *Suppose H is a hypergraph with no st split for some $s, t \in V(H)$. Let $H' = H/\{s, t\}$, where $\{s, t\}$ is contracted to node $v_{\{s,t\}}$. Let $\mathcal{D}'$ be a prime decomposition of H', and let $G' \in \mathcal{D}'$ such that G' contains node $v_{\{s,t\}}$. And let G be obtained through uncontracting $v_{\{s,t\}}$ in G' with respect to H.*

```
PRIME(H, λ)
  if |V(H)| ≥ 4
      x ← a new marker node
      query the split oracle with H and λ
      if oracle returns a split (S, V(H) − S)
          {H1, H2} ← REFINE(H, (S, V(H) − S), x)
          return PRIME(H1, λ) ∪ PRIME(H2, λ)
      else the oracle returns {s, t}
          D' ← PRIME(H/{s, t}, λ), {s, t} contracts to v_{s,t}
          G' ← the member of D' that contains v_{s,t}
          G ← uncontract v_{s,t} in G' with respect to H
          if ({s, t}, V(G) − {s, t}) is a split in G
              {G1, G2} ← refinement of G induced by {s, t}
              D ← (D' − {G'}) ∪ {G1, G2}
          else
              D ← (D' − {G'}) ∪ G
          return D
  else
      return {H}
```

Figure 3.2: The algorithm for computing a prime decomposition.

1. *Suppose $\{s,t\}$ defines a split in G and let $\{G_1, G_2\}$ be a simple refinement of G based on this split. Then $\mathcal{D} = (\mathcal{D}' - \{G'\}) \cup \{G_1, G_2\}$ is a prime decomposition of H.*

2. *If $\{s,t\}$ does not define a split in G then $\mathcal{D} = (\mathcal{D}' - \{G'\}) \cup \{G\}$ is a prime decomposition of H.*

Proof. Every split in H' is a split in H. Therefore $(\mathcal{D}' - \{G'\}) \cup \{G\}$ is a decomposition of H. Other than G, all other elements in $(\mathcal{D}' - \{G'\}) \cup \{G\}$ are prime.

If G is not prime, then there is a split. There is no *st* split in G because H does not have any *st* split. Any split in G must have the form $(A, V(G) - A)$ where $\{s,t\} \subseteq A$. If $A \neq \{s,t\}$, then there exist some other node $v \in A$, which implies $|A - \{s,t\} \cup \{v_{\{s,t\}}\}| \geq 2$, and $(A - \{s,t\} \cup \{v_{\{s,t\}}\}, V(G') - A)$ is a split in G', a contradiction to the fact that G' is prime. Hence $(\{s,t\}, V(G) - \{s,t\})$ is the unique split in G. Therefore the simple refinement of G based on this unique split are both prime, and we reach the first case.

If G is prime, then we are done, as we reach the second case. □

Theorem 3.12. PRIME(H, λ) *outputs a prime decomposition in $O(n(p + T(n,m,p)))$ time. Where $T(n,m,p)$ is the time to query split oracle with a hypergraph of n nodes, m edges and sum of degree p.*

Proof. Using induction and Lemma 3.12, the correctness of the algorithm is clear. PRIME is called at most $2n$ times, and each call takes $O(p + T(n, m, p))$ time. □

Using the split oracle from Theorem 3.8 we obtain the following corollary.

Corollary 3.2. *A prime decomposition of a capacitated hypergraph can be computed in $O(np + n^2 \log n)$ time. For uncapacitated hypergraphs it can be computed $O(np)$ time.*

3.4.5 Reducing space usage

The description of computing the prime and canonical decompositions did not focus on the space usage. A naive implementation can use $\Omega(np)$ space if we store each hypergraph in the decomposition explicitly. Here we briefly describe how one can reduce the space usage to $O(p)$ by storing a decomposition implicitly via a *decomposition tree*.

Consider a decomposition $\mathcal{D} = \{H_1, \ldots, H_k\}$ of $H = (V, E)$. We associate a decomposition tree $T = (A, F)$ with $\mathcal{D}$ where $A = \{a_1, \ldots, a_k\}$, one node per hypergraph in $\mathcal{D}$; there is an edge $a_i a_j \in F$ iff H_i and H_j share a marker node. With each a_i we also store $V(H_i)$ which includes the marker nodes and some nodes from $V(H)$. This is stored in a map $\psi : A \to \cup_i V(H_i)$. It is easy to see that the total storage for the tree and storing the node sets is $O(n)$; a marker node appears in exactly two of the hypergraphs of a decomposition and a node of H in exactly one of the hypergraphs in the decomposition.

Given the decomposition tree T and ψ and a node $a_i \in A$, we can recover the hypergraph H_i (essentially the edges of H_i since we store the node sets explicitly) associated with a node a_i in $O(p)$ time. For each edge e incident to a_i in T, let C_e be the component of $T - e$ that does not contain a_i. $V(H) \cap (\cup_{a_j \in C_e} \psi(a_j))$ are the set of nodes in H which are contracted to a single marker node in H_i corresponding to the edge e. We collect all this contraction information and then apply the contraction to the original hypergraph H to recover the edge set of H_i. It is easy to see that this can be done in $O(p)$ time.

3.4.6 Hypercactus representation

For a hypergraph H, a hypercactus representation is a hypercactus H^* and a function $\phi : V(H) \to V(H^*)$ such that for all $S \subseteq V(H)$, S is a min-cut in H if and only if $\phi(S)$ is a min-cut in H^*. This is a generalization of the cactus representation when H is a graph.

Note the similarity of Theorem 3.10 and the definition of the hypercactus representation. It is natural to ask if there is a hypercactus representation that is essentially a canonical decomposition. Indeed, given the canonical decomposition of H, Cheng showed that one

can construct a "structure hypergraph" that captures all min-cuts [19], which Fleiner and Jordan later point out is a hypercactus representation [18]. The process to construct such a hypercactus representation from a canonical decomposition is simple. We describe the details for the sake of completeness.

Assume without loss of generality that $\lambda(H) = 1$. We construct a hypercactus if the hypergraph is prime or a solid polygon. If H is a solid polygon, then it consists of a cycle and a hyperedge containing all the nodes. If the cycle has non-zero capacity, let H^* to be H with the hyperedge containing all the nodes removed, and assign a capacity of $\frac{1}{2}$ to each edge of the cycle. If the cycle has zero capacity, then let H^* to be a single hyperedge containing all nodes, the hyperedge has capacity 1. In both cases H^* together with the the identity function on $V(H)$ forms a hypercactus representation for H. If H is prime, let V' be the set of nodes that induce a trivial min-cut, i.e. $v \in V'$ iff $\{v\}$ is a min-cut in H. Introduce a new node v_H, and let $H^* = (\{v_H\} \cup V', \{\{v_H, v'\} | v' \in V'\})$, with capacity 1 for each edge; in other words we create a star with center v_H and leaves in V'. Define $\phi : V(H) \to V(H^*)$ as

$$\phi(u) = \begin{cases} u & u \in V' \\ v_H & u \notin V' \end{cases} \tag{3.25}$$

Then H^* and ϕ form a hypercactus representation.

For the more general case, let $\mathcal{D}^* = \{H_1, \ldots, H_k\}$ be the canonical decomposition of H. For each i, construct hypercactus representation (H_i^*, ϕ_i) of H_i as described earlier. We observe that that if x is a marker node in H_i, then it is also present in H_i^*. If H_i is a solid polygon this is true because $V(H_i) = V(H_i^*)$. If H_i is prime, then every marker node induces a trivial min-cut in H_i, hence also preserved in H_i^*. Construct H^* from $H_1^*, \ldots, H_k^*$ by identifying marker nodes. This also gives us $\phi : V(H) \to V(H^*)$ by gluing together $\phi_1, \ldots, \phi_k$ naturally. (H^*, ϕ) is the desired hypercactus representation.

The construction takes $O(np)$ time and $O(p)$ space.

Theorem 3.13. *A hypercactus representation of a capacitated hypergraph can be found in $O(n(p + n \log n))$ time and $O(p)$ space for capacitated hypergraphs, and in $O(np)$ time for uncapacitated hypergraphs.*

Proof. We combine Theorem 3.11 and Corollary 3.2. The space usage can be made $O(p)$ based on the discussion in Section 3.4.5. □

If H is a graph, the hypercactus representation constructed is a cactus representation. Theorem 3.13 matches the best known algorithm for cactus representation construction of graphs in both time and space [17], and is conceptually simpler.

Via sparsification we obtain a faster algorithm for uncapacitated hypergraphs.

Theorem 3.14. *A hypercactus representation of an uncapacitated hypergraph can be found in $O(p + \lambda n^2)$ time and $O(p)$ space.*

Proof. Find the min-cut value λ, and a $(\lambda + 1)$-sparsifier H' of H in $O(p + \lambda n^2)$ time. Theorem 3.5 shows that every min-cut in H is a min-cut in H', and vice versa. Therefore the hypercactus for H' is a hypercactus for H. Apply Theorem 3.13 to H'. □

For graphs Gabow [15] achieves a running time of $O(m + \lambda n^2)$ which is faster than the running time of $O(m + \lambda n^2)$ given by the preceding theorem. Gabow's algorithm is much more complex and it is an open problem whether one can achieve similar running time even for hypergraphs with fixed rank.

3.5 NEAR-LINEAR TIME $(2 + \epsilon)$ APPROXIMATION FOR MIN-CUT

Matula gave an elegant use of MA-ordering to obtain a $(2+\epsilon)$-approximation for the min-cut in an uncapacitated undirected graph in $O(m/\epsilon)$ time [21]. Implicit in his paper is an algorithm that gives a $(2+\epsilon)$-approximation for capacitated graphs in $O(\frac{1}{\epsilon}(m \log n + n \log^2 n))$ time; this was explicitly pointed out by Karger [22]. Here we extend Matula's idea to hypergraphs. We describe an algorithm that outputs a $(2+\epsilon)$-approximation for hypergraph min-cut in $O(\frac{1}{\epsilon}(p \log n + n \log^2 n))$ time for capacitated case, and in $O(p/\epsilon)$ time for the uncapacitated case. In fact, we show a more general result. For any non-negative symmetric submodular function whose connectivity function d_f is *subadditive*, we can produce a $(2+\epsilon)$-approximation of the minimizer using $O(\frac{n^2 \log n}{\epsilon})$ value oracle calls to the function f. Note that Queyranne's algorithm that outputs an exact minimizer requires $\Omega(n^3)$ oracle calls.

A connectivity function of f, d_f, is *subadditive* if $d_f(A, B \cup C) \leq d_f(A, B) + d_f(A, C)$ for all disjoint A, B, C. d_f is *additive* if preceding inequality holds with equality. We say a function is nice, if it is normalized, symmetric, submodular and its connectivity function is subadditive. It is easy to see that non-negative capacitated sum of nice functions is also nice.

Theorem 3.15. *The cut function of a hypergraph is nice.*

Proof. We just have to prove the theorem for hypergraphs with at most 1 edge, then we can use the fact every hypergraph cut function is sum of hypergraph cut function realized by a single edge hypergraph.

Consider the hypergraph on n nodes V, and a single edge e. Consider arbitrary disjoint sets of nodes A, B and C, we want to show

$$d_c(A, B \cup C) \leq d_c(A, B) + d_c(A, C). \tag{3.26}$$

If $d_c(A, B \cup C) = 0$, then the inequality is clearly true. Hence we can assume $d(A, B \cup C)$ is either $\frac{1}{2}$ or 1. If $d_c(A, B) = 1$. Since we know $d_c(A, B) = \frac{1}{2}(d(A, B) + d'(A, B))$ for all A and B, we know $d(A, B \cup C) = d'(A, B \cup C) = 1$. This shows $e \cap (B \cup C)$ is non-empty and $e \subseteq A \cup B \cup C$. Either $e \cap B$ or $e \cap C$ is non-empty. Therefore either $d(A, B)$ or $d(A, C)$ is 1. Assume $d(A, B) = 1$. If we also have $d(A, C) = 1$, then we have $d_c(A, B \cup C) = 1 = \frac{1}{2}d(A, B) + \frac{1}{2}d(A, C) \leq d_c(A, B) + d_c(A, B)$. If otherwise $d(A, C) = 0$, this implies $e \cap C$ is empty, and $e \subseteq A \cup B$. Hence $d'(A, B) = 1$. We have $d_c(A, B \cup C) = 1 = \frac{1}{2}(d(A, B) + d'(A, B)) = d_c(A, B) \leq d_c(A, B) + d_c(A, C)$.

Now, we assume that $d_c(A, B \cup C) = \frac{1}{2}$. This shows $d(A, B \cup C) = 1$, and either $d(A, B)$ or $d(A, C)$ is 1. Either way $d_c(A, B \cup C) = \frac{1}{2} \leq \frac{1}{2}d(A, B) + \frac{1}{2}d(A, C) \leq d_c(A, B) + d_c(A, C)$. □

Note the proof above shows the connectivity of graph functions are additive.

Are all nice functions hypergraph cut functions? The answer is no. Yamaguchi showed there exists a hypergraph cut function h and a graph cut function g such that $h - g$ is a normalized symmetric submodular function but not a hypergraph cut function [88, Remark 2.]. It's easy to see $h - g$ has subadditive connectivity since g have additive connectivity.

We now consider nice functions instead of hypergraphs and after the algorithm and analysis in this more general setting we return to estimate the running time of the algorithm for hypergraphs. We define $\sigma(f) = \sum_{v \in V} f(v)$ and $\delta(f) = \min_{v \in V} f(v)$. The function f' obtained from contracting S into s in f is defined such that $f'(X) = f(X)$ if $s \notin X$, and $f'(X) = f(X \cup S \setminus \{s\})$ if $s \in X$. One can define inductively a function obtained by contracting a few disjoint sets. It is not hard to see that nice functions are closed under contraction.

Recall the Queyranne ordering of the nodes in f is an ordering of the nodes $v_1, \ldots, v_n$, such that

$$d_f(V_i, v_{i+1}) \leq d_f(V_i, v_j) \tag{3.27}$$

for all $j \geq i + 1$, where $d_f(A, B) = \frac{1}{2}(f(A) + f(B) - f(A \cup B))$.

Let $v_1, \ldots, v_n$ be a Queyranne ordering of the given function f. Given a non-negative number α, a set of consecutive nodes in the ordering $v_a, v_{a+1}, \ldots, v_b$ where $a \leq b$ is called α-tight if $d_f(V_i, v_{i+1}) \geq \alpha$ for all $a \leq i < b$. The *maximal* α-tight sets partition V. We obtain a new function by contracting each maximal α-tight set into a single node. We call

the contracted function an α-contraction. Note that the contraction depends both on α and the specific Queyranne ordering.

Lemma 3.13. *Let f be any normalized symmetric set function on $\{v_1, \ldots, v_n\}$. Consider any ordering of the nodes $v_1, \ldots, v_n$, then*

$$\sum_{i=1}^{n} d_f(V_{i-1}, v_i) = \frac{1}{2}\sigma(f) \tag{3.28}$$

Proof. The proof follows from the following series of simple equalities.

$$\sum_{i=1}^{n} d_f(V_{i-1}, v_i) = \frac{1}{2}\sum_{i=1}^{n}(f(V_{i-1}) + f(v_i) - f(V_i)) = \frac{1}{2}\sum_{i=1}^{n}(f(V_{i-1}) - f(V_i)) + \frac{1}{2}\sum_{i=1}^{n} f(v_i) \tag{3.29}$$

$$= \frac{1}{2}(f(V_0) - f(V_n) + \frac{1}{2}\sum_{i=1}^{n} f(v_i) = \frac{1}{2}\sum_{i=1}^{n} f(v_i) = \frac{1}{2}\sigma(f) \tag{3.30}$$

□

One important aspect of α-contraction of a nice function f is that $\sigma(f') \leq 2\alpha n$ if f' is the function obtained from α-contraction.

Lemma 3.14. *Let f' be an α-contraction of a given nice function f. Then $\sigma(f') \leq 2\alpha n$ where $n = |\operatorname{dom}(f)|$.*

Proof. Assume the α-tight partition of V is $X_1, \ldots, X_h$ where the ordering of the parts is induced by the Queyranne ordering. For $1 \leq i \leq h$, let $A_i = \bigcup_{j=1}^{i} X_j$. Since each X_i is a maximal α-tight set, $d_f(A_i, x) < \alpha$ for all $x \in X_{i+1}$. We have the following set of inequalities:

$$\sigma(f') = \sum_{i=1}^{h} f(X_i) \tag{3.31}$$

$$= 2\sum_{i=1}^{h} d_f(A_{i-1}, X_i) \qquad \text{(Lemma 3.13)} \tag{3.32}$$

$$\leq 2\sum_{i=1}^{h} \sum_{x \in X_{i+1}} d_f(A_i, x) \qquad \text{(subadditivity)} \tag{3.33}$$

$$< 2\sum_{i=1}^{h} \alpha |X_{i+1}| \leq 2\alpha n. \tag{3.34}$$

□

The second important property of α-contraction is captured by the following lemma.

Lemma 3.15. *Let f be a non-negative symmetric submodular function over V and let $v_1, \ldots, v_n$ be a Queyranne ordering of the node set V. If v_i and v_j are in an α-tight set then $\lambda_f(v_i, v_j) \geq \alpha$.*

Proof. Assume without loss of generality that $i < j$. Consider any k such that $i \leq k < j$. We have $d_f(V_k, v_{k+1}) \geq \alpha$ because i and j are in the same α-tight set. By Theorem 3.2, $\lambda_f(v_k, v_{k+1}) \geq d_f(V_k, v_{k+1}) \geq \alpha$. By induction and using the fact that for any $a, b, c \in V$, $\lambda_f(a, c) \geq \min(\lambda_f(a, b), \lambda_f(b, c))$, we have $\lambda_f(v_i, v_j) \geq \alpha$. □

Figure 3.3 describes a simple recursive algorithm for finding an approximate min-cut.

```
APPROXIMATE-MINIMIZER(f)
  if (|dom(f)| ≥ 2)
      δ ← δ(f)
      if (δ = 0)
          return 0
      α ← 1/(2+ε) δ
      Compute Queyranne ordering of f
      f' ← α-contraction of f
      λ' ← APPROXIMATE-MINIMIZER(f')
      return min(δ, λ')
  else
      return ∞
```

Figure 3.3: Description of $(2+\epsilon)$-approximation algorithm. It is easy to remove the recursion.

Theorem 3.16. APPROXIMATE-MIN-CUT *outputs a $(2+\epsilon)$-approximation to an input nice function f, and can be implemented in $O(\frac{n^2}{\epsilon} \log \frac{n\delta(f)}{\lambda(f)})$ oracle calls.*

Proof. We first argue about the termination and run-time. From Lemma 3.14, $\sigma(f') \leq \frac{2}{2+\epsilon} n\delta(f)$. Since $\sigma(f) \geq n\delta(f)$, we see that each recursive call reduces the σ value of the function by a factor of $\frac{2}{2+\epsilon}$. This ensures termination.

Let f'' be any non-trivial normalized symmetric submodular function (that has at least two nodes) that arises in the recursion. The minimizer value does not reduce by contraction and hence $\lambda(f'') \geq \lambda(f)$ which in particular implies that $\delta(f'') \geq \lambda(f)$, and hence $\sigma(f'') \geq 2|\operatorname{dom}(f'')|\lambda(f)$. After the first recursive call, $\sigma(f') \leq \frac{2}{2+\epsilon} n\delta(f)$. Thus the total number of recursive calls is $O(\epsilon^{-1} \log(\frac{n\delta(f)}{\lambda(f)}))$. The work in each call is dominated by the time to compute a Queyranne ordering which can be done in $O(n^2)$ oracle calls [68]. This time gives the desired upper bound on the run-time of the algorithm.

We now argue about the correctness of the algorithm which is by induction on n. It is easy to see that the algorithm correctly outputs the min-cut value if $n = 1$ or if $\delta = 0$. Assume $n \geq 2$ and $\delta(f) > 0$. The number of nodes in f' is strictly less than n if $\delta(f) > 0$ since the σ strictly decreases. Since contraction does not reduce the non-trivial minimizer value, $\lambda(f') \geq \lambda(f)$. By induction, $\lambda(f') \leq \lambda' \leq (2+\epsilon)\lambda(f')$. If $\lambda(f') = \lambda(f)$ then the algorithm outputs a $(2+\epsilon)$-approximation since $\delta \geq \lambda(f)$. The more interesting case is if $\lambda(f') > \lambda(f)$. This implies that there are two distinct nodes x and y in f such that $\lambda(x, y; f) = \lambda(f)$ and x and y are contracted together in the α-contraction. By Lemma 3.15, $\lambda(x, y; f) \geq \alpha = \frac{1}{2+\epsilon}\delta$ which implies that $\delta \leq (2+\epsilon)\lambda(f)$. Since the algorithm returns $\min(\delta, \lambda')$ we have that the output is no more than $(2+\epsilon)\lambda(f)$. □

Since $\delta(f)$ can be much larger than $\lambda(f)$, we can preprocess the function to reduce δ to at most $n\lambda(f)$ to obtain a strongly polynomial run time.

Lemma 3.16. *Let $\beta = \min_{i>1} d_f(V_{i-1}, v_i)$ for a given Queyranne ordering $v_1, \ldots, v_n$ of a non-negative symmetric submodular function f. Then $\beta \leq \lambda(f) \leq n\beta$.*

Proof. From Lemma 3.15, $\lambda_f(u, v) \geq \beta$ for all $u, v \in V$ because V is a β-tight set. Therefore $\lambda(f) \geq \beta$. Let $i^* = \arg\min_{i>1} d_f(V_{i-1}, v_i)$. Then,

$$\lambda(f) \leq f(V_{i^*-1}) = d_f(V_{i^*-1}, V \setminus V_{i^*-1}) \leq \sum_{j=i^*}^{n} d_f(V_{i^*-1}, v_j) \leq (n+1-i^*)\beta \leq n\beta. \quad (3.35)$$

□

Let β be the value in Lemma 3.16, then a $2n\beta$-contraction of f yields a nice function f' where $\sigma(f') = O(n^2\beta)$. This also implies that $\delta(f') = O(n^2\beta)$. Applying the $(2+\epsilon)$ approximation algorithm to f' gives us the following corollary.

Corollary 3.3. *A $(2+\epsilon)$ approximation for a nice function can be computed in $O(\epsilon^{-1}n^2 \log n)$ oracle calls.*

We now specialize the algorithm to hypergraphs. The dominating term in each iteration is the time to compute a Queyranne ordering, which takes $O(p + n \log n)$ time for capacitated hypergraphs, and $O(p)$ time for uncapacitated hypergraphs. The number of recursive calls is $O(\frac{1}{\varepsilon} \log n)$.

Corollary 3.4. *A $(2+\epsilon)$ approximation for hypergraph min-cut can be computed in $O(\epsilon^{-1}(p + n \log n) \log n)$ time for capacitated hypergraphs, and in $O(\epsilon^{-1}p)$ time for uncapacitated hypergraphs.*

The analysis assumed that f is a normalized symmetric submodular function. Suppose f is not normalized but has subadditive connectivity. We find a $(2+\epsilon)$ approximate minimizer of the normalization f' (the function f' defined by $f'(S) = f(S) - f(\emptyset)$) instead. It can be seen that the output is also a $(2+\epsilon)$ approximate minimizer of f.

3.6 STRENGTH ESTIMATION AND CUT SPARSIFIERS

The goal of this section describe a fast algorithm for computing approximate strengths of edges of a capacitated hypergraph. These estimates can be used to do importance sampling and obtain a cut sparsifier as shown in [24].

The *strength* of an edge e, denoted by $\gamma_H(e)$, is defined as $\max_{e \subseteq U \subseteq V} \lambda(H[U])$; in other words the largest connectivity of a node induced subhypergraph that contains e. We also define the cost $\zeta_H(e)$ of e as the inverse of the strength; that is $\zeta_H(e) = 1/\gamma_H(e)$. We drop the subscript H if there is no confusion regarding the hypergraph. The preceding definitions generalize easily to capacitated hypergraphs. The strength of an edge e is the maximum min-cut value over all node induced subhypergraphs that contain e.

The main technical result of this section is the following.

Theorem 3.17. *Let $H = (V, E)$ be a capacitated hypergraph on n nodes. There is an efficient algorithm that computes for each edge e an approximate strength $\gamma'(e)$ such that the following properties are satisfied:*

1. *lower bound property: $\gamma'(e) \leq \gamma_H(e)$ and*
2. *α-cost property: $\sum_{e \in E} \frac{1}{\gamma'(e)} \leq \alpha \cdot (n-1)$ where $\alpha = O(r)$.*

For uncapacitated hypergraphs the running time of the algorithm is $O(p \log^2 n)$ and for capacitated hypergraphs the running time is $O(p \log p \log^2 n)$.

We refer to estimates that satisfy the properties of the preceding theorem as α-approximate strengths. One idea that comes to mind in finding $O(r)$-approximate strengths is to convert the given hypergraph into a graph and compute approximate strengths in the graph. For example, we could replace each hyperedge with a clique, or a star spanning the nodes contained in the edge. The minimum strength of the replaced edges might give us a $O(\alpha)$-approximation. Unfortunately, this will not work even when then rank is 3. Consider the following hypergraph H with nodes $\{v_1, \ldots, v_n\}$. There is an edge $e = \{v_1, v_2\}$, and edge $\{v_1, v_2, v_i\}$ for all $3 \leq i \leq n$. The strength of e in H is 1. Let G be a graph where each $\{v_1, v_2, v_i\}$ in H is replaced with a star centered at v_1 and spans $\{v_1, v_2, v_i\}$. The strength of e in G is $n-1$. This bound also holds if each hyperedge $\{v_1, v_2, v_i\}$ is replaced by a clique.

The proof of the preceding theorem closely follows the corresponding theorem for graphs from [23]. A key technical tool for graphs is the deterministic sparsification algorithm for edge-connectivity due to Nagamochi and Ibaraki [9]. Here we rely on the generalization to hypergraphs from Section 3.3. Before we describe the proof we discuss some applications in the next subsection.

3.6.1 Applications

A capacitated hypergraph $H' = (V, E')$ is a $(1 \pm \epsilon)$-cut-sparsifier of a capacitated hypergraph $H = (V, E)$, if for every $S \subseteq V$, the cut value of S in H' is within $(1 \pm \epsilon)$ factor of the cut value of S in H. Below we state formally the sampling theorem from [24].

Theorem 3.18 ([24]). *Let H be a rank r capacitated hypergraph where edge e has capacity $c(e)$. Let H_ϵ be a hypergraph obtained by independently sampling each edge e in H with probability $p_e = \min\left(\frac{3((d+2)\ln n + r)}{\gamma_H(e)\epsilon^2}, 1\right)$ and capacity $c(e)/p_e$ if sampled. With probability at least $1 - O(n^{-d})$, the hypergraph H_ϵ has $O(n(r+\log n)/\epsilon^2)$ edges and is a $(1\pm\epsilon)$-cut-sparsifier of H.*

The expected capacity of each edge in H_ϵ is the capacity of the edge in H. It is not difficult to prove that c-approximate strength also suffices to get a $(1 \pm \epsilon)$-cut-sparsifier. That is, if we replace γ with α-approximate strength γ', the sampling algorithm will still output a $(1\pm\epsilon)$-cut-sparsifier of H. Indeed, the lower bound property shows each edge will be sampled with a higher probability, therefore the probability of obtaining a $(1 \pm \varepsilon)$-sparsifier increases. However, the number of edges in the sparsifier increases. The α-cost property shows the hypergraph H_ϵ will have $O(\alpha n(r + \log n)/\epsilon^2)$ edges.

From Theorem 3.17, we can find an $O(r)$-approximate strength function γ' in $O(p \log^2 n \log p)$ time.

Corollary 3.5. *A $(1 \pm \epsilon)$-cut-sparsifier of H with $O(nr(r + \log n)/\epsilon^2)$ edges can be found in $O(p \log^2 n \log p)$ time with high probability.*

The number of edges in the $(1 \pm \epsilon)$-cut-sparsifier is worse than Theorem 3.18 by a factor of r. It is an open problem whether the extra factor of r can be removed while maintaining the near-linear running time.

As is the case for graphs, cut sparsification allows for faster approximation algorithms for cut problems. We mention two below.

min-cut: The best running time known currently to compute a (global) min-cut in a hypergraph is $O(pn)$ [69, 70]. We already described how to obtain a $(2+\varepsilon)$-approximation in $\tilde{O}(p/\varepsilon)$ time [7]. Via Corollary 3.5 we can first sparsify the hypergraph and then apply the $O(pn)$ time algorithm to the sparsified graph. This gives a randomized algorithm that outputs a $(1+\varepsilon)$-approximate min-cut in a rank r hypergraph in $O(n^2r^2(r+\log n)/\varepsilon^2 + p\log^2 n \log p)$ time and works with high probability. For small r the running time is $\tilde{O}(p + n^2/\varepsilon^2)$ for a $(1+\varepsilon)$-approximation.

min-*st*-cut: The standard technique to compute a min-*st*-cut in a hypergraph is computing a *st* maximum flow in an associated capacitated digraph with $O(n+m)$ nodes and $O(p)$ edges [80]. A $(1-\varepsilon)$ approximation algorithm of *st* maximum flow in such graph can be found in $\tilde{O}(p\sqrt{n+m})$ time [89]. Via sparsification Corollary 3.5, we can obtain a randomized algorithm to find a $(1+\varepsilon)$-approximate min-*st*-cut in a rank r hypergraph in $\tilde{O}(n^{3/2}r^2(r+\log n)^{3/2}/\varepsilon^3 + p)$ time. For small r the running time is $\tilde{O}(p + n^{3/2}/\varepsilon^3)$ for a $(1+\varepsilon)$-approximation.

3.6.2 Properties of edge strengths

A *k-strong component* of H is a inclusion-wise maximal set of nodes $U \subseteq V$ such that $H[U]$ is k-edge-connected. An edge e is *k-strong* if $\gamma(e) \geq k$, otherwise e is *k-weak*. $\kappa(H)$ is the number of components of a hypergraph H.

Proposition 3.1. *Deleting an edge does not increase the strength of any remaining edge. Contracting an edge does not decrease the strength of any remaining edge.*

Lemma 3.17. *Let $H = (V, E)$ be a hypergraph. If an edge $e \in E$ crosses a cut of value k, then $\gamma_H(e) \leq k$.*

Proof. Suppose S is a cut such that $|\delta_H(S)| \leq k$ and $e \in \delta_H(S)$. Consider any $U \subseteq V$ that contains e as a subset and let $H' = H[U]$. It is easy to see that $|\delta_{H'}(S \cap U)| \leq k$ and hence $\lambda(H') \leq k$. Therefore, $\gamma(e) = \max_{e \subseteq U \subseteq V} \lambda(H[U]) \leq k$. □

Lemma 3.18 (Extends Lemma 4.5,4.6 [23])**.** *Let e, e' be distinct edges in a hypergraph $H = (V, E)$.*

1. *If $\gamma_H(e) \geq k$ and e' is a k-weak edge then $\gamma_H(e) = \gamma_{H'}(e)$ where $H' = H \setminus e'$.*
2. *Suppose $\gamma_H(e) < k$ (that is e is a k-weak edge) and e' is a k-strong edge. Let $H' = H/e'$ obtained by contracting e'. If f is the corresponding edge of e in H' then $\gamma_H(e) = \gamma_{H'}(f)$.*

In short, contracting a k-strong edge does not increase the strength of a k-weak edge and deleting a k-weak edge does not decrease the strength of a k-strong edge.

Proof. We prove the two claims separately.

Deleting a k-weak edge: Since e is k-strong in H there is $U \subseteq V$ such that $\lambda(H[U]) \geq k$ and $e \subseteq U$. If $e' \subseteq U$, e' would be a k-strong edge. Since e' is k-weak, e' does not belong to $H[U]$. Thus $H'[U] = H[U]$ which implies that $\gamma_{H'}(e) \geq k$.

Contracting a k-strong edge: Let $H' = H/e'$ where e' is a k-strong edge in H. Let $v_{e'}$ be the node in H' obtained by contracting e'. Let $U' \subseteq V(H')$ be the set that certifies the strength of f, that is $\gamma_{H'}(f) = \lambda(H'[U'])$ and $f \subseteq U'$. If $v_{e'} \notin U'$ then it is easy to see that $U' \subseteq V(H)$ and $H[U] = H'[U']$ which would imply that $\gamma_{H'}(f) = \gamma_H(e)$. Thus, we can assume that $v_{e'} \in U'$. Let U be the set of nodes in $V(H)$ obtained by uncontracting $v_{e'}$. We claim that $\lambda(H[U]) \geq \lambda(H'[U'])$. If this is the case we would have $\gamma_H(e) \geq \lambda(H[U]) \geq \lambda(H'[U']) = \gamma_{H'}(f)$. We now prove the claim. Let W be the set that certifies the strength of e', namely $\gamma_H(e') = \lambda(H[W])$. Consider any cut $S \subseteq U \cup W$ in $H[U \cup W]$. If S crosses W or strictly contained in W, then S has cut value at least $\lambda(H[W])$. Otherwise, $S \subseteq U \setminus W$ or $W \subseteq S$. By symmetry, we only consider $S \subseteq U \setminus W$. $S \neq U$ because $e' \subseteq U \cap W$. S is also a cut in $H'[U']$, and $|\delta_{H[U]}(S)| = |\delta_{H'[U']}(S)|$. This shows that

$$\lambda(H[U \cup W]) \geq \min(\lambda(H'[U']), \lambda(H[W])) \tag{3.36}$$

Because $\lambda(H[W]) > \lambda(H[U])$, and $\lambda(H[U]) \geq \lambda(H[U \cup W])$, we arrive at $\lambda(H[U]) \geq \lambda(H'[U'])$. □

Theorem 3.19 (Extends Lemma 4.10 [23])**.** *Consider a connected uncapacitated hypergraph $H = (V, E)$. A capacitated hypergraph $H' = (V, E)$ with capacity $\zeta_H(e)$ on each edge e has minimum cut value* 1*.*

Proof. Consider a cut S of value k in H; that is $|\delta_H(S)| = k$. For each edge $e \in \delta(S)$, $\gamma_H(e) \leq k$ by Lemma 3.17. Therefore e has capacity at least $1/k$ in H'. It follows the cut value of S in H' is at least $k \cdot 1/k \geq 1$. Thus, the min-cut of H' is at least 1.

Let S be a min-cut in H whose value is k^*. It is easy to see that for each $e \in \delta_H(S)$, $\gamma_H(e) = k^*$. Thus the value of the cut S in H' is exactly $k^* \cdot 1/k^* = 1$. □

Lemma 3.19 (Extends Lemma 4.11 [23])**.** *For a uncapacitated hypergraph $H = (V, E)$ with at least* 1 *node,*

$$\sum_{e \in E} \zeta(e) \leq n - \kappa(H) \tag{3.37}$$

Proof. Let $t = n - \kappa(H)$. We prove the theorem by induction on t.

For the base case, if $t = 0$, then H has no edges and therefore the sum is 0.

Otherwise, let $t > 0$. Let U be a connected component of H with at least 2 nodes. There exists a cut X of cost 1 in $H[U]$ by Theorem 3.19.

Let $H' = H - \delta(S)$. H' has at least one more connected component than H. By Proposition 3.1, for each $e \in E(H')$ $\gamma_H(e) \geq \gamma_{H'}(e)$; hence $\zeta_{H'}(e) \geq \zeta_H(e)$. By the inductive hypothesis, $\sum_{e \in E(H')} \zeta_{H'}(e) \leq (n - \kappa(H')) \leq t - 1$. The edges in H are exactly $\delta(X) \cup E(H')$. By the same argument as in the preceding lemma, $\sum_{e \in \delta(X)} \zeta_H(e) = 1$. Therefore,

$$\sum_{e \in E(H)} \zeta_H(e) = \sum_{e \in \delta(X)} \zeta_H(e) + \sum_{e \in E(H')} \zeta_H(e) \tag{3.38}$$

$$= 1 + \sum_{e \in E(H')} \zeta_H(e) \tag{3.39}$$

$$\leq 1 + \sum_{e \in E(H')} \zeta_{H'}(e) \tag{3.40}$$

$$\leq 1 + (t - 1) = t. \tag{3.41}$$

□

Corollary 3.6. *The number of k-weak edges in an uncapacitated hypergraph H on n nodes is at most $k(n - \kappa(H))$.*

Lemma 3.20. *The k-strong components are pairwise disjoint.*

Proof. Consider k-strong components A and B. Assume $A \cap B \neq \emptyset$, then $\lambda(G[A \cup B]) \geq \min(\lambda(G[A]), \lambda(G[B])) \geq k$ using triangle inequality for connectivity. This shows $A = A \cup B = B$ by maximality of A and B. □

3.6.3 Estimating strengths in uncapacitated hypergraphs

In this section, we consider uncapacitated hypergraphs and describe a near-linear time algorithm to estimate the strengths of all edges as stated in Theorem 3.17. Let $H = (V, E)$ be the given hypergraph. The high-level idea is simple. We assume that there is a fast algorithm WeakEdges(H, k) that returns a set of edges $E' \subseteq E$ such that E' contains all the k-weak edge in E; the important aspect here is that the output may contain some edges which are *not* k-weak, however, the algorithm should not output too many such edges (this will be quantified later).

The estimation algorithm is defined in Figure 3.4. The algorithm repeatedly calls WEAKEDGES(H, k) for increasing values of k while removing the edges found in previous iterations.

```
ESTIMATION(H)
  Compute a number k such that γ_H(e) ≥ k for all e ∈ E(H)
  H_0 ← H, i ← 1
  while there are edges in H_{i-1}
      F_i ← WEAKEDGES(H_{i-1}, 2^i k)
      for e ∈ F_i
          γ'(e) ← 2^{i-1} k
      H_i ← H_{i-1} − F_i
      i ← i + 1
  return γ'
```

Figure 3.4: The estimation algorithm

Lemma 3.21. *Let $H = (V, E)$ be an uncapacitated hypergraph. Then,*

1. *For each $e \in F_i$, $\gamma(e) \geq 2^{i-1}k$. That is, the strength of all edges deleted in iteration i is at least $2^{i-1}k$.*

2. *For each $e \in E$, $\gamma'(e) \leq \gamma(e)$.*

Proof. $\gamma_{H_i}(e) \leq \gamma_H(e)$ for all i and $e \in E(H_i)$ because deleting edges cannot increase strength. Let E_i denote the set of edges in H_i with $E_0 = E$.

We prove that $2^i k \leq \gamma_{H_i}(e)$ for all $e \in E_i$ by induction on i. If $i = 0$, then $k \leq \gamma_{H_0}(e)$ for all $e \in E_0$. Now we assume $i > 0$. At end of iteration $(i-1)$, by induction, we have that $\gamma_{H_{i-1}}(e) \geq 2^{i-1}k$ for each $e \in E_{i-1}$. In iteration i, F_i contains all $2^i k$-weak edges in the graph H_{i-1}. We hav $E_i = E_{i-1} - F_i$. Thus, for any edge $e \in E_i$, $\gamma_H(e) \geq \gamma_{H_{i-1}}(e) \geq 2^i k$. This proves the claim.

Since $F_i \subseteq E_{i-1}$, it follows from the previous claim that $\gamma_H(e) \geq 2^{i-1}k$ for all $e \in F_i$. Since the F_i form a partition of E, we have $\gamma'(e) \leq \gamma_H(e)$ for all $e \in E$. □

Note that in principle WEAKEDGES(H, k) could output all the edges of the graph for any $k \geq \lambda(H)$. This would result in a high cost for the resulting strength estimate. Thus, we need some additional properties on the output of the procedure WEAKEDGES(H, k). Let $H = (V, E)$ be a hypergraph. A set of edges $E' \subseteq E$ is called *ℓ-light* if $|E'| \leq \ell(\kappa(H - E') - \kappa(H))$. Intuitively, on average, we remove ℓ edges in E' to increase the number of components of H by 1.

Lemma 3.22. *If* WeakEdges(H, k) *outputs a set of* αk*-light edges for all* k*, then the output* γ' *of the algorithm* Estimation(H) *satisfies the* 2α*-cost property. That is,* $\sum_{e \in E} \frac{1}{\gamma'(e)} \leq 2\alpha(n-1)$.

Proof. From the description of Estimation(H), and using the fact that the edges sets $F_1, F_2, \ldots,$ partition E, we have $\sum_{e \in E} \frac{1}{\gamma'(e)} = \sum_{i \geq 1} |F_i| \frac{1}{2^{i-1}k}$.

F_i is the output of WeakEdges$(H_{i-1}, 2^i k)$. From the lightness property we assumed, $|F_i| \leq \alpha 2^i k(\kappa(H_i) - \kappa(H_{i-1}))$. Combining this with the preceding equality,

$$\sum_{e \in E} \frac{1}{\gamma'(e)} = \sum_{i \geq 1} |F_i| \frac{1}{2^{i-1}k} \leq \sum_{i \geq 1} 2\alpha(\kappa(H_i) - \kappa(H_{i-1})) \leq 2\alpha(n-1). \tag{3.42}$$

□

Implementing WeakEdges

We now show describe an implementation of WeakEdges(H, k) that outputs a $4rk$-light set.

Let $H = (V, E)$ be a hypergraph. An edge e is *k-crisp* with respect to H if it crosses a cut of value less than k. In other words, there is a cut X, such that $e \in \delta(X)$ and $|\delta(X)| < k$. Note that any k-crisp edge is k-weak. A set of edges $E' \subseteq E$ is a *k-partition*, if E' contains all the k-crisp edges in H. A k-partition may contain non-k-crisp edges.

We will assume access to a subroutine Partition(H, k) that given H and integer k, it finds a $2k$-light k-partition of H. We will show how to implement Partition later. See Figure 3.5 for the implementation of WeakEdges.

```
WeakEdges(H, k)
  E' ← ∅
  repeat 1 + log_2 n times:
      E' ← E' ∪ Partition(H, 2rk)
      H ← H − E'
  return E'
```

Figure 3.5: Algorithm for returning a $4rk$-light set of all k-weak edges in H.

Theorem 3.20. WeakEdges(H, k) *returns a* $4rk$*-light set* E' *such that* E' *contains all the* k*-weak edges of* H *with* $O(\log n)$ *calls to* Partition.

Proof. First, we assume H has no k-strong component with more than 1 node and that H is connected. Then all edges are k-weak and the number of k-weak edges in H is at most

$k(n-1)$ by Corollary 3.6. It also implies that $\sum_v \deg(v) \leq rk(n-1)$. By Markov's inequality at least half the nodes have degree less than $2rk$. For any node v with degree less than $2rk$, all edges incident to it are $2rk$-crisp. Thus, after the first iteration, all such nodes become isolated since PARTITION$(H, 2rk)$ contains all $2rk$-crisp edges. If H is not connected then we can apply this same argument to each connected component and deduce that at least half of the nodes in each component will be isolated in the first iteration Therefore, in $\log n$ iterations, all nodes become isolated. Hence WEAKEDGES(H, k) returns all the edges of H.

Now consider the general case when H may have k-strong components. We can apply the same argument as above to the hypergraph obtained by contracting each k-strong component into a single node. The is well defined because the k-strong components are disjoint by Lemma 3.20.

Let the edges removed in the ith iteration to be E_i, and the hypergraph before the edge removal to be H_i. So $H_{i+1} = H_i - E_i$. Recall that PARTITION$(H_i, 2rk)$ returns a $4rk$-light set. Hence we know $|E_i| \leq 4rk(\kappa(H_{i+1}) - \kappa(H_i))$.

$$|E'| = \sum_{i \geq 1} |E_i| \leq \sum_{i \geq 1} 4rk(\kappa(H_{i+1}) - \kappa(H_i)) = 4rk(\kappa(H - E') - \kappa(H)) \tag{3.43}$$

This shows E' is $4rk$-light. □

It remains to implement PARTITION(H, k) that returns a $2k$-light k-partition. To do this, we use k-sparse certificates. Recall by Theorem 3.7, we can obtain a k-sparse certificate in $O(p)$ time.

A k-sparse certificate E' is certainly a k-partition. However, E' may contain too many edges to be $2k$-light. Thus, we would like to find a smaller subset of E'. Note that every k-crisp edge must be in a k-sparse certificate and hence no edge in $E \setminus E'$ can be k-crisp. Hence we will contract the edges in $E \setminus E'$, and find a k-sparse certificate in the hypergraph after the contraction. We repeat the process until eventually we reach a $2k$-light set. See Figure 3.6 for the formal description of the algorithm.

Theorem 3.21. PARTITION*(H, k) outputs a $2k$-light k-partition in $O(p \log n)$ time.*

Proof. If the algorithm either returns all the edges of the graph H in the first step then it is easy to see that the output is a $2k$-light k-partition since the algorithm explicitly checks for the lightness condition.

Otherwise let E' a k-sparse certificate of H. As we argued earlier, $E - E'$ contains no k-crisp edges and hence contracting them is safe. Moreover, all the original k-crisp edges remain k-crisp after the contraction. Since the algorithm recurses on the new graph, this establishes the correctness of the output.

```
PARTITION(H, k)
  if number of edges in H ≤ 2k(n − κ(H))
      E′ ←edges in H
      return E′
  else
      E′ ← k-sparse certificate of H
      H ← contract all edges of H − E′
      return PARTITION(H, k)
```

Figure 3.6: Algorithm for returning a $2k$-light k-partition.

We now argue for termination and running time by showing that the number of nodes halves in each recursive call. Assume H contains n nodes, the algorithm finds a k-sparse certificate and contracts all edges not in the certificate. The resulting hypergraph has n' nodes and m' edges. We have $m' \leq k(n-1)$ by Theorem 3.7. If $n' - 1 \leq (n-1)/2$, then the number of nodes halved. Otherwise $n' - 1 > (n-1)/2$, then the number of edges $m' \leq k(n-1) < 2k(n'-1)$, and the algorithm terminates in the next recursive call.

The running time of the algorithm for a size p hypergraph with n nodes is $T(p,n)$. $T(p,n)$ satisfies the recurrence $T(p,n) = O(p) + T(p,n/2) = O(p \log n)$. □

Putting things together, ESTIMATION(H) finds the desired $O(r)$-approximate strength.

Theorem 3.22. *Let $H = (V,E)$ be a uncapacitated hypergraph. The output γ' of* ESTIMATION(H) *is a $O(r)$-approximate strength function of H.* ESTIMATION(H) *can be implemented in $O(p \log^2 n \log p)$ time.*

Proof. Combining Lemma 3.21, Lemma 3.22 and Theorem 3.20, we get the output γ' of ESTIMATION(H) is a $O(r)$-approximate strength function. The maximum strength in the graph is at most p, all edges with be removed at the $(1 + \log p)$th iteration of the while loop. In each iteration, there is one call to WEAKEDGES. Each call of WEAKEDGES takes $O(p \log^2 n)$ time by combining Theorem 3.21 and Theorem 3.20. The step outside the while loop takes linear time, since we can set k to be 1 as a lower bound of the strength. Hence overall, the running time is $O(p \log^2 n \log p)$. □

3.6.4 Estimating strengths in capacitated hypergraphs

Consider a capacitated hypergraph $H = (V,E)$ with an associated capacity function $c : E \to \mathbb{N}_+$; that is, we assume all capacities are non-negative integers. For proving correctness, we consider an uncapacitated hypergraph H' that simulates H. Let H' contain

$c(e)$ copies of edge e for every edge e in H; one can see that the strength of each of the copies of e in H' is the same as the strength of e in H. Thus, it suffices to compute strengths of edges in H'. We can apply the correctness proofs from the previous sections to H'. In the remainder of the section we will only be concerned with the running time issue since we do not wish to explicitly create H'. We say H is the implicit representation of H'.

By Theorem 3.7, we can find k-sparse certificate E' of H' such that $|E'| \leq k(n-1)$, in $O(p + n \log n)$ time, where p is the number of edges in the implicit representation.

The remaining operations in PARTITION and WEAKEDGES consist only of adding edges, deleting edges and contracting edges. These operations take time only depending on the size of the implicit representation. Therefore the running time in Theorem 3.21 and Theorem 3.20 still holds for capacitated hypergraph.

Lemma 3.23. *Suppose we are given a hypergraph $H = (V, E)$ and a lower bound b on the strengths of the edges. If the total capacity of all edges is at most bM, then* ESTIMATION(H) *can be implemented in $O(p \log^2 n \log M)$ time.*

Proof. Because b is a lower bound of the strength, we can set k to be b in the first step. The maximum strength in the graph is at most bM, all edges will be removed at the $(1+\log M)$th iteration of the while loop. Each iteration calls WEAKEDGES once, hence the running time is $O(p \log^2 n \log M)$. □

The running time in Lemma 3.23 can be improved to strongly polynomial time by using the windowing technique [23].

Assume we have disjoint intervals $I_1, \ldots, I_t$ where for every $e \in E$, $\gamma(e) \in I_i$ for some i. In addition, assume $I_i = [a_i, b_i]$, $b_i \leq p^2 a_i$ and $b_i \leq a_{i+1}$ for all i. We can essentially apply the estimation algorithm to edges with strength inside each interval. Let E_i be the set of edges whose strengh lies in interval I_i. Indeed, let H_i to be the graph obtained from H by contracting all edges in E_j where $j > i$, and deleting all edges in $E_{j'}$ where $j' < i$. For edge $e \in E_i$ let e' be its corresponding edge in H_i. From Lemma 3.18, $\gamma_{H_i}(e') = \gamma_H(e)$. The total capacity of H_i is at most $pb_i \leq p^3 a_i$. We can run ESTIMATION(H_i) to estimate γ_{H_i} since the ratio between the lower bound a_i and upper bound b_i is p^3. Let p_i be the size of H_i, and n_i be the number of nodes in H_i, the running time for ESTIMATION(H_i) is $O(p_i \log^2 n_i \log p_i)$ by Lemma 3.23. The total running time of ESTIMATION over all H_i is $O(\sum_i p_i \log^2 n_i \log p_i) = O(p \log^2 n \log p)$. Constructing H_i from H_{i+1} takes $O(p_i + p_{i+1})$ time: contract all edges in H_{i+1} and then add all the edges e_i where $\gamma(e) \in I_i$. Therefore we can construct all $H_1, \ldots, H_t$ in $O(p)$ time.

It remains to find the intervals $I_1, \ldots, I_t$. For each edge e, we first find values d_e, such

that $d_e \leq \gamma(e) \leq pd_e$. The maximal intervals in $\bigcup_{e \in E}[d_e, pd_e]$ are the desired intervals. We now describe the procedure to find the values d_e for all $e \in E$.

Definition 3.3. *The star approximate graph $A(H)$ of H is a capacitated graph obtained by replacing each hyperedge e in H with a star S_e, where the center of the star is an arbitrary node in e, and the star spans each node in e. Every edge in S_e has capacity equal to the capacity of e.*

It is important that $A(H)$ is a multigraph: parallel edges are distinguished by which hyperedge it came from. We define a correspondence between the edges in $A(H)$ and H by a function π. For an edge e' in $A(H)$, $\pi(e') = e$ if $e' \in S_e$. Let T be a maximum capacity spanning tree in $A(H)$. For $e \in E$, define T_e to be the minimal subtree of T that contains all nodes in e. Note that all the leaves of T_e are nodes from e.

For any two nodes u and v, we define d_{uv} to be the capacity of the minimum capacity edge in the unique u-v-path in T. For each edge $e \in E$ we let $d_e = \min_{u,v \in e} d_{uv}$. We will show d_e satisfies the property that $d_e \leq \gamma(e) \leq pd_e$.

Let $V_e = \bigcup_{e' \in T_e} \pi(e')$. Certainly, $\gamma(e) \geq \lambda(H[V_e])$, because all nodes of e are contained in V_e. $\lambda(H[V_e]) \geq d_e$ because every cut in $H[V_e]$ has to cross some $\pi(e')$ where $e' \in T_e$, and the capacity of $\pi(e')$ is at least d_e. Hence $\gamma(e) \geq d_e$.

We claim if we remove all edges in H with capacity at most d_e, then it will disconnect some $s, t \in e$. If the claim is true then e crosses a cut of value at most pd_e. By Lemma 3.17, $\gamma(e) \leq pd_e$. Assume that the claim is not true. Then, in the graph $A(H)$ we can remove all edges with capacity at most d_e and the nodes in e will still be connected. We can assume without loss of generality that the maximum capacity spanning tree T in $A(H)$ is computed using the greedy Kruskal's algorithm [90]. This implies that T_e will contain only edges with capacity strictly greater than d_e. This contradicts the definition of d_e.

$A(H)$ can be constructed in $O(p)$ time. The maximum spanning tree T can be found in $O(p + n \log n)$ time. We can construct a data structure on T in $O(n)$ time, such that for any $u, v \in V$, it returns d_{uv} in $O(1)$ time. [91] To compute d_e, we fix some node v in e, and compute $d_e = \min_{u \in e, v \neq u} d_{uv}$ using the data structure in $O(|e|)$ time. Computing d_e for all e takes in $O(\sum_{e \in E} |e|) = O(p)$ time. The total running time is $O(p + n \log n)$. We conclude the following theorem.

Lemma 3.24. *Given a capacitated hypergraph H, we can find a value d_e for each edge e, such that $d_e \leq \gamma(e) \leq pd_e$ in $O(p + n \log n)$ time.*

The preceding lemma gives us the desired intervals. Using Lemma 3.23, we have the desired theorem.

Theorem 3.23. *Given a rank r capacitated hypergraph H with capacity function c, in $O(p \log^2 n \log p)$ time, one can find a $O(r)$-approximate strength function of H.*

3.7 OPEN PROBLEMS

We close with some open problems. The main one is to find an algorithm for hypergraph min-cut that is faster than the current one that runs in $O(np + n^2 \log n)$ time. We do not know a better deterministic run-time even when specialized to graphs. However we have a randomized near-linear time algorithm for graphs [46]. Can Karger's algorithm be extended to hypergraphs with fixed rank r? Recently there have been several fast st max-flow algorithms for graphs and digraphs. The algorithms for digraphs [89, 92] have straight forward implications for hypergraph st-cut computation via the equivalent digraph. However, hypergraphs have additional structure and it may be possible to find faster (approximate) algorithms.

We described a linear time algorithm to find a maximum flow between the last two nodes of a tight-ordering of a hypergraph (the flow is in the equivalent digraph of the hypergraph). We believe that such a linear time algorithm is also feasible for the last two nodes of an MA-ordering of a hypergraph.

We obtained a fast algorithm that outputs $O(r)$-approximate strengths of a hypergraph. Can we compute $O(1)$-approximate strengths in near linear time? This would allow us to avoid the extra factor of r in Corollary 3.5. Kogan and Krauthgamer [24] given an upper bound on the sparsifier in terms of edges. This translates to an upper bound on the representation size but when r is large it may not be tight. Can one prove lower bounds on the number of edges or the representation size of a $(1 + \varepsilon)$-cut sparsifier for hypergraphs?

Chapter 4: Hypergraph k-cut and constant span hedge k-cut

The hypergraph k-cut problem is the following:

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 node set into k parts with minimum number of hyperedges crossing the partition. This is an extension of the classic hypergraph min-cut problem studied in the previous chapter. Hypergraph partitioning problems were discussed as early as in 1973 by Lawler [80] and have several applications including clustering in VLSI design and network reliability (e.g., see [1, 2, 27, 93]).

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 been investigated thoroughly in the literature. When k is part of the input, Goldschmidt and Hochbaum showed that Graph-k-Cut is **NP-Hard** [27] while Saran and Vazirani designed a 2-approximation algorithm [94]. 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 nodes in the input graph [27]. Karger and Stein [29] designed a randomized algorithm for Graph-k-Cut 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 run-time for solving Graph-k-Cut has been improved over a series of works [30, 31, 95] with the current best run-time being $\tilde{O}(n^{2k})$ due to Thorup [28].

The complexity of Hypergraph-k-Cut has remained an intriguing open problem since the works of Goldschmidt and Hochbaum (1994) and Saran and Vazirani (1995). As we have seen in the previous chapter, the case of $k = 2$, denoted Hypergraph-2-Cut, is well-known to admit deterministic polynomial time algorithms [69, 70, 80]. When k is part of the input, Hypergraph-k-Cut is **NP-Hard** as observed from Graph-k-Cut. Chekuri and Li [26] 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 [96]. Chekuri and 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 [97] gave a polynomial time algorithm for Hypergraph-k-Cut in constant rank hypergraphs. A randomized polynomial time algorithm for Hypergraph-k-Cut in constant rank hypergraphs for constant k can also be obtained using the *uniform random contraction* technique of Karger and Stein [29] as illustrated by Kogan and Krauthgamer [24]. Moving to arbitrary rank hypergraphs, Xiao [93] showed a non-crossing structural property of an optimal solution and used it to to design a polynomial time algorithm for Hypergraph-3-Cut. Okumoto, Fukunaga and Nagamochi [98] reduced Hypergraph-k-Cut for constant k to the node-capacitated 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 $k \geq 4$.

We present a randomized polynomial time algorithm to solve Hypergraph-k-Cut for constant k in arbitrary rank hypergraphs. This is the first polynomial time algorithm for Hypergraph-k-Cut for constant k.

In the $s-t$ hedge cut problem (abbreviated st-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 st-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 st-Hedge-Cut is **NP-Hard** [39], even if each hedge consists of exactly two edges [40], while Ghaffari et al. showed that Hedge-2-Cut admits a randomized polynomial time approximation scheme [3]. Ghaffari et al. [3] 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 nodes incident to the edges in the hedge) and is hence solvable efficiently. The complexity of Hedge-2-Cut for constant span hedgegraphs

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

was raised as an open problem by Coudert et al. [99]. We generalize the 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:

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 node set into k parts with minimum number of hedges crossing the partition . We show that HEDGE-k-CUT for hedgegraphs with constant span is tractable for constant k.

We next generalize the ideas behind the randomized polynomial time approximation scheme for HEDGE-2-CUT by Ghaffari, Karger and Panigrahi [3] to obtain a randomized polynomial time approximation scheme for HEDGE-k-CUT for constant k (for all input hedgegraphs irrespective of their span). The algorithms and analysis also lead to combinatorial bounds on the number of optimal solutions.

4.1 RESULTS

Recall p represents the input-size of the hedgegraphs.

Theorem 4.1. *For every non-negative constant integer s, there exists a randomized polynomial time algorithm to solve* HEDGE-k-CUT *in hedgegraphs with span at most s that runs in time $O(mpn^{ks+k-s}\log n)$ and succeeds with probability at least $1-1/n$.*

For an input hypergraph, let n denote the number of nodes and let p denote the sum of the cardinality of the hyperedges. We show a reduction from HYPERGRAPH-k-CUT to HEDGE-k-CUT in 1-span hedgegraphs. This reduction in conjunction with Theorem 4.1 immediately leads to a randomized polynomial time algorithm for HYPERGRAPH-k-CUT with run-time $O(mpn^{2k-1}\log n)$. We save a factor of m on this run-time by specializing the run-time analysis of the same algorithm that is used in Theorem 4.1 for hypergraphs.

Theorem 4.2. *There exists a randomized polynomial time algorithm to solve* HYPERGRAPH-k-CUT *that runs in time $O(pn^{2k-1}\log n)$ and succeeds with probability at least $1-1/n$.*

We mention that for the special case of $k=2$, namely HYPERGRAPH-2-CUT, Ghaffari et al. gave an algorithm based on random contractions [3]. 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, the contraction algorithm for HYPERGRAPH-2-CUT that follows from Theorem 4.2 *does not* require knowledge of the optimum cut value and is easy to implement. More importantly, it generalizes naturally to resolve the complexity of the more general problem of HYPERGRAPH-k-CUT.

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 *min-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 min-k-cut-sets:

Corollary 4.1. *The number of distinct minimum k-cut-sets in an n-node hypergraph is $O(n^{2(k-1)})$.*

As special cases, the bound stated in Corollary 4.1 recovers (i) the bound on the number of min-k-cut-sets in graphs by Karger and Stein [29] as well as (ii) the bound on the number of min-cut-sets in hypergraphs [3, 7].

Next, we generalize the techniques underlying the randomized polynomial time approximation scheme for HEDGE-2-CUT by Ghaffari, Karger and Panigrahi [3] to obtain a randomized polynomial time approximation scheme for HEDGE-k-CUT. In contrast to Theorem 4.1, 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 4.3. *For any given $\epsilon > 0$, there exists a randomized algorithm to find a $(1+\epsilon)$-approximate minimum hedge k-cut-set in time $pn^{O(\log(1/\epsilon))} \log n$ 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 4.3 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 for the algorithm mentioned in Theorem 4.3 is in the exponent 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 min-k-cut-sets in hedgegraphs.

Theorem 4.4. *The number of distinct minimum hedge k-cuts-set in an n-node hedgegraph with minimum hedge k-cut-set value λ is $n^{O(k+\log \lambda)}$.*

Organization. We present the preliminaries in Section 4.2, prove Theorems 4.1 and 4.2 in Section 4.3, and Theorems 4.3 and 4.4 in Section 4.4.

4.1.1 Related work

For Graph-k-Cut with k being a part of the input, Saran and Vazirani [94] designed a polynomial-time 2-approximation algorithm. Manurangsi [100] showed that there is no polynomial-time $(2-\epsilon)$-approximation for any constant $\epsilon > 0$ assuming the *Small Set Expansion Hypothesis* [101]. Gupta, Lee and Li [102] designed an algorithm that runs in time $O(2^{k^6}n^4)$ to achieve an approximation factor of $2-\delta$ for some constant $\delta > 0$.

Approximation algorithms for the hypergraph k-cut problem, the hypergraph k-partitioning problem, and more generally, submodular partitioning problems have been well-studied in the literature. The hypergraph k-partitioning problem is similar in flavor to the hypergraph k-cut problem but it minimizes a different objective. In the hypergraph k-partitioning problem, the input is a hypergraph and the goal is to find a partitioning of the node 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). The hypergraph k-partitioning problem and the hypergraph k-cut problem coincide to Graph-k-Cut when the input hypergraph is a graph. The hypergraph k-partitioning problem 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 $V_1, \ldots, V_k$ so that $\sum_{i=1}^{k} f(V_i)$ is minimum. Submodular k-partitioning for the case of $k = 3$ admits an efficient algorithm [98] while approximation algorithms have been designed for larger constants k [98, 103]. Submodular k-partitioning for $k = 4$ admits an efficient algorithm if the input function is symmetric [35].

We also mention that approximation algorithms are known for the variant of submodular partitioning that separates a given set of k elements [103, 104]. 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 $V_1, \ldots, V_k$ with $v_i \in V_i \ \forall \ i \in \{1, \ldots, k\}$ so that

$\sum_{i=1}^{k} f(V_i)$ is minimum. This generalizes the hypergraph k-way cut problem (where the goal is to delete the smallest number of hyperedges in order to disconnect a given collection of k nodes in the input hypergraph). The current-best known approximation factor 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 of link failures have been considered in the literature: a link could fail if either all or at least one of the resources that the link depends upon fails [99]. The 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, Karger and Panigrahi [3] who also showed that Hedge-2-Cut has a randomized polynomial time approximation scheme.

4.2 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 and $\binom{a}{b}$ is 0.

Let $G = (V, E)$ be a hedgegraph. For a hedge e, we use $r(e)$ to denote the number of nodes incident to the edges in e. 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 nodes in $G[e]$. Let $V(e)$ denote the nodes 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 *size* of a hedgegraph is the number of edges in the underlying hedgegraph.

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 on hedgegraphs formally for the sake of completeness. Let $U \subseteq V$ be a subset of nodes in G. We define G *contract* U, denoted G/U, to be a hedgegraph on node set $V' := (V - U) \cup \{u\}$, where u is a newly introduced node, and on hedge set E', where E' is obtained as follows: for each hedge $e \in E$, we define the hedge e' to be

$$e' := \{(\{a, b\} - U) \cup \{u\} : |\{a, b\} \cap U| = 1, \{a, b\} \in e\} \tag{4.1}$$

$$\cup \{\{a, b\} : \{a, b\} \cap U = \emptyset, \{a, b\} \in e\} \tag{4.2}$$

and obtain $E' := \{e' : e' \neq \emptyset, e \in E\}$. For a hedge $e \in E$, let $C_1, \ldots, C_{s(e)}$ denote the node

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 node set of each component in $G[e]$ individually, i.e., $G/e := G/C_1/C_2/\ldots/C_{s(e)}$. We observe that contracting a hedge does not increase the span.

We need the following technical lemma.

Lemma 4.1 (Majorization inequality, e.g., see Theorem 108 in [105])**.** *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*

$$\sum_{i=1}^{\ell} f(y_i) \leq \sum_{i=1}^{\ell} f(x_i). \tag{4.3}$$

4.3 HEDGE K-CUT IN CONSTANT SPAN HEDGEGRAPHS

In this section, we design an algorithm to solve Hedge-k-Cut in constant span hedgegraphs. In Section 4.3.1, we give an outline of the algorithm for Hypergraph-2-Cut to present the main ideas underlying our algorithm for Hypergraph-k-Cut. In Section 4.3.2, we present the complete algorithm and the analysis for Hedge-k-Cut in constant span hedgegraphs. In Section 4.3.3, we improve on the run-time analysis of the same algorithm by specializing it to hypergraphs and also conclude a combinatorial bound on the number of distinct minimum k-cut-sets.

4.3.1 Overview

We recall Karger's uniform random contraction algorithm for graphs (more generally, multigraphs): pick an edge *uniformly at random*, contract it and repeat until there are 2 nodes left at which point output the edges between the two nodes. 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 node) 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 $2/n$.

Now consider the above algorithm for hypergraphs under the standard definition of hyperedge contraction (the graph G/e is obtained by removing the nodes of the hyperedge e, introducing a new node v and for every other hyperedge f in G that intersects e, replacing f by $(f \setminus e) \cup \{v\}$ and removing hyperedges with cardinality one). 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-node hypergraphs for which a hyperedge from a min-cut could be chosen for contraction with probability as large as a constant and not at most $2/n$. We overcome this issue by choosing a hyperedge to contract from a non-uniform probability distribution.

We now present this non-uniform distribution along with the analysis. Let n be the number of nodes in the input hypergraph $G = (V, E)$. For each hyperedge $e \in E$, we define a "dampening factor" δ_e to be the probability that a uniformly random node from V does *not* belong to e. We use $r(e)$ to denote the rank of a hyperedge e. Thus,

$$\delta_e = \frac{n - r(e)}{n}. \tag{4.4}$$

We note that $\delta_e = 0$ implies that e contains every node in the graph and hence is in every cut. Our algorithm for Hypergraph-2-Cut is the following: pick a hyperedge e with probability proportional to δ_e, contract and repeat until either (i) the number of nodes is at most 4 in which case, output all optimum solutions in the constant-sized current hypergraph or (ii) the dampening factor of all hyperedges is zero in which case, output all hyperedges in the current graph. The run-time of the algorithm is polynomial as contraction and updating the dampening factors can be implemented in polynomial-time.

In order to analyze the correctness probability of this algorithm, let us define

$$q_n := \min_{\substack{G\text{: } n\text{-node hypergraph} \\ C^*\text{: hyperedges of a min-cut in } G}} \mathsf{Pr}(\text{Algorithm outputs } C^* \text{ when executed on } G). \tag{4.5}$$

We show that $q_n \geq \binom{n}{2}^{-1}$ by induction on n. For the base case, we consider $n \leq 4$ and observe that the algorithm is correct with probability one on such n-node hypergraphs. For the induction step, let $n > 4$. Fix a n-node hypergraph $G = (V, E)$ and the hyperedges C^* of a min-cut in G that together achieve the minimum for q_n. We may assume that there exists a hyperedge in G whose dampening factor is not zero for otherwise, all hyperedges are in every cut and hence, the output is in fact an optimum and consequently, the correctness probability is one. Now, the probability that the algorithm returns C^* when executed on G is at least the probability of choosing a hyperedge $e \in E \setminus C^*$, contracting it and succeeding on the remaining hypergraph. The number of nodes in the hypergraph after contracting e is $n - r(e) + 1$. Let p_e be the probability of contracting e. Hence,

$$q_n \geq \sum_{e \in E \setminus C^*} p_e q_{n-r(e)+1} = \frac{1}{\sum_{f \in E} \delta_f} \cdot \sum_{e \in E \setminus C^*} \delta_e q_{n-r(e)+1}. \tag{4.6}$$

Now, consider a hyperedge $e \in E \setminus C^*$. We have $n - r(e) \geq 1$ for otherwise the hyperedge e is in every cut and would be in C^*. Moreover, $r(e) \geq 2$ since our current hypergraph never contains a hyperedge with cardinality one. Thus, $n - r(e) + 1 \leq n - 1$. By the inductive hypothesis, we have $q_{n-r(e)+1} \geq \binom{n-r(e)+1}{2}^{-1}$. Hence,

$$\delta_e q_{n-r(e)+1} \geq \frac{n - r(e)}{n} \cdot \frac{1}{\binom{n-r(e)+1}{2}} = \frac{2}{n(n - r(e) + 1)} \geq \frac{2}{n(n-1)} = \frac{1}{\binom{n}{2}} \tag{4.7}$$

and consequently,

$$q_n \geq \frac{1}{\sum_{f \in E} \delta_f} \sum_{e \in E \setminus C^*} \frac{1}{\binom{n}{2}} = \frac{1}{\binom{n}{2}} \cdot \frac{|E \setminus C^*|}{\sum_{f \in E} \delta_f}. \tag{4.8}$$

It remains to argue that $|E \setminus C^*| \geq \sum_{f \in E} \delta_f$. By rewriting the inequality, it suffices to show that the cardinality $|C^*|$ of a min-cut is at most $|E| - \sum_{f \in E} \delta_f$. We will show this by an averaging argument. Consider the size of the set F_v of hyperedges containing a node v. Since F_v is a cut (as it isolates the node v), the cardinality $|C^*|$ of a min-cut is at most $|F_v|$ for every $v \in V$. Now, consider F_v when v is chosen uniformly. We have (by using $1_{v \in f}$ to denote the indicator function that indicates if the node v is in the hyperedge f)

$$|C^*| \leq \mathbb{E}_{v \in V}(|F_v|) = \mathbb{E}_{v \in V} \left(\sum_{f \in E} 1_{v \in f} \right) \tag{4.9}$$

$$= \sum_{f \in E} \mathsf{Pr}_{v \in V}(v \in f) \tag{4.10}$$

$$= \sum_{f \in E} (1 - \mathsf{Pr}_{v \in V}(v \notin f)) \tag{4.11}$$

$$= \sum_{f \in E} (1 - \mathsf{Pr}_{v \in V}(v \notin f)) \tag{4.12}$$

$$= \sum_{f \in E} (1 - \delta_f) \qquad \text{(by the definition of } \delta_f\text{)} \tag{4.13}$$

$$= |E| - \sum_{f \in E} \delta_f. \tag{4.14}$$

Thus, the algorithm succeeds with probability at least $\binom{n}{2}^{-1}$. Executing it $O(n^2 \log n)$ times and returning the best answer among all executions gives an optimum solution with high probability. Our algorithm for Hypergraph-k-Cut and for Hedge-k-Cut in constant span hedgegraphs extends the above algorithm by a careful generalization of the dampening factor.

4.3.2 The contraction algorithm

For ease of description and analysis, we will focus on the minimum cardinality variant of Hedge-k-Cut. We will specify how to adapt it to solve the minimum cost variant at the end of the section. 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 nodes in the input hedgegraph $G = (V, E)$. For a hedge $e \in E$, we recall that $r(e)$ is the number of nodes incident to the edges in e and define

$$\delta_e := \frac{\binom{n-r(e)}{k-1}}{\binom{n}{k-1}}. \tag{4.15}$$

As per the convention established in the preliminaries, we emphasize that $\delta_e = 0$ if $n - r(e) < k - 1$. 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 nodes and $r(e)$ for every $e \in E$ and repeat until the number of nodes is small. When the number of nodes 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 nodes. 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 4.1.

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

$$m - \lambda \geq \sum_{e \in E} \delta_e. \tag{4.16}$$

Proof. We will prove the lemma by exhibiting an upper bound on λ by the probabilistic method. Let W be a subset of $k - 1$ nodes chosen uniformly at random among all subset of

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 $n \leq 2(k-1)(s+1)$, 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 $|V(G/e)| \geq k$:
      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 δ_e.
   (e) Contract and update: $G \leftarrow G/e$, $n \leftarrow |V(G)|$ and update δ_e for every hedge e in the contracted graph G.
3. Output all hedge k-cut-sets with minimum value among the candidates.

Figure 4.1: Contraction algorithm for constant span hedgegraphs.

nodes of size $k-1$. Now consider the k-partition of the node 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 4.3. *Suppose $n > 2(k-1)(s+1)$. Then, for every hedge e with $r(e) \in \{2, \ldots, n-k+1\}$, we have*

$$\delta_e \binom{n - r(e) + s(e)}{(k-1)(s+1)}^{-1} \geq \binom{n}{(k-1)(s+1)}^{-1}. \tag{4.17}$$

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 4.2. Since $r(e) \leq n-k+1$, we have $\binom{n-r(e)}{k-1} \geq 1$. Thus,

$$\delta_e = \binom{n-r(e)}{k-1}\binom{n}{k-1}^{-1} \tag{4.18}$$

$$\geq \binom{n}{k-1}^{-1} \tag{4.19}$$

$$\geq \binom{n}{(k-1)(s+1)}^{-1} \tag{4.20}$$

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

For the rest of the proof, we will assume that $n - r(e) + s(e) \geq (k-1)(s+1)$. 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

$$\delta_e \binom{n-r(e)+s(e)}{t}^{-1} \geq \binom{n}{t}^{-1}. \tag{4.21}$$

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

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

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

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

$$\binom{n-x}{k-1}\binom{n}{k-1}^{-1}\binom{n-x+s}{t}^{-1} \geq \binom{n}{t}^{-1}. \tag{4.22}$$

Consider the LHS of (4.22) 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

$$\text{LHS}(x) = C_{n,k,s}\frac{(n-x)!(n-x+s-t)!}{(n-x-k+1)!(n-x+s)!} \tag{4.23}$$

$$= C_{n,k,s}\frac{(n-x+s-t)!}{(n-x-k+1)!\prod_{i=1}^{s}(n-x+i)} \tag{4.24}$$

$$= C_{n,k,s}\left(\frac{1}{\prod_{i=k-1}^{t-s-1}(n-x-i)}\right)\left(\frac{1}{\prod_{i=1}^{s}(n-x+i)}\right). \tag{4.25}$$

The last equation follows since $t = (k-1)(s+1)$, and hence $k-1 \leq t-s$. From the above

expression, we have that LHS(x) is an increasing function of x. Thus we only need to show inequality (4.22) when x is the minimum value in the domain of interest, i.e., $x = r(e) = s+1$. Hence, it suffices to show that

$$\binom{n-s-1}{k-1}\binom{n}{k-1}^{-1}\binom{n-1}{t}^{-1}\binom{n}{t} \geq 1. \tag{4.26}$$

To show the above, we write out the LHS of (4.26):

$$\text{LHS of (4.26)} = \frac{(n-s-1)!(n-k+1)!}{(n-s-k)!(n-1)!(n-t)} \tag{4.27}$$

$$= \frac{\prod_{i=s+1}^{s+k-1}(n-i)}{(n-t)\prod_{i=1}^{k-2}(n-i)}. \tag{4.28}$$

In order to show that LHS of (4.26) ≥ 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

$$\sum_{i=s+1}^{s+k-1}(-\log(n-i)) \leq \sum_{i=1}^{k-2}(-\log(n-i)) - \log(n-t). \tag{4.29}$$

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 (4.29) follows by applying Lemma 4.1 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 $r(e) \leq s$. By the assumptions of the lemma, we have $r(e) \geq 2$ and hence $s \geq 2$. We recall that $r(e)$ is the number of nodes 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, $r(e) - s(e) \geq r(e)/2$. Consequently, we have

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

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

$$\binom{n-x}{k-1}\binom{n}{k-1}^{-1}\binom{n-x/2}{t}^{-1} \geq \binom{n}{t}^{-1}. \tag{4.30}$$

Proceeding similarly to the analysis in Case 1 above, we can show that the LHS of (4.30) is

an increasing function of x. Then we only need to show inequality (4.30) for $x = 2$, i.e., we need to show that

$$\binom{n-2}{k-1}\binom{n}{k-1}^{-1}\binom{n-1}{t}^{-1}\binom{n}{t} \geq 1. \tag{4.31}$$

Inequality (4.31) is a special case of inequality (4.26), 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 4.5. *For an n-node input hedgegraph with span s, the contraction algorithm given in Figure 4.1 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 $O(nmp)$, where m is the number of hedges in the input hedgegraph.

Proof. For a hedgegraph H, let $\mathcal{O}(H)$ denote the set of min-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-node hedgegraphs with span at most s. We define

$$q_n := \inf_{H \in \mathcal{G}_{n,s}} \min_{C \in \mathcal{O}(H)} q(H, C). \tag{4.32}$$

We will prove by induction on n that $q_n \geq \binom{n}{(k-1)(s+1)}^{-1}$. 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 nodes 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 $n \leq 2(k-1)(s+1)$. 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 $|V(G/e)| \geq k$. 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 $|V(G/e)| \geq k$, 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) $E \setminus C \neq \emptyset$. 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 $p_e \geq 0 \ \forall e \in E$ 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, the algorithm 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 $n - r(e) + s(e) < n$ nodes 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, $q(G/e, C) \geq q_{n-r(e)+s(e)}$ by definition.

Thus, we have

$$
\begin{aligned}
q(G, C) &\geq \sum_{e \in E \setminus C} p_e \cdot q(G/e, C) && (4.33)\\
&\geq \sum_{e \in E \setminus C} p_e \cdot q_{n-r(e)+s(e)} && (4.34)\\
&= \frac{1}{\sum_{e \in E} \delta_e} \sum_{e \in E \setminus C} \delta_e \cdot q_{n-r(e)+s(e)} \qquad (\text{since } p_e = \delta_e / \sum_{e \in E} \delta_e) && (4.35)\\
&\geq \frac{1}{\sum_{e \in E} \delta_e} \sum_{e \in E \setminus C} \delta_e \cdot \binom{n - r(e) + s(e)}{(k-1)(s+1)}^{-1}. \qquad (\text{by induction hypothesis}) && (4.36)
\end{aligned}
$$

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 4.3,

for every hedge $e \in E \setminus C$, we have

$$
\delta_e \cdot \binom{n - r(e) + s(e)}{(k-1)(s+1)}^{-1} \geq \binom{n}{(k-1)(s+1)}^{-1}. \tag{4.37}
$$

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

$$q(G,C) \geq \frac{1}{\sum_{e\in E}\delta_e} \sum_{e\in E\setminus C} \binom{n}{(k-1)(s+1)}^{-1} \tag{4.38}$$

$$= \frac{m-\lambda}{\sum_{e\in E}\delta_e} \binom{n}{(k-1)(s+1)}^{-1} \qquad \text{(since } |C| = \lambda \text{ and } |E| = m\text{)} \tag{4.39}$$

$$\geq \binom{n}{(k-1)(s+1)}^{-1}. \qquad \text{(by Lemma 4.2)} \tag{4.40}$$

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

$$q_n = \inf_{H\in\mathcal{G}_{n,s}} \min_{C\in\mathcal{O}(H)} q(H,C) \geq \binom{n}{(k-1)(s+1)}^{-1}. \tag{4.41}$$

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

We analyze the run-time for one iteration of Step 2. The brute-force operation in Step 2(a) takes $O(pk^{2(k-1)(s+1)})$ time. The for-loop in Step 2(b) applies at most $O(m)$ contractions. Each contraction takes $O(p)$ time and each brute-force search for minimum hedge-k-cut-set takes $O(pk^{k+s})$ time. Hence Step 2(b) runs in time $O(mp)$. 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 $O(p)$ 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 $O(mp)$.

Since the number of nodes 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 $O(nmp)$ 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 $O(np)$. □

Theorem 4.1 follows from Theorem 4.5 by executing the contraction algorithm $\log n \binom{n}{(k-1)(s+1)}$ times and outputting a hedge k-cut-set with the minimum value among all executions.

The contraction algorithm can be adapted to solve the min-cost variant, where each hedge e has capacity $w(e)$, and the goal is to find a subset of hedges of minimum total capacity to remove so that the underlying graph has at least k connected components. In this case, we set $\delta_e := w(e)\binom{n-r(e)}{k-1}/\binom{n}{k-1}$ and run the same contraction algorithm as above. The correctness and run-time arguments are analogous to the one in Theorem 4.5 and we avoid repeating in the interests of brevity.

4.3.3 Contraction Algorithm for Hypergraph-k-Cut

We next focus on the special case of Hypergraph-k-Cut. We restate and prove Theorem 4.2.

Theorem 4.2. *There exists a randomized polynomial time algorithm to solve* Hypergraph-k-Cut *that runs in time* $O(pn^{2k-1}\log n)$ *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 4.1, such a transformation immediately gives a randomized polynomial time algorithm to solve Hypergraph-k-Cut that runs in time $O(nmpn^{2(k-1)}\log n)$ 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 node $v \in e$ 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 nodes 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 node 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 node v of e'. Without loss of generality, let $v \in V_1$. Then there exists a node $u \in e' \cap V_j$ 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 node 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 prove the improved run-time bound. We obtain the improvement by showing 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 $O(np)$ as analyzed in the proof of Theorem 4.5. The theorem now follows by running the contraction algorithm $\binom{n}{2(k-1)} \log n$ times and returning the hedge k-cut-set with minimum value among all executions. □

We now bound the number of optimal cut-sets. We restate and prove Corollary 4.1.

Corollary 4.1. *The number of distinct minimum k-cut-sets in an n-node hypergraph is $O(n^{2(k-1)})$.*

Proof. We recall that a k-cut-set is a set C of hyperedges in a hypergraph G whose removal results in a hypergraph with at least k connected components. Let $S_1, \ldots, S_\ell$ be the min-k-cut-sets in a given n-node hypergraph. Let $\mathcal{E}_i$ be the event that S_i survives the contraction algorithm until there are at most $4(k-1)$ nodes. Then, by Theorem 4.5 (by considering $s = 1$ for hypergraphs), we have that $\Pr(\mathcal{E}_i) \geq \binom{n}{2(k-1)}^{-1}$. The number of possible min-k-cut-sets for HYPERGRAPH-k-CUT in a hypergraph with $4(k-1)$ nodes is at most $k^{4(k-1)}$ and hence, $\sum_{i=1}^{\ell} \Pr(\mathcal{E}_i) \leq k^{4(k-1)}$. Hence, $\ell \leq k^{4(k-1)} \binom{n}{2(k-1)} = O(n^{2(k-1)})$. □

4.4 RPTAS FOR HEDGE-K-CUT

In this section, we provide a randomized 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, Karger and Panigrahi [3]. Their contraction algorithm distinguishes 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 appropriately.

Let $G = (V, E)$ be a hedgegraph with n nodes. 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 4.2.

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

1. If G has k nodes, 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 4.2: Contraction algorithm for arbitrary span hedgegraphs

The following lemma bounds the number of branching steps performed by the algorithm.

Lemma 4.4. *The total number of branching steps in one execution of the contraction algorithm on an n-node hedgegraph is at most*

$$\log_{\frac{8(k-1)}{8(k-1)-1}} n. \tag{4.42}$$

Proof. We prove this lemma by induction on n. Let $G = (V, E)$ be an n-node hedgegraph. We consider $n = k$ as base case. For such n, the algorithm terminates without a branching step, so the statement is true.

We now show the induction step. If the graph underlying G has at least k components, then the algorithm terminates without a branching step, so the statement is again true.

If G is small, then the inductive hypothesis directly implies the statement since no branching step is performed.

If G is large, then we go to either branch (a) or branch (b). In branch (a), the algorithm recurses on a small hedgegraph H_1. Since the number of nodes spanned by each hedge never increases during the execution of the algorithm, the hedgegraph H_1 will not become a large hedgegraph, and hence will not encounter another branching, until its number of nodes is halved. Then by the inductive hypothesis, the total number of branchings in the algorithm is at most

$$1 + \log_{\frac{8(k-1)}{8(k-1)-1}} \left(\frac{n}{2}\right) \leq \log_{\frac{8(k-1)}{8(k-1)-1}} n. \tag{4.43}$$

We have proved the induction step for branch (a). Next we will show the induction step for branch (b). In branch (b), let e be the contracted hedge. Then the algorithm recurses on the hedgegraph H_2 which has $n - r(e) + s(e)$ nodes. Since each connected component of any hedge has at least two nodes, we have $s(e) \leq r(e)/2$, and thus $n - r(e) + s(e) \leq n - r(e)/2$. Since e is either a large or a moderate hedge, we have that $r(e) \geq n/(4(k-1))$, and hence the hedgegraph H_2 has at most $n - r(e)/2 \leq n \cdot (1 - 1/(8(k-1))) = n \cdot (8(k-1)-1)/(8(k-1))$ nodes. Therefore, by the inductive hypothesis, the total number of branchings in the algorithm is at most

$$1 + \log_{\frac{8(k-1)}{8(k-1)-1}} \left(\frac{8(k-1)-1}{8(k-1)} \cdot n\right) = \log_{\frac{8(k-1)}{8(k-1)-1}} n. \tag{4.44}$$

□

We next show that the algorithm always outputs a feasible solution and that it can be implemented to run in polynomial-time.

Lemma 4.5. *The contraction algorithm given in Figure 4.2 always outputs a hedge k-cut-set. Moreover, the algorithm can be implemented to run in time $O(pn)$.*

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 (nodes). So, the algorithm always outputs a hedge k-cut-set.

Next, we show that the contraction algorithm can be implemented to run in $O(pn)$ 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 4.5, a contraction step can be implemented to run in $O(p)$ time by

processing hedges one by one to mark contracted nodes and replacing them with a new node. Since in one execution of the contraction algorithm there can be at most n contractions and $O(\log n)$ branching steps by Lemma 4.4, the contraction algorithm can be implemented to run in $O(pn)$ time.

□

Next, we state a few helper lemmas which 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 nodes incident to the edges in e is denoted by $r(e)$.

Lemma 4.6. *Let $G = (V, E)$ be a hedgegraph with minimum hedge k-cut-set value λ. Let W be a subset of $k-1$ nodes and let $E(W) := \{e \in E : |V(e) \cap W| \geq 1\}$. Then*

$$|E(W)| \geq \lambda. \tag{4.45}$$

Proof. Let $\mathcal{P} = \{\{v\} | v \in W\} \cup \{V \backslash W\}$ be the k-partitioning of the node set induced by W. Let $E(\mathcal{P})$ be the set of hedges that cross $\mathcal{P}$. Since each hedge contains at least two nodes, every hedge crossing $\mathcal{P}$ must contain at least one node in W. Therefore, $E(\mathcal{P}) \subseteq E(W)$, so $|E(W)| \geq |E(\mathcal{P})|$. Since $E(\mathcal{P})$ is a hedge k-cut-set and λ is the minimum hedge k-cut-set value, we have $|E(\mathcal{P})| \geq \lambda$ and hence, $|E(W)| \geq \lambda$.

□

Lemma 4.7. *Let $G = (V, E)$ be an n-node hedgegraph with minimum hedge k-cut-set value λ. Then,*

$$\sum_{e \in E} r(e) \geq \frac{n\lambda}{k-1}. \tag{4.46}$$

Proof. Let $\mathcal{U}$ denote the set of all subsets of nodes of size $k-1$. Then $|\mathcal{U}| = \binom{n}{k-1}$. For a subset W of $k-1$ nodes, let $E(W) := \{e \in E : |V(e) \cap W| \geq 1\}$. For a node v, we use $\deg(v)$ to denote the number of hedges in E which have edges incident to v.

To prove this lemma, we observe that $\sum_{W \in \mathcal{U}} \sum_{v \in W} \deg(v) = \binom{n-1}{k-2} \sum_{v \in V} \deg(v)$. Indeed, for each $v \in V$, the term $\deg(v)$ appears in the LHS whenever $v \in W$. There are $\binom{n-1}{k-2}$ sets

W in $\mathcal{U}$ that contain v, so the observation follows. Therefore,

$$\sum_{e\in E} r(e) = \sum_{v\in V} \deg(v) \tag{4.47}$$

$$= \binom{n-1}{k-2}^{-1} \sum_{W\in\mathcal{U}} \sum_{v\in W} \deg(v) \qquad \text{(by the above observation)} \tag{4.48}$$

$$= \binom{n-1}{k-2}^{-1} \sum_{W\in\mathcal{U}} \sum_{e\in E} |V(e)\cap W| \tag{4.49}$$

$$\geq \binom{n-1}{k-2}^{-1} \sum_{W\in\mathcal{U}} \sum_{e\in E, |V(e)\cap W|\geq 1} 1 \tag{4.50}$$

$$\geq \binom{n-1}{k-2}^{-1} \sum_{W\in\mathcal{U}} |E(W)| \qquad \text{(by definition of } E(W)) \tag{4.51}$$

$$\geq \binom{n-1}{k-2}^{-1} \sum_{W\in\mathcal{U}} \lambda \qquad \text{(by Lemma 4.6)} \tag{4.52}$$

$$= \binom{n-1}{k-2}^{-1} \binom{n}{k-1} \lambda \tag{4.53}$$

$$= \frac{n\lambda}{k-1}. \tag{4.54}$$

□

Lemma 4.8. *If $G = (V, E)$ is an n-node 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) $m \geq 2\lambda$.

Proof. Since C contains λ hedges, and each hedge $e \in E$ has $r(e) \leq n/(2(k-1))$, we have that $\sum_{e\in C} r(e) \leq n\lambda/2(k-1)$. Hence, by Lemma 4.7, we have that $\sum_{e\in E\setminus C} r(e) \geq n\lambda/(2(k-1))$.

Moreover, $\sum_{e\in E} r(e) \leq m(n/2(k-1))$ since every hedge e has $r(e) \leq n/(2(k-1))$. Again, by Lemma 4.7, we have that $m \geq 2\lambda$. □

Lemma 4.9. *For every $x \in (0, 1/2)$ and $c \geq 4$, we have $(1-x)\cdot(1-x/c)^{-3c/2} \geq 1$.*

Proof. Let $f(x) := (1-x)\cdot(1-x/c)^{-3c/2}$. Then

$$f'(x) = \left(1-\frac{x}{c}\right)^{-\frac{3c}{2}} \cdot \left(\frac{3c(1-x)}{2(c-x)} - 1\right).$$

Since the first factor $(1 - x/c)^{-3c/2} > 0$ for all $x \in (0, 1/2), c \geq 4$, then the sign of $f'(x)$ depends on the sign of $3c(1-x)/(2(c-x)) - 1$. If we solve $f'(x) = 0$ for x, we have $x = c/(3c-2)$. Since $c \geq 4$, $2/c < 1$, so $c/(3c-2) = 1/(3-2/c) \leq 1/2$. Then $c/(3c-2)$ divides the interval $(0, 1/2)$ into two pieces and we look at the sign of $f'(x)$ in these two pieces separately. When $0 < x < c/(3c-2)$,

$$\frac{3c(1-x)}{2(c-x)} - 1 = \frac{c - x(3c-2)}{2(c-x)} > 0.$$

When $x > c/(3c-2)$,

$$\frac{c - x(3c-2)}{2(c-x)} < 0.$$

Therefore, we know $f'(x) \geq 0$ for $x \in (0, c/(3c-2)]$ and $f'(x) < 0$ for $x \in (c/(3c-2), 1/2)$. Since $f(0) = 1$, $f(x) \geq 1$ for $x \in (0, c/(3c-2)]$. Now we only need to show that $f(x) \geq 1$ for $x \in (c/(3c-2), 1/2)$. Since $f'(x) < 0$ for $x \in (c/(3c-2), 1/2)$, we only need to show that $f(x) \geq 1$ when $x = 1/2$. When $x = 1/2$, $f(x) = 1/2 \cdot (1 - 1/(2c))^{-3c/2}$, which is an increasing function of c. If $c = 4$, $1/2 \cdot (1 - 1/(2c))^{-3c/2} = 1/2 \cdot (7/8)^{-6} > 1$. Therefore, $1/2 \cdot (1 - 1/(2c))^{-3c/2} > 1$ for $c \geq 4$, so $f(x) > 1$ when $x = 1/2$.

□

We now lower bound the success probability of the algorithm.

Lemma 4.10. *For an n-node input hedgegraph, the contraction algorithm given in Figure 4.2 outputs a $(1+\epsilon)$-approximate minimum hedge k-cut-set with probability $n^{-O(\log(1/\epsilon))}$.*

Proof. Let $\mathcal{H}(n, \ell)$ be the family of hedgegraphs on n nodes 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

$$q_{n,\ell} := \inf_{H \in \mathcal{H}(n,\ell)} q(H). \tag{4.55}$$

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

$$q_{n,\ell} \geq n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell} \quad \forall\, n \geq k. \tag{4.56}$$

Let $G \in \mathcal{H}(n, \ell)$, with node set $V = [n]$ and hedge set E. Let $m := |E|$. Let us fix a minimum hedge k-cut-set C of G and suppose that its value is λ.

To base the induction, we consider $n = k$. Then, G has only one hedge k-cut-set and the algorithm returns it. So $q(G) = 1$ and hence $q_{n,\ell} = 1$.

We next show the induction step. If the graph underlying G has at least k components, then, the empty set is the only minimum hedge k-cut-set and the algorithm returns it. Hence, $q(G) = 1$. Thus, we may assume that $n > k$ and the graph underlying G has fewer than k components. 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 $n - r(e) + s(e)$ nodes. 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,

$$
\begin{aligned}
q(G) &\geq \frac{1}{m} \sum_{e \in E \setminus C} q(G/e) && (4.57)\\
&\geq \frac{1}{m} \sum_{e \in E \setminus C} q_{n-r(e)+s(e),\ell} \qquad \text{(by definition of } q_{n,l}) && (4.58)\\
&\geq \frac{1}{m} \sum_{e \in E \setminus C} (n - r(e) + s(e))^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell} \qquad \text{(by inductive hypothesis)} && (4.59)\\
&\geq \frac{1}{m} \sum_{e \in E \setminus C} \left(n - \frac{r(e)}{2}\right)^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell} && (4.60)\\
&\text{(by } n - r(e) + s(e) \leq n - r(e)/2) && (4.61)\\
&= \frac{m - \lambda}{m} \left(\frac{\gamma}{2}\right)^{\ell} \frac{1}{m - \lambda} \sum_{e \in E \setminus C} \left(n - \frac{r(e)}{2}\right)^{-6(k-1)}. && (4.62)
\end{aligned}
$$

   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

$$
\frac{1}{m - \lambda} \sum_{e \in E \setminus C} \left(n - \frac{r(e)}{2}\right)^{-6(k-1)} \geq \left(n - \frac{\sum_{e \in E \setminus C} r(e)}{2(m - \lambda)}\right)^{-6(k-1)}. \tag{4.63}
$$

Therefore, we have

$$q(G) \geq \frac{m-\lambda}{m} \cdot \left(\frac{\gamma}{2}\right)^{\ell} \cdot \left(n - \frac{\sum_{e \in E \setminus C} r(e)}{2(m-\lambda)}\right)^{-6(k-1)} \tag{4.64}$$

$$\geq \frac{m-\lambda}{m} \cdot \left(\frac{\gamma}{2}\right)^{\ell} \cdot \left(n - \frac{n\lambda/(2(k-1))}{2m}\right)^{-6(k-1)} \qquad \text{(by Lemma 4.8)} \tag{4.65}$$

$$= (1 - \frac{\lambda}{m}) \left(\frac{\gamma}{2}\right)^{\ell} n^{-6(k-1)} \left(1 - \frac{\lambda}{4(k-1)m}\right)^{-6(k-1)} \tag{4.66}$$

$$= \left(\frac{\gamma}{2}\right)^{\ell} n^{-6(k-1)} \cdot (1-x) \left(1 - \frac{x}{4(k-1)}\right)^{-6(k-1)}. \tag{4.67}$$

The last equality follows by setting $x := \lambda/m$. We have $x \in (0, 1/2)$ since $m \geq 2\lambda$ by Lemma 4.8. 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 4.9, 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 $|L \backslash C| \geq \gamma \cdot |L|$. 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 $|L \backslash C| \geq \gamma \cdot |L|$, the probability of picking a hedge $e \in L \setminus C$ to contract is at least 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 $n - r(e) + s(e)$ nodes where $n - r(e) + s(e) \leq n - r(e)/2 \leq n - 1$. We also note that $H_2 \in \mathcal{H}(|V(H_2)|, \ell - 1)$. Therefore,

$$q(G) \geq \frac{1}{2} \cdot \gamma \cdot q(H_2) \tag{4.68}$$

$$\geq \frac{1}{2} \cdot \gamma \cdot q_{|V(H_2)|, \ell-1} \qquad \text{(by definition of } q_{n,l}) \tag{4.69}$$

$$\geq \frac{1}{2} \cdot \gamma \cdot (|V(H_2)|)^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell-1} \qquad \text{(by inductive hypothesis)} \tag{4.70}$$

$$\geq \frac{1}{2} \cdot \gamma \cdot n^{-6(k-1)} \left(\frac{\gamma}{2}\right)^{\ell-1} \tag{4.71}$$

$$= n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell}. \tag{4.72}$$

3. Suppose G is large and $|L \backslash C| < \gamma \cdot |L|$. Since $|L \backslash C| < \gamma \cdot |L|$, 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,

$$q(G) \geq \frac{1}{2} \cdot q(H_1) \geq \frac{1}{2} \cdot q_{n,\ell-1}. \tag{4.73}$$

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 4.4.1. *$q(G) \geq n^{-6(k-1)}$ 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 $n > k$ nodes 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

$$q(G) \geq n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell} = n^{-6(k-1)}. \tag{4.74}$$

by the same proof as that of case 1.

□

By Claim 4.4.1, 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

$$q(G) \geq \frac{1}{2} \cdot q_{n,\ell-1} \geq \frac{1}{2} \cdot \left(\frac{\gamma}{2}\right)^{\ell-1} \cdot n^{-6(k-1)} \geq \left(\frac{\gamma}{2}\right)^{\ell} \cdot n^{-6(k-1)}. \tag{4.75}$$

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

$$q_{n,\ell} = \inf_{H \in \mathcal{H}(n,\ell)} q(H) \geq n^{-6(k-1)} \cdot \left(\frac{\gamma}{2}\right)^{\ell}. \tag{4.76}$$

We know that $\gamma < 1$. Substituting the upper bound on ℓ from Lemma 4.4, we obtain that

$$q_{n,\ell} \geq n^{-6(k-1)} \left(\frac{\gamma}{2}\right)^{\ell} \tag{4.77}$$

$$\geq n^{-6(k-1)} \left(\frac{\gamma}{2}\right)^{\log_{\frac{8(k-1)}{8(k-1)-1}} n}. \tag{4.78}$$

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

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

□

Theorem 4.3 follows from Lemmas 4.10 and 4.5 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. Next, in order to prove Theorem 4.4, we bound the probability that the contraction algorithm returns any fixed optimum solution.

Lemma 4.11. *For an n-node input hedgegraph, the contraction algorithm given in Figure 4.2 outputs any fixed minimum hedge k-cut-set with value λ with probability $n^{-O(k+\log \lambda)}$.*

Proof. For a hedgegraph H, let $\mathcal{O}(H)$ denote the set of min-k-cut-sets in H. Let $\mathcal{H}(n,\ell)$ be the family of hedgegraphs on n nodes for which the contraction algorithm will always terminate using at most ℓ branchings. Let $\mu(H,C)$ denote the probability that the algorithm returns C on H. We define

$$\mu_{n,\ell} := \inf_{H \in \mathcal{H}(n,\ell)} \min_{C \in \mathcal{O}(H)} \mu(H,C). \tag{4.79}$$

We will prove by induction on n that

$$\mu_{n,\ell} \geq n^{-6(k-1)} \cdot \left(\frac{1}{2(1+\lambda)}\right)^{\ell} \quad \forall\, n \geq k. \tag{4.80}$$

Let $G \in \mathcal{H}(n,\ell)$, with node set $V = [n]$ and hedge set E. Let $m := |E|$. Let us fix a minimum hedge k-cut-set C of G and suppose that its value is λ. Arguments identical to that in the proof of Lemma 4.10 address the base case of $n = k$ and when the graph underlying G has at least k components. So, we may again assume that G has fewer than k components. We again distinguish three cases.

1. Suppose G is small. The arguments for this case are identical to that of the first case in the proof of Lemma 4.10. We avoid repeating in the interests of brevity.

2. Suppose G is large and $|L \setminus C| \geq 1$. The algorithm returns C on G if it goes to branch (b), contracts a hedge $e \in L \setminus C$, and returns C on the resulting graph $H_2 = G/e$. The probability of picking a hedge $e \in L \setminus C$ to contract is $|L \setminus C|/|L|$. Since $|L \setminus C| \geq 1$ and moreover $|L \cap C| \leq |C| = \lambda$, we have $|L \setminus C|/|L| \geq 1/(1+\lambda)$. The algorithm contracts either a moderate or a large hedge e for which $r(e) \geq n/(4(k-1))$ and H_2 has $n - r(e) + s(e)$ nodes where $n - r(e) + s(e) \leq n - r(e)/2$. Hence, H_2 has at most $n - r(e)/2 \leq n - 1$ nodes. We also note that $H_2 \in \mathcal{H}(|V(H_2)|, \ell - 1)$ and if the hedge e that is contracted to obtain H_2 is not in C, then $C \in \mathcal{O}(H_2)$. Therefore,

$$\begin{aligned}
\mu(G, C) &\geq \frac{1}{2} \cdot \frac{1}{(1+\lambda)} \cdot \mu(H_2, C) && (4.81)\\
&\geq \frac{1}{2(1+\lambda)} \cdot \mu_{|V(H_2)|,\ell-1} \quad \text{(by definition of } \mu_{n,l}) && (4.82)\\
&\geq (|V(H_2)|)^{-6(k-1)} \cdot \left(\frac{1}{2(1+\lambda)}\right)^{\ell} \quad \text{(by inductive hypothesis)} && (4.83)\\
&\geq n^{-6(k-1)} \cdot \left(\frac{1}{2(1+\lambda)}\right)^{\ell} && (4.84)
\end{aligned}$$

3. Suppose G is large and $L \subseteq C$. In this case, the algorithm returns C on G if it goes to branch (a) and returns $C - L$ on $H_1 = G - L$. We note that $H_1 \in \mathcal{H}(n, \ell - 1)$ and $C \setminus L \in \mathcal{O}(H_1)$. Therefore,

$$\mu(G, C) \geq \frac{1}{2} \cdot \mu(H_1, C \setminus L) \geq \frac{1}{2} \cdot \mu_{n,\ell-1}. \tag{4.85}$$

   By induction on ℓ, we may now show that $\mu(G, C) \geq n^{-6(k-1)} \cdot (1/2(1+\lambda))^{\ell}$. The rest of the proof is identical to that of case 3 of the proof of Lemma 4.10, so we avoid repeating it.

Therefore, for all $n \geq k$, we have

$$\mu_{n,\ell} \geq n^{-6(k-1)} \cdot \left(\frac{1}{2(1+\lambda)}\right)^{\ell}. \tag{4.86}$$

Substituting the upper bound on ℓ from Lemma 4.4 gives

$$\mu_{n,\ell} \geq n^{-6(k-1)} \cdot \left(\frac{1}{2(1+\lambda)}\right)^{\log_{\frac{8(k-1)}{8(k-1)-1}} n} = n^{-O(k+\log\lambda)}. \tag{4.87}$$

□

Lemma 4.11 leads to Theorem 4.4 similar to the proof of Corollary 4.1 from Lemma 4.5. Lemmas 4.11 and 4.5 also lead to the following corollary by executing the contraction algorithm $n^{O(\log \lambda)} \log n$ times and returning a hedge k-cut-set with the minimum value among all executions (the value λ can be found by a binary search).

Corollary 4.2. *There exists a randomized algorithm to solve* Hedge-k-Cut *that runs in time* $pn^{O(\log \lambda)} \log n$*, where* λ *is the value of a minimum hedge* k*-cut-set in the input hedge-graph.*

4.5 OPEN PROBLEMS

We gave a quasipolynomial time algorithm for Hedge-k-Cut. Can we solve Hedge-k-Cut in polynomial time? It is unknown even for $k = 2$. We described a polynomial time algorithm for Hypergraph-k-Cut. Can the running time be improved? For example, the technique in Karger-Stein might be applicable [29]. Finally, our algorithm for Hypergraph-k-Cut is randomized. Is there a deterministic polynomial time algorithm?

Chapter 5: Global vs. Fixed-terminal cuts

The dichotomy in complexity between global and fixed-terminal k-cut problems for graphs is well-known since the early 90s. For concreteness, we recall from chapter 4 the Graph-k-Cut problem and the Graph-k-Way-Cut problem for $k \geq 3$. While the global variant, namely Graph-k-Cut, admits an efficient algorithm [27, 29], the fixed-terminal variant, namely Graph-k-Way-Cut, is **NP-Hard** [38].

In this chapter, we further investigate this dichotomy by studying different kinds of cut problems.

5.1 *ST*-SEP-*K*-CUT

We consider the following variant of Graph-k-Cut. It is an intermediary between Graph-k-Cut and Graph-k-Way-Cut.

***st*-Sep-k-Cut:** Given a *graph* $G = (V, E)$ with two specified nodes $s, t \in V$, find a smallest subset of edges to remove so that the resulting graph has at least k connected components with s and t being in different components.

The complexity of *st*-Sep-k-Cut for constant k was posed as an open problem by Queyranne [34]. We resolve this open problem by showing that *st*-Sep-k-Cut is solvable in polynomial-time for every constant k.

Theorem 5.1. *For every constant k, there is a polynomial-time algorithm to solve st-Sep-k-Cut.*

Notations. Let $G = (V, E)$ be an graph. Let $\gamma^q(G)$ denote the value of an optimum q-Cut in G, i.e.,

$$\gamma^q(G) := \min\{c(\{V_1, \ldots, V_q\}) : V_i \neq \emptyset \ \forall\ i \in [q], \\ V_i \cap V_j = \emptyset \ \forall\ i, j \in [q], \cup_{i=1}^{q} V_i = V\}.$$

Proof of Theorem 5.1. Let γ^* denote the optimum value of *st*-Sep-k-Cut in $G = (V, E)$ and let H denote the graph obtained from G by adding an edge of infinite capacity between s and t. The algorithm is based on the following observation (we recommend the reader to consider $k = 3$ for ease of understanding):

Proposition 5.1. *Let $\{V_1, \ldots, V_k\}$ be a partition of V corresponding to an optimal solution of st-*Sep-k-Cut*, where s is in V_{k-1} and t is in V_k. Then $c(\{V_1, \ldots, V_{k-2}, V_{k-1} \cup V_k\}) \leq 2\gamma^{k-1}(H)$.*

Proof. Let $W_1, \ldots, W_{k-1}$ be a minimum $(k-1)$-cut in H. Clearly, s and t are in the same part, so we may assume that they are in W_{k-1}. Let U_1, U_2 be a minimum st-cut in $G[W_{k-1}]$. Then $\{W_1, \ldots, W_{k-2}, U_1, U_2\}$ gives an st-separating k-cut, showing that

$$\gamma^* \leq c(\{W_1, \ldots, W_{k-2}, U_1, U_2\}) = \gamma^{k-1}(H) + \lambda_{G[W_{k-1}]}(s,t). \tag{5.1}$$

By Menger's theorem, we have $\lambda_G(s,t)$ pairwise edge-disjoint paths $P_1, \ldots, P_{\lambda_G(s,t)}$ between s and t in G. Consider one of these paths, say P_i. If all nodes of P_i are from $V_{k-1} \cup V_k$, then P_i has to use at least one edge from $\delta(V_{k-1}, V_k)$. Otherwise, P_i uses at least two edges from $\delta(V_1 \cup \cdots \cup V_{k-2})$. Hence the maximum number of pairwise edge-disjoint paths between s and t is

$$\lambda_G(s,t) \leq d(V_{k-1}, V_k) + \frac{1}{2}\left(c(V_1 \cup \cdots \cup V_{k-2})\right). \tag{5.2}$$

Thus, we have

$$\begin{aligned}
\gamma^* &= d(V_{k-1}, V_k) + c(V_1 \cup \cdots \cup V_{k-2}) + \sum_{i<j\leq k-2} d(V_i, V_j) && (5.3)\\
&\geq \lambda_G(s,t) + \frac{1}{2}\left(c(V_1 \cup \cdots \cup V_{k-2}) + \sum_{i<j\leq k-2} d(V_i, V_j)\right) && (5.4)\\
&= \lambda_G(s,t) + \frac{1}{2}c(\{V_1, \ldots, V_{k-2}, V_{k-1} \cup V_k\}) && (5.5)\\
&\geq \lambda_{G[W_{k-1}]}(s,t) + \frac{1}{2}c(\{V_1, \ldots, V_{k-2}, V_{k-1} \cup V_k\}) && (5.6)
\end{aligned}$$

that is,

$$\gamma^* \geq \lambda_{G[W_{k-1}]}(s,t) + \frac{1}{2}c(\{V_1, \ldots, V_{k-2}, V_{k-1} \cup V_k\}). \tag{5.7}$$

By combining (5.1) and (5.7), we get $c(\{V_1, \ldots, V_{k-2}, V_{k-1} \cup V_k\}) \leq 2\gamma^{k-1}(H)$, proving the proposition. □

Karger and Stein [29] showed that the number of feasible solutions to Graph-k-Cut in an graph G with value at most $2\gamma^k(G)$ is $O(n^{4k})$. All these solutions can be enumerated in polynomial-time for fixed k [29, 106, 107]. This observation together with Proposition 5.1 gives the algorithm for finding an optimal solution to st-Sep-k-Cut. The algorithm is summarized in Figure 5.1.

Algorithm for st-Sep-k-Cut

Input: graph $G = (V, E)$ with $s, t \in V$

1. Let H be the graph obtained from G by adding an edge of infinite capacity between s and t. In H, enumerate all feasible solutions to $(k-1)$-Cut—namely the node partitions $\{W_1, \ldots, W_{k-1}\}$—whose cut value $c(\{W_1, \ldots, W_{k-1}\})$ is at most $2\gamma^{k-1}(H)$. Without loss of generality, assume $s, t \in W_{k-1}$.

2. For each feasible solution to $(k-1)$-Cut in H listed in Step 1, find a minimum st-cut in $G[W_{k-1}]$, say U_1, U_2.

3. Among all feasible solutions $\{W_1, \ldots, W_{k-1}\}$ to $(k-1)$-Cut listed in Step 1 and the corresponding U_1, U_2 found in Step 2, return the k-cut $\{W_1, \ldots, W_{k-2}, U_1, U_2\}$ with minimum $c(\{W_1, \ldots, W_{k-2}, U_1, U_2\})$.

Figure 5.1: Algorithm for st-Sep-k-Cut

The correctness of the algorithm follows from Proposition 5.1: one of the choices enumerated in Step 1 will correspond to the partition $(V_1, \ldots, V_{k-2}, V_{k-1} \cup V_k)$, where $(V_1, \ldots, V_k)$ is the partition corresponding to the optimal solution. □

5.2 S-SIZE-K-CUT

Let $s = (s_1, \ldots, s_k)$ be a non-decreasing vector of positive integers. For a graph G, a set of edges C is *s-size k-cut* if we can find a k-partition $(V_1, \ldots, V_k)$, such that $|V_i| \geq s_i$ for all $1 \leq i \leq k$, and C are the edges crossing the k-partition. The s-Size-k-Cut problem asks one to find a s-size k-cut of minimum capacity for an input graph G. We recover Graph-k-Cut when s is the all 1 vector.

We note that we can also define the fixed-terminal variant, namely s-size k-way-cut. It is **NP-Hard** for $k \geq 3$, and if $k = 2$, it reduces to $O(n^{s_1+s_2-2})$ maximum flow computations. Hence there is a gap between the global and the fixed-terminal variants.

We use the greedy tree packing algorithm by Thorup, which was used to solve Graph-k-Cut. We recall that a k-cut is a set of edges such that after its removal, the graph has at least k components. We recall that $\gamma_k(G)$ is the capacity of the minimum k-cut in G.

Theorem 5.2 ([28])**.** *Let k be a constant and G be a capacitated graph on m edges and n nodes. There exists a set of $\tilde{O}(m)$ spanning trees $\mathcal{T}$, such that for every k-cut F with value at most $\gamma_k(G)$ (i.e. a minimum k-cut), there is a $T \in \mathcal{T}$ such $|F \cap E(T)| \leq 2k-2$.*

A closer observation of the proof shows that [28] actually proves a more general theorem.

Theorem 5.3. *Let k be a constant and G be a capacitated graph on m edges and n nodes. There exists a set of $\tilde{O}(m)$ spanning trees $\mathcal{T}$, such that for every set of edges F with value at most $\gamma_k(G)$, there is a $T \in \mathcal{T}$ such $|F \cap E(T)| \leq 2k-2$. Furthermore, $\mathcal{T}$ can be found in $\tilde{O}(nm) = \tilde{O}(n^3)$ time.*

For any $t \leq k$, the t-cuts with value at most a min-k-cut can be enumerate in polynomial time. First, find $\tilde{O}(m)$ spanning trees of G' described in Theorem 5.3. For each tree, remove all possible choice of $2k-2$ edges, which breaks the tree into components $C_1, \ldots, C_{2k-1}$. We group the components into t groups and consider the union of each group. This will be a candidate partition. See Figure 5.2 for the formal algorithm to generate all candidate t-partitions for input G, t and k.

```
CandidatePartition(G, t, k):
  T ← Õ(m) spanning trees in Theorem 5.3
  for each tree T ∈ T
      for each set of 2(k − 1) edges F in T
          C ← the components of T − F
          for each ordered t partition (C_1, ..., C_t) of C
              for 1 ≤ i ≤ t
                  V_i ← ⋃_{X∈C_i} X
              add (V_1, ..., V_t) as a candidate partition
  output all candidate partitions
```

Figure 5.2: Algorithm for returning all candidate t partitions

In particular, the algorithm in Figure 5.2 takes $\tilde{O}(n^{2(k-1)+1}) = \tilde{O}(n^{2k-1})$ time to compute all candidate partitions. As a consequence, we obtain a faster deterministic algorithm for s-Size-k-Cut.

Theorem 5.4. *Let $s = (s_1, \ldots, s_k)$ and $s_i \geq s_{i+1}$ for all $i \leq k-1$. s-Size-k-Cut for graph G can be solved in $O(n^{2(\sigma - s_1 + 2)})$ time, where $\sigma = \sum_{i=1}^{k} s_i$ and n is the number of nodes of G.*

Proof. Let $\sigma' = \sigma - s_1$. Assume n is at least $s_1(\sigma'+1)$, otherwise brute force takes $O(1)$ time because both σ' and s_1 are constants. We show that the value of the min s-size k-cut is at most the value of a min-$(\sigma'+1)$-cut when $n \geq s_1(\sigma'+1)$. Consider any $(\sigma'+1)$-cut with node partition $V_1, \ldots, V_{\sigma'+1}$, each has size $n_1, \ldots, n_{\sigma'+1}$ respectively. Assume $n_i \geq n_{i+1}$. We will construct a s-size constrained k-cut $U_1, \ldots, U_k$. By pigeonhole principle, $|V_1| \geq s_1$. Let

$U_1 = V_1$. We consider an arbitrary partition of the remaining σ' sets into $k-1$ partition classes, such that the ith partition class contain s_{i+1} of the sets. This is feasible since $\sigma' = \sum_{i=2}^{k} s_i$. Let U_{i+1} be the union of the sets in the ith partition class. The resulting $\{U_1, \ldots, U_k\}$ induces a s-size k-cut that only uses edges in a minimum $(\sigma'+1)$-cut. Hence we have shown that the minimum s-size k-cut is bounded above by minimum $(\sigma'+1)$-cut. Therefore the optimal partition is returned by CandidatePartition$(G, k, \sigma'+1)$, which takes running time $\tilde{O}(n^{2(\sigma-s_1+1)+1})$. Given the partition, it takes $O(m)$ time to evaluate the cut value. Hence the total running time is $\tilde{O}(n^{2(\sigma-s_1+1)}m) = \tilde{O}(n^{2(\sigma-s_1+2)})$. □

5.3 $(S, *, T)$-LINEAR-3-CUT

A set of edges C is a *linear-k-way-cut* for an k-tuple of terminals $(t_1, \ldots, t_k)$, if there is no path from t_i to t_j for all $i < j$ in $G - C$. C is a *linear-k-cut* if it is a linear-k-way-cut for some k-tuple of terminals. Linear-k-Way-Cut and Linear-k-Cut denote the problem of finding a minimum linear-k-way-cut and a minimum linear-k-cut. Linear-k-Way-Cut was studied in approximating multicuts [41]. Linear-k-Way-Cut is **NP-Hard** for all $k \geq 3$. A $\sqrt{2}$-approximation algorithm exists for the case when $k = 3$, and it is tight assuming the Unique Game Conjecture [108]. Linear-k-Way-Cut was introduced by Erbacher et al. in [36]. They showed that the problem is fixed-parameter tractable when parameterized by the size of the solution. It is unknown if Linear-k-Cut is tractable.

As a sub-problem in the algorithm for solving BiCut in Theorem 5.6, we need to study the following problem.

$(s, *, t)$-Linear-3-Cut (abbreviating linear 3-cut)**:** Given a digraph $D = (V, E)$ and two specified nodes $s, t \in V$, find a smallest subset of edges to remove so that there exists a node r with the property that s cannot reach r and t, and r cannot reach t in the resulting graph.

$(s, *, t)$-Linear-3-Cut is a semi-global variant of (s, r, t)-Linear-3-Cut, introduced in [36], where the input specifies three terminals s, r, t and the goal is to find a smallest subset of edges whose removal achieves the property above. A simple reduction from 3-way-Cut shows that (s, r, t)-Linear-3-Cut is **NP-Hard**. The approximability of (s, r, t)-Linear-3-Cut was studied by Chekuri and Madan [109]. They showed that the inapproximability factor coincides with the flow-cut gap of an associated *path-blocking linear program* assuming the Unique Games Conjecture. However, the exact approximability factor is still unknown. On the positive side, there exists a simple combinatorial 2-approximation algorithm for (s, r, t)-Linear-3-Cut.

A 2-approximation for $(s, *, t)$-Linear-3-Cut can be obtained by iterating over all choices

for the terminal r and using the above-mentioned 2-approximation for (s, r, t)-Linear-3-Cut. However, for the purposes of getting a strictly better than 2-approximation for BiCut, we need a strictly better than 2-approximation for $(s, *, t)$-Linear-3-Cut. We obtain the following improved approximation factor:

Theorem 5.5. *There exists a polynomial-time* 3/2*-approximation algorithm for* $(s, *, t)$-Linear-3-Cut.

Remark The 3/2 approximation result was later superseded by [108], since their algorithm can be used to give a $\sqrt{2}$-approximation algorithm.

We emphasize that we do not know if $(s, *, t)$-Linear-3-Cut is **NP-Hard**.

In this section, we prove Theorems 5.5. Theorem 5.5 gives a 3/2-approximation for $(s, *, t)$-Linear-3-Cut and is a necessary component of our proof of Theorem 5.6.

One of our main tools used in the approximation algorithm for BiCut is a 3/2-approximation algorithm for $(s, *, t)$-Linear-3-Cut. We present this algorithm now. We recall the problem $(s, *, t)$-Linear-3-Cut: Given a digraph with specified nodes s, t, find a smallest subset of edges whose removal ensures that the graph contains a node r with the property that s cannot reach r and t, and r cannot reach t.

Notations. Let V be the node set of a graph. For two nodes $s, t \in V$, a subset $X \subseteq V$ is an *$\overline{s}t$-set* if $t \in X \subseteq V - s$. A family $\mathcal{C}$ of subsets of V is a *chain* if for every pair of sets $A, B \in \mathcal{C}$, we have $A \subseteq B$ or $B \subseteq A$. We observe that a chain family can have at most $|V|$ non-empty sets. Two sets A and B are *uncomparable* if $A \setminus B$ and $B \setminus A$ are non-empty, and *comparable* otherwise. A set A is *compatible* with a chain $\mathcal{C}$ if $\mathcal{C} \cup \{A\}$ is a chain, and it is *incompatible* otherwise.

For two sets $A, B \subseteq V$, let

$$\beta(A, B) := |\delta^{in}(A) \cup \delta^{in}(B)|, \text{ and} \tag{5.8}$$

$$\sigma(A, B) := |\delta^{in}(A)| + |\delta^{in}(B)|. \tag{5.9}$$

We first rephrase the problem in a convenient way.

Lemma 5.1. $(s, *, t)$-Linear-3-Cut *in a digraph* $D = (V, E)$ *is equivalent to*

$$\min \{\beta(A, B) : t \in A \subsetneq B \subseteq V - \{s\}\}. \tag{5.10}$$

Proof. Let $F \subseteq E$ be an optimal solution for $(s, *, t)$-Linear-3-Cut in D and let

$$(A, B) := \operatorname{argmin}\{\beta(A, B) : t \in A \subsetneq B \subseteq V - s\}. \tag{5.11}$$

Let us fix an arbitrary node $r \in B - A$. Since the deletion of $\delta^{in}(A) \cup \delta^{in}(B)$ results in a graph with no directed path from s to r, from r to t and from s to t, the edge set $\delta^{in}(A) \cup \delta^{in}(B)$ is a feasible solution to (s, r, t)-Linear-3-Cut in D, thus implying that $|F| \leq \beta(A, B)$.

On the other hand, F is a feasible solution for (s, r', t)-Lin-3-Cut in D for some $r' \in V - \{s, t\}$. Let A' be the set of nodes that can reach t in $D - F$, and R' be the set of nodes that can reach r' in $D - F$. Then, $F \supseteq \delta^{in}(A')$. Moreover, $F \supseteq \delta^{in}(R' \cup A')$ since $R' \cup A'$ has in-degree 0 in $D - F$, and s is not in $R' \cup A'$ because it cannot reach r' and t in $D - F$. Therefore, taking $B' = R' \cup A'$ we get $F \supseteq \delta^{in}(A') \cup \delta^{in}(B')$. □

The above reformulation shows that the optimal solution is given by a chain consisting of two $\overline{s}t$-sets. The following lemma shows that we can obtain a 3/2-approximation to the required chain.

Lemma 5.2. *There exists a polynomial-time algorithm that given a digraph $D = (V, E)$ with nodes $s, t \in V$ returns a pair of $\overline{s}t$-sets $A \subsetneq B \subseteq V$ such that*

$$\beta(A, B) \leq \frac{3}{2} \min\{\beta(A, B) : t \in A \subsetneq B \subseteq V - \{s\}\}. \tag{5.12}$$

Proof. The objective is to find a chain of two $\overline{s}t$-sets A, B with minimum $\beta(A, B)$. We can assume $|V| \geq 4$, otherwise we check all possibilities. To obtain an approximation, we build a chain $\mathcal{C}$ of $\overline{s}t$-sets with the property that, for some value $k \in \mathbb{Z}_+$,

(i) every set $C \in \mathcal{C}$ is an $\overline{s}t$-set with $d^{in}(C) \leq k$, and

(ii) every $\overline{s}t$-set T with $d^{in}(T)$ strictly less than k is in $\mathcal{C}$.

We use the following procedure to obtain such a chain: We initialize with k being the minimum $\overline{s}t$-cut value and $\mathcal{C}$ consisting of the sink-side of a single minimum $\overline{s}t$-cut. In a general step, we find two $\overline{s}t$-sets: an $\overline{s}t$-set Y compatible with the current chain $\mathcal{C}$, i.e. $\mathcal{C} \cup \{Y\}$ forming a chain, with minimum $d^{in}(Y)$ and an $\overline{s}t$-set Z *not* compatible with the current chain $\mathcal{C}$, i.e. crossing at least one member of $\mathcal{C}$, with minimum $d^{in}(Z)$. Note that it is possible that Y or Z does not exist; the former happens when the chain is maximal, while the latter happens when $\mathcal{C} = \{\{t\}\}$ or $\mathcal{C} = \{V - \{s\}\}$ or $\mathcal{C} = \{\{t\}, V - \{s\}\}$. Since $|V| \geq 4$, at least one of Y and Z exist.

The required sets Y and Z can be found in polynomial-time as follows: let $t \in C_1 \subseteq \ldots, \subseteq C_q \subseteq V - s$ denote the members of $\mathcal{C}$, and let $C_0 = \{t\}$, $C_{q+1} = V - s$. (i) Find a minimum cut $Y_i \notin \mathcal{C}$ with $C_i \subseteq Y_i \subseteq C_{i+1}$ for $i = 0, \ldots, q$, and choose Y to be a set with minimum cut value among these cuts. (ii) For each $i \in \{1, \ldots, q\}$ and for each pair x, y of nodes with $y \in C_i \subseteq V - x$, find a minimum cut Z^i_{xy} with $\{t, x\} \subseteq Z^i_{xy} \subseteq V - \{s, y\}$, and choose Z to be a set with minimum cut value among these cuts. Since $\mathcal{C}$ is a chain, we have that $q \leq |V|$ and hence both sets Y and Z can be found in polynomial-time. If Z does not exist or $d^{in}(Y) \leq d^{in}(Z)$, then we add Y to $\mathcal{C}$, and set k to $d^{in}(Y)$; otherwise we set k to $d^{in}(Z)$ and stop.

Proposition 5.2. *Let $\mathcal{C}$ denote the chain before any general step of the above-mentioned procedure. Then, for every $C \in \mathcal{C}$ and for every $\overline{s}t$-set A that is not in $\mathcal{C}$, we have*

$$d^{in}(C) \leq d^{in}(A). \tag{5.13}$$

Proof. Let A be an $\overline{s}t$-set that is not in $\mathcal{C}$. Suppose for the sake of contradiction that $d^{in}(A) < d^{in}(C)$ for some $C \in \mathcal{C}$. Let $\mathcal{C}'$ denote the chain consisting of those members of $\mathcal{C}$ that were added before C. Since $A \notin \mathcal{C}$ and C is a set of minimum cut value compatible with $\mathcal{C}'$, we have that A should cross at least one member of $\mathcal{C}'$. Hence, by $d^{in}(A) < d^{in}(C)$, the procedure stops before adding C to the chain $\mathcal{C}'$, a contradiction to C being in $\mathcal{C}$. □

Proposition 5.3. *The chain $\mathcal{C}$ and the value k obtained at the end of the above-mentioned procedure satisfy* (i) *and* (ii)*.*

Proof. The construction immediately guarantees that every set $C \in \mathcal{C}$ is an $\overline{s}t$-set. By Proposition 5.2 and by construction of $\mathcal{C}$ and k, we have that $d^{in}(C) \leq k$ for every $C \in \mathcal{C}$ and hence, we have (i).

By construction, $\mathcal{C}$ contains all $\overline{s}t$-sets T with $d^{in}(T) < k$ that are compatible with $\mathcal{C}$. Suppose for the sake of contradiction, we have an $\overline{s}t$-set T with $d^{in}(T) < k$ that is not in $\mathcal{C}$. Then, the set T should be incompatible with $\mathcal{C}$. We note that the procedure terminates by setting k to be the minimum cut value $d^{in}(Z)$ among Z that are incompatible with $\mathcal{C}$. Hence, the procedure should have set k to be a value that is at most $d^{in}(T)$ and terminated. This is a contradiction to $d^{in}(T) < k$. Therefore, there does not exist an $\overline{s}t$-set T with $d^{in}(T) < k$ that is not in $\mathcal{C}$ and hence, we have (ii). □

By the above, the procedure stops with a chain $\mathcal{C}$ containing all $\overline{s}t$-sets of cut value less than k, and an $\overline{s}t$-set Z of cut value exactly k which crosses some member X of $\mathcal{C}$. If the optimum value of our problem is less than k, then both members of the optimal pair (A, B)

belong to the chain $\mathcal{C}$, and we can find them by taking the minimum of $\beta(A', B')$ where $A' \subseteq B'$ with $A', B' \in \mathcal{C}$.

We can thus assume that the optimum is at least k. Since $d^{in}(Z) = k$ and $d^{in}(X) \leq k$, the submodularity of the in-degree function implies

$$d^{in}(X \cap Z) + d^{in}(X \cup Z) \leq d^{in}(Z) + d^{in}(X) \leq 2k. \tag{5.14}$$

Hence either $d^{in}(X \cap Z) \leq k$ or $d^{in}(X \cup Z) \leq k$. Since

$$d(X \setminus Z, X \cap Z) + d(Z \setminus X, X \cap Z) \leq d^{in}(X \cap Z) \text{ and} \tag{5.15}$$

$$d(V \setminus (X \cup Z), X \setminus Z) + d(V \setminus (X \cup Z), Z \setminus X) \leq d^{in}(X \cup Z), \tag{5.16}$$

at least one of the following four possibilities holds:

1. $d^{in}(X \cap Z) \leq k$ and $d(X \setminus Z, X \cap Z) \leq \frac{1}{2}k$. Choose $A = X \cap Z$, $B = X$. Then $\beta(A, B) = d(X \setminus Z, X \cap Z) + d^{in}(X) \leq \frac{1}{2}k + k = \frac{3}{2}k$.

2. $d^{in}(X \cap Z) \leq k$ and $d(Z \setminus X, X \cap Z) \leq \frac{1}{2}k$. Choose $A = X \cap Z$, $B = Z$. Then $\beta(A, B) = d(Z \setminus X, X \cap Z) + d^{in}(Z) \leq \frac{1}{2}k + k = \frac{3}{2}k$.

3. $d^{in}(X \cup Z) \leq k$ and $d(V \setminus (X \cup Z), X \setminus Z) \leq \frac{1}{2}k$. Choose $A = Z$, $B = X \cup Z$. Then $\beta(A, B) = d^{in}(Z) + d(V \setminus (X \cup Z), X \setminus Z) \leq k + \frac{1}{2}k = \frac{3}{2}k$.

4. $d^{in}(X \cup Z) \leq k$ and $d(V \setminus (X \cup Z), Z \setminus X) \leq \frac{1}{2}k$. Choose $A = X$, $B = X \cup Z$. Then $\beta(A, B) = d^{in}(X) + d(V \setminus (X \cup Z), Z \setminus X) \leq k + \frac{1}{2}k = \frac{3}{2}k$.

Thus a pair (A, B) can be obtained by taking the minimum among the four possibilities above and $\beta(A', B')$ where $A' \subseteq B'$ with $A', B' \in \mathcal{C}$, concluding the proof of the approximation factor. The algorithm is summarized in Figure 5.3. It remains to ensure that the algorithm can be implemented to run in polynomial-time. We have already seen that Steps 2(a) and 2(b) can be implemented to run in polynomial-time above. Furthermore, the size of the chain $\mathcal{C}$ is at most $|V|$ and hence, Step 3 can also be implemented to run in polynomial time.

□

Theorem 5.5 is a consequence of Lemmas 5.1 and 5.2. The approximation algorithm is summarized in Figure 5.3.

Approximation Algorithm for $(s, *, t)$-Linear-3-Cut

Input: digraph $D = (V, E)$ with $s, t \in V$. We assume $|V| \geq 4$.

1. Let S denote the sink-side of a minimum $s \to t$ cut and α denote its value. Initialize $\mathcal{C} \leftarrow \{S\}$ and $k \leftarrow \alpha$.
2. Repeat:
   (a) $Y \leftarrow \arg\min\{d^{in}(Y) : Y$ is a $\overline{s}t$-set compatible with $\mathcal{C}\}$
   (b) $Z \leftarrow \arg\min\{d^{in}(Z) : Z$ is a $\overline{s}t$-set incompatible with $\mathcal{C}\}$
   (c) If $d^{in}(Y) \leq d^{in}(Z)$, then update $\mathcal{C} \leftarrow \mathcal{C} \cup \{Y\}$ and $k \leftarrow d^{in}(Y)$.
   (d) Else, update $k \leftarrow d^{in}(Z)$, set X to be a set in $\mathcal{C}$ that crosses Z and go to Step 3.
3. Let $(A, B) \leftarrow \arg\min\{\beta(A, B) : A, B \in \mathcal{C}, A \neq B\}$.
4. Let $(S, T) \leftarrow \arg\min\{\beta(X \cap Z, X), \beta(X \cap Z, Z), \beta(Z, X \cup Z), \beta(X, X \cup Z)\}$
5. Return $\arg\min\{\beta(A, B), \beta(S, T)\}$.

Figure 5.3: Approximation Algorithm for $(s, *, t)$-Linear-3-Cut.

5.4 BICUT

The global minimum cut problem in graphs is a classic interdiction problem that admits efficient algorithms. We study the following generalization of this problem from graphs to digraphs:

BiCut: Given a digraph, find a smallest subset of edges whose removal ensures that *there exist* two nodes s and t such that s cannot reach t and t cannot reach s.

A natural approach to solving BiCut is by iterating over all pairs of distinct nodes s and t in the input graph and solving the following fixed-terminal bicut problem:

***st*-BiCut:** Given a digraph with two specified terminal nodes s, t, find a smallest subset of edges whose removal ensures that s cannot reach t and t cannot reach s.

Clearly, *st*-BiCut is equivalent to 2-terminal multiway-cut in digraphs (the goal in k-terminal multiway cut is to remove a smallest subset of edges to ensure that s cannot reach t and t cannot reach s for every pair $\{s, t\}$ of the given k terminals). A classic result by Garg, Vazirani and Yannakakis shows that *st*-BiCut is **NP-Hard** [110]. A simple 2-approximation algorithm is to return the union of a minimum $s \to t$ cut and a minimum $t \to s$ cut in the input digraph. The approximability of *st*-BiCut has seen renewed interest in the last few

months culminating in inapproximability results matching the best-known approximability factor [109, 111]: *st*-BiCut has no efficient $(2-\epsilon)$-approximation for any constant $\epsilon > 0$ assuming the Unique Games Conjecture [37]. These results suggest that we have a very good understanding of the complexity and the approximability of the fixed-terminal variant, i.e., *st*-BiCut. In contrast, even the complexity of the global variant, i.e., BiCut, is still an open problem.

The motivations for studying BiCut are multifold. In several network defense/attack applications, global cuts and connectivity are more important than connectivity between fixed pairs of terminals. On the one hand, BiCut is a fundamental global cut problem with interdiction applications involving digraphs. On the other hand, there is no known complexity theoretic result for BiCut. The fundamental nature of the problem coupled with the lack of basic tractability results are compelling reasons to investigate this problem.

In this section, we exhibit a dichotomy in the approximability of BiCut and *st*-BiCut. While *st*-BiCut is inapproximable to a constant factor better than 2 assuming UGC, we show that BiCut is approximable to a constant factor that is strictly better than 2. The following is our main result:

Theorem 5.6. *There exists a polynomial-time* $(2-1/448)$*-approximation algorithm for* BiCut.

We emphasize that the complexity of BiCut is still an open problem.

In this section, we present our approximation algorithm (Theorem 5.6) for BiCut. We begin with the high-level ideas of the approximation algorithm in Section 5.4.1. The full algorithm and the proof of its approximation ratio are presented in Section 5.4.2.

We recall the problem BiCut: Given a digraph, find a smallest number of edges in it whose removal ensures that there exist two distinct nodes s and t such that s cannot reach t and t cannot reach s. We begin with a reformulation of BiCut that is helpful for the purposes of designing an algorithm. We recall that two sets A and B are *uncomparable* if $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. We also recall that for two sets of nodes A and B, the quantity $\beta(A,B) = |\delta^{in}(A) \cup \delta^{in}(B)|$.

Definition 5.1. *For a digraph* $D = (V, E)$*, let*

$$\beta := \min\{\beta(A,B) : \ A \text{ and } B \text{ are uncomparable}\}. \tag{5.17}$$

The following lemma shows that bicut is equivalent to finding an uncomparable pair of subsets of nodes A, B with minimum $\beta(A,B)$.

Lemma 5.3. *Let $D = (V, E)$ be a digraph. The minimum number of edges in D whose removal ensures that there exist two distinct nodes s and t such that s cannot reach t and t cannot reach s is exactly equal to β.*

Proof. We show the inequality in both directions. Suppose β is attained by two sets $A, B \subseteq V$ such that A and B are uncomparable. Let $s \in A \backslash B$ and $t \in B \backslash A$. For $F := \delta^{in}(A) \cup \delta^{in}(B)$, we consider the graph $D' := D - F$. Since there are no edges entering the set B in D', the node s cannot reach the node t in D'. Since there are no edges entering the set A in D', the node t cannot reach the node s in D'. Thus, the minimum number of edges whose removal ensures that there exist two distinct nodes s and t such that s cannot reach t and t cannot reach s is at most $|F| = \beta$. Suppose F is a smallest set of edges of D such that the graph $D' := D - F$ has two nodes s and t such that s cannot reach t and t cannot reach s. Let A be the set of nodes that can reach s in D' and B be the set of nodes that can reach t in D'. The sets A and B are uncomparable since $s \in A \setminus B$ and $t \in B \setminus A$. Moreover, $|\delta^{in}_{D'}(A) \cup \delta^{in}_{D'}(B)| = 0$. Thus, we have $\beta \leq \beta(A, B) \leq |F|$.

□

Using the above formulation, and by recalling that $\sigma(A, B) = |\delta^{in}(A)| + |\delta^{in}(B)|$, we have the following natural relaxation of bicut:

Definition 5.2. *For a digraph $D = (V, E)$, let*

$$\sigma := \min\{\sigma(A, B) : \ A \text{ and } B \text{ are uncomparable}\}. \tag{5.18}$$

A pair where the latter value is attained is called a minimum uncomparable cut-pair.

5.4.1 Overview of the Approximation Algorithm

In this section, we sketch the argument for a $(2-\epsilon)$-approximation for some small enough ϵ. We observe that for every pair of subsets of nodes (A, B), we have

$$\beta(A, B) = \sigma(A, B) - d(V \setminus (A \cup B), A \cap B). \tag{5.19}$$

Therefore, $\beta(A, B) \leq \sigma(A, B) \leq 2\beta(A, B)$ for every pair of subsets of nodes (A, B) and hence $\beta \leq \sigma \leq 2\beta$. Furthermore, σ can be computed in polynomial-time (see Lemma 5.4), and the optimal solution is a $(2 - \epsilon)$-approximation for BiCut if $\sigma \leq (2 - \epsilon)\beta$. On the other hand, if $\sigma > (2-\epsilon)\beta$, then $d(V \setminus (A \cup B), A \cap B) > (1-\epsilon)\beta$ for every minimizer (A, B) of $\beta(A, B)$, thus providing a structural handle on optimal solutions. Our algorithm proceeds by making

several further attempts at finding pairs (A', B') that could give a $(2-\epsilon)$-approximation. Each attempt that is unsuccessful at giving a $(2-\epsilon)$-approximation implies some structural property of the optimal solution. These structural properties are together exploited by the last attempt to succeed.

Let us fix an uncomparable minimizer (A, B) for $\beta(A, B)$. From (5.19), we note that if $A \cap B$ or $V \setminus (A \cup B)$ is empty, then $\sigma(A, B) = \beta(A, B)$ and hence, computing σ would have found the optimum bicut value already. So, we may assume that $A \cap B$ and $V \setminus (A \cup B)$ are non-empty. In the subsequent attempts, we guess nodes $x \in A \setminus B$, $y \in B \setminus A$, $w \in V \setminus (A \cup B)$, and $z \in A \cap B$. We use the notation $X := A \setminus B$, $Y := B \setminus A$, $W := V \setminus (A \cup B)$, and $Z := A \cap B$ (see Figure 5.4).

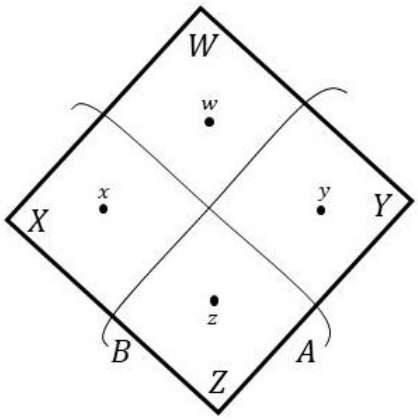


Figure 5.4: The partitioning of the node set in the graph D. Here, (A, B) denotes the optimum bicut that is fixed.

We now observe that A is the sink-side of a $\{w, y\} \to \{x, z\}$-cut while B is the sink-side of a $\{w, x\} \to \{y, z\}$-cut. Our next attempt in the algorithm is to find (X', Y'), where X' is the sink-side of the unique inclusionwise minimal minimum $\{w, y\} \to \{x, z\}$-cut, and Y' is the sink-side of the unique inclusionwise minimal minimum $\{w, x\} \to \{y, z\}$-cut. The hope behind this attempt is that X' could recover A and Y' could recover B as these are feasible solutions to the respective problems and thus, they would together help us recover the optimal solution. Unfortunately, this favorable best-case scenario may not happen. Yet, owing to the feasibility of A and B for the respective problems, we may conclude that $\sigma(X', Y') \leq \sigma(A, B) \leq 2\beta(A, B) = 2\beta$.

Our subsequent attempts are more complex and proceed by modifying X' and Y'. We observe that Z is the sink-side of a $\{w, x, y\} \to \{z\}$-cut. So, our next attempt in the algorithm would be to find Z' as the sink-side of a minimum $\{w, x, y\} \to \{z\}$-cut and expand X' and Y' to include Z' thereby obtaining an uncomparable pair $(A' = X' \cup Z', B' = Y' \cup Z')$.

Our hope is to find a Z' so that the resulting $\beta(A', B')$ is small. While finding Z', we prefer not to have many edges of $E[X'] \cup E[Y']$ in the new bicut (A', B'). This is because such edges enter only one among the two sets A' and B'. (We recall that if we have an uncomparable pair (A', B') with lot of edges from $V \setminus (A' \cup B')$ to $A' \cap B'$, then the value of $\beta(A', B')$ is going to be much less than $\sigma(A', B')$—e.g., see (5.19)—thus leading to a $(2-\epsilon)$-approximation.) So, in order to avoid the edges of $E[X'] \cup E[Y']$ in the new bicut (A', B'), we make such edges more expensive by duplicating them before finding Z'. Let D_1 be the digraph obtained by duplicating the edges in $E[X'] \cup E[Y']$, and let Z' be the sink-side of the minimum $\{w, x, y\} \to \{z\}$-cut in D_1. We then show that the pair $(X' \cup Z', Y' \cup Z')$ is a $(2-\epsilon)$-approximation unless $|\delta^{in}_{D_1}(Z)| > (2-3\epsilon)\beta$, thus giving us more structural handle on the optimum solution.

We next make an analogous attempt by shrinking X' and Y' instead of expanding. Let D_2 be the digraph obtained by duplicating the edges in $E[V \setminus X'] \cup E[V \setminus Y']$, and let W' be the source-side of the minimum $\{w\} \to \{x, y, z\}$-cut in D_2. We obtain that the pair $(X' \setminus W', Y' \setminus W')$ is a $(2-\epsilon)$-approximation unless $|\delta^{out}_{D_2}(W)| > (2-3\epsilon)\beta$.

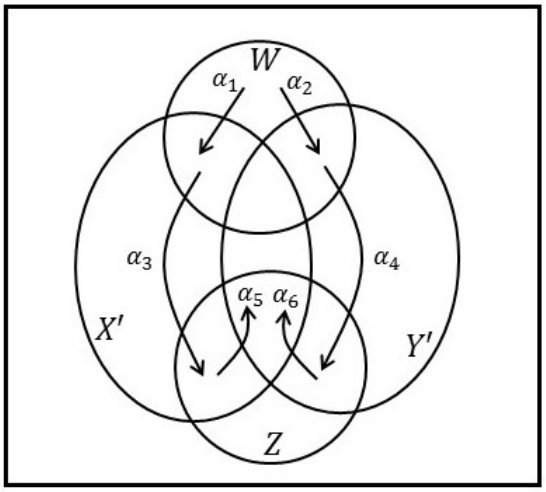


Figure 5.5: The quantities $\alpha_1, \ldots, \alpha_6$.

Let $\alpha_1, \ldots, \alpha_6$ be the number of edges in each position indicated in Figure 5.5. If the attempts so far are unsuccessful, then we use the structural properties derived so far to arrive at the following:

1. All but $O(\epsilon\beta)$ edges in $\delta^{in}(X') \cup \delta^{in}(Y') \cup \delta^{out}(W) \cup \delta^{in}(Z)$ are as positioned in Figure 5.5.

2. The quantities $\alpha_1, \alpha_3, \alpha_5$ are within $O(\epsilon\beta)$ of each other (see (5.87), (5.88), (5.89)) and

so are $\alpha_2, \alpha_4, \alpha_6$.

3. Furthermore, $(1 - O(\epsilon))\beta = \alpha_3 + \alpha_4 \leq \beta$ (see Proposition 5.7).

Without loss of generality, we may assume that $\alpha_3 \geq \alpha_4$. Hence, by conclusion (3) from above, we have that $\alpha_3 \geq \beta/2 - O(\epsilon)\beta$.

Our final attempt in the algorithm to obtain a $(2-\epsilon)$-approximate bicut is to expand Y' by including some nodes from $X' \setminus Y'$ and to shrink X' by excluding some nodes from $X' \setminus Y'$. We now explain the motivation behind this choice of expanding and shrinking. Consider $S := Y' \cup (X' \cap Z)$, which is obtained by expanding Y' by including some nodes from $X' \setminus Y'$ and $T := X' \setminus (X' \cap (W \setminus Y'))$, which is obtained by shrinking X' by excluding some nodes from $X' \setminus Y'$ (see Figure 5.6). By definition, (S, T) is an uncomparable pair. We will now see that the bicut value of (S, T) is much smaller than 2β. Using conclusions (1) and (2) from above, we obtain that

$$\begin{aligned}
\beta(S,T) &= |\delta^{in}(Y' \cup (X' \cap Z)) \cup \delta^{in}(X' \setminus (X' \cap (W \setminus Y')))| \\
&= |\delta^{in}(Y')| - \alpha_5 + \alpha_3 + |\delta^{in}(X')| - \alpha_1 + O(\epsilon)\beta && (5.20)\\
&= \sigma(X', Y') - \alpha_1 - \alpha_5 + \alpha_3 + O(\epsilon)\beta && (5.21)\\
&\leq 2\beta - \alpha_3 + O(\epsilon)\beta && (5.22)\\
&\leq \frac{3}{2}\beta + O(\epsilon)\beta. && (5.23)
\end{aligned}$$

In the above, equation (5.20) is by using conclusion (1), equation (5.21) is by definition of σ, inequality (5.22) is by using conclusion (2) and $\sigma(X', Y') \leq \sigma(A, B) \leq 2\beta$, and inequality (5.23) is because $\alpha_3 \geq \beta/2 - O(\epsilon)\beta$.

Although (S, T) is a good approximation to the optimal bicut, we cannot obtain the sets S and T without the knowledge of W and Z (which, in turn, depend on the optimal bicut (A, B)). Instead, our algorithmic attempt is to expand Y' by including some nodes from $X' \setminus Y'$ and to shrink X' by excluding some nodes from $X' \setminus Y'$. In other words, our candidate is a pair $(B', Y' \cup A')$ for some $X' \cap Y' \subseteq A' \subsetneq B' \subseteq X'$ (we need the condition $A' \subsetneq B'$ because B' and $Y' \cup A'$ should be uncomparable) with minimum $\beta(B', Y' \cup A')$ value. When choosing A' and B', we ignore the edges whose contribution to the objective do not depend on A' and B'. Let H be the digraph obtained by removing the edges in $E[Y' \cup (V \setminus X')]$. Our aim is to minimize $|\delta^{in}_H(B') \cup \delta^{in}_H(Y' \cup A')|$. However, using conclusion (1), we note that this quantity differs from $|\delta^{in}_H(A') \cup \delta^{in}_H(B')|$ by $O(\epsilon\beta)$, so we may instead aim to minimize the latter.

The crucial observation now is that this latter minimization problem is an instance of

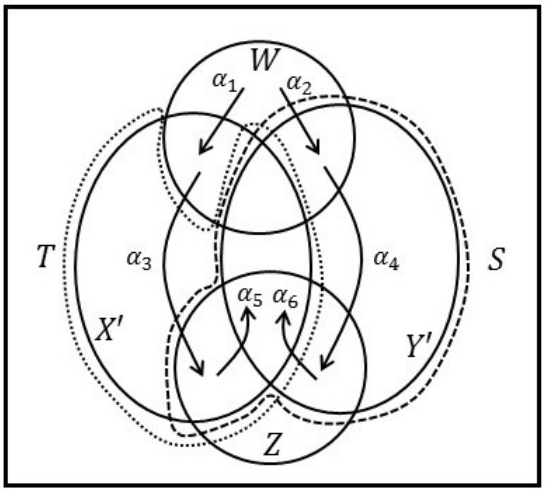


Figure 5.6: The motivation behind the last attempt.

$(s, *, t)$-Linear-3-Cut. While we do not know how to solve $(s, *, t)$-Linear-3-Cut optimally, we can obtain a 3/2-approximation in polynomial-time by Theorem 5.5. By the reformulation of $(s, *, t)$-Linear-3-Cut in Lemma 5.1, we get a pair of subsets (A', B') for which $X' \cap Y' \subseteq A' \subsetneq B' \subseteq X'$ and which is a 3/2-approximation. In particular, $|\delta^{in}_H(A') \cup \delta^{in}_H(B')| \leq (3/2)|\delta^{in}_H((X' \cap (Z \cup Y')) \cup \delta^{in}_H(X' \setminus (W \setminus Y'))| \leq 3(\alpha_3 + O(\epsilon)\beta)/2$. Using this and proceeding similar to the calculations shown above to obtain the bound on $\beta(S, T)$ (i.e., 5.20, 5.21, 5.22, and 5.23), we derive that $\beta(B', Y' \cup A') \leq (7/4 + O(\epsilon))\beta$, concluding the proof.

5.4.2 Approximation Algorithm and Analysis

In this section we prove Theorem 5.6 by giving a polynomial-time $(2 - \epsilon)$-approximation algorithm for BiCut for a constant $\epsilon > 0$. We will describe the algorithm, analyze its approximation factor to show that it is $(2 - \epsilon)$ for some constant $\epsilon > 0$ and compute the value of ϵ at the end of the analysis.

We begin by showing that a natural relaxation of β, namely σ, can be solved.

Lemma 5.4. *For a digraph $D = (V, E)$, there exists a polynomial-time algorithm to find a minimum uncomparable cut-pair.*

Proof. For fixed nodes a and b, there is an efficient algorithm to find A and B such that $a \in A \setminus B$ and $b \in B \setminus A$ and $\sigma(A, B)$ is minimized. Indeed, this is precisely finding the sink side of a min $a \to b$ cut and that of a min $b \to a$ cut. Trying all distinct pairs of nodes a

and b and taking the minimum gives the desired result. □

We need the following definition.

Definition 5.3. *If c is a capacity function on a digraph D, then $d_c^{in}(U) = \sum_{e \in \delta^{in}(U)} c(e)$ is the sum of the capacities of incoming edges of U. Similarly, $d_c^{out}(U) = \sum_{e \in \delta^{out}(U)} c(e)$.*

The rest of the section is devoted to presenting the approximation algorithm and its analysis (i.e., proving Theorem 5.6).

Proof of Theorem 5.6. The algorithm is summarized in Figure 5.7. We first note that the algorithm indeed returns the bicut value of an uncomparable pair. The run-time of the algorithm being polynomial follows from Lemmas 5.2 and 5.4. In the rest of the proof, we analyze the approximation factor. We will show that the algorithm achieves a $(2-\epsilon)$-approximation factor and compute ϵ at the end.

We note that both values μ_1, μ_2 computed by the algorithm are bicut values of uncomparable pairs of sets. Indeed, by definition, it is clear that μ_1 is the bicut value of an uncomparable pair of sets. The value μ_2' computed in Step 3(xii) is also the bicut value of an uncomparable pair of sets: A pair $(P, Q) \in \{(X', Y'), (X' \cup Z', Y' \cup Z'), (X' \setminus W', Y' \setminus W')\}$ is uncomparable since the node $x \in P \setminus Q$ while the node $y \in Q \setminus P$. The pair $(B_1, Y' \cup A_1)$ is uncomparable since the node $y \in (Y' \cup A_1) \setminus B_1$ while the set $B_1 \setminus (Y' \cup A_1)$ is non-empty since $A' \subsetneq B'$.

To analyze the approximation factor, let us fix a minimizer (A, B) for BiCut in the input graph $D = (V, E)$, i.e. fix an uncomparable pair (A, B) such that $\beta(A, B) = \beta$. Let $X := A \setminus B$, $Y := B \setminus A$, $Z := A \cap B$, and $W := V \setminus (A \cup B)$ (see Figure 5.4). With this notation, we have

$$\beta = d(W \cup Y, X) + d(W \cup X, Y) + d^{in}(Z) = d(Y, X \cup Z) + d(X, Y \cup Z) + d^{out}(W). \quad (5.24)$$

We may assume that both Z and W are non-empty, otherwise $\beta(A, B) = \sigma(A, B)$ and consequently, the algorithm finds the optimum since it returns a value $\mu \leq \mu_1 \leq \sigma(A, B) = \beta(A, B)$. Let $\epsilon > 0$ be a constant whose value will be determined later.

Approximation Algorithm for BiCut

Input: digraph $D = (V, E)$

1. Compute $(S, T) \leftarrow \arg\min\{\sigma(S, T) : S \text{ and } T \text{ are uncomparable}\}$ using Lemma 5.4 and set $\mu_1 \leftarrow \beta(S, T)$
2. Initialize $\mu_2 \leftarrow \infty$
3. For each ordered tuple of nodes (x, y, z, w)
   - (i) $X' \leftarrow$ sink-side of the unique inclusionwise minimal minimum $\{w, y\} \to \{x, z\}$-cut
   - (ii) $Y' \leftarrow$ sink-side of the unique inclusionwise minimal minimum $\{w, x\} \to \{y, z\}$-cut
   - (iii) $E_1 \leftarrow E[X'] \cup E[Y']$
   - (iv) $E_2 \leftarrow E[V \setminus X'] \cup E[V \setminus Y']$
   - (v) $D_1 \leftarrow D$ with the arcs in E_1 duplicated
   - (vi) $D_2 \leftarrow D$ with the arcs in E_2 duplicated
   - (vii) $Z' \leftarrow$ sink-side of minimum $\{w, x, y\} \to \{z\}$-cut in D_1
   - (viii) $W' \leftarrow$ source-side of minimum $\{w\} \to \{x, y, z\}$-cut in D_2
   - (ix) $H \leftarrow$ contract $X' \cap Y'$ to z', contract $V \setminus X'$ to w', remove all $w'z'$ arcs
   - (x) In H, find $\overline{w'}z'$-sets $A' \subsetneq B'$ such that $\beta(A', B')$ is at most $(3/2)\min\{\beta(A, B) : z' \in A \subsetneq B \subseteq V - \{w'\}\}$ using Lemma 5.2
   - (xi) $A_1 \leftarrow (A' \setminus \{z'\}) \cup (X' \cap Y')$ and $B_1 \leftarrow (B' \setminus \{z'\}) \cup (X' \cap Y')$
   - (xii) $\mu_2' \leftarrow \min\{\beta(X', Y'), \beta(X' \cup Z', Y' \cup Z'), \beta(X' \setminus W', Y' \setminus W'), \beta(B_1, Y' \cup A_1)\}$.
   - (xiii) If $\mu_2' < \mu_2$, update $\mu_2 \leftarrow \mu_2'$
4. Return $\mu \leftarrow \min\{\mu_1, \mu_2\}$.

Figure 5.7: Approximation Algorithm for BiCut

Lemma 5.5. *If one of the following is true, then $\sigma \leq (2 - \epsilon)\beta$:*

(i) $d(W, Z) \leq (1 - \epsilon)\beta$.

(ii) For every $z \in Z$, there exists a subset U of nodes containing z but not all nodes of Z with $d^{in}(U) < (1 - \epsilon)\beta$.

(iii) For every $w \in W$, there exists a subset U of nodes not containing w but intersecting W with $d^{in}(U) < (1 - \epsilon)\beta$.

Proof. (i) If $d(W, Z) \leq (1-\epsilon)\beta$, then $\sigma(A, B) = \beta(A, B) + d(W, Z) \leq (2-\epsilon)\beta$. The pair (A, B) is uncomparable, and hence $\sigma \leq \sigma(A, B) \leq (2-\epsilon)\beta$.

(ii) Suppose condition *(ii)* holds. This in particular, implies that $|Z| \geq 2$. Since condition *(ii)* holds, there exist sets M with in-degree less than $(1-\epsilon)\beta$ such that $Z \setminus M \neq \emptyset$. Among all such sets, consider a set M with inclusionwise maximal intersection with Z. Let $z \in Z \setminus M$. There exists a set U containing z but not Z with $d^{in}(U) < (1-\epsilon)\beta$. Because of the maximal intersection of M with Z, we have that $M \not\subseteq U$. Hence M and U are uncomparable and therefore $\sigma \leq \sigma(M, U) \leq (2-2\epsilon)\beta$.

(iii) An argument similar to the proof of *(ii)* shows that $\sigma \leq (2-2\epsilon)\beta$ if condition *(iii)* holds.

□

Our aim is to show that the algorithm in Figure 5.7 achieves a $(2-\epsilon)$-approximation. Therefore, we may assume for the rest of the proof that

$$\sigma > (2-\epsilon)\beta \tag{5.25}$$

since otherwise, the algorithm returns $\mu \leq \mu_1 = \sigma \leq (2-\epsilon)\beta$. By Lemma 5.5, we have

$$d(W, Z) \geq (1-\epsilon)\beta. \tag{5.26}$$

We also have nodes $z \in Z$ and $w \in W$ violating conditions *(ii)* and *(iii)* of Lemma 5.5 respectively. Let us fix such nodes, i.e.,

(a) if $|Z| = 1$, then fix $z \in Z$, else if $|Z| \geq 2$, then fix $z \in Z$ such that $d^{in}(U) \geq (1-\epsilon)\beta$ for all subsets U of nodes containing z but not all nodes of Z, and

(b) if $|W| = 1$, then fix $w \in W$, else if $|W| \geq 2$, then fix $w \in W$ such that $d^{in}(U) \geq (1-\epsilon)\beta$ for all subsets U of nodes not containing w but intersecting W.

Also let us fix an arbitrary choice of $x \in X, y \in Y$ (since A and B are uncomparable, we have that X and Y are non-empty and hence such an x and y can be chosen). Henceforth, we will consider the iteration of Step 3 in the algorithm for this choice of x, y, z, w.

We note that (X', Y') form an uncomparable pair since $x \in X' \setminus Y'$ and $y \in Y' \setminus X'$. If $\beta(X', Y') \leq (2-\epsilon)\beta$, then the algorithm returns $\mu \leq \mu_2 \leq (2-\epsilon)\beta$. Therefore, we may assume that

$$\beta(X', Y') \geq (2-\epsilon)\beta. \tag{5.27}$$

Also, we have $d^{in}(X') \leq d^{in}(X \cup Z)$ because X' is the sink-side of a min $\{w, y\} \to \{x, z\}$ cut. Since $d^{in}(X \cup Z) = d^{in}(A) \leq \beta$, we have that

$$d^{in}(X') \leq \beta. \tag{5.28}$$

Similarly,

$$d^{in}(Y') \leq d^{in}(Y \cup Z) \leq \beta. \tag{5.29}$$

Consequently,

$$\sigma(X', Y') \leq d^{in}(X') + d^{in}(Y') \leq 2\beta. \tag{5.30}$$

We consider four cases depending on the relations between W and $X' \cup Y'$, and between Z and $X' \cap Y'$.

Case 0. Suppose $W \cap (X' \cup Y') = \emptyset$ and $Z \subseteq X' \cap Y'$ (see Figure 5.8). In this case $\delta^{in}(X')$ and $\delta^{in}(Y')$ both contain all edges counted in $d(W, Z)$. Hence $\beta(X', Y') \leq \sigma(X', Y') - d(W, Z) \leq (1 + \epsilon)\beta$. The second inequality here is because $\sigma(X', Y') \leq 2\beta$ by (5.30) and $d(W, Z) \geq (1 - \epsilon)\beta$ by (5.26). This shows that (X', Y') is a $(1 + \epsilon)$-approximation. This completes the proof for this case.

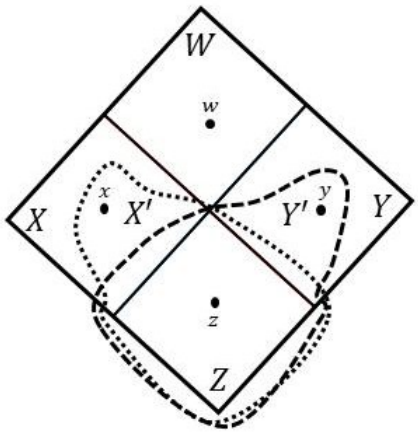


Figure 5.8: The case where $W \cap (X' \cup Y') = \emptyset$ and $Z \subseteq X' \cap Y'$.

For the remaining three cases, we will use the next lemma. Let c be the capacity function obtained by increasing the capacity of each edge in E_1 to 2, and let $\bar{c}$ be the capacity function obtained by increasing the capacity of each edge in E_2 to 2. Recall that Z' is the sink-side of a minimum $\{w, x, y\} \to \{z\}$-cut in D_1, and W' is the source-side of minimum $\{w\} \to \{x, y, z\}$-cut in D_2.

Lemma 5.6. *If $d^{in}(X' \cap Z') \geq (1-\epsilon)\beta$ and $d^{in}(Y' \cap Z') \geq (1-\epsilon)\beta$, then $\beta(X' \cup Z', Y' \cup Z') \leq 2\epsilon\beta + d_c^{in}(Z)$. If $d^{out}(W' \setminus X') \geq (1-\epsilon)\beta$ and $d^{out}(W' \setminus Y') \geq (1-\epsilon)\beta$, then $\beta(X' \setminus W', Y' \setminus W') \leq 2\epsilon\beta + d_c^{out}(W')$.*

Proof. If $d^{in}(X' \cap Z') \geq (1-\epsilon)\beta$, then $d^{in}(X') - d^{in}(X' \cap Z') \leq \epsilon\beta$. So

$$\begin{aligned} d^{in}(X' \cup Z') &= d^{in}(Z') + d^{in}(X') - d^{in}(X' \cap Z') \\ &\qquad - d(X' \setminus Z', Z' \setminus X') - d(Z' \setminus X', X' \setminus Z') \\ &\leq d^{in}(Z') + \epsilon\beta - d(X' \setminus Z', Z' \setminus X') - d(Z' \setminus X', X' \setminus Z'). \end{aligned} \tag{5.31}$$

Hence, we have

$$d^{in}(X' \cup Z') \leq d^{in}(Z') + \epsilon\beta - d(X' \setminus Z', Z' \setminus X'). \tag{5.32}$$

Similarly,

$$d^{in}(Y' \cup Z') \leq d^{in}(Z') + \epsilon\beta - d(Y' \setminus Z', Z' \setminus Y'). \tag{5.33}$$

We need the following proposition.

Proposition 5.4.

$$\begin{aligned} \beta(X' \cup Z', Y' \cup Z') \leq \sigma(X' \cup Z', Y' \cup Z') + d_c^{in}(Z') - 2d^{in}(Z') \\ + d(X' \setminus Z', Z' \setminus X') + d(Y' \setminus Z', Z' \setminus Y'). \end{aligned} \tag{5.34}$$

Proof. By counting the edges entering Z', we have

1. $d_c^{in}(Z') = d^{in}(Z') + |\delta^{in}(Z') \cap E_1|$.
2. $d^{in}(Z') = d(V \setminus (X' \cup Y' \cup Z'), Z') + |\delta^{in}(Z') \cap E_1| + d(X' \setminus Z', Z' \setminus X') + d(Y' \setminus Z', Z' \setminus Y') - d((X' \cap Y') \setminus Z', Z' \setminus (X' \cup Y'))$.

The first equation can be rewritten as

$$d_c^{in}(Z') - 2d^{in}(Z') = -d^{in}(Z') + |\delta^{in}(Z') \cap E_1|. \tag{5.35}$$

Using this and the second equation, we get

$$d_c^{in}(Z') - 2d^{in}(Z') + d(X' \setminus Z', Z' \setminus X') + d(Y' \setminus Z', Z' \setminus Y') \tag{5.36}$$

$$= -d(V \setminus (X' \cup Y' \cup Z'), Z') + d((X' \cap Y') \setminus Z', Z' \setminus (X' \cup Y')). \tag{5.37}$$

Thus, the desired inequality (5.34) simplifies to

$$\beta(X' \cup Z', Y' \cup Z') \leq \sigma(X' \cup Z', Y' \cup Z') - d(V \setminus (X' \cup Y' \cup Z'), Z') \tag{5.38}$$
$$+ d((X' \cap Y') \setminus Z', Z' \setminus (X' \cup Y')). \tag{5.39}$$

To prove this inequality, we observe that the edges counted by $d(V \setminus (X' \cup Y' \cup Z'), Z')$ are counted twice in $\sigma(X' \cup Z', Y' \cup Z')$. Hence we have the desired relation (5.34). □

Using (5.34), (5.33) and (5.32) we get

$$\beta(X' \cup Z', Y' \cup Z') \leq d^{in}(X' \cup Z') + d^{in}(Y' \cup Z') + d_c^{in}(Z') - 2d^{in}(Z') \tag{5.40}$$
$$+ d(X' \setminus Z', Z' \setminus X') + d(Y' \setminus Z', Z' \setminus Y') \tag{5.41}$$
$$\leq d^{in}(Z') + \epsilon\beta + d^{in}(Z') + \epsilon\beta + d_c^{in}(Z') - 2d^{in}(Z') \tag{5.42}$$
$$= 2\epsilon\beta + d_c^{in}(Z') \tag{5.43}$$
$$\leq 2\epsilon\beta + d_c^{in}(Z).$$

The last inequality above is because Z is a feasible solution for the minimization problem that obtains Z' and hence $d_c^{in}(Z') \leq d_c^{in}(Z)$. This completes the proof of the first part of the lemma. The second part follows by a symmetric argument. □

We are now ready to prove the three remaining cases.

Case 1. Suppose $W \cap (X' \cup Y') = \emptyset$ and $Z \not\subseteq X' \cap Y'$. Without loss of generality, let $Z \not\subseteq X'$. This implies that $|Z| \geq 2$. The set $X' \cap Z'$ contains z but not the whole Z, hence $d^{in}(X' \cap Z') \geq (1-\epsilon)\beta$ by (a).

We first consider the subcase where $d^{in}(Y' \cap Z') < (1-\epsilon)\beta$. By the choice of the node z and (a), this means that $Z \subseteq Y' \cap Z'$. In this case $Y' \cap Z'$ crosses X', because X' does not contain all nodes in Z, and $Y' \cap Z'$ does not contain x. Thus $(X', Y' \cap Z')$ is an uncomparable pair. Now we observe that $\sigma(X', Y' \cap Z') = d^{in}(X') + d^{in}(Y' \cap Z') \leq (2-\epsilon)\beta$. Thus, $\sigma \leq (2-\epsilon)\beta$, a contradiction to (5.25).

Next we consider the other subcase where $d^{in}(Y' \cap Z') \geq (1-\epsilon)\beta$. Then, by Lemma 5.6, we get

$$\beta(X' \cup Z', Y' \cup Z') \leq 2\epsilon\beta + d_c^{in}(Z). \tag{5.44}$$

We are in the case where $(X' \cup Y') \cap W = \emptyset$, so $d_c^{in}(Z) \leq d^{in}(Z) + d(X, Z) + d(Y, Z)$. We now note that $d^{in}(Z) + d(X, Z) + d(Y, Z) = 2d^{in}(Z) - d(W, Z) \leq 2\beta - (1-\epsilon)\beta = (1+\epsilon)\beta$ since $d(W, Z) \geq (1-\epsilon)\beta$ and $d^{in}(Z) \leq \beta$ which follows from (5.24). Hence we have

$\beta(X' \cup Z', Y' \cup Z') \leq (1+3\epsilon)\beta$. Since $(X' \cup Z', Y' \cup Z')$ is an uncomparable pair, we have that $\mu_2 \leq (1+3\epsilon)\beta$.

Case 2. Suppose $W \cap (X' \cup Y') \neq \emptyset$ and $Z \subseteq X' \cap Y'$. This is similar to Case 1 by symmetry. The uncomparable pair of sets that are of interest in this case are $(X' \setminus W', Y' \setminus W')$.

Case 3. Suppose $W \cap (X' \cup Y') \neq \emptyset$ and $Z \not\subseteq X' \cap Y'$. Consequently, we have that $|Z|, |W| \geq 2$ and hence by (a) and (b), we have that $d^{in}(U) \geq (1-\epsilon)\beta$ for all subsets U of nodes containing z but not all nodes of Z and for all subsets U of nodes not containing w but intersecting W. For the rest of the proof, we may also assume that

$$\mu_2 > (2-\epsilon)\beta \tag{5.45}$$

for otherwise, the algorithm returns $\mu \leq \mu_2 \leq (2-\epsilon)\beta$ and we are done. With this, we have the following proposition.

Proposition 5.5.

$$d_c^{in}(Z) \geq (2-3\epsilon)\beta, \ \textit{and} \tag{5.46}$$

$$d_{\bar{c}}^{out}(W) \geq (2-3\epsilon)\beta. \tag{5.47}$$

Proof. We know that $Z \not\subseteq X' \cap Y'$. Without loss of generality, suppose $Z \not\subseteq X'$. The set $X' \cap Z'$ contains z but not the whole Z, hence $d^{in}(X' \cap Z') \geq (1-\epsilon)\beta$. By the same argument as in the first subcase of Case 1 (first paragraph), we may assume that $d^{in}(Y' \cap Z') \geq (1-\epsilon)\beta$ (otherwise, $\sigma \leq (2-\epsilon)\beta$, a contradiction to (5.25)). The inequality $\beta(X' \cup Z', Y' \cup Z') \leq 2\epsilon\beta + d_c^{in}(Z)$ holds using Lemma 5.6. If $d_c^{in}(Z) \leq (2-3\epsilon)\beta$, then these imply $\beta(X' \cup Z', Y' \cup Z') \leq (2-\epsilon)\beta$. Since $(X' \cup Z', Y' \cup Z')$ is an uncomparable pair, we would thus have $\mu_2 \leq (2-\epsilon)\beta$, a contradiction to (5.45). Similarly, if $d_{\bar{c}}^{out}(W) \leq (2-3\epsilon)\beta$, then we obtain $\mu_2 \leq \beta(X' \setminus W', Y' \setminus W') \leq (2-\epsilon)\beta$, a contradiction to (5.45). Thus, we have the conclusion. □

Let us define the following quantities (see Figure 5.5):

1. $\alpha_1 := d(W \setminus (X' \cup Y'), W \cap (X' \setminus Y'))$,
2. $\alpha_2 := d(W \setminus (X' \cup Y'), W \cap (Y' \setminus X'))$,
3. $\alpha_3 := d(W \cap (X' \setminus Y'), Z \cap (X' \setminus Y'))$,
4. $\alpha_4 := d(W \cap (Y' \setminus X'), Z \cap (Y' \setminus X'))$,

5. $\alpha_5 := d(Z \cap (X' \setminus Y'), X' \cap Y' \cap Z)$, and

6. $\alpha_6 := d(Z \cap (Y' \setminus X'), X' \cap Y' \cap Z)$.

In propositions 5.6, 5.7, 5.8, 5.9, 5.10 and 5.11, we show a sequence of inequalities involving these quantities.

Proposition 5.6. *Each of the values $d^{in}(X' \cap Y')$, $d^{in}(X' \cup Y')$, $d^{in}(X' \cap Z)$, $d^{in}(X' \cup Z)$, $d^{in}(Y' \cap Z)$, $d^{in}(Y' \cup Z)$ is at least $(1-\epsilon)\beta$ and is at most $(1+\epsilon)\beta$.*

Proof. By submodularity,

$$d^{in}(X' \cap Y') + d^{in}(X' \cup Y') \leq d^{in}(X') + d^{in}(Y') \leq 2\beta. \tag{5.48}$$

We note that $d^{in}(X' \cap Y') \geq (1-\epsilon)\beta$ by the choice of the node z. This shows $d^{in}(X' \cup Y') \leq (1+\epsilon)\beta$. Similarly, $d^{in}(X' \cup Y') \geq (1-\epsilon)\beta$ by the choice of the node w, and hence $d^{in}(X' \cap Y') \leq (1+\epsilon)\beta$.

We argue the bounds for $d^{in}(X' \cup Z)$ and $d^{in}(X' \cap Z)$. The bounds for $d^{in}(Y' \cup Z)$ and $d^{in}(Y' \cap Z)$ follow using a similar proof strategy. By the assumption of Case 3, we have $Z \not\subseteq X' \cap Y'$. We will argue that $d^{in}(X' \cap Z) \geq (1-\epsilon)\beta$ by considering two sub-cases. Sub-case (i): Suppose $Z \not\subseteq X'$. Hence $X' \cap Z$ contains z but not all of Z. By the choice of the node z, we have $d^{in}(X' \cap Z) \geq (1-\epsilon)\beta$. Sub-case (ii): Suppose $Z \subsetneq X'$. Then, $d^{in}(X' \cap Z) = d^{in}(Z)$. We have $d^{in}(Z) \geq d(W, Z) \geq (1-\epsilon)\beta$ using (5.26) and hence, $d^{in}(X' \cap Z) \geq (1-\epsilon)\beta$.

By submodularity,

$$d^{in}(X' \cup Z) \leq d^{in}(X') + d^{in}(Z) - d^{in}(X' \cap Z) \leq 2\beta - (1-\epsilon)\beta = (1+\epsilon)\beta. \tag{5.49}$$

Next, we notice that $X' \cup Z$ and Y' are uncomparable, so $\sigma(X' \cup Z, Y') \geq (2-\epsilon)\beta$ by (5.25). However, we have

$$\sigma(X' \cup Z, Y') = d^{in}(X' \cup Z) + d^{in}(Y') \leq d^{in}(X' \cup Z) + \beta. \tag{5.50}$$

Hence, $d^{in}(X' \cup Z) \geq (1-\epsilon)\beta$. Using submodularity, we obtain $d^{in}(X' \cap Z) \leq (1+\epsilon)\beta$. □

Proposition 5.7. $(1-6\epsilon)\beta \leq \alpha_3 + \alpha_4 \leq \beta$.

Proof. We have $\alpha_3 + \alpha_4 \leq d(W, Z)$ by definition of α_3 and α_4. Moreover, $d(W, Z) \leq \beta$ by (5.24). Hence, we have the upper bound that $\alpha_3 + \alpha_4 \leq \beta$. We next show the lower bound. From (5.46), we recall that $(2-3\epsilon)\beta \leq d_c^{in}(Z) = d^{in}(Z) + |\delta^{in}(Z) \cap E_1|$ and from (5.47), we

recall that $(2-3\epsilon)\beta \leq d^{out}_{\overline{c}}(W) = d^{out}(W) + |\delta^{out}(W) \cap E_2|$. Moreover, we have $d^{in}(Z) \leq \beta$ and $d^{out}(W) \leq \beta$ by (5.24).

Let C be the set of edges from W to Z, i.e. those counted by $d(W, Z)$. We next argue that $\alpha_3 + \alpha_4 = |C \cap E_1 \cap E_2|$. In order to show this equality, we show the inequality in both directions. For the first direction, we observe that every edge e that is counted by $\alpha_3 + \alpha_4$ is in C as well as E_1 as well as E_2, and hence $\alpha_3 + \alpha_4 \leq |C \cap E_1 \cap E_2|$. For the other direction, consider $e \in C \cap E_1 \cap E_2$. Then, e is counted either in α_3 or α_4 but not both. Hence, $|C \cap E_1 \cap E_2| \leq \alpha_3 + \alpha_4$.

Let $a := |\delta^{in}(Z) \setminus C|$ and $b := |\delta^{out}(W) \setminus C|$. Using (5.24), we have

$$\begin{aligned}
\beta &= d(W \cup Y, X) + d(W \cup X, Y) + d^{in}(Z) && (5.51)\\
&\geq d(W, X) + d(W, Y) + d^{in}(Z) && (5.52)\\
&= b + d^{in}(Z) && (5.53)\\
&= b + |C| + |\delta^{in}(Z) \setminus C| && (5.54)\\
&= b + |C| + a. && (5.55)
\end{aligned}$$

Thus, we have $|C| + a + b \leq \beta$. Furthermore, we have $|C \cap E_1| \geq |\delta^{in}(Z) \cap E_1| - a$ and $|C \cap E_2| \geq |\delta^{out}(W) \cap E_2| - b$. From all the above, we get the following sequence of inequalities that shows the lower bound:

$$\begin{aligned}
|C \cap E_1 \cap E_2| \geq& |C| - |C \setminus E_1| - |C \setminus E_2| && (5.56)\\
=& |C| - (|C| - |C \cap E_1|) - (|C| - |C \cap E_2|) && (5.57)\\
=& |C \cap E_1| + |C \cap E_2| - |C| && (5.58)\\
\geq& |\delta^{in}(Z) \cap E_1| - a + |\delta^{out}(W) \cap E_2| - b - |C| && (5.59)\\
\geq& (2-3\epsilon)\beta - d^{in}(Z) + (2-3\epsilon)\beta - d^{out}(W) - (a + b + |C|) && (5.60)\\
\geq& (4-6\epsilon)\beta - 3\beta && (5.61)\\
=& (1-6\epsilon)\beta.
\end{aligned}$$

□

Proposition 5.8. *$(1-8\epsilon)\beta \leq \alpha_1 + \alpha_2 \leq (1+\epsilon)\beta$ and $(1-8\epsilon)\beta \leq \alpha_5 + \alpha_6 \leq (1+\epsilon)\beta$.*

Proof. We first show the upper bounds. We have $\alpha_1 + \alpha_2 \leq d^{in}(X' \cup Y')$ which is at most $(1+\epsilon)\beta$ by Proposition 5.6. Similarly, we have $\alpha_5 + \alpha_6 \leq d^{in}(X' \cap Y') \leq (1+\epsilon)\beta$. We next show the lower bounds.

We first note that

$$\alpha_5 + \alpha_6 \geq d^{in}(X' \cap Y' \cap Z) - |\delta^{in}(Z) \cap \delta^{in}(X' \cap Y' \cap Z)| \\ - d(V \setminus (X' \cup Y'), X' \cap Y' \cap Z). \tag{5.62}$$

We bound each of the terms in the RHS now. We observe that $X' \cap Y' \cap Z$ contains z but not all nodes in Z, hence

$$d^{in}(X' \cap Y' \cap Z) \geq (1 - \epsilon)\beta. \tag{5.63}$$

Moreover, we have

$$|\delta^{in}(X') \cap \delta^{in}(Y')| = d^{in}(X') + d^{in}(Y') - |\delta^{in}(X') \cup \delta^{in}(Y')| \tag{5.64}$$
$$= \sigma(X', Y') - \beta(X', Y') \tag{5.65}$$
$$\leq 2\beta - (2 - \epsilon)\beta \qquad \text{(Using (5.30) and (5.27))} \tag{5.66}$$
$$= \epsilon\beta. \tag{5.67}$$

Here, $|\delta^{in}(X') \cap \delta^{in}(Y')| \leq \epsilon\beta$ implies that we have at most $\epsilon\beta$ edges entering $X' \cap Y' \cap Z$ from $V \setminus (X' \cup Y')$. Thus, we have

$$d(V \setminus (X' \cup Y'), X' \cap Y' \cap Z) \leq \epsilon\beta. \tag{5.68}$$

We further have $|\delta^{in}(Z) \cap \delta^{in}(X' \cap Y' \cap Z)| \leq d^{in}(Z) - \alpha_3 - \alpha_4$. Using Proposition 5.7, we obtain that

$$|\delta^{in}(Z) \cap \delta^{in}(X' \cap Y' \cap Z)| \leq d^{in}(Z) - \alpha_3 - \alpha_4 \leq 6\epsilon\beta. \tag{5.69}$$

Substituting the bounds from (5.63), (5.68), and (5.69) in (5.62), we obtain that $\alpha_5 + \alpha_6 \geq (1 - \epsilon)\beta - 6\epsilon\beta - \epsilon\beta = (1 - 8\epsilon)\beta$.

We proceed by a similar argument now to show the lower bound for $\alpha_1 + \alpha_2$. We note that

$$\alpha_1 + \alpha_2 \geq d^{out}(W \setminus (X' \cup Y')) - |\delta^{out}(W) \cap \delta^{out}(W \setminus (X' \cup Y'))| \\ - d(W \setminus (X' \cup Y'), X' \cap Y'). \tag{5.70}$$

We bound each of the terms in the RHS now. We observe that $d^{out}(W \setminus (X' \cup Y')) = d^{in}(V \setminus (W \setminus (X' \cup Y')))$. The set $V \setminus (W \setminus (X' \cup Y'))$ does not contain w but intersects W, hence

$$d^{out}(W \setminus (X' \cup Y')) = d^{in}(V \setminus (W \setminus (X' \cup Y'))) \geq (1 - \epsilon)\beta. \tag{5.71}$$

Moreover, by $|\delta^{in}(X') \cap \delta^{in}(Y')| \leq \epsilon\beta$ derived as above, we have at most $\epsilon\beta$ edges entering $X' \cap Y'$ from $W \setminus (X' \cup Y')$. Thus, we have

$$d(W \setminus (X' \cup Y'), X' \cap Y') \leq \epsilon\beta. \tag{5.72}$$

We further have $|\delta^{out}(W) \cap \delta^{out}(W \setminus (X' \cup Y'))| \leq d^{out}(W) - \alpha_3 - \alpha_4$. Using Proposition 5.7, we obtain that

$$|\delta^{out}(W) \cap \delta^{out}(W \setminus (X' \cup Y'))| \leq d^{out}(W) - \alpha_3 - \alpha_4 \leq 6\epsilon\beta. \tag{5.73}$$

Substituting the bounds from (5.71), (5.72), and (5.73) in (5.70), we obtain that $\alpha_1 + \alpha_2 \geq (1-\epsilon)\beta - 6\epsilon\beta - \epsilon\beta = (1-8\epsilon)\beta$. □

Proposition 5.9. $(1-16\epsilon)\beta \leq \alpha_1 + \alpha_6 \leq \beta$ *and* $(1-16\epsilon)\beta \leq \alpha_2 + \alpha_5 \leq \beta$.

Proof. The upper bounds follow by $\alpha_1 + \alpha_6 \leq d^{in}(X') \leq \beta$ and $\alpha_2 + \alpha_5 \leq d^{in}(Y') \leq \beta$. On the other hand, combining the two inequalities in Proposition 5.8 gives $(2-16\epsilon)\beta \leq \alpha_1 + \alpha_2 + \alpha_5 + \alpha_6$. Now using the upper bound $\alpha_2 + \alpha_5 \leq \beta$ gives $(1-16\epsilon)\beta \leq \alpha_1 + \alpha_6$. Similarly, we obtain $(1-16\epsilon)\beta \leq \alpha_2 + \alpha_5$. □

Proposition 5.10. $(1-23\epsilon)\beta \leq \alpha_3 + \alpha_6 \leq (1+\epsilon)\beta$.

Proof. Consider the set $M := X' \cap Z$. We note that $\alpha_3 + \alpha_6 \leq d^{in}(X' \cap Z)$. By Proposition 5.6, we have $d^{in}(M) \leq (1+\epsilon)\beta$, which gives the upper bound. We now show the lower bound.

By Proposition 5.6, we have

$$(1-\epsilon)\beta \leq d^{in}(M). \tag{5.74}$$

Next we have

$$d^{in}(M) = \alpha_6 + d((Z \setminus X') \cap Y', M \setminus Y') + d(Z \setminus (X' \cup Y'), M) + d(V \setminus Z, M). \tag{5.75}$$

Also,

$$\alpha_1 + \alpha_6 + d((Z \setminus X') \cap Y', M \setminus Y') + d(Z \setminus (X' \cup Y'), M) \leq d^{in}(X') \leq \beta.$$

Using Proposition 5.9, we thus obtain

$$d((Z \setminus X') \cap Y', M \setminus Y') + d(Z \setminus (X' \cup Y'), M) \leq 16\epsilon\beta. \tag{5.76}$$

We next note that $\alpha_4 + d(V \setminus Z, M) \leq d^{in}(Z) \leq \beta$. Using Proposition 5.7, we have $(1 - 6\epsilon)\beta - \alpha_3 \leq \alpha_4$. We thus obtain

$$d(V \setminus Z, M) \leq 6\epsilon\beta + \alpha_3. \tag{5.77}$$

Using (5.74), (5.75), (5.76), and (5.77), we obtain

$$\begin{aligned}
(1-\epsilon)\beta &\leq d^{in}(M) && (5.78)\\
&= \alpha_6 + d((Z \setminus X') \cap Y', M \setminus Y') && (5.79)\\
&\qquad + d(Z \setminus (X' \cup Y'), M) + d(V \setminus Z, M) && (5.80)\\
&\leq \alpha_6 + 16\epsilon\beta + \alpha_3 + 6\epsilon\beta && (5.81)\\
&= \alpha_3 + \alpha_6 + 22\epsilon\beta. && (5.82)
\end{aligned}$$

Rewriting the final inequality gives $(1 - 23\epsilon)\beta \leq \alpha_3 + \alpha_6$. □

Proposition 5.11. $\alpha_1 + \alpha_5 \geq 2\alpha_3 - 51\epsilon\beta$.

Proof. The above propositions give us a chain of relations:

$$\begin{aligned}
(1 - 16\epsilon)\beta - \alpha_6 \leq \alpha_1 \leq \beta - \alpha_6, && (5.83)\\
(1 - 8\epsilon)\beta - \alpha_1 \leq \alpha_2 \leq (1 + \epsilon)\beta - \alpha_1, && (5.84)\\
(1 - 16\epsilon)\beta - \alpha_2 \leq \alpha_5 \leq \beta - \alpha_2, && (5.85)\\
(1 - 23\epsilon)\beta - \alpha_3 \leq \alpha_6 \leq (1 + \epsilon)\beta - \alpha_3. && (5.86)
\end{aligned}$$

By substitution, we get

$$\begin{aligned}
\alpha_3 - 17\epsilon\beta \leq \alpha_1 \leq \alpha_3 + 23\epsilon\beta, && (5.87)\\
\alpha_1 - 17\epsilon\beta \leq \alpha_5 \leq \alpha_1 + 8\epsilon\beta. && (5.88)
\end{aligned}$$

By substituting again, we get

$$\alpha_3 - 34\epsilon\beta \leq \alpha_5 \leq \alpha_3 + 31\epsilon\beta. \tag{5.89}$$

Using (5.87) and (5.89), we obtain $\alpha_1 + \alpha_5 \geq 2\alpha_3 - 51\epsilon\beta$. □

Without loss of generality, we may assume that $\alpha_3 \geq (\alpha_3 + \alpha_4)/2$, since if not, there is another iteration of the algorithm where x and y are switched and thus, the unique choices

of X' and Y' also get switched. Therefore, by Proposition 5.7, we have

$$\alpha_3 \geq (1/2 - 3\epsilon)\beta. \tag{5.90}$$

Let H be the digraph obtained in Step 3(ix) of the algorithm, i.e., by contracting $X' \cap Y'$ to a node z', contracting $V \setminus X'$ to a node w', and removing all $w'z'$ arcs. Let

$$A_0 := ((X' \cap Z) \setminus Y') \cup \{z'\} \text{ and} \tag{5.91}$$
$$B_0 := (X' \setminus (W \cup Y')) \cup \{z'\}. \tag{5.92}$$

We note that (A_0, B_0) is a feasible solution for Step 3(x) of the algorithm: both sets contain z' and do not contain w' by definition and moreover $A_0 \subsetneq B_0$ since the node x is in $B_0 \setminus A_0$. The following proposition shows an upper bound on the value of $\beta(A_0, B_0)$ in H:

Proposition 5.12.

$$|\delta_H^{in}(A_0) \cup \delta_H^{in}(B_0)| \leq \alpha_3 + 39\epsilon\beta. \tag{5.93}$$

Proof. For notational convenience, we will use $d(P, Q)$ to denote $d_D(P, Q)$ for two subsets $P, Q \subseteq V$. We have that

$$\begin{aligned} |\delta_H^{in}(A_0)| &= |\delta_H(V(H) \setminus A_0, (X' \cap Z) \setminus Y')| + |\delta_H(X' \setminus (Y' \cup Z), z')| \\ &= d(V \setminus ((X' \cap Z) \cup (X' \cap Y')), (X' \cap Z) \setminus Y') \\ &\quad + d(X' \setminus (Y' \cup Z), X' \cap Y'), \end{aligned} \tag{5.94}$$

and

$$\begin{aligned} |\delta_H^{in}(B_0) \setminus \delta_H^{in}(A_0)| &= d(V \setminus X', X' \setminus (Y' \cup W \cup Z)) \\ &\quad + d((X' \cap W) \setminus Y', X' \setminus (Y' \cup W \cup Z)). \end{aligned} \tag{5.95}$$

We would like to bound the sum of the above four terms. We further decompose the first term as follows:

$$\begin{aligned} d(V \setminus ((X' \cap Z) \cup (X' \cap Y')), (X' \cap Z) \setminus Y') = \\ d(Z \setminus X', (X' \cap Z) \setminus Y') \\ + d(V \setminus ((X' \cap Z) \cup (X' \cap Y') \cup Z), (X' \cap Z) \setminus Y'). \end{aligned} \tag{5.96}$$

We note that $d(V \setminus ((X' \cap Z) \cup (X' \cap Y') \cup Z), (X' \cap Z) \setminus Y')$ counts a subset of the edges entering Z. Since we have $d(W, Z) \geq (1-\epsilon)\beta$, while $d^{in}(Z) \leq \beta$, it follows that all but $\epsilon\beta$ edges entering Z are from W. Hence,

$$\begin{aligned} & d(V \setminus ((X' \cap Z) \cup (X' \cap Y') \cup Z), (X' \cap Z) \setminus Y') \\ & \quad \leq d((V \setminus ((X' \cap Z) \cup (X' \cap Y') \cup Z)) \cap W, (X' \cap Z) \setminus Y') + \epsilon\beta \\ & \quad = d(W \setminus (X' \cap Y'), (X' \cap Z) \setminus Y') + \epsilon\beta. \end{aligned} \tag{5.97}$$

The last equation above is because the set $(V \setminus ((X' \cap Z) \cup (X' \cap Y') \cup Z)) \cap W$ is precisely $W \setminus (X' \cap Y')$. Hence, using (5.94), (5.95), (5.96), and (5.97), we have that $|\delta_H^{in}(A_0) \cup \delta_H^{in}(B_0)| - \epsilon\beta$ is at most

$$\begin{aligned} & d(Z \setminus X', (X' \cap Z) \setminus Y') + d(W \setminus (X' \cap Y'), (X' \cap Z) \setminus Y') \\ & \quad + d(X' \setminus (Y' \cup Z), X' \cap Y') \\ & \quad + d(V \setminus X', X' \setminus (Y' \cup W \cup Z)) \\ & \quad + d((X' \cap W) \setminus Y', X' \setminus (Y' \cup W \cup Z)). \end{aligned} \tag{5.98}$$

We now bound this sum by suitably grouping the terms.

1. The first term $d(Z \setminus X', (X' \cap Z) \setminus Y')$ and the fourth term $d(V \setminus X', X' \setminus (Y' \cup W \cup Z))$ together count a subset of the edges entering X'. We have

   $$d(Z \setminus X', (X' \cap Z) \setminus Y') + d(V \setminus X', X' \setminus (Y' \cup W \cup Z)) + \alpha_1 + \alpha_6 \leq d^{in}(X') \leq \beta. \tag{5.99}$$

   Using $\alpha_1 + \alpha_6 \geq (1 - 16\epsilon)\beta$ from Proposition 5.9, we obtain

   $$d(Z \setminus X', (X' \cap Z) \setminus Y') + d(V \setminus X', X' \setminus (Y' \cup W \cup Z)) \leq 16\epsilon\beta. \tag{5.100}$$

2. The third term $d(X' \setminus (Y' \cup Z), X' \cap Y')$ counts a subset of the edges entering Y'. We have

   $$d(X' \setminus (Y' \cup Z), X' \cap Y') + \alpha_2 + \alpha_5 \leq d^{in}(Y') \leq \beta. \tag{5.101}$$

   Using $\alpha_2 + \alpha_5 \geq (1 - 16\epsilon)\beta$ from Proposition 5.9, we obtain

   $$d(X' \setminus (Y' \cup Z), X' \cap Y') \leq 16\epsilon\beta. \tag{5.102}$$

3. The second term $d(W \setminus (X' \cap Y'), (X' \cap Z) \setminus Y')$ and the fifth term $d((X' \cap W) \setminus Y', X' \setminus$

$(Y' \cup W \cup Z))$ together count a subset of the edges leaving W. We have

$$d(W \setminus (X' \cap Y')), (X' \cap Z) \setminus Y') + d((X' \cap W) \setminus Y', X' \setminus (Y' \cup W \cup Z)) + \alpha_4 \leq d^{out}(W) \leq \beta.$$

Using $\alpha_3 + \alpha_4 \geq (1 - 6\epsilon)\beta$ from Proposition 5.7, we obtain

$$d(W \setminus (X' \cap Y'), (X' \cap Z) \setminus Y') + d((X' \cap W) \setminus Y', X' \setminus (Y' \cup W \cup Z)) \leq 6\epsilon\beta + \alpha_3. \quad (5.103)$$

Thus, the total contribution of the five terms is at most $38\epsilon\beta + \alpha_3$, thus proving the proposition.

□

Using Proposition 5.12, Step 3(x) of the algorithm finds $\overline{w'}z'$-sets $A' \subsetneq B'$ such that

$$\begin{aligned} |\delta^{in}_H(A') \cup \delta^{in}_H(B')| &\leq \frac{3}{2}|\delta^{in}_H(A_0) \cup \delta^{in}_H(B_0)| \\ &\leq \frac{3}{2}(\alpha_3 + 39\epsilon\beta) = \frac{3}{2}\alpha_3 + \frac{117}{2}\epsilon\beta. \end{aligned} \quad (5.104)$$

Let $A_1 := (A' \setminus \{z'\}) \cup (X' \cap Y')$ and $B_1 := (B' \setminus \{z'\}) \cup (X' \cap Y')$ as obtained in Step 3(xi) of the algorithm, i.e., A_1 and B_1 are the corresponding sets in V obtained by replacing z' by $X' \cap Y'$ (see Figure 5.9). Now we consider the pair $(B_1, Y' \cup A_1)$. Since $A' \subsetneq B'$, we have that

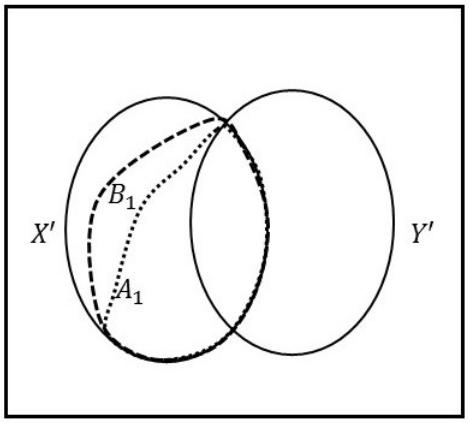


Figure 5.9: The sets A_1 and B_1 are completely contained in X'.

$B_1 \setminus (Y' \cup A_1) \neq \emptyset$. Moreover, the node $y \in (Y' \cup A_1) \setminus B_1$ and hence $(Y' \cup A_1) \setminus B_1 \neq \emptyset$. Hence, $(B_1, Y' \cup A_1)$ is an uncomparable pair. We next compute the bicut value $\beta(B_1, Y' \cup A_1)$ of this pair in the original digraph. The next proposition will help in bounding the bicut value.

Proposition 5.13.

$$\beta(B_1, Y' \cup A_1) + \alpha_5 + \alpha_1 \leq \sigma(X', Y') + |\delta_H^{in}(A') \cup \delta_H^{in}(B')|. \tag{5.105}$$

Proof. The proposition follows by counting the edges on the left hand side. We use a figure to easily visualize the counting argument. We recall that $X' \cap Y' \subseteq A_1 \subseteq B_1 \subseteq X'$.

We use Figure 5.10. Each arrow represents that all edges from the set of nodes in the rectangle containing its tail to the set of nodes in the rectangle containing its head are counted in the left hand side of Proposition 5.13. In particular, edges corresponding to $\delta^{in}(B_1)$ are marked as thin continuous arrows and $\delta^{in}(Y' \cup A_1) \setminus \delta^{in}(B_1)$ are marked as thin dotted arrows. Edges counted by α_1, i.e., corresponding to $\delta(W \setminus (X' \cup Y'), W \cap (X' \setminus Y'))$, are marked as thick $\rightarrow W$ arrows to indicate that the head v of the edges are in $W \cap S$ where S is the set of nodes in the rectangle containing the head. Edges counted by α_5, i.e., corresponding to $\delta(Z \cap (X' \setminus Y'), X' \cap Y' \cap Z)$, are marked as thick dotted $Z \rightarrow Z$ arrows to indicate that the tail u and the head v of the edges are in $Z \cap S_1$ and $Z \cap S_2$ respectively where S_1 and S_2 are the set of nodes in the rectangles containing the tail and head respectively.

We note that the edges that are counted twice in the left hand side are exactly the ones in the following four sets:

1. $\delta(Z \cap (X' \setminus B_1), Z \cap (X' \cap Y'))$ since these edges are also contained in $\delta(X' \setminus B_1, X' \cap Y')$,
2. $\delta(Z \cap (B_1 \setminus A_1), Z \cap (X' \cap Y'))$ since these edges are also contained in $\delta(B_1 \setminus A_1, X' \cap Y')$,
3. $\delta(W \setminus (X' \cup Y'), W \cap (B_1 \setminus A_1))$ since these edges are also contained in $\delta(W \setminus (X' \cup Y'), B_1 \setminus A_1))$, and
4. $\delta(W \setminus (X' \cup Y'), W \cap (A_1 \setminus Y'))$ since these edges are also contained in $\delta(W \setminus (X' \cup Y'), A_1 \setminus Y')$.

In order to prove the proposition, we need to show that every edge in the left hand side that is counted exactly once is counted in the right hand side and moreover, those edges that are counted twice in the left hand side are counted by two different terms in the right hand side. In order to show this, we mark the tail of the arrows as follows: $\square$ indicates that the edge is counted in $\delta^{in}(X')$, $\diamond$ indicates that the edge is counted in $\delta^{in}(Y')$ and $\circ$ indicates that the edge is counted in $\delta_H^{in}(A') \cup \delta_H^{in}(B')$. We note that the edges that are counted twice have different tail marks as follows:

1. the tail marks of $\delta(Z \cap (X' \setminus B_1), Z \cap (X' \cap Y'))$ and $\delta(X' \setminus B_1, X' \cap Y')$ are different,
2. the tail marks of $\delta(Z \cap (B_1 \setminus A_1), Z \cap (X' \cap Y'))$ and $\delta(B_1 \setminus A_1, X' \cap Y')$ are different,

3. the tail marks of $\delta(W \setminus (X' \cup Y'), W \cap (B_1 \setminus A_1))$ and $\delta(W \setminus (X' \cup Y'), B_1 \setminus A_1))$ are different, and

4. the tail marks of $\delta(W \setminus (X' \cup Y'), W \cap (A_1 \setminus Y'))$ and $\delta(W \setminus (X' \cup Y'), A_1 \setminus Y')$ are different.

Thus, the left hand side is at most the right hand side.

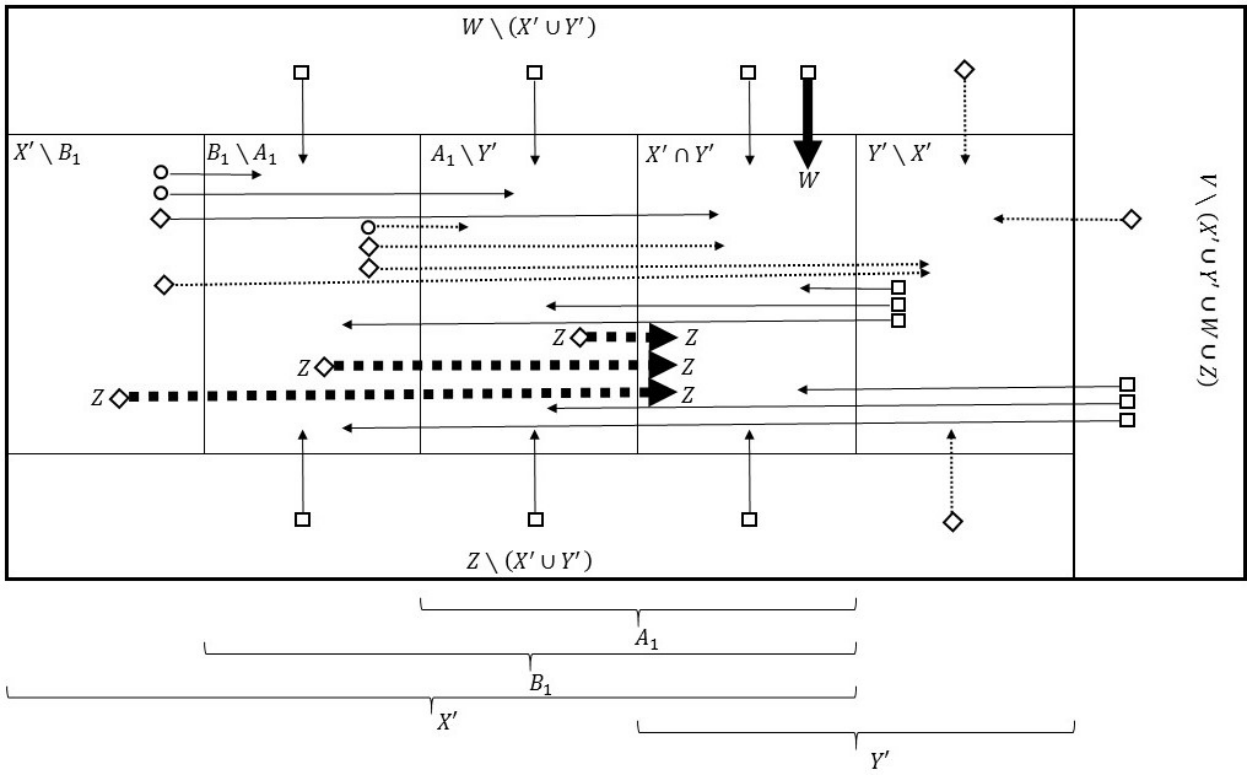


Figure 5.10: Proof of Proposition 5.13.

□

Using Proposition 5.13 and inequality (5.104), we get

$$\beta(B_1, Y' \cup A_1) \leq \sigma(X', Y') + |\delta_H^{in}(A') \cup \delta_H^{in}(B')| - \alpha_5 - \alpha_1 \tag{5.106}$$

$$\leq 2\beta + \frac{3}{2}\alpha_3 + \frac{117}{2}\epsilon\beta - \alpha_5 - \alpha_1. \tag{5.107}$$

Next, using Proposition 5.11, we get

$$\beta(B_1, Y' \cup A_1) \leq 2\beta + \frac{3}{2}\alpha_3 + \frac{117}{2}\epsilon\beta - (2\alpha_3 - 51\epsilon\beta) = 2\beta - \frac{1}{2}\alpha_3 + \frac{219}{2}\epsilon\beta. \tag{5.108}$$

Finally, we recall that $\alpha_3 \geq (1/2 - 3\epsilon)\beta$ from (5.90) and hence,

$$\beta(B_1, Y' \cup A_1) \leq \left(2 + \frac{219}{2}\epsilon\right)\beta - \frac{1}{2}\left(\frac{1}{2} - 3\epsilon\right)\beta = \left(\frac{7}{4} + 111\epsilon\right)\beta. \tag{5.109}$$

Based on all the cases analyzed above, the approximation factor is at most

$$\max\left\{1 + \epsilon, 1 + 3\epsilon, 2 - \epsilon, \frac{7}{4} + 111\epsilon\right\} = \max\left\{2 - \epsilon, \frac{7}{4} + 111\epsilon\right\}. \tag{5.110}$$

In order to minimize the factor, we set $\epsilon = 1/448$ to get the desired approximation factor, thus concluding the proof of Theorem 5.6. □

5.5 OPEN PROBLEMS

We gave a (3/2)-approximation for $(s, *, t)$-Linear-3-Cut. In fact, a $\sqrt{2}$-approximation algorithm exists [108]. We do not know if $(s, *, t)$-Linear-3-Cut is NP-hard. Can we determine the complexity of $(s, *, t)$-Linear-3-Cut? On the other hand, any improvement in the approximation ratio of $(s, *, t)$-Linear-3-Cut improves the approximation of Bi-Cut. Can we find algorithms with better approximation ratio? We described a $(2 - 1/448)$-approximation for BiCut, yet we do not know if the problem is NP-hard. A central problem is the resolution of the complexity of BiCut. Finally, what are other natural problems that exhibit the complexity gap between the global and local variants?

References

[1] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integr. VLSI J.*, vol. 19, no. 1-2, pp. 1–81, Aug 1995.

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

[3] M. Ghaffari, D. R. Karger, and D. Panigrahi, "Random contractions and sampling for hypergraph and hedge connectivity," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '17. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=3039686.3039757 pp. 1101–1114.

[4] R. Diestel, *Graph Theory, 4th Edition*, ser. Graduate texts in mathematics. Springer, 2012, vol. 173.

[5] C. Chekuri, T. Rukkanchanunt, and C. Xu, "On element-connectivity preserving graph simplification," in *Algorithms ESA 2015*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015, vol. 9294, pp. 313–324.

[6] R. Hassin and A. Levin, "Flow trees for vertex-capacitated networks," *Discrete Applied Mathematics*, vol. 155, no. 4, pp. 572 – 578, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166218X06003581

[7] C. Chekuri and C. Xu, "Computing minimum cuts in hypergraphs," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611974782.70 pp. 1085–1100.

[8] C. Chekuri and C. Xu, "A note on approximate strengths of edges in a hypergraph," *CoRR*, vol. abs/1703.03849, 2017. [Online]. Available: http://arxiv.org/abs/1703.03849

[9] H. Nagamochi and T. Ibaraki, "A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph." *Algorithmica*, vol. 7, no. 5&6, pp. 583–596, 1992.

[10] H. N. Gabow, "A Matroid Approach to Finding Edge Connectivity and Packing Arborescences," *Journal of Computer and System Sciences*, vol. 50, no. 2, pp. 259–273, Apr 1995.

[11] S. Guha, A. McGregor, and D. Tench, "Vertex and hyperedge connectivity in dynamic graph streams," in *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, ser. PODS '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2745754.2745763 pp. 241–247.

[12] E. Dinits, A. Karzanov, and M. Lomonosov, "On the structure of a family of minimal weighted cuts in graphs," in *Studies in Discrete Mathematics*, A. Fridman, Ed. Nauka (Moskva), 1976, pp. 290–306.

[13] H. Nagamochi, Y. Nakao, and T. Ibaraki, "A fast algorithm for cactus representations of minimum cuts," *Japan Journal of Industrial and Applied Mathematics*, vol. 17, no. 2, p. 245, Jun 2000.

[14] H. Nagamochi and T. Kameda, "Constructing cactus representation for all minimum cuts in an undirected network," *Journal of the Operations Research Society of Japan*, vol. 39, no. 2, pp. 135–158, 1996.

[15] H. N. Gabow, "The minset-poset approach to representations of graph connectivity," *ACM Trans. Algorithms*, vol. 12, no. 2, pp. 24:1–24:73, Feb 2016.

[16] L. Fleischer, "Building chain and cactus representations of all minimum cuts from HaoOrlin in the same asymptotic run time," *Journal of Algorithms*, vol. 33, no. 1, pp. 51 – 72, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677499910398

[17] H. Nagamochi, S. Nakamura, and T. Ishii, "Constructing a cactus for minimum cuts of a graph in $O(mn+n^2 \log n)$ time and $O(m)$ space," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 2, pp. 179–185, 2003.

[18] T. Fleiner and T. Jordán, "Coverings and structure of crossing families," *Mathematical Programming*, vol. 84, no. 3, pp. 505–518, Apr 1999.

[19] E. Cheng, "Edge-augmentation of hypergraphs," *Mathematical Programming*, vol. 84, no. 3, pp. 443–465, Apr 1999.

[20] W. H. Cunningham, "Decomposition of submodular functions," *Combinatorica*, vol. 3, no. 1, pp. 53–68, 1983.

[21] W. D. Matula, "A Linear Time $2+\epsilon$ Approximation Algorithm for Edge Connectivity," in *SODA*, 1993, pp. 500–504.

[22] D. R. Karger, "Random Sampling in Graph Optimization Problems," Ph.D. dissertation, Stanford University, Feb 1995.

[23] A. A. Benczúr and D. R. Karger, "Randomized approximation schemes for cuts and flows in capacitated graphs," *SIAM J. Comput.*, vol. 44, no. 2, pp. 290–319, 2015, preliminary versions appeared in STOC '96 and SODA '98. [Online]. Available: http://dx.doi.org/10.1137/070705970

[24] D. Kogan and R. Krauthgamer, "Sketching cuts in graphs and hypergraphs," in *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ser. ITCS '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2688073.2688093 pp. 367–376.

[25] K. Chandrasekaran, C. Xu, and X. Yu, "Hypergraph k-cut in randomized polynomial time," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611975031.94 pp. 1426–1438.

[26] C. Chekuri and S. Li, "A note on the hardness of the k-way hypergraph cut problem," November 2015, unpublished manuscript available at http://chekuri.cs.illinois.edu/papers/hypergraph-kcut.pdf.

[27] O. Goldschmidt and D. S. Hochbaum, "A Polynomial Algorithm for the k-cut Problem for Fixed k," *Mathematics of Operations Research*, vol. 19, no. 1, pp. 24–37, 1994.

[28] M. Thorup, "Minimum k-way cuts via deterministic greedy tree packing," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*. ACM, 2008, pp. 159–166.

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

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

[31] Y. Kamidoi, N. Yoshida, and H. Nagamochi, "A deterministic algorithm for finding all minimum k-way cuts." *SIAM J. Comput.*, vol. 36, no. 5, pp. 1329–1341, 2006.

[32] K. Bérczi, K. Chandrasekaran, T. Király, E. Lee, and C. Xu, "Global and Fixed-Terminal Cuts in Digraphs," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 81. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 2:1–2:20.

[33] K. Kawarabayashi and C. Xu, "Minimum violation vertex maps and their application to cut problems," 2018, unpublished Manuscript.

[34] M. Queyranne, "On Optimum k-way Partitions with Submodular Costs and Minimum Part-size Constraints," Talk url: https://smartech.gatech.edu/bitstream/handle/1853/43309/Queyranne.pdf, 2012.

[35] F. Guiñez and M. Queyranne, "The size-constrained submodular k-partition problem," Manuscript, https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxmbGF2aW9ndWluZXpob21lcGFnZXxneDo0NDVlMThkMDg4ZWRlOGI1, 2012. [Online]. Available: https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxmbGF2aW9ndWluZXpob21lcGFnZXxneDo0NDVlMThkMDg4ZWRlOGI1

[36] R. Erbacher, T. Jaeger, N. Talele, and J. Teutsch, "Directed multicut with linearly ordered terminals," Preprint arXiv:1407.7498, 2014. [Online]. Available: https://arxiv.org/abs/1407.7498

[37] S. Khot, "On the power of unique 2-prover 1-round games," in *Proceedings of the 34th annual ACM Symposium on Theory of Computing*, ser. STOC '02, 2002, pp. 767–775.

[38] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis, "The complexity of multiterminal cuts," *SIAM Journal on Computing*, vol. 23, no. 4, pp. 864–894, 1994.

[39] 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*, vol. 21, no. 2, pp. 192–208, 2011.

[40] P. Zhang and B. Fu, "The label cut problem with respect to path length and label frequency," *Theoretical Computer Science*, vol. 648, pp. 72 – 83, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0304397516303930

[41] C. Chekuri and V. Madan, "Approximating multicut and the demand graph," in *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '17, 2017, pp. 855–874.

[42] H. Hind and O. Oellermann, "Menger-Type Results for Three or More Vertices," *Congressus Numerantium*, pp. 179–204, 1996.

[43] C. Chekuri and N. Korula, "A graph reduction step preserving element-connectivity and packing steiner trees and forests," *SIAM Journal on Discrete Mathematics*, vol. 28, no. 2, pp. 577–597, 2014, preliminary version in *Proc. of ICALP*, 2009.

[44] K.-i. Kawarabayashi and M. Thorup, "Deterministic global minimum cut of a simple graph in near-linear time," in *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, ser. STOC '15. New York, NY, USA: ACM, 2015. [Online]. Available: http://doi.acm.org/10.1145/2746539.2746588 pp. 665–674.

[45] M. Henzinger, S. Rao, and D. Wang, *Local Flow Partitioning for Faster Edge Connectivity*, pp. 1919–1938. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611974782.125

[46] D. R. Karger, "Minimum cuts in near-linear time," *J. ACM*, vol. 47, no. 1, pp. 46–76, Jan. 2000.

[47] R. Duan, "Breaking the $O(n^{2.5})$ Deterministic Time Barrier for Undirected Unit-Capacity Maximum Flow," in *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA: ACM-SIAM, 2013, pp. 1171–1179.

[48] R. Hariharan, T. Kavitha, D. Panigrahi, and A. Bhalgat, "An $\tilde{O}(mn)$ gomory-hu tree construction algorithm for unweighted graphs," in *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1250790.1250879 pp. 605–614.

[49] H. Gabow and P. Sankowski, "Algebraic algorithms for b-matching, shortest undirected paths, and f-factors," in *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, Oct 2013, pp. 137–146.

[50] H. Y. Cheung, L. C. Lau, and K. M. Leung, "Graph connectivities, network coding, and expander graphs," in *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, ser. FOCS '11. Washington, DC, USA: IEEE Computer Society, 2011. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2011.55 pp. 190–199.

[51] H. N. Gabow, "Using expander graphs to find vertex connectivity," in *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000, pp. 410–420.

[52] M. R. Henzinger, S. Rao, and H. N. Gabow, "Computing vertex connectivity: New bounds from old techniques," *Journal of Algorithms*, vol. 34, no. 2, pp. 222 – 250, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677499910556

[53] S. Even and R. Tarjan, "Network flow and testing graph connectivity," *SIAM Journal on Computing*, vol. 4, no. 4, pp. 507–518, 1975. [Online]. Available: http://dx.doi.org/10.1137/0204043

[54] J. Cheriyan and M. Salavatipour, "Packing Element-Disjoint Steiner Trees," *ACM Transactions on Algorithms*, vol. 3, no. 4, 2007, preliminary version in *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 52-61, 2005.

[55] A. Aazami, J. Cheriyan, and K. Jampani, "Approximation Algorithms and Hardness Results for Packing Element-Disjoint Steiner Trees in Planar Graphs," *Algorithmica*, vol. 63, no. 1–2, pp. 425–456, 2012, preliminary version in APPROX 2009.

[56] A. A. Benczúr, "Counterexamples for directed and node capacitated cut-trees," *SIAM Journal on Computing*, vol. 24, no. 3, pp. 505–510, 1995. [Online]. Available: http://dx.doi.org/10.1137/S0097539792236730

[57] L. Lovász, "On Some Connectivity Properties of Eulerian Graphs," *Acta Mathematica Hungarica*, vol. 28, no. 1, pp. 129–138, 1976.

[58] W. Mader, "A Reduction Method for Edge-Connectivity in Graphs," *Annals of Discrete Mathematics*, vol. 3, pp. 145–164, 1978.

[59] L. C. Lau and C. K. Yung, "Efficient edge splitting-off algorithms maintaining all-pairs edge-connectivities," *SIAM Journal on Computing*, vol. 42, no. 3, pp. 1185–1200, 2013.

[60] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Springer Verlag, Berlin Heidelberg, 2003.

[61] A. Frank, T. Ibaraki, and H. Nagamochi, "On sparse subgraphs preserving connectivity properties," *Journal of graph theory*, vol. 17, no. 3, pp. 275–281, 1993.

[62] H. N. Gabow and R. E. Tarjan, "Algorithms for Two Bottleneck Optimization Problems," *Journal of Algorithms*, vol. 9, no. 3, pp. 411–417, 1988.

[63] H. N. Gabow, "Efficient splitting off algorithms for graphs," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94. New York, NY, USA: ACM, 1994. [Online]. Available: http://doi.acm.org/10.1145/195058.195436 pp. 696–705.

[64] F. Granot and R. Hassin, "Multi-terminal maximum flows in node-capacitated networks," *Discrete Appl. Math.*, vol. 13, no. 2-3, pp. 131–156, Mar. 1986. [Online]. Available: http://dx.doi.org/10.1016/0166-218X(86)90078-8

[65] C. Chekuri, "Some open problems in element connectivity," September 2015, unpublished Survey. Available at http://chekuri.cs.illinois.edu/papers/elem-connectivity-open-probs.pdf.

[66] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM*, vol. 44, no. 4, pp. 585–591, 1997.

[67] H. Nagamochi and T. Ibaraki, *Algorithmic Aspects of Graph Connectivity*, 1st ed. New York, NY, USA: Cambridge University Press, 2008.

[68] M. Queyranne, "Minimizing symmetric submodular functions," *Mathematical Programming*, vol. 82, no. 1, pp. 3–12, 1998.

[69] R. Klimmek and F. Wagner, "A simple hypergraph min cut algorithm," Bericht FU Berlin Fachbereich Mathematik und Informatik, Tech. Rep. B 96-02, 1996, available at http://edocs.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000297/1996_02.pdf.

[70] W.-K. Mak and D. Wong, "A fast hypergraph min-cut algorithm for circuit partitioning," *Integration, the VLSI Journal*, vol. 30, no. 1, pp. 1 – 11, 2000.

[71] A. Frank, T. Király, and M. Kriesell, "On decomposing a hypergraph into k connected sub-hypergraphs," *Discrete Applied Mathematics*, vol. 131, no. 2, pp. 373 – 383, 2003, submodularity.

[72] A. Frank, *Connections in Combinatorial Optimization*, ser. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2011.

[73] D. R. Karger and D. Panigrahi, "A near-linear time algorithm for constructing a cactus representation of minimum cuts," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '09. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496770.1496798 pp. 246–255.

[74] W. H. Cunningham and J. Edmonds, "A combinatorial decomposition theory," *Canadian Journal of Mathematics*, vol. 32, no. 3, pp. 734–765, 1980.

[75] S. Fujishige, "Canonical decomposotions of symmetric submodular functions," *Discrete Applied Mathematics*, vol. 5, no. 2, pp. 175–190, 1983.

[76] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, "A general framework for graph sparsification," in *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, ser. STOC '11. New York, NY, USA: ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/1993636.1993647 pp. 71–80.

[77] J. Batson, D. A. Spielman, and N. Srivastava, "Twice-Ramanujan sparsifiers," *SIAM Journal on Computing*, vol. 41, no. 6, pp. 1704–1721, 2012.

[78] Y. T. Lee and H. Sun, "An sdp-based algorithm for linear-sized spectral sparsification," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2017. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3055399.3055477 pp. 678–687.

[79] H. Aissi, A. R. Mahjoub, S. T. McCormick, and M. Queyranne, "Strongly polynomial bounds for multiobjective and parametric global minimum cuts in graphs and hypergraphs," *Math. Program.*, vol. 154, no. 1-2, pp. 3–28, 2015.

[80] E. L. Lawler, "Cutsets and partitions of hypergraphs," *Networks*, vol. 3, no. 3, pp. 275–285, 1973. [Online]. Available: http://dx.doi.org/10.1002/net.3230030306

[81] J. B. Orlin, "Max flows in o(nm) time, or better," in *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13. New York, NY, USA: ACM, 2013, pp. 765–774.

[82] E. A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation," *Soviet Math Doklady*, vol. 11, pp. 1277–1280, 1970.

[83] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *J. Comput. Syst. Sci.*, vol. 26, no. 3, pp. 362–391, June 1983.

[84] R. Rizzi, "NOTE On Minimizing Symmetric Set Functions," *Combinatorica*, vol. 20, no. 3, pp. 445–450, 2000.

[85] M. Brinkmeier, "Minimizing symmetric set functions faster," *CoRR*, vol. abs/cs/0603108, 2006. [Online]. Available: http://arxiv.org/abs/cs/0603108

[86] J. S. Provan and D. R. Shier, "A paradigm for listing (s, t)-cuts in graphs," *Algorithmica*, vol. 15, no. 4, pp. 351–372, 1996.

[87] S. R. Arikati and K. Mehlhorn, "A Correctness certificate for the Stoer-Wagner min-cut algorithm," *Information Processing Letters*, vol. 70, no. 5, pp. 251–254, 1999.

[88] Y. Yamaguchi, "Realizing symmetric set functions as hypergraph cut capacity," *Discrete Mathematics*, vol. 339, no. 8, pp. 2007–2017, aug 2016. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0012365X16300267

[89] Y. T. Lee and A. Sidford, "Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{\text{rank}})$ iterations and faster algorithms for maximum flow," in *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 2014, pp. 424–433.

[90] J. B. Kruskal, Jr., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proc. Amer. Math. Soc.*, vol. 7, pp. 48–50, 1956.

[91] B. Chazelle, "Computing on a free tree via complexity-preserving mappings," *Algorithmica*, vol. 2, no. 1, pp. 337–361, 1987.

[92] A. Madry, "Navigating central path with electrical flows: From flows to matchings, and back," in *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE, 2013, pp. 253–262.

[93] M. Xiao, "Finding minimum 3-way cuts in hypergraphs," *Information Processing Letters (Preliminary version in TAMC 2008)*, vol. 110, no. 14, pp. 554–558, 2010.

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

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

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

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

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

[99] 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.

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

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

[102] A. Gupta, E. Lee, and J. Li, "An FPT Algorithm Beating 2-Approximation for k-Cut," in *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2018, pp. 2821–2837.

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

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

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

[106] D. Karger and R. Motwani, "Derandomization through approximation," in *Proceedings of the 26th annual ACM symposium on Theory of computing*, ser. STOC '94, 1994, pp. 497–506.

[107] V. Vazirani and M. Yannakakis, "Suboptimal cuts: Their enumeration, weight and number (extended abstract)," in *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, ser. ICALP '92, 1992, pp. 366–377.

[108] K. Bérczi, K. Chandrasekaran, T. Király, and V. Madan, "A tight $\sqrt{2}$-approximation for linear 3-cut," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '18. [Online]. Available: http://epubs.siam.org/doi/abs/10.1137/1.9781611975031.92 pp. 1393–1406.

[109] C. Chekuri and V. Madan, "Simple and fast rounding algorithms for directed and node-weighted multiway cut," in *To appear in Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '17, 2017.

[110] N. Garg, V. Vazirani, and M. Yannakakis, "Multiway cuts in directed and node weighted graphs," in *Proceedings of the 20th International Colloquium on Automata, Languages and Programming*, ser. ICALP '94, 1994, pp. 487–498.

[111] E. Lee, "Improved Hardness for Cut, Interdiction, and Firefighter Problems," in *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming*, ser. ICALP, 2017, pp. 92:1–92:14.