Minimum cuts and sparsification in hypergraphs *

Chandra Chekuri† Chao Xu‡

Abstract

We study algorithmic and structural aspects of connectivity in hypergraphs. Given a hypergraph $H = (V, E)$ with $n = |V|$, $m = |E|$ and $p = \sum_{e \in E} |e|$ the fastest known algorithm to compute a global minimum cut in H runs in $O(np)$ time for the uncapacitated case, and in $O(np + n^2 \log n)$ time for the capacitated case. We show the following new results.

- Given an uncapacitated hypergraph H and an integer k we describe an algorithm that runs in $O(p)$ time to find a (trimmed) subhypergraph H' with sum of degrees $O(kn)$ that preserves all edge-connectivities up to k (a k-sparse certificate). This generalizes the corresponding result of Nagamochi and Ibaraki from graphs to hypergraphs. Using this sparsification we obtain an $O(p + \lambda n^2)$ time algorithm for computing a global minimum cut of H where λ is the minimum cut value.
- We show that a hypercactus representation of *all* the global minimum cuts of a capacitated hypergraph can be computed in $O(np + n^2 \log n)$ time and $O(p)$ space matching the asymptotic time to find a single minimum cut.
- We obtain a $(2 + \varepsilon)$-approximation to the global minimum cut of a capacitated hypergraph in $O(\frac{1}{\varepsilon}(p \log n + n \log^2 n))$ time, and for uncapacitated hypergraphs in $O(p/\varepsilon)$ time. We achieve this by generalizing Matula's algorithm for graphs to hypergraphs.
- We describe an algorithm to compute approximate strengths of all the edges of a hypergraph in $O(p \log^2 n \log p)$ time. This gives a near linear time algorithm for finding a $(1 + \varepsilon)$-cut sparsifier based on the work of Kogan and Krauthgamer. As a byproduct we obtain faster algorithms for various cut and flow problems in hypergraphs of small rank.

Our results build upon properties of vertex orderings that were inspired by the maximum adjacency ordering for graphs due to Nagamochi and Ibaraki. Unlike graphs we observe that there are several orderings for hypergraphs and these yield different insights.

1 Introduction

We consider algorithmic and structural aspects of connectivity in hypergraphs. A hypergraph $H = (V, E)$ consists of a finite vertex set V and a set of hyperedges E where each edge e is a subset of vertices. Undirected loopless graphs are a special case of hypergraphs where all edges are sets of size two. For the most part we use n to denote the number of vertices $|V|$, m to denote the number of edges $|E|$, and p to denote $\sum_{e \in E} |e|$. Note that $p = \sum_{v \in V} \deg(v)$ where $\deg(v)$ is the degree of v (the number of hyperedges that contain v). We observe that p is the natural representation size of a connected hypergraph, and p is the number of edges in the standard representation of H as a bipartite graph $G_H = (V \cup E, F)$ with $F = \{(v, e) | v \in V, e \in E, v \in e\}$. The *rank* of a hypergraph is the maximum edge size, that is, $\max_{e \in E} |e|$. We let r denote the rank. Note that graphs are hypergraphs of rank 2. A number of results on hypergraphs assume that rank is a fixed constant. In this paper our focus is on general hypergraphs without assumptions on r although some results are more meaningful for the case of small r.

Hypergraphs arise in a number of settings in both theory and practice. Some of the most basic algorithmic questions regarding hypergraphs have to do with connectivity and cuts. Given a hypergraph $H = (V, E)$ and a vertex subset $S \subseteq V$, let $\delta_H(S)$ denote the set of edges that intersect both S and $V \setminus S$. It is well-known that the set function $|\delta_H(S)|$ defines a symmetric submodular set function over the ground set V. The connectivity (or the global minimum cut value) of a hypergraph H, denoted by $\lambda(H)$, is defined as $\min_{\emptyset \subsetneq S \subsetneq V} |\delta_H(S)|$; equivalently, it is

*This paper contains and extends results in two preliminary versions [10, 11]. Work on this paper supported in part by NSF grant CCF-1526799.

†University of Illinois Urbana-Champaign, Urbana, IL 61801. chekuri@illinois.edu

‡*Current Affiliation:* Yahoo! Research, New York, NY 10003. *Previous Affiliation:* University of Illinois Urbana-Champaign, Urbana, IL 61801. chao.xu@oath.com

the minimum number of edges that need to be removed such that H is disconnected. For distinct nodes $s, t \in V$ we denote by $\lambda_H(s,t)$ (or sometimes by $\lambda(s,t;H)$) the edge-connectivity between s and t in H which is defined as $\min_{S \subseteq V, |S \cap \{s,t\}|=1} |\delta_H(S)|$. These definitions readily generalize to capacitated hypergraphs where each edge $e \in E$ has a non-negative capacity $c(e)$ associated with it. In this paper we study algorithmic and structural questions that arise in computing $\lambda(H)$. In the sequel we use the term mincut to refer to the global mincut.

Algorithms for mincuts and s-t mincuts in graphs have been extensively studied. Traditional algorithms for mincut were based on computing a sequence of $(n-1)$ s-t mincuts; s-t mincuts are most efficiently computed via network flow although one can also compute them via submodular function minimization. The first algorithm for finding a mincut in an *undirected* graph that avoided the use of flows was due to Nagamochi and Ibaraki [45]. They devised a surprising and influential algorithm based on maximum-adjacency orderings (MA-ordering) which is an ordering of the vertices 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 vertices in the ordering then $\{t\}$ is an s-t mincut. This yields a simple $O(mn + n^2 \log n)$ time algorithm [54] for computing a mincut 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 [46]. Karger devised another highly influential technique based on random contractions [30] which led to a randomized $O(n^2 \log^3 n)$-time Monte Carlo algorithm for computing a mincut in capacitated graphs [33]. Subsequently, using sampling techniques for cut-sparsification and tree packings, Karger devised a randomized $O(m \log^3 n)$ time Monte Carlo algorithm [31]. More recently Kawarabayashi and Thorup [34] devised a deterministic $O(m \log^{12} n)$ time algorithm for *simple* uncapacitated graphs. Henzinger, Rao and Wang [29] improved the deterministic running time in simple uncapacitated graphs to $O(m \log^2 n \log\log n)$ using flow partitioning.

What about hypergraphs? A simple and well-known reduction shows that $\lambda_H(s,t)$ can be computed via s-t network flow in the vertex capacitated bipartite graph G_H associated with H. Thus, using $(n-1)$ network flows one can compute $\lambda(H)$. However, Queyranne [50] showed that the Nagamochi-Ibaraki ordering approach generalizes to find the mincut 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 [35] and Mak and Wong [42]. 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 will later show that we can exploit their different properties in our algorithms.

Apart from the above mentioned results, very little else is known in the algorithms literature on mincut 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 [28, 36]. Our initial motivation to address these algorithmic questions came from the study of algorithms for element-connectivity and related problems which are closely related to hypergraphs — we refer the reader to the survey [8]. In this paper the three main questions we address are the following.

- Are there faster (approximation) algorithms for mincut computation in hypergraphs?
- How many distinct mincuts can there be in a hypergraph? Is there a fast algorithm to compute a compact representation called the hypercactus representation that is known to exist [12, 17]? For graphs it is well-known that there are at most $\binom{n}{2}$ mincuts and that there exists a compact $O(n)$-sized data structure called the *cactus* to represent all of them.
- Are there fast algorithms to *sparsify* a hypergraph to (approximately) preserve cuts? In graphs such results have found powerful applications in addition to mathematical elegance.

1.1 Overview of Results

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

Sparse certificate and fast algorithm for small mincuts: 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

[1] For a submodular function $f : 2^V \to \mathbb{R}$ the mincut 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.

$s, t \in V$. Nagamochi and Ibaraki [44] showed, via an implicit use of 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 [28]. 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 vertex $v \in e$ 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 [21] (see [19, 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 algorithm for computing the mincut 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 weighted hypergraphs.

All mincuts and the hypercactus representation: We consider the problem of finding all the mincuts in a hypergraph. For any capacitated graph G on n vertices, it is well-known, originally from the work of Dinitz, Karzanov and Lomonosov [16], that there is a compact $O(n)$ sized data structure, namely a cactus graph, that represents all the mincuts 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 mincuts 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 mincuts of a graph in $O(mn + n^2 \log n)$ time [47] or in $O(nm \log(n^2/m))$-time [18, 25]. For uncapacitated graphs, there is an $O(m + \lambda^2 n \log(n/\lambda))$-time algorithm [25]. There is also a Monte Carlo algorithm that runs in $\tilde{O}(m)$ time [32] building on the randomized near-linear time algorithm of Karger [31]. In effect, the time to compute the cactus representation is the same as the time to compute the global mincut! We note, however, that all the algorithms are fairly complicated, in particular the deterministic algorithms.

The situation for hypergraphs is not as straightforward. First, how many distinct mincuts 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 mincut 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 mincuts 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 [14], Fujishige [22], and Cunningham [13] on submodular functions from the early 1980s. Recently Ghaffari et al. also proved this fact through a random contraction algorithm [26]. Cheng [12], building on Cunningham's work [13], explicitly showed that the mincuts of a hypergraph admit a compact structure which we call a hypercactus representation. Later Fleiner and Jordán [17] showed that a similar structure exists for any symmetric submodular function defined over crossing families. However, these papers were not as concerned with algorithmic considerations.

In this paper we show that the hypercactus representation of the mincuts 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 mincut. 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 [12, 13] to build the canonical decomposition of a hypergraph which then allows one to build a hypercactus representation easily. The algorithmic step needed for constructing the canonical decomposition is conceptually simpler and relies on an efficient algorithm for finding a non-trivial mincut (one in which both sides have at least two nodes) in a hypergraph H if there is one. Our 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 [42]. Our algorithm yields a conceptually simple algorithm for graphs as well, and highlights the power of the decomposition theory for graphs and submodular functions [13, 14, 22].

$(2+\varepsilon)$ approximation for global mincut: Matula [43], building on the properties of MA-ordering, showed that a $(2+\varepsilon)$ approximation for the global mincut of an uncapacitated graph can be computed in deterministic $O(m/\varepsilon)$

time. The algorithm generalizes to capacitated graphs and runs in $O(\frac{1}{\varepsilon}(m \log n + n \log^2 n))$ time (as mentioned by Karger [30]). Although the approximation is less interesting in light of the randomized $\tilde{O}(m)$ algorithm of Karger [31] [3], it is a useful building block that allows one to deterministically estimate the value of a mincut. For hypergraphs there was no such approximation known. In fact, the survey [8] 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[4].

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

Approximating strengths and $(1+\varepsilon)$-cut sparsifiers: Benczúr and Karger, in their seminal work [4], showed that all cuts of a capacitated graph $G = (V, E)$ on n vertices 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 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 [4] 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 [24] has shown a general framework for cut sparsification where the sampling probability p_e can also be chosen to be the approximate connectivity between the end points of e. In another seminal work, Batson, Spielman and Srivastava [3] 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 [40].

Guha et al. [28] devised a simple method for cut sparsifier of uncapacitated hypergraphs in the streaming setting. The approach follows [4]. They showed that there is a sketch of $O(\varepsilon^{-2} n \operatorname{polylog} n)$ space where a $(1 \pm \varepsilon)$-cut sparsifier can be constructed. The running time is $\tilde{O}(np)$. However, their results cannot be generalized to capacitated hypergraphs. Kogan and Krauthgamer [36] 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 paper, we develop a near-linear time algorithm for computing approximate strengths of edges in a capacitated hypergraph. For this purpose we rely on sparsification for unweighted 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 6.

1.2 Other Related Work

Kogan and Krauthgamer's [36] 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-mincuts 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. Aissi et al. later generalized it to all real $\alpha \geq 1$ [1]. Kogan and Krauthgamer's cut-sparsification results rely on the number of approximate minimum cuts. In particular, given a n-vertex capacitated hypergraph $H = (V, E)$ of rank r they show that there is a capacitated hypergraph $H' = (V, E')$ with $O(\frac{n}{\varepsilon^2}(r + \log n))$ edges such that every cut capacity in H is preserved to within a $(1 \pm \varepsilon)$ factor in H'. Aissi et al. [1] considered parametric mincuts in graphs and hypergraphs of fixed rank and obtained polynomial bounds on the number of distinct mincuts.

Hypergraph cuts have also been studied in the context of k-way cuts. Here the goal is to partition the vertex set V into k non-empty sets so as to minimize the number of hyperedges crossing the partition. For $k \leq 3$ a polynomial time algorithm is known [56] while the complexity was, until recently, unknown for any fixed $k \geq 4$. The problem is NP-Complete when k is part of the input even for graphs [27]. Fukunaga [23] obtained a polynomial-time algorithm for k-way cut when k and the rank r are fixed; this generalizes the polynomial-time algorithm for graphs [27, 55]. Karger's contraction algorithm also yields a randomized algorithm when k and the rank r are

[3] $\tilde{O}$ hides polylog factors.

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

fixed. Recently, it was shown that the k-way cut can be solved in randomized polynomial time for constant k [6]. When k is part of the input, k-way cut in graphs admits a $2(1-1/k)$-approximation [52]. This immediately yields a $2r(1-1/k)$-approximation for hypergraphs. If r is not fixed and k is part of the input, it was recently shown [9] that the approximability of the k-way cut problem is related to that of the k-densest subgraph problem, and hence it is unlikely to admit a sub-polynomial approximation ratio.

Hypergraph cuts arise in several other contexts with terminals such as the s-t cut problem or its generalizations such as multi-terminal cut problem and the multicut problem. In some of these problems one can reduce the hypergraph cut problem to a node-capacitated undirected graph cut problem and vice-versa.

Organization: Section 2 sets up requisite notation, and also describes different vertex orderings and their properties which underpin most of the results in the paper. Section 3 describes sparsification for unweighted hypergraphs. Section 4 describes our algorithm for computing the cactus representation of all the minimum cuts. Section 5 describes our 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 6 describes our algorithm to compute approximate strengths of edges in a weighted hypergraph that yields a near linear-time cut sparsification algorithm.

2 Preliminaries

A hypergraph $H = (V, E)$ is capacitated if there is a non-negative edge capacity function $c : E \to \mathbb{R}_+$ associated with it. If all capacities are 1 we call the hypergraph uncapacitated; we allow multiple copies of an edge in the uncapacitated case. A cut $(S, V-S)$ is a bipartition of the vertices, where S and $V-S$ are both non-empty. We will abuse the notation and call a set S a cut to mean the cut $(S, V-S)$. Two sets A and B *cross* if $A \cap B$, $A \setminus B$ and $B \setminus A$ are all non-empty. For $S \subseteq V$, $\delta_H(S)$ is defined to be the set of all edges in $E(H)$ that have an endpoint in both S and $V-S$. We will drop H from the notation if the hypergraph is clear from the context. The capacity of the cut S, denoted by $c(S)$, is defined to be $\sum_{e \in \delta(S)} c(e)$. The function $c : 2^V \to \mathbb{R}$ defined above is called the *cut function*. The function $\lambda(H)$ is the edge-connectivity of H and is defined as $\min_{\emptyset \subsetneq S \subsetneq V} c(S)$. A cut S is a mincut of H if $c(S) = \lambda(H)$. A hypergraph is *k-edge-connected* if $\lambda(H) \geq k$. A set of edges F is an edge-cut-set if $F = \delta(S)$ for some cut S. A set of edges is a min edge-cut-set if $F = \delta(S)$, where S is a mincut. For distinct vertices $s, t \in V(H)$, an s-t cut is a cut S such that $|S \cap \{s, t\}| = 1$. $\lambda(s, t; H)$ is the value of the s-t mincut(minimum s-t cut) in H. The function 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. For $U \subseteq V$, $H[U] = (U, \{e | e \subseteq U, e \in E\})$ denotes the vertex induced subhypergraph of H.

For pairwise disjoint vertex subsets $A_1, \ldots, A_k$, $E(A_1, \ldots, A_k; H) = \{e \mid e \cap A_i \neq \emptyset, 1 \leq i \leq k\}$ is the set of edges that have an end point in each of the sets $A_1, \ldots, A_k$. Let $d(A_1, \ldots, A_k; H) = \sum_{e \in E(A_1, \ldots, A_k; H)} c(e)$ denotes the total capacity of the edges in $E(A_1, \ldots, A_k; H)$. A related quantity for two disjoint sets A and B is $d'(A, B; H) = \sum_{e \in E(A,B;H), e \subseteq A \cup B} c(e)$ 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. We abuse the notation and write singletons in place of sets for the above functions, i.e. for vertices u and v, $E(u, v)$ denotes $E(\{u\}, \{v\})$.

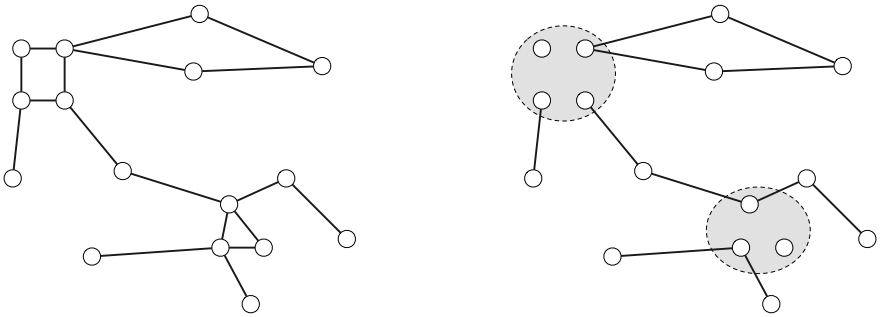
A hypergraph $H' = (V', E')$ is a *subhypergraph* of H if $V' \subseteq V$ and $E' \subseteq E$. Thus a subhypergraph of H is obtained by deleting vertices and edges. Removing a vertex $v \in e$ from e is called *trimming* e [19]. H' is a *trimmed subhypergraph* of H if there is an injection $\phi : E \to E'$ where $\phi(e) \subseteq e$. Namely, H' is obtained by deleting vertices and trimming edges.

Contracting an edge e in a hypergraph H results in a new hypergraph $H' = (V', E')$ where all vertices in e are identified into a single new vertex 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 vertex v_e. We denote H' by H/e.

For simplicity, given a hypergraph H, we use n as the number of vertices, m as the number of edges, and $p = \sum_{e \in E} |e|$ as the sum of degrees.

Equivalent digraph: s-t mincut in a hypergraph H can be computed via an s-t maximum flow in an associated capacitated digraph (see [38]) $\vec{H} = (\vec{V}, \vec{E})$ that we call the *equivalent digraph*. The 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 $c(e)$.



(a) A cactus.

(b) A hypercactus.

Figure 2.1: Examples of cactuses and hypercactuses. Grey regions are hyperedges.

For any pair $s, t \in V(H)$, there is bijection between the finite capacity s-t cuts in $\vec{H}$ and s-t cuts in H. We omit further details of this simple fact. The number of vertices in the equivalent digraph is $O(m)$, and number of edges is $O(p)$. Suppose we wish to compute an s-t minimum cut in the equivalent digraph where $s, t \in V$. If we use Orlin's algorithm [48] naively we obtain a running time of $O(mp)$ for max s-t-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 s-t-flow on the equivalent digraph using Dinic's algorithm is $O(np \log n)$ [15, 53].

Cut representations and hypercactus representation: Consider a hypergraph H. A hypergraph H^* and a vertex map $\phi : V(H) \to V(H^*)$ is a *(cut) representation* of H if the following two properties are true.

1. For each mincut S of H, $S^* = \phi(S)$ is a mincut in H^*.
2. For each mincut S^* of H^*, $\phi^{-1}(S^*)$ is a mincut of H.

If (H^*, ϕ) is a representation, then we can find all mincuts of H by finding all mincuts of H^*. Indeed, by the second property, if S^* is a mincut of H^*, we can obtain a mincut $S = \phi^{-1}(S^*)$ in H. By the first property, all mincuts in H can be obtained this way. We are interested in finding a representation that is in a simple class of hypergraphs such that finding all mincuts for a hypergraph in that class is easy.

A vertex is an *articulation vertex* if its removal disconnects the hypergraph (i.e., the remaining hypergraph has at least two components.) A *block* is an inclusion-wise maximal induced subgraph that does not have an articulation vertex. A *cactus* is a graph such that each of its blocks is either a single edge or a cycle. Another way to characterize a cactus is that it is a graph in which no two cycles share an edge. A *hypercactus* is a hypergraph such that each of its blocks is a single hyperedge or a cycle. A hypercactus can also be considered as a hypergraph obtained by repeatedly replacing a cycle in a cactus with a single hyperedge covering all the vertices of the cycle. See fig. 2.1 for examples. As we mentioned, every graph has a cactus cut representation. Similarly, for each hypergraph, there exists a hypercactus cut representation [12]; Cheng refers to his representation as "structure hypergraph". Fleiner and Jordan [17] also defined a "hypercactus" for the more general setting of symmetric submodular functions over crossing families. Their hypercactus representation is not the same as our definition here. Under their definition a cycle is not considered a hypercactus, which in our view, is counterintuitive.

2.1 Vertex orderings

Nagamochi and Ibaraki's work on vertex orderings for graphs has been generalized to symmetric submodular functions by Queyranne [50] and further extended by Rizzi [51]. We set up the requisite notation in the abstract context before addressing the case of hypergraphs.

Let $f : 2^V \to \mathbb{R}$ be a real-valued set function defined over a finite set V. We confine our attention to symmetric functions, that is functions which satisfy the property that $f(A) = f(V \setminus A)$ for all $A \subseteq V$. A set 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 *normalized* if $f(\emptyset) = 0$. It is well-known that the cut capacity function of a hypergraph is symmetric and submodular.

For a given set function f over V, a *non-trivial minimizer* is a set in $\arg\min\{f(S) \mid \emptyset \subsetneq S \subsetneq V\}$. The *domain* of f, denoted $\mathrm{dom}(f)$, is V. We define $\lambda(x,y;f) = \min\{f(S) : S \subseteq V, x \in S, y \notin S\}$, and $\lambda(f) = \min_{x,y\in V} \lambda(x,y;f)$. Hence $\lambda(f)$ is the value of a non-trivial minimizer of f. The value of a hypergraph mincut is precisely $\lambda(c)$, where c is the cut function of H.

An ordered pair of vertices (s,t) is called a *pendant pair* for a function f if $f(\{t\}) = \lambda(s,t;f)$. A pendant pair for a hypergraph is defined naturally as a pendant pair of its cut function. There are three algorithms for computing a hypergraph mincut following the Nagamochi-Ibaraki approach of finding a pendant pair, contracting the pair, and recursing [35, 42]. This approach has been generalized to symmetric submodular functions by Queyranne [50], and even further by Rizzi [51], 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 vertex ordering is an ordering of the vertices $v_1, \ldots, v_n$. The set of vertices $V_i = \{v_1, \ldots, v_i\}$ are the first i vertices in the ordering. If $f(S) = g(S, \bar{S})$ for all $S \subseteq V$, we say g *realizes* f. Rizzi considered a *max-back ordering* of g. That is, an ordering of the vertices $v_1, \ldots, v_n$ such that

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

for all $j \geq i$.

Theorem 2.1 ([51]) *Suppose g realizes f, and let $v_1, \ldots, v_n$ be a max-back ordering of an order inducing function g over V. Then $f(\{v_n\}) = \lambda(v_{n-1}, v_n; f) = g(V_{n-1}, v_n)$.*

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 2.2 ([5]) *Suppose g realizes f, and let $v_1, \ldots, v_n$ be a max-back ordering of an order inducing function g over V. Then for $1 < i \leq n$, $\lambda(v_{i-1}, v_i; f) \geq g(V_{i-1}, v_i)$.*

It is easy to prove the following theorem via induction.

Theorem 2.3 *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 if $f(V) = 0$. In particular, we define a *Queyranne ordering* of a symmetric submodular function f as a max-back ordering with respect to d_f.

Lemma 2.4 *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 $d(A,B) = d'(A,B) = 0$ and it is 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, then $e \cap B = \emptyset$. This shows $d(A,B) = d'(A,B) = 0$ and $c(B) = 0$. The edge 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 $d_c(A,B) = \frac{1}{2}(1 + 1 - 1) = 1/2 = \frac{1}{2}(d(A,B) + d'(A,B))$, the desired result. □

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

Vertex orderings for hypergraphs We work with several vertex orderings defined for hypergraphs. Given a hypergraph $H=(V,E)$ and an ordering of the vertices $v_1,\ldots,v_n$, 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 vertex of e in the ordering. An ordering of the edges $e_1,\ldots,e_m$ is called a *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 vertex order.

Definition An ordering of vertices $v_1,\ldots,v_n$ 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 vertex 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 vertices 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 vertex 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 [35]. Mak and Wong used the tight ordering [42]. Queyranne defined an ordering for symmetric submodular functions [50] which when specialized to cut functions of hypergraphs is the one we define, see lemma 2.4. It is easy to verify the 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, 4 and 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 orderings can be used to find the mincut because of a more general phenomenon; every α-ordering is an order inducing function that realizes the hypergraph cut function.

Lemma 2.5 *The functions 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 is clear that d is symmetric. We have $d(A,B)=1$ 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 vertex in B, and at least one vertex in $A\cup C$. This shows $d(A,B\cup C)=1$ 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 $d'(A,B\cup C)=1$, 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 2.2.

Lemma 2.6 *Let $v_1,\ldots,v_n$ be an α-ordering of a hypergraph, then $\{v_n\}$ is a min v_{n-1}-v_n-cut.*

3 k-trimmed certificate and faster mincut 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 $A \subseteq V$. That is, 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 [28]. The fact that such a certificate exists is not hard to prove. One can generalize the forest decomposition technique for graphs [44] in a straightforward 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 vertices 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. If H' is a k-certificate of H and $\mathsf{size}(H') \leq 2k(n-1)$, 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 3.1 *Let $H = (V, E)$ be a hypergraph on n vertices and m edges with $\mathsf{size}(H) = p$. There is an algorithm that for a 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 $O(kn)$ time.*

The above theorem is tight for graphs. Our proof is an adaptation of that of Frank, Ibaraki and Nagamochi [20] 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 an 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 vertex 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.

Our goal is to show that H_k is a k-trimmed certificate of H, and there is a data structure to obtain it quickly.

First, we show that H_k can be retrieved quickly.

Lemma 3.2 *Let $H = (V, E)$ be a hypergraph on n vertices and m edges with $\mathsf{size}(H) = p$. There is an algorithm that creates a data structure in $O(p)$ time such that H_k can be retrieved in $O(\mathsf{size}(H_k))$ time.*

Proof: 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 $O(\mathsf{size}(H_k))$ 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 vertex v, we can sort all the backward edges of v using the head ordering in $O(p)$ time as follows: we maintain a queue Q_v for each vertex 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 vertex 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 vertex of e into Q_e). At the end of the process, each queue Q_e contains the vertices of an edge in E_k. The running time is $O(\mathsf{size}(H_k))$ since we only process $D_k(v)$ for each v. □

Next, we show that $\mathsf{size}(H_k) \leq 2k(n-1)$.

Lemma 3.3 *Let $H = (V, E)$ be a hypergraph on n vertices, for each k, $\mathsf{size}(H_k) \leq 2k(n-1)$.*

Proof: For a vertex v, the number of backward edges containing v is called the *backward sum of degrees*. Note the sum of degrees is no larger than twice the backward sum of degrees. Indeed, the number of edges cannot exceed the backward sum of degrees because each edge is a backward edge for some vertex, and each backward edge has exactly one head. Now we bound the backward sum of degrees of H_k. Each vertex v that is not v_1 is in at most k backward edges in H_k, therefore the backward sum of degrees is at most $k(n-1)$, and the sum of degrees is at most $2k(n-1)$. □

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

For all $i \leq j$,

$$d(V_{i-1}, v_i; H_k) \geq \min\{k, d(V_{i-1}, v_i; H)\} \geq \min\{k, d(V_{i-1}, v_j; H)\} \geq d(V_{i-1}, v_j; H_k).$$

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 vertex 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*

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

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

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

□

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: We fix j and prove by induction on i. For the base case consider $i = 1$. Indeed, in this case both sides of the inequality are 0 and the desired inequality holds trivially. Assume lemma is true for all $i' < i$. 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:

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

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 $v = v_i$,

$$\begin{aligned} |\gamma(V_{i-1}) \cap \delta(A')| &\geq |\gamma(V_{i-2}) \cap \delta(A')| + |E(v_{i-1}, v_i) \setminus E(V_{i-2}, v_{i-1}, v_i)| \\ &\geq d(V_{i-2}, v_i) + |E(v_{i-1}, v_i) \setminus E(V_{i-2}, v_{i-1}, v_i)| \\ &\quad \text{(apply induction on } i-1, A') \\ &= d(V_{i-1}, v_i) \\ &\geq d(V_{i-1}, v_j) \quad \text{(from MA-ordering)}. \end{aligned}$$

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 3.7 *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 $k = 0$.

Consider any $k > 0$. $|\delta_{H_i}(A)| \leq |\delta_{H_k}(A)|$ for all $i \leq k$, because H_i is a trimmed subhypergraph of H_k. If $|\delta_H(A)| = k' < k$, then by induction,

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

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 $\bar{A}$). Since e' does not cross A in H_k, there is a $v_j \in e \cap \bar{A}$ 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 $E(V_{j-1}, v_j; H_k) = E(V_i, v_j; H_k)$ and hence $d(V_{j-1}, v_j; H_k) = d(V_i, v_j; H_k) = k$.

$$\begin{aligned} k &= d(V_{j-1}, v_j; H_k) = d(V_i, v_j; H_k) \\ &= d(V_{i-1}, v_j; H_k) + d(v_i, v_j; H_k) - d(V_{i-1}, v_i, v_j; H_k). \end{aligned}$$

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

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, $|\gamma_{H_k}(V_{i-1}) \cap \delta_{H_k}(A)| \geq d(V_{i-1}, v_j; H_k)$. Combining this inequality with the preceding one,

$$|\gamma_{H_k}(V_{i-1}) \cap \delta_{H_k}(A)| \geq d(V_{i-1}, v_j; H_k) = k - d(v_i, v_j; H_k) + d(V_{i-1}, v_i, v_j; H_k). \tag{1}$$

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

$$\begin{aligned} k &> |\delta_{H_k}(A)| \quad \text{(by assumption)} \\ &\geq |\gamma_{H_k}(V_i) \cap \delta_{H_k}(A)| \\ &= |\gamma_{H_k}(V_{i-1}) \cap \delta_{H_k}(A)| + |E(v_i, v_j; H_k) \setminus E(V_{i-1}, v_i, v_j; H_k)| \\ &= |\gamma_{H_k}(V_{i-1}) \cap \delta_{H_k}(A)| + d(v_i, v_j; H_k) - d(V_{i-1}, v_i, v_j; H_k) \\ &\geq \big(k - d(v_i, v_j; H_k) + d(V_{i-1}, v_i, v_j; H_k)\big) + d(v_i, v_j; H_k) - d(V_{i-1}, v_i, v_j; H_k) \\ &\qquad \text{(from eq. (1))} \\ &= k \end{aligned}$$

This finishes the proof. □

Proof (Proof of theorem 3.1): A direct consequence of lemma 3.2, lemma 3.3 and theorem 3.7. □

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 $E = \{\{0, 1, 2\}, \{0, 2, 3\}, \{1, 2\}\}$. The sequence $0, 1, 2, 3$ is a tight ordering. Hypergraph H_2 is H with edge $\{1, 2\}$ removed. One can see $\lambda_{H_2}(1, 2) = 1 < 2 = \lambda_H(1, 2)$.

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

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

$0, 1, 2, 3, 4$ is a Queyranne ordering. Hypergraph 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 global mincut we obtain the following theorem.

Theorem 3.8 *The global mincut of an uncapacitated hypergraph H with n vertices and $\mathsf{size}(H) = p$ can be computed in $O(p + \lambda n^2)$ time, where λ is the value of the mincut of H.*

Proof: Using theorem 3.1, 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 $\mathsf{size}(H_k) = O(\lambda n)$. To find k we apply exponential search for the smallest i such that $2^i > \lambda$. Each search computes hypergraph mincut on H_{2^i}, which takes $O(2^i n^2)$ time. For any $k > \lambda$, the value of the mincut on the k-trimmed certificate equals to λ. Therefore, the search stops when the value of mincut 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 s-t 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 $T(n, m, \lambda(s, t)n)$. A k-trimmed certificate can also help in computing α-approximate mincuts 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 [28]. 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 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 vertices $v_1, \ldots, v_n$. Let $e_1, \ldots, e_m$ be the induced head ordering of the edges, just as before. The algorithm is similar to the uncapacitated version. For a vertex 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 3.9 *There is an algorithm that fir a 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.*

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 mincuts. Cunningham [13] proved that such decomposition exists for an arbitrary non-negative submodular function, following previous work by Cunningham and Edmonds [14] and Fujishige [22]. Cheng [12] showed that the canonical decomposition can be used to efficiently, and relatively easily, build a hypercactus representation. Subsequently Fleiner and Jordán [17] 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 mincut. Here we show an efficient algorithm for finding such a mincut in a hypergraph, and then use it to build a canonical decomposition. We can then construct a hypercactus representation from the canonical decomposition as shown in [12]. 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 vertex. A *split* is a non-trivial mincut. An s-t split is a split that separates s and t.

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 mincut value λ, outputs either a split in H or a pair of vertices $\{s,t\}$ such that there is no s-t 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 s-t flow in $\vec{H}$ to help decide whether there is an s-t split, and output one if it exists.

Lemma 4.1 *Given a maximum s-t flow in the equivalent digraph of H, and the value of mincut λ in H, there is an algorithm that in $O(p)$ time either finds a s-t split, or certifies that no s-t split exists in H.*

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

Suppose a directed graph G has k minimum u-v-cuts for some vertex pair (u,v). Given a maximum u-v flow in G and an integer ℓ, there is an enumeration algorithm [49] 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-s-t cut in $\vec{H}$ corresponds to a min-s-t cut in H. If the algorithm returns at most two cuts and both are trivial then there is no s-t split. Otherwise one of the output cuts is an s-t split. The running time is $O(p)$ since the number of edges in $\vec{H}$ is $O(p)$. □

One can find a maximum s-t flow in $\vec{H}$ using standard flow algorithms but that would not lead to a near-linear time algorithm. In graphs, Arikati and Mehlhorn [2] devised a linear-time algorithm that computes the maximum flow between the last two vertices 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 vertices 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 vertices of e under the tight ordering. The key observation is the following.

Lemma 4.2 *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 $1 \leq i < j \leq n$, $d'(V_i, v_j; G) = d'(V_i, v_j; H)$.*

Proof: Consider any edge e counted in $d'(V_i, v_j; H)$, e' is counted in $d'(V_i, v_j; G)$. This shows that $d'(V_i, v_j; H) \leq d'(V_i, v_j; G)$.

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

The preceding lemma implies that the tight ordering for H is a tight ordering for G. From lemma 2.6,

$$\lambda(v_{n-1}, v_n; G) = d'(V_{n-1}, v_n; G) = d'(V_{n-1}, v_n; H) = \lambda(v_{n-1}, v_n; H).$$

Letting $s = v_{n-1}$ and $t = v_n$, we see that $\lambda(s,t;G) = \lambda(s,t;H)$. Moreover, an s-t flow in G can be easily lifted to an s-t flow in $\vec{H}$. Thus, we can compute an s-t max flow in G in linear-time using the algorithm of [2] and this can be converted, in linear time, into an s-t max-flow in $\vec{H}$.

This gives the following theorem.

Theorem 4.3 *The split oracle can be implemented in $O(p + n\log n)$ time for capacitated hypergraphs, and in $O(p)$ time for uncapacitated hypergraphs.*

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 mincuts of H are trivial. A capacitated hypergraph is called a *solid polygon* if it consists of a graph cycle where each edge has the same capacity a, and a hyperedge containing all vertices 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 vertices. For a semi-brittle hypergraph with at least 4 vertices, every split consists of two edges of the graph cycle and the hyperedge covering all vertices. For a brittle hypergraph with at least 4 vertices, 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 vertex 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* vertex 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 mincut 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 vertices. 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 vertices. 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 vertex.

The important theorem below is due to [13], and stated again in [12] specifically for hypergraphs.

Theorem 4.4 ([13]) *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 vertices.*

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 mincut information in H as stated below.

Theorem 4.5 ([12, 13]) *Let $\mathcal{D} = \{H_1, \ldots, H_k\}$ be a canonical decomposition of H.*

1. *For each mincut S of H, there is a unique i, such that $\phi_{H_i}(S)$ is a mincut of H_i.*
2. *For each mincut S of H_i, $\phi_{H_i}^{-1}(S)$ is a mincut 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 mincuts 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 vertex sets of the hypergraphs in $\mathcal{D}$.

Recall that a set of edges $E' \subseteq E$ is called a min edge-cut-set if $E' = \delta(S)$ for some mincut S. As a corollary of the preceding theorem, one can easily prove that there are at most $\binom{n}{2}$ distinct min edge-cut-sets in a hypergraph. This fact is not explicitly stated in [12, 13].

Corollary 4.6 *A hypergraph with n vertices has at most $\binom{n}{2}$ distinct min edge-cut-sets.*

Proof: Let H be a hypergraph on n vertices. If H is prime, then there are at most n min edge-cut-sets. If H is a solid-polygon, then there are at most $\binom{n}{2}$ min edge-cut-sets. Let $\mathcal{D}$ be a canonical decomposition of H. The decomposition $\mathcal{D}$ is obtained via a simple refinement $\{H_1, H_2\}$ of H with size a and b, followed by further refinements. Then, by induction, there are at most $\binom{a}{2} + \binom{b}{2}$ min edge-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 [26].

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 $(A, \bar{A})$ and $(B, \bar{B})$ 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 vertices. 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 good splits [14, Theorem 3] via the following simple algorithm. If H is prime or a solid polygon return $\{H\}$ itself. Otherwise find a good split $(A, \bar{A})$ 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 expensive.

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 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 [13], 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 vertex 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 vertex 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

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

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 4.7 *$\mathcal{D}_x$ 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 4.8 *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 vertex, 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 vertices 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 vertices. 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 4.9 *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, if 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 $O(np)$. Our 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 then $\mathcal{D}_i = \{H_i\}$. If H_i is a solid polygon then $\mathcal{D}_i$ is a standard decomposition of H_i. Our 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 4.8, 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 4.10 *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.

4.4 Computing a prime decomposition

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

Definition Consider a hypergraph H. Let $H' = H/\{s, t\}$, where $\{s, t\}$ is contracted to vertex $v_{\{s,t\}}$. 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

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

See fig. 4.1 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 s-t split.

```
PRIME(H, λ)
  if |V(H)| ≥ 4
      x ← a new marker vertex
      query the split oracle with H and λ
      if oracle returns a split (S, V(H)−S)
          {H_1, H_2} ← REFINE(H, (S, V(H)−S), x)
          return PRIME(H_1, λ) ∪ PRIME(H_2, 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
              {G_1, G_2} ← refinement of G induced by {s,t}
              D ← (D' − {G'}) ∪ {G_1, G_2}
          else
              D ← (D' − {G'}) ∪ G
          return D
  else
      return {H}
```

Figure 4.1: The algorithm for computing a prime decomposition.

Lemma 4.11 *Suppose H is a hypergraph with no s-t split for some $s,t \in V(H)$. Let $H' = H/\{s,t\}$, where $\{s,t\}$ is contracted to vertex $v_{\{s,t\}}$. Let $\mathcal{D}'$ be a prime decomposition of H', and let $G' \in \mathcal{D}'$ such that G' contains vertex $v_{\{s,t\}}$. Let G be obtained through uncontracting $v_{\{s,t\}}$ in G' with respect to H.*

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 s-t split in G because H does not have any s-t 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 vertex $v \in A$, 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 4.12 PRIME(H, λ) *outputs in $O(n(p + T(n,m,p)))$ time a prime decomposition, where $T(n,m,p)$ is the time to query split oracle with a hypergraph of n vertices, m edges and sum of degree p.*

Proof: Using induction and lemma 4.11, 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 4.3 we obtain the following corollary.

Corollary 4.13 *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.*

4.5 Reducing space usage

Our 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 vertex. With each a_i we also store $V(H_i)$ which includes the marker vertices and some vertices from $V(H)$. It is easy to see that the total storage for the tree and storing the vertex sets is $O(n)$; a marker vertex appears in exactly two of the hypergraphs of a decomposition and a vertex of H in exactly one of the hypergraphs in the decomposition.

Given the decomposition tree T, vertex sets $V(H_1), \ldots, V(H_k)$ and a node $a_i \in A$, we can recover the hypergraph H_i (essentially the edges of H_i since we store the vertex 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. The set of vertices in H which are contracted to a single marker vertex in H_i corresponding to the edge e is $V(H) \cap (\cup_{a_j \in C_e} V(H_j))$. 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.

4.6 Hypercactus representation

Note the similarity of theorem 4.5 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 hypercactus representation that captures all mincuts [12] (he called it a "structure hypergraph"). When H is a graph, the hypercactus is a cactus. We describe the details of the construction of a hypercactus from the canonical decomposition, for the sake of completeness.

Assume without loss of generality that $\lambda(H) = 1$. We construct the hypercactus given the canonical decomposition. First, we consider the case where the decomposition consists of only a prime or a solid polygon. If H is a solid polygon, then it consists of a cycle and a hyperedge containing all the vertices. If the cycle has non-zero capacity, we define H^* to be the hypergraph consisting of a single cycle obtained from the solid polygon H by removing the hyperedge that contains all the vertices. Each edge of the cycle in H^* is assigned a capacity of $\frac{1}{2}$. If the cycle has zero capacity, then let H^* is single hyperedge that contains all vertices, and this hyperedge is assigned capacity 1. In both cases, (H^*, ϕ), where ϕ is the identity function, is a hypercactus representation.

If H is prime, let V' be the set of vertices that induce a trivial mincut, i.e. $v \in V'$ iff $\{v\}$ is a mincut in H. Introduce a new vertex v_H, and define $H^* = (\{v_H\} \cup V', \{\{v_H, v'\} \mid v' \in V'\})$, with capacity 1 for each edge; in other words we create a star with center v_H and the leaves being the vertices of V'. Define $\phi : V(H) \to V(H^*)$ as

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

Then H^* is a hypercactus, and (H^*, ϕ) is a representation of H, respectively.

For the more general case, let $\mathcal{D}^* = \{H_1, \ldots, H_k\}$ be the canonical decomposition of H. For each i, construct the hypercactus cut representation (H_i^*, ϕ_i) of H_i as described earlier. We observe that if x is a marker vertex in H_i, then it is also present in H_i^*. The hypergraph H^* is defined to be the hypergraph constructed from $H_1^*, \ldots, H_k^*$ by identifying marker vertices. The hypergraph H^* is a hypercactus; each marker vertex is an articulation vertex and each H_i^* is either a block (if H_i^* is a single hyperedge or a cycle), or H_i^* is a union of blocks where each block is an edge (if H_i^* is a star). The construction also gives us the mapping $\phi : V(H) \to V(H^*)$ obtained by gluing together $\phi_1, \ldots, \phi_k$ in the natural fashion. The pair (H^*, ϕ) is a hypercactus representation of H.

The preceding construction and the algorithm for computing the canonical decomposition gives the following theorem.

Theorem 4.14 *A hypercactus representation of a capacitated hypergraph can be found in $O(n(p + n \log n))$ time and $O(p)$ space. For uncapacitated hypergraphs the run time improves to $O(np)$.*

Proof: We combine theorem 4.10 and corollary 4.13. The space usage can be made $O(p)$ based on the discussion in Section 4.5. □

If H is a graph, (H^*, ϕ) constructed in theorem 4.14 is a cactus representation. The asymptotic time and space bounds obtained in theorem 4.14 match those of the algorithm from [47]. We believe that our approach is conceptually simpler.

Via sparsification we obtain a faster algorithm for uncapacitated hypergraphs.

Theorem 4.15 *A hypercactus representation of an uncapacitated hypergraph can be found in $O(p + \lambda n^2)$ time and $O(p)$ space.*

Proof: We can find the mincut value λ, and a $(\lambda+1)$-sparsifier H' of H in $O(p+\lambda n^2)$ time. theorem 3.7 shows that every mincut in H is a mincut in H', and vice versa. Therefore the hypercactus representation for H' is a hypercactus representation for H. We apply theorem 4.14 to H' to obtain a hypercactus of H in the claimed time and space bounds. □

For graphs Gabow [25] achieves a running time of $O(m+\lambda^2 n\log(n/\lambda))$ which is faster than the $O(p+\lambda n^2) = O(m+\lambda n^2)$ run time given by the preceding theorem when $\lambda = O(n)$. Gabow's algorithm is quite complex and it is an open problem whether one can achieve a similar running time for hypergraphs, even when they have fixed rank.

5 Near-linear time $(2+\varepsilon)$ approximation for mincut

Matula gave an elegant use of MA-ordering to obtain a $(2+\varepsilon)$-approximation for the mincut in an uncapacitated undirected graph in $O(m/\varepsilon)$ time [43]. Implicit in his paper there is an algorithm that gives a $(2+\varepsilon)$-approximation for capacitated graphs in $O(\frac{1}{\varepsilon}(m\log n + n\log^2 n))$ time; this was explicitly pointed out by Karger [30]. Here we extend Matula's idea to hypergraphs. We describe an algorithm that outputs a $(2+\varepsilon)$-approximation for hypergraph mincut in $O(\frac{1}{\varepsilon}(p\log n + n\log^2 n))$ time for the capacitated case, and in $O(p/\varepsilon)$ 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 $(2+\varepsilon)$-approximation of the minimizer using $O(\frac{n^2\log n}{\varepsilon})$ 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 $d_f(A, B\cup C) \le d_f(A,B) + d_f(A,C)$ for all disjoint A, B, C. The function d_f is *additive* if the 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 weighted sum of nice functions is also nice.

Theorem 5.1 *The cut function of a hypergraph is nice.*

Proof: It suffices to prove the theorem for hypergraphs with a single edge since we can then use the fact that a general hypergraph cut function is the sum of single-edge hypergraph cut functions.

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

$$d_c(A, B\cup C) \le d_c(A,B) + d_c(A,C).$$

If $d_c(A, B\cup C) = 0$, then the inequality is clearly true. Hence we can assume $d_c(A, B\cup C)$ is either $\frac{1}{2}$ or 1. Suppose $d_c(A, B\cup C) = 1$. Since $d_c(A,B) = \frac{1}{2}(d(A,B) + d'(A,B))$ for all A and B, we have $d(A, B\cup C) = d'(A, B\cup C) = 1$. 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 $d_c(A, B\cup C) = 1 = \frac{1}{2}d(A,B) + \frac{1}{2}d(A,C) \le d_c(A,B) + d_c(A,B)$. 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 $d_c(A, B\cup C) = 1 = \frac{1}{2}(d(A,B) + d'(A,B)) = d_c(A,B) \le d_c(A,B) + d_c(A,C)$.

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 $d_c(A, B\cup C) = \frac{1}{2} \le \frac{1}{2}d(A,B) + \frac{1}{2}d(A,C) \le d_c(A,B) + d_c(A,C)$. □

The preceding proof shows that the connectivity function of a graph cut function is additive. Are all nice functions hypergraph cut functions? The answer is no. Yamaguchi showed that 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 [57, Remark 2.]. It is easy to see that $h-g$ has subadditive connectivity since g has additive connectivity.

We now consider nice functions instead of hypergraph cut functions. After we describe the algorithm and analysis in the more general setting, we estimate the running time of the algorithm for the specific case of hypergraphs. We define $\sigma(f) = \sum_{v\in V} f(v)$ and $\delta(f) = \min_{v\in V} f(v)$. Given a set function f over V and $S \subseteq V$ we can obtain a derived set function f' by contracting S into a new element s as follows. The function $f' : 2^{V'} \to \mathbb{R}$ where $V' = (V\setminus S)\cup\{s\}$ is defined as: $f'(X) = f(X)$ if $s \notin X$, and $f'(X) = f(S\cup(X\setminus\{s\}))$ if $s \in X$. Similarly one can define, inductively, a function obtained from f by contracting several disjoint sets. It is not hard to see that nice functions are closed under the contraction operation.

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

$$d_f(V_i, v_{i+1}) \geq d_f(V_i, v_j)$$

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

Lemma 5.2 *Let f be any normalized symmetric set function on $\{v_1, \ldots, v_n\}$. Consider any ordering of the vertices $v_1, \ldots, v_n$, then*

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

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

$$\begin{aligned}
\sum_{i=1}^{n} d_f(V_{i-1}, v_i) &= \frac{1}{2}\sum_{i=1}^{n}(f(V_{i-1}) + f(v_i) - f(V_i)) \\
&= \frac{1}{2}\sum_{i=1}^{n}(f(V_{i-1}) - f(V_i)) + \frac{1}{2}\sum_{i=1}^{n} f(v_i) \\
&= \frac{1}{2}(f(V_0) - f(V_n)) + \frac{1}{2}\sum_{i=1}^{n} f(v_i) \\
&= \frac{1}{2}\sum_{i=1}^{n} f(v_i) \\
&= \frac{1}{2}\sigma(f)
\end{aligned}$$

□

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 5.3 *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:

$$\begin{aligned}
\sigma(f') &= \sum_{i=1}^{h} f(X_i) \\
&= 2\sum_{i=1}^{h} d_f(A_{i-1}, X_i) && \text{(lemma 5.2)} \\
&\leq 2\sum_{i=1}^{h}\sum_{x \in X_i} d_f(A_{i-1}, x) && \text{(subadditivity)} \\
&< 2\sum_{i=1}^{h} \alpha |X_i| \leq 2\alpha n.
\end{aligned}$$

□

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

```
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 5.1: Description of $(2+\varepsilon)$-approximation algorithm. It is easy to remove the recursion.

Lemma 5.4 *Let f be a non-negative symmetric submodular function over V and let $v_1, \ldots, v_n$ be a Queyranne ordering of the vertex set V. If v_i and v_j are in an α-tight set then $\lambda(v_i, v_j; f) \geq \alpha$.*

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 v_i and v_j are in the same α-tight set. By theorem 2.2, $\lambda(v_k, v_{k+1}; f) \geq d_f(V_k, v_{k+1}) \geq \alpha$. By induction and using the fact that for any $a, b, c \in V$, $\lambda(a, c; f) \geq \min(\lambda(a, b; f), \lambda(b, c; f))$, we have $\lambda(v_i, v_j; f) \geq \alpha$. □

fig. 5.1 describes a simple recursive algorithm for finding an approximate mincut.

Theorem 5.5 APPROXIMATE-MINIMIZER *outputs a $(2+\varepsilon)$-approximation to an input nice function f, and can be implemented in $O(\frac{n^2}{\varepsilon} \log \frac{n\delta(f)}{\lambda(f)})$ oracle calls.*

Proof: We first argue about the termination and run-time. From lemma 5.3, $\sigma(f') \leq \frac{2}{2+\varepsilon} n\delta(f)$. Since $\sigma(f) \geq n\delta(f)$, we see that each recursive call reduces the σ value of the function by a factor of $\frac{2}{2+\varepsilon}$. The number of vertices strictly decreases after each recursive call, because otherwise σ value would not change. This ensures termination.

Let f'' be any non-trivial normalized symmetric submodular function (that has at least two vertices) 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, $\sigma(f') \leq \frac{2}{2+\varepsilon} n\delta(f)$. Thus the total number of recursive calls is $O(\varepsilon^{-1} \log(\frac{n\delta(f)}{\lambda(f)}))$. 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 [50]. 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 mincut value if $n = 1$ or if $\delta = 0$. Assume $n \geq 2$ and $\delta(f) > 0$. The number of vertices 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, $\lambda(f') \leq \lambda' \leq (2+\varepsilon)\lambda(f')$. If $\lambda(f') = \lambda(f)$ then the algorithm outputs a $(2+\varepsilon)$-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 5.4, $\lambda(x, y; f) \geq \alpha = \frac{1}{2+\varepsilon}\delta$ which implies that $\delta \leq (2+\varepsilon)\lambda(f)$. Since the algorithm returns $\min(\delta, \lambda')$ we have that the output is no more than $(2+\varepsilon)\lambda(f)$. □

Since $\delta(f)$ can be much larger than $\lambda(f)$, we can preprocess the function to reduce δ to at most $n^2\lambda(f)$ to obtain a strongly polynomial run time.

Lemma 5.6 *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 5.4, $\lambda(u, v; f) \geq \beta$ 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,

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

□

Let β be the value in lemma 5.6, 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 $(2+\varepsilon)$ approximation algorithm to f' gives us the following corollary.

Corollary 5.7 *A $(2+\varepsilon)$ approximation for a nice function can be computed in $O(\varepsilon^{-1}n^2 \log n)$ oracle calls.*

We now specialize the algorithm to hypergraphs.

Corollary 5.8 *A $(2+\varepsilon)$ approximation for hypergraph mincut can be computed in $O(\varepsilon^{-1}(p + n\log n)\log n)$ time for capacitated hypergraphs, and in $O(\varepsilon^{-1}p)$ time for uncapacitated hypergraphs.*

Proof: 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. The number of recursive calls is $O(\frac{1}{\varepsilon}\log n)$. For the capacitated hypergraphs, we obtain the running time $O(\varepsilon^{-1}(p + n\log n)\log n)$.

For uncapacitated hypergraphs, recall p is $\sigma(f)$, where f is the cut function of the hypergraph. The running time in each recursive call is dominated by computing a Queyranne ordering, which takes $O(p)$ time. In the proof of Theorem 5.5, p reduces by a factor of $2/(2+\varepsilon)$ after each recursive call. The running time is therefore $O(\sum_{i=1}^{\infty}(\frac{2}{2+\varepsilon})^i p) = O(\varepsilon^{-1}p)$. □

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

6 Strength estimation and cut sparsifiers

The goal of this section is to 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 [36].

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 vertex 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 mincut value over all vertex induced subhypergraphs that contain e.

The main technical result of this section is the following.

Theorem 6.1 *Let $H = (V, E)$ be a rank r capacitated hypergraph on n vertices. 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 $O(\alpha)$-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 vertices contained in the edge. The minimum strength of the replaced edges might be an $O(\alpha)$-approximation. Unfortunately, this will not work even when then rank is 3. Consider the following hypergraph H with vertices $\{v_1, \ldots, v_n\}$. 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.

Our proof for the preceding theorem closely follows the corresponding theorem for graphs from [4]. A key technical tool for graphs is the deterministic sparsification algorithm for edge-connectivity due to Nagamochi and Ibaraki [44]. Here we rely on our generalization to hypergraphs from Section 3. Before we describe the proof we discuss some applications in the next subsection.

6.1 Applications

A capacitated hypergraph $H' = (V, E')$ is a $(1 \pm \varepsilon)$-cut approximation of a capacitated hypergraph $H = (V, E)$, if for every $S \subseteq V$, the cut value of S in H' is within $(1 \pm \varepsilon)$ factor of the cut value of S in H. Below we state formally the sampling theorem from [36].

Theorem 6.2 ([36]) *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 $p_e = \min\left(\frac{3((d+2)\ln n + r)}{\gamma_H(e)\varepsilon^2}, 1\right)$ and capacity $c(e)/p_e$ if sampled. With probability at least $1 - O(n^{-d})$, the hypergraph H_ε has $O(n(r + \log n)/\varepsilon^2)$ edges and is a $(1 \pm \varepsilon)$-cut-approximation 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 α-approximate strength also suffices to get a $(1 \pm \varepsilon)$-cut-approximation. That is, if we replace γ with α-approximate strength γ', the sampling algorithm will still output a $(1 \pm \varepsilon)$-cut-approximation of H. Indeed, the lower bound property shows that 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 that the hypergraph H_ε will have $O(\alpha n(r + \log n)/\varepsilon^2)$ edges.

From theorem 6.1, we can find an $O(r)$-approximate strength function γ' in $O(p \log^2 n \log p)$ time.

Corollary 6.3 *A $(1 \pm \varepsilon)$-cut approximation of H with $O(nr(r + \log n)/\varepsilon^2)$ edges can be found in $O(p \log^2 n \log p)$ time with high probability.*

The number of edges in the $(1 \pm \varepsilon)$-cut approximation is worse than theorem 6.2 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.

Mincut: The best running time known currently to compute a (global) mincut in a hypergraph is $\tilde{O}(pn)$ [35, 42]. We have already seen, in Section 5, an algorithm that obtains a $(2+\varepsilon)$-approximation in $\tilde{O}(p/\varepsilon)$ time. Via Corollary 6.3 we can first sparsify the hypergraph and then apply the $\tilde{O}(pn)$ time mincut algorithm to the sparsified graph. This gives a randomized algorithm that, with high probability, outputs a $(1+\varepsilon)$-approximate mincut in a rank r hypergraph in $O(n^2 r^2(r + \log n)/\varepsilon^2 + p \log^2 n \log p)$ time. For small r the running time is $\tilde{O}(p + n^2/\varepsilon^2)$ for a $(1+\varepsilon)$-approximation.

s-t mincut: The standard technique to compute an s-t mincut in a hypergraph is computing an s-t maximum flow in an associated capacitated digraph with $O(n+m)$ vertices and $O(p)$ edges [38]. An s-t maximum flow in such graph can be found in $\tilde{O}(p\sqrt{n+m})$ time [39]. Via sparsification corollary 6.3, we can obtain a randomized algorithm to find a $(1+\varepsilon)$-approximate s-t mincut in a rank r hypergraph in $\tilde{O}(n^{3/2} r^{5/2}(r + \log n)^{3/2}/\varepsilon^3 + p)$ time. For small r the running time is $\tilde{O}(p + n^{3/2}/\varepsilon^3)$ for a $(1+\varepsilon)$-approximation.

6.2 Properties of edge strengths

A *k-strong component* of hypergraph H is an inclusion-wise maximal set of vertices $U \subseteq V$ 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 connected components of a hypergraph H.

We start with a simple observation that simplifies later proofs.

Proposition 6.4 *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 6.5 *Let $A, B \subseteq V$ such that $A \cap B \neq \emptyset$. We have $\lambda(H[A \cup B]) \geq \min(\lambda(H[A]), \lambda(H[B]))$. In particular the k-strong components are pairwise disjoint.*

Proof: Note that $H[A \cup B]$ contains every edge that belongs to $H[A]$ or $H[B]$. Consider an arbitrary cut S in $H[A \cup B]$; S crosses A or S crosses B which implies that $|\delta_{H[A\cup B]}(S)| \geq \min\{\lambda(H[A]), \lambda(H[B])\}$. Since S was arbitrary we have $\lambda(H[A \cup B]) \geq \min\{\lambda(H[A]), \lambda(H[B])\}$ as desired.

Consider k-strong components A and B such that $A \cap B \neq \emptyset$. From the preceding argument, $\lambda(H[A \cup B]) \geq \min(\lambda(H[A]), \lambda(H[B])) \geq k$. Maximality of k-strong components implies that $A = A \cup B = B$. □

Lemma 6.6 *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 6.7 (Extends Lemma 4.5 and Lemma 4.6 [4]) *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, deleting a k-weak edge does not decrease the strength of a k-strong edge, and contracting a k-strong edge does not increase the strength of a k-weak edge.

Proof: We prove the two claims separately. By proposition 6.4, to prove the equality of either claim, only one side of the inequality has to be proven.

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 vertex in H' obtained by contracting e'. Let $U' \subseteq V(H')$ be a 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 $\gamma_{H'}(f) = \gamma_H(e)$. Now we consider the case when $v_{e'} \in U'$. Let U be the set of vertices in $V(H)$ obtained by uncontracting $v_{e'}$ in U'. That is, $U = (U' \setminus v_{e'}) \cup e'$. Note that $e \subseteq U$.

Let W be a set that certifies the strength of e' in H, namely $\gamma_H(e') = \lambda(H[W])$ and $e' \subseteq W$. We now consider the graph $H[U \cup W]$ and prove a lower bound on its mincut value. Consider any cut $S \subseteq U \cup W$ in $H[U \cup W]$. If S crosses W or is strictly contained in W (that is $S \subsetneq W$), 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 the case when $S \subseteq U \setminus W$. Since $e' \subseteq U \cap W$, $S \neq U$ and S is also a cut in $H'[U']$, and therefore $|\delta_{H[U]}(S)| = |\delta_{H'[U']}(S)|$. From these two observations we have

$$\lambda(H[U \cup W]) \geq \min(\lambda(H'[U']), \lambda(H[W])).$$

But $\lambda(H[W]) = \gamma_H(e') > \gamma_H(e) \geq \lambda(H[U \cup W])$. Therefore, $\gamma_H(e) \geq \lambda(H[U \cup W]) \geq \lambda(H'[U']) = \gamma_H(f)$.

□

Theorem 6.8 (Extends Lemma 4.10 [4]) *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 6.6. 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 mincut of H' is at least 1.

Let S be a mincut 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 6.9 (Extends Lemma 4.11 [4]) *For an uncapacitated hypergraph $H = (V, E)$ with at least* 1 *vertex,*

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

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 vertices. There exists a cut X of cost 1 in $H[U]$ by theorem 6.8.

Let $H' = H - \delta(X)$. H' has at least one more connected component than H. By Proposition 6.4, 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, $\sum_{e \in E(H')} \zeta_{H'}(e) \leq (n - \kappa(H')) \leq t - 1$.

```
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 6.1: The estimation algorithm

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,

$$\begin{aligned}
\sum_{e \in E(H)} \zeta_H(e) &= \sum_{e \in \delta(X)} \zeta_H(e) + \sum_{e \in E(H')} \zeta_H(e) \\
&= 1 + \sum_{e \in E(H')} \zeta_H(e) \\
&\leq 1 + \sum_{e \in E(H')} \zeta_{H'}(e) \\
&\leq 1 + (t-1) = t.
\end{aligned}$$

□

Corollary 6.10 *The number of k-weak edges in an uncapacitated hypergraph H on n vertices is at most $k(n - \kappa(H))$.*

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 6.1. 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 k-weak edge in E; the important aspect here is that E' 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 fig. 6.1. The algorithm repeatedly calls WEAKEDGES(H, k) for increasing values of k while removing the edges found in previous iterations.

Lemma 6.11 *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 have $E_i = E_{i-1} - F_i$. Thus, for any edge $e \in E_i$, $\gamma_{H_i}(e) \geq 2^i k$. This proves the claim.

Since $F_i \subseteq E_{i-1}$, it follows from the previous claim that $\gamma_H(e) \geq \gamma_{H_{i-1}}(e) \geq 2^{i-1}k$ 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 6.12 *If* WEAKEDGES(H,k) *an αk-light set of 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 edge 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,

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

□

6.3.1 Implementing WEAKEDGES

We now 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 fig. 6.2 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 6.2: Algorithm for returning a $4rk$-light set of all k-weak edges in H.

Theorem 6.13 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 vertex 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 6.10. It also implies that $\sum_v \deg(v) \leq rk(n-1)$. By Markov's inequality at least half the vertices have degree less than $2rk$. For any vertex v with degree less than $2rk$, all edges incident to it are $2rk$-crisp. Thus, after the first iteration, all such vertices 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 vertices in each component will be isolated in the first iteration. Therefore, in $\log n$ iterations, all vertices 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 vertex. The is well defined because the k-strong components are disjoint by lemma 6.5.

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 $|E_i| \leq 4rk(\kappa(H_{i+1})-\kappa(H_i))$.

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

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 3.9, 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

```
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 6.3: Algorithm for returning a $2k$-light k-partition.

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 fig. 6.3 for the formal description of the algorithm.

Theorem 6.14 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' be 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.

We now argue for termination and running time by showing that the number of vertices halves in each recursive call. Assume H contains n vertices, the algorithm finds a k-sparse certificate and contracts all edges not in the certificate. The resulting hypergraph has n' vertices and m' edges. We have $m' \leq k(n-1)$ by theorem 3.9. If $n' - 1 \leq (n-1)/2$, then the number of vertices halved. Otherwise $n' - 1 > (n-1)/2$, then the number of edges is $m' \leq k(n-1) < 2k(n'-1)$, and the algorithm terminates in the next recursive call.

The running time of the algorithm for a size p hypergraph with n vertices is $T(p,n)$. The running time $T(p,n)$ 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 6.15 *Let $H = (V,E)$ be an 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 6.11, lemma 6.12 and theorem 6.13, 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, so 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 6.14 and theorem 6.13. 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 $O(p \log^2 n \log p)$. □

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 3.9, 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 6.14 and theorem 6.13 still holds for capacitated hypergraph.

Lemma 6.16 *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 6.16 can be improved to strongly polynomial time by using the windowing technique [4].

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 6.7, $\gamma_{H_i}(e') = \gamma_H(e)$. The total capacity of H_i is at most $p b_i \leq p^3 a_i$. We can run ESTIMATION(H_i) to estimate γ_{H_i} since the ratio between the lower bound a_i and upper bound $p b_i$ is p^3. Let p_i be the size of H_i, and n_i be the number of vertices in H_i, so the running time for ESTIMATION(H_i) is $O(p_i \log^2 n_i \log p_i)$ by lemma 6.16. 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, 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 $d_e \leq \gamma(e) \leq p d_e$. The maximal intervals in $\bigcup_{e \in E} [d_e, p d_e]$ are the desired intervals. We now describe the procedure to find the values d_e for all $e \in E$.

Definition 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 vertex in e, and the star spans each vertex 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 vertices in e. Note that all the leaves of T_e are vertices from e.

For any two vertices 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 $d_e \leq \gamma(e) \leq p d_e$.

Let $V_e = \bigcup_{e' \in T_e} \pi(e')$. Certainly, $\gamma(e) \geq \lambda(H[V_e])$, because all vertices 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 $p d_e$. By lemma 6.6, $\gamma(e) \leq p d_e$. 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 vertices 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 Kruskal's greedy algorithm [37]. 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. [7] To compute d_e, we fix some vertex 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 6.17 *Given a capacitated hypergraph H, we can find a value d_e for each edge e, such that $d_e \leq \gamma(e) \leq p d_e$ in $O(p + n \log n)$ time.*

The preceding lemma gives us the desired intervals. Using lemma 6.16, we have the desired theorem.

Theorem 6.18 *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.*

7 Concluding Remarks

We close with some open problems. The main one is to find an algorithm for hypergraph mincut 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 [31]. Can Karger's algorithm be extended to hypergraphs with fixed rank r? Recently there have been several fast s-t max-flow algorithms for undirected and directed graphs. The algorithms for directed graphs [39, 41] have straightforward implications for hypergraph s-t cut computation via the equivalent digraph. However, hypergraphs have additional structure and it may be feasible to find faster (approximate) algorithms.

We described a linear time algorithm to find a maximum flow between the last two vertices 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 vertices 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 6.3. Kogan and Krauthgamer [36] gave 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?

Some of the research in this paper was inspired by work on element connectivity and we refer the reader to [8] for related open problems.

Acknowledgments

We thank Yosef Pogrow for pointing out a flaw in the proof of theorem 3.7 in a previous version of the paper. We also like to thank Tao Du for pointing out an issue in sentences leading up to Corollary 5.8. We thank Yutaro Yamaguchi for discussions on realizing hypergraph cut functions. We thank the reviewers for several useful comments and corrections.

References

[1] Hassene Aissi, Ali Ridha Mahjoub, S. Thomas McCormick, and Maurice Queyranne. Strongly polynomial bounds for multiobjective and parametric global minimum cuts in graphs and hypergraphs. *Math. Program.*, 154(1-2):3–28, 2015.

[2] Srinivasa R. Arikati and Kurt Mehlhorn. A Correctness certificate for the Stoer-Wagner min-cut algorithm. *Information Processing Letters*, 70(5):251–254, 1999.

[3] Joshua Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721, 2012.

[4] András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM J. Comput.*, 44(2):290–319, 2015. Preliminary versions appeared in STOC '96 and SODA '98.

[5] Michael Brinkmeier. Minimizing symmetric set functions faster. *CoRR*, abs/cs/0603108, 2006.

[6] Karthekeyan Chandrasekara, Chao Xu, and Xilin Yu. Hypergraph k-cut in randomized polynomial time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1426–1438.

[7] Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(1):337–361, 1987.

[8] Chandra Chekuri. Some open problems in element connectivity. Unpublished Survey. Available at http://chekuri.cs.illinois.edu/papers/elem-connectivity-open-probs.pdf, September 2015.

[9] Chandra Chekuri and Shi Li. A note on the hardness of the k-way hypergraph cut problem. Unpublished manuscript available at http://chekuri.cs.illinois.edu/papers/hypergraph-kcut.pdf, November 2015.

[10] Chandra Chekuri and Chao Xu. Computing minimum cuts in hypergraphs. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1085–1100, 2017.

[11] Chandra Chekuri and Chao Xu. A note on approximate strengths of edges in a hypergraph. *CoRR*, abs/1703.03849, 2017.

[12] Eddie Cheng. Edge-augmentation of hypergraphs. *Mathematical Programming*, 84(3):443–465, Apr 1999.

[13] William H. Cunningham. Decomposition of submodular functions. *Combinatorica*, 3(1):53–68, 1983.

[14] William H. Cunningham and Jack Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32(3):734–765, 1980.

[15] E. A. Dinic. Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. *Soviet Math Doklady*, 11:1277–1280, 1970.

[16] M.V. Lomonosov E.A. Dinits, A.V. Karzanov. On the structure of a family of minimal weighted cuts in graphs. In A.A. Fridman, editor, *Studies in Discrete Mathematics*, pages 290–306. Nauka (Moskva), 1976.

[17] Tamás Fleiner and Tibor Jordán. Coverings and structure of crossing families. *Mathematical Programming*, 84(3):505–518, Apr 1999.

[18] Lisa Fleischer. Building chain and cactus representations of all minimum cuts from Hao–Orlin in the same asymptotic run time. *Journal of Algorithms*, 33(1):51 – 72, 1999.

[19] András Frank. *Connections in Combinatorial Optimization*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2011.

[20] András Frank, Toshihide Ibaraki, and Hiroshi Nagamochi. On sparse subgraphs preserving connectivity properties. *Journal of graph theory*, 17(3):275–281, 1993.

[21] András Frank, Tamás Király, and Matthias Kriesell. On decomposing a hypergraph into k connected sub-hypergraphs. *Discrete Applied Mathematics*, 131(2):373 – 383, 2003. Submodularity.

[22] Satoru Fujishige. Canonical decomposotions of symmetric submodular functions. *Discrete Applied Mathematics*, 5(2):175–190, 1983.

[23] Takuro Fukunaga. Computing minimum multiway cuts in hypergraphs. *Discrete Optimization*, 10(4):371 – 382, 2013.

[24] Wai Shing Fung, Ramesh Hariharan, Nicholas J.A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 71–80, New York, NY, USA, 2011. ACM.

[25] Harold N. Gabow. The minset-poset approach to representations of graph connectivity. *ACM Trans. Algorithms*, 12(2):24:1–24:73, Feb 2016.

[26] Mohsen Ghaffari, David R. Karger, and Debmalya Panigrahi. Random contractions and sampling for hypergraph and hedge connectivity. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1101–1114, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

[27] O. Goldschmidt and D.S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Mathematics of Operations Research*, pages 24–37, 1994.

[28] Sudipto Guha, Andrew McGregor, and David Tench. Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems*, PODS '15, pages 241–247, New York, NY, USA, 2015. ACM.

[29] Monika Henzinger, Satish Rao, and Di Wang. *Local Flow Partitioning for Faster Edge Connectivity*, pages 1919–1938.

[30] David R Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, Feb 1995.

[31] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, January 2000.

[32] David R. Karger and Debmalya 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*, SODA '09, pages 246–255, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

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

[34] Ken-ichi Kawarabayashi and Mikkel 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*, STOC '15, pages 665–674, New York, NY, USA, 2015. ACM.

[35] Regina Klimmek and Frank Wagner. A simple hypergraph min cut algorithm. Technical Report B 96-02, Bericht FU Berlin Fachbereich Mathematik und Informatik, 1996. Available at http://edocs.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000297/1996_02.pdf.

[36] Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 367–376, New York, NY, USA, 2015. ACM.

[37] Joseph B. Kruskal, Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, 7:48–50, 1956.

[38] E. L. Lawler. Cutsets and partitions of hypergraphs. *Networks*, 3(3):275–285, 1973.

[39] Yin Tat Lee and Aaron 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*, pages 424–433. IEEE, 2014.

[40] Yin Tat Lee and He Sun. An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 678–687, New York, NY, USA, 2017. ACM.

[41] Aleksander 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*, pages 253–262. IEEE, 2013.

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

[43] W. David Matula. A Linear Time $2+\epsilon$ Approximation Algorithm for Edge Connectivity. In *SODA*, pages 500–504, 1993.

[44] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.

[45] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[46] Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[47] Hiroshi Nagamochi, Shuji Nakamura, and Toshimasa 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*, E86-D(2):179–185, 2003.

[48] James B. Orlin. Max flows in o(nm) time, or better. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 765–774, New York, NY, USA, 2013. ACM.

[49] J S Provan and D R Shier. A paradigm for listing (s,t)-cuts in graphs. *Algorithmica*, 15(4):351–372, 1996.

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

[51] Romeo Rizzi. NOTE – On Minimizing Symmetric Set Functions. *Combinatorica*, 20(3):445–450, 2000.

[52] Huzur Saran and Vijay V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, February 1995.

[53] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983.

[54] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

[55] Mikkel Thorup. Minimum k-way cuts via deterministic greedy tree packing. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 159–166. ACM, 2008.

[56] Mingyu Xiao. Finding minimum 3-way cuts in hypergraphs. *Information Processing Letters*, 110(14-15):554–558, 2010. Preliminary version in TAMC 2008.

[57] Yutaro Yamaguchi. Realizing symmetric set functions as hypergraph cut capacity. *Discrete Mathematics*, 339(8):2007–2017, aug 2016.