Packing hypertrees and the k-cut problem in Hypergraphs

Mourad Baïou, Francisco Barahona

▶ **To cite this version:**

Mourad Baïou, Francisco Barahona. Packing hypertrees and the k-cut problem in Hypergraphs. 2021. hal-03454487

HAL Id: hal-03454487
https://hal.science/hal-03454487v1

Preprint submitted on 29 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Packing hypertrees and the k-cut problem in Hypergraphs

Mourad Baïou[1] and Francisco Barahona[2]

[1] Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France.
[2] IBM Research AI

Abstract. We give a combinatorial algorithm to find a maximum packing of hypertrees in a capacitated hypergraph. Based on this we extend to hypergraphs several algorithms for the k-cut problem, that are based on packing spanning trees in a graph. In particular we give a γ-approximation algorithm for hypergraphs of rank γ, extending the work of Ravi and Sinha [22] for graphs. We also extend the work of Chekuri, Quanrud and Xu [7] in graphs, to give an algorithm for the k-cut problem in hypergraphs that is polynomial if k and the rank of the hypergraph are fixed. We also give a combinatorial algorithm to solve a linear programming relaxation of this problem in hypergraphs.

Keywords: k-cut · Packing hypertrees · Hypergraphic matroids.

1 Introduction

Hypergraphic matroids were introduced by Lorea [17] and later studied by Frank, Kiraly and Kriesell [10]. They showed that the notion of circuit-matroid of graphs can be generalized to hypergraphs. The notion of spanning trees generalizes to hypertrees. Then Frank et al. [10] extended a theorem of Tutte [25] and Nash-Williams [21] about spanning trees, to a similar theorem giving the maximum number of disjoint hypertrees contained in a hypergraph. Based on this we give an algorithm to find a maximum packing of hypertrees in a capacitated hypergraph.

Spanning tree packing has been used to study the k-cut problem in graphs. It was used by Naor and Rabani [20] to derive a linear programming relaxation, and by Thorup [24] to develop an algorithm that is polynomial for fixed k. The linear programming relaxation was further studied by Chekuri, Quanrud and Xu [7], shedding light on the connections among several of these results. Here we show that hypertree packing and other algorithms for hypergraphic matroids, can be used to extend to the k-cut problem in hypergraphs, several of the results mentioned above.

Below we describe previous work, then we give more details about our contribution, and we outline the organization of this paper.

1.1 Previous Work

Based on the Theorem of Tutte [25] and Nash-Williams [21], polynomial algorithms for packing spanning trees in a graph have been given by Barahona [2] and Gabow and Manu [13]. The k-cut problem in graphs is NP-hard if k is part of the input, see [15]. If k is fixed, Goldschmidt and Hochbaum [15] gave the first polynomial algorithm. Later other algorithms improving the running time have been found. For a graph $G = (V, E)$, let $n = |V|$ and $m = |E|$. Thorup [24] gave an $O(mn^{2k-2})$ algorithm. Chekuri, Quanrud and Xu [7] improved the running time to $O(mn^{2k-3})$. Also they presented a framework that unifies the tree packing approach of Thorup [24] and the linear programming approach of Naor and Rabani [20]. When k is part of the input, several 2-approximation algorithms have been developed. Saran and Vazirani [23], gave a $(2 - 2/k)$ approximation. Nagamochi and Kamidoi [19], and Kapoor [16], found a similar approximation. Naor and Rabani [20] used a linear programming relaxation to obtain a $2(1-1/n)$ approximation. Ravi and Sinha [22] used Lagrangian relaxation to also give a $2(1 - 1/n)$ approximation. Under the Small Set Expansion hypothesis [18], a factor of 2 is the best possible approximation. For more references on the k-cut problem see [7].

For a hypergraph $H = (V, E)$, let $n = |V|$, $m = |E|$, and $\gamma = \max\{|e| : e \in E\}$. This last number is called the *rank* of H. For the k-cut problem in hypergraphs, Fukunaga [12] extended Thorup's algorithm and gave an $O(m^2 n^{\gamma k-1})$ algorithm. Chandrasekaran, Xu and Yu [5] obtained a randomized algorithm that runs in $\tilde{O}(pn^{2k-1})$ time, where $p = \sum_{e \in E} |e|$. Fox, Panigrahi and Zhang [9] improved the randomized run-time to $\tilde{O}(mn^{2k-2})$. Recently Chandrasekaran and Chekuri [4] gave two deterministic algorithms with complexities $O(n^{3k(k-1)/2})$ and $O(n^{8k})$. When k is part of the input, the k-cut problem in hypergraphs is hard to approximate to within large factors under the Exponential Time Hypothesis, see [6], [18]; recall that for graphs there are several 2-approximation algorithms.

1.2 Our Contribution

We give a combinatorial algorithm for packing hypertrees in a hypergraph. Then we use this algorithm and the tools developed in [1], to study the k-cut problem in hypergraphs. First we extend the algorithm of Ravi and Sinha [22] for graphs, to obtain a γ-approximation algorithm for hypergraphs of rank γ. Then we study a linear programming relaxation for hypergraphs similar to the one used by Naor and Rabani [20] for graphs. We give a combinatorial algorithm to solve this linear relaxation. Then we extend to hypergraphs the analysis done by Cheruki et al. [7] for graphs, and show that the integrality gap is γ. We also build on their analysis to show that a maximum hypertree packing gives an $O(mn^{\gamma k-3})$ algorithm for k-cut in hypergraphs of rank γ. This improves by a factor of $O(mn^2)$ the time of the algorithm of [12], that is based on an approximate hypertree packing. In summary, our work shows that the use of hypergraphic matroids leads to

natural extensions of several algorithms for the k-cut problem, that were initially developed for graphs.

1.3 Organization

Section 2 contains definitions, notation and some preliminary results. In Section 3 we give the algorithm for packing hypertrees. Section 4 contains lower and upper bounds for the value of a minimum k-cut. In Section 5 we study a linear programming relaxation. Section 6 contains a polynomial algorithm for fixed k and fixed rank.

2 Preliminaries

Let $H = (V, E)$ be hypergraph. For a non-empty set $X \subset V$ and $F \subseteq E$, $F[X]$ denotes the set of hyperedges in F contained in X. For $S \subset V$ we use $H(S)$ to denote the hypergraph $(S, E[X])$. Let $\mathcal{P} = \{V_1, \ldots, V_k\}$ be a family of non-empty subsets of V with $V_i \cap V_j = \emptyset$ for $i \neq j$, we denote by $\delta_F(\mathcal{P})$ the set of hyperedges in F included in $\cup_i V_i$ and that intersect at least two sets in $\mathcal{P}$. Notice that $\mathcal{P}$ is not necessarily a partition of V. For a hypergraph $H' = (V, E')$ sometimes we use $\delta_{H'}(\mathcal{P})$ instead of $\delta_{E'}(\mathcal{P})$. Also when there is no confusion we use $\delta(\mathcal{P})$ instead of $\delta_F(\mathcal{P})$. We say that H is *connected* if $\delta(S, V \setminus S) \neq \emptyset$, for all $S \subset V$, $\emptyset \neq S \neq V$. For $S \subset V$, *shrinking* S means creating a new hypergraph $H' = (V', E')$. Here $V' = (V \setminus S) \cup \{s\}$, where s is a new node that represents S. And $E' = E_1 \cup E_2$, where $E_1 = \{e \in E : e \cap S = \emptyset\}$, and $E_2 = \{(e \setminus S) \cup \{s\} : e \in E, e \cap S \neq \emptyset, e \cap (V \setminus S) \neq \emptyset\}$. For a vector $x \in \mathbb{R}^E$, and $S \subseteq E$ we use $x(S)$ to denote $\sum_{e \in S} x(e)$.

Let $D = (V, A)$ be a directed graph, for $S \subseteq V$, we denote by $\delta^+(S)$ the set $\delta^+(S) = \{(u, v) \in A \,|\, u \in S, v \notin S\}$. Given two distinguished vertices s and t, for a set $S \subset V$, with $s \in S$, $t \notin S$, the set of arcs $\delta^+(S)$ is called an *st-cut*. Given a capacity vector $c \in \mathbb{R}^A_+$, a *minimum st*-cut is an *st*-cut $\delta^+(S)$ such that $c(\delta^+(S))$ is minimum. A minimum *st*-cut can be found in $O(|V|^3)$ time with the push-preflow algorithm of [14].

For a hypergraph $H = (V, E)$, a *hyperforest* is a set $F \subseteq E$ such that $|F[X]| \leq |X| - 1$ for every non-empty $X \subseteq V$. A hyperforest F is called a *hypertree* of H if $|F| = |V| - 1$. If H is a graph, $F \subseteq E$ is a hypertree if and only if F is a spanning tree. It was proven by Lorea [17] that the hyperforests of a hypergraph form the family of independent sets of a matroid. These are called *hypergraphic matroids*. Frank et al. [10] further studied hypergraphic matroids. In particular they gave the following formula for the rank $r(F)$ of $F \subseteq E$, $r(F) = \min\{|V| - |\mathcal{P}| + |\delta_F(\mathcal{P})| : \mathcal{P} \text{ a partition of } V\}$. Notice that matroid rank and rank of a hypergraph are completely different concepts.

Remark 1 *It follows from the formula above that if T is a hypertree then $|\delta_T(\mathcal{P})| \geq |\mathcal{P}| - 1$, for every partition $\mathcal{P}$ of V.*

Frank et al. [10] extended a theorem of Tutte [25] and Nash-Williams [21] giving the maximum number of spanning trees in a graph. They gave a similar formula for the maximum number of disjoint hypertrees contained in a hypergraph. This is in the theorem below.

Theorem 2 *[10] A hypergraph contains k disjoint hypertrees if and only if*

$$|\delta(\mathcal{P})| \geq k(|\mathcal{P}| - 1) \tag{1}$$

holds for every partition $\mathcal{P}$ of V.

Based on this theorem we give an algorithm to find a maximum packing of hypertrees in a hypergraph. For that we use three algorithms mentioned below, that were developed in [1].

2.1 Separation of partition inequalities

Since there is an exponential number of inequalities (1), we need a polynomial algorithm to test if there is any of them that is violated. For that we assume that $\bar{x} \in \mathbb{R}^E_+$ is an input vector and we solve

$$\text{minimize} \quad \bar{x}(\delta(\mathcal{P})) - \beta(|\mathcal{P}| - 1). \tag{2}$$

Where the minimum is taken among all partitions $\mathcal{P}$ of V, and $\beta > 0$ is a fixed number. Since $\mathcal{P} = \{V\}$ is a partition, the minimum is always less than or equal to zero. This gives us a most violated inequality, if there is any. In [1] this was reduced to a sequence of $|V|$ minimum cut problems in a graph with $O(|V|+|E|)$ nodes.

2.2 Strength of a network

Given a hypergraph $H = (V, E)$, with a capacity vector $c \in \mathbb{R}^E_+$, we also need to find the maximum value of k so that $c(\delta(\mathcal{P})) \geq k(|\mathcal{P}| - 1)$, for all partitions $\mathcal{P}$ of V. We compute $s = \min \frac{c(\delta(\mathcal{P}))}{|\mathcal{P}|-1}$, where the minimum is taken over all partitions $\mathcal{P}$ of V, with $|\mathcal{P}| \geq 2$. For graphs this was called *Network Strength* in [8]. Thus we call the value s, the *strength* of H. Then $k = \lfloor s \rfloor$. The strength can be found with the same asymptotic complexity as $|V|$ applications of the push-preflow algorithm [14], in a graph with $O(|V| + |E|)$ nodes, see [1].

2.3 Network Reinforcement

The following problem was studied in [8]. Given a graph, a number k and a set of candidate edges, each of them with an associated cost, find a minimum cost set of candidate edges to be added to the network so it has strength equal to k. We need to solve a similar problem for a hypergraph. An algorithm for it was given in [1]. It requires to solve $|E||V|$ minimum cut problems in a graph with $O(|V| + |E|)$ nodes.

3 Packing Hypertrees

Here we give an algorithmic proof of Theorem 2. If H contains k disjoint hypertrees then it follows from Remark 1 that (1) holds for every partition. So we have to prove the other direction.

A partition is called *tight* if (1) holds as equation. We proceed by induction on $|V| + |E|$. So we assume that the statement is true for any hypergraph $H' = (V', E')$ with $|V'| + |E'| < |V| + |E|$. If an edge does not belong to any tight partition, we remove it and we apply the induction hypothesis. So we assume that every edge appears in a tight partition.

Case 1. Suppose that the partition $\{S_1, \ldots, S_p\}$ is tight, and at least one set, S_1 say, has $|S_1| > 1$. We shrink S_1 to form the hypergraph H'. From the induction hypothesis we know that there are k disjoint hypertrees in H'. Now consider $H'' = H(S_1)$. If there is a partition $T_1, \ldots, T_l$ of S_1 with $\delta_{H''}(T_1, \ldots, T_l) < k(l-1)$, then $\delta_H(T_1, \ldots, T_l, S_2, \ldots, S_p) < k(l-1) + k(p-1) = k(l+p-2)$, a contradiction. Thus by our induction hypothesis there are k disjoint hypertrees in $H(S_1)$. Lemma 3 below shows that any hypertree of H' can be combined with a hypertree of H'' to obtain a hypertree of H.

Lemma 3 *Let T' be a hypertree of H', and T'' a hypertree of H''. Then $T = T' \cup T''$ is a hypertree of $H = (V, E)$.*

Proof. Suppose that for a partition $\{U_1, \ldots, U_p\}$ of V, we have $|\delta_T(U_1, \ldots, U_p)| < (p-1)$. After renumbering the sets $\{U_i\}$, we can assume that $S_1 \subseteq \cup_{i=1}^{i=r} U_i$, and $S_1 \cap U_i \neq \emptyset$ for $i = 1, \ldots, r$. Also $|\delta_T(U_1, \ldots, U_p)| < (r-1) + (p-r) = (p-1)$. But this is not possible because $|\delta_T(U_1, \ldots, U_r)| \geq (r-1)$, and $|\delta_T(\cup_{i=1}^{i=r} U_i, U_{r+1}, \ldots, U_p)| \geq (p-r)$. Thus $|\delta_T(\mathcal{P})| \geq |\mathcal{P}| - 1$, for every partition $\mathcal{P}$ of V. Since $r(T) = \min\{|V| - |\mathcal{P}| + |\delta_T(\mathcal{P})| : \mathcal{P} \text{ a partition of } V\}$, we have $r(T) = |V| - 1$. In view of $|T| = |V| - 1$, we conclude that T is a hypertree of H.

Case 2. Now we assume that the partition $\{S_1, \ldots, S_p\}$ is tight, and all sets $\{S_i\}$ are singletons. We have

$$|E| = k(|V| - 1). \tag{3}$$

If every hyperedge has exactly two elements we have a graph. Then the result follows from the Theorem of Tutte [25] and Nash-Williams [21]. In this case the algorithms of [2] or [13] give the packing of spanning trees.

Suppose that a hyperedge e has at least three elements. We remove one element v from e, and test if all inequalities (1) are satisfied. If so we keep working with the new hypergraph. If not, there is a tight partition $\mathcal{P}$ whose inequality becomes violated after removing v from e. This is not the partition of all singletons, because (3) is not violated after removing v from e. Then we can treat $\mathcal{P}$ as in Case 1. This completes the proof.

3.1 Integral Packing

Based on the above proof, now we derive an algorithm. We assume that for a hypergraph $H = (V, E)$, we have a capacity vector $c \in \mathbb{Z}_+^E$. We denote by $\mathcal{T}(H)$ the set of hypertrees of H. A *maximum integral packing of hypertrees* is a solution of the following.

$$\max \sum_{T \in \mathcal{T}(H)} y_T; \quad \sum_{T\,:\,e \in T} y_T \leq c(e), \text{ for each edge } e;\ y \geq 0, \text{ integer valued.} \quad (4)$$

The algorithm has several stages as follows.

- First we compute
$$k = \min \left\lfloor \frac{c(\delta(\mathcal{P}))}{|\mathcal{P}| - 1} \right\rfloor, \quad (5)$$
over all partitions $\mathcal{P}$ of V. This is the maximum value of k such that $c(\delta(\mathcal{P})) \geq k(|\mathcal{P}| - 1)$ for every partition $\mathcal{P}$ of V. This is the strength of the hypergraph, as defined in Sub-section 2.2. For that we use the algorithm in [1]. This requires $|V|$ applications of the push-preflow algorithm [14], in a graph with $O(|V| + |E|)$ nodes. This gives the value of the maximum in (4), but not the values for the variables y.
- Once the value k is known, we should adjust the capacities so that every hyperedge appears at least in a tight partition. For that we solve the linear program below.

$$\min x(E) \quad (6)$$
$$x(\delta(\mathcal{P})) \geq k(|\mathcal{P}| - 1), \text{ for all partitions } \mathcal{P} \text{ of } V, \quad (7)$$
$$0 \leq x(e) \leq c(e). \quad (8)$$

This is called Network Reinforcement, and as mentioned in Sub-section 2.3, it reduces to $|E||V|$ minimum cut problems in a graph with $O(|V|+|E|)$ nodes, see [1]. If the capacities are integers, this algorithm produces an integer solution. Denote by $\bar{x}$ the solution obtained. We lower the capacities c to $\bar{x}$, i.e., we set $c \leftarrow \bar{x}$.
- Now we have to treat Cases 1 and 2 above. We have to find a tight partition. For that we pick an edge e, we decrease by one its capacity $c(e)$, and find a most violated partition inequality, with the algorithm of [1]. This involves $|V|$ minimum cut problems in a graph with $O(|V| + |E|)$ nodes. A violated inequality is a tight inequality if we do not decrease $c(e)$. Let $\{S_1, \ldots, S_p\}$ be the associated partition of V.
In Case 1 we assume that one set, S_1 say, has $|S_1| > 1$. We shrink S_1 to a single node and we denote by H' the resulting hypergraph. Then we look for a packing of hypertrees of value k in H'. We also denote by $H'' = H(S_1)$, and look for a packing of hypertrees of value k in H''. Then we combine hypertrees in H' with hypertrees in H'' to obtain a packing of hypertrees of value k in H. This is done as below.

Let $S' = \{T'_1, \ldots, T'_r\}$ be a set of hypertrees of H' with positive weights $\{\alpha_1, \ldots, \alpha_r\}$, and let $S'' = \{T''_1, \ldots, T''_s\}$ be a set of hypertrees of H'' with positive weights $\{\beta_1, \ldots, \beta_s\}$. Pick any hypertree in the first set, T'_1 say, and any hypertree in the second set, T''_1 say, and form a hypertree in H, $T = T'_1 \cup T''_1$ with weight $\gamma = \min\{\alpha_1, \beta_1\}$. Subtract γ from α_1 and β_1, and remove from S' and S'' any hypertree with zero weight. Continue until S' and S'' are empty. This procedure produces at most $r + s$ hypertrees of H. If the weights $\{\alpha_i\}$ and $\{\beta_j\}$ are integers, then the new weights are also integer.

In Case 2 we assume that $\{S_1, \ldots, S_p\}$ is the partition of all singletons. If all hyperedges have exactly two elements, we have a graph. Then we can apply the algorithms of [2] or [13]. If the capacities are integer these two algorithms produce an integral packing. The algorithm of [13] has complexity $O(|V|^3|E|\log(|V|^2/|E|))$ and produces at most $2|E|+2|V|-2$ spanning trees. Assume now that there is a hyperedge e with at least three nodes and with capacity $c(e)$. We remove a node v and look for a most violated partition inequality. If there is none, we keep working with the new hypergraph. Otherwise, let α be the violation. We create a new hyperedge $e' = e \setminus \{v\}$ with capacity $c(e) - \alpha$, and give a capacity α to e. Then we have a tight partition that is treated as in Case 1. Notice that Case 1 arises at most $|V|$ times, therefore during the entire execution of this algorithm, at most $|V|$ new hyperedges are created.

Since all arithmetic operations are additions and subtractions, and the capacities are integer, this algorithm produces an integral packing. Now we analyze the complexity of this algorithm. Notice that it requires finding at most $|V|$ violated partition inequalities. This amounts to at most $|V|^2$ minimum cut problems in a graph with $O(|V| + |E|)$ nodes. So the complexity of this part is $O(|V|^2(|V| + |E|)^3)$. This dominates the complexity of the algorithm for finding a packing of spanning trees that is $O(|V|^3|E|\log(|V|^2/|E|))$.

After some transformations our algorithm requires finding packings of spanning trees in graphs. For each of these trees, each edge is contained in a hyperedge of the original hypergraph. The algorithm of [13] produces $2m+2n$ different spanning trees for a graph with n nodes and m edges. Thus we conclude that our algorithm produces at most $O(|E| + |V|)$ hypertrees.

3.2 Fractional Packing

Now we consider problem (4), but without requiring integrality of the variables y. The only difference here is that formula (5) is replaced by $k' = \min \frac{c(\delta(\mathcal{P}))}{|\mathcal{P}|-1}$. Then k' might be a non-integer number. All other steps of the algorithm remain the same. Notice that in Case 2 when a new hyperedge is created, it receives a fraction of the capacity of the original hyperedge. We illustrate this below with a simple example.

Consider $H = (V = \{a, b, c, d\}, E = \{V\})$. Let $c(V) = 1$. Then using formula above we get $k' = 1/3$, given by the partition of all singletons. Thus in

our algorithm, this partition is tight. As in Case 2, removing node a from the hyperedge V gives a violation of 1/3 for the partition inequality associated with $\{\{a\},\{b,c,d\}\}$. Thus we give capacity 1/3 to V and capacity 2/3 to the new hyperedge $\{b,c,d\}$; then the partition $\{\{a\},\{b,c,d\}\}$ is tight. When we shrink $\{b,c,d\}$ to single node we obtain a graph with one edge. Then we have to keep working with $H(\{b,c,d\})$. Here the partition of all singletons is tight, so we remove b from the hyperedge $\{b,c,d\}$. Then the partition inequality associated with $\{\{b\},\{c,d\}\}$ is violated by 1/3. Thus we give capacity 1/3 to $\{b,c,d\}$, and capacity 1/3 to $\{c,d\}$. When we shrink $\{c,d\}$, we obtain a graph with one edge, and the hypergraph $H(\{c,d\})$ is also a graph with one edge. In both cases the packing is trivial to find. In summary, the algorithm made three copies of the hyperedge V (or subsets of it), each of them with capacity 1/3, and gave the weight 1/3 to the resulting hypertree.

4 A relaxation of the *k*-cut problem

Consider a hypergraph $H=(V,E)$, with a weight vector $w\in\mathbb{R}^E_+$, and a fixed number k. The *k-cut problem* consists of finding a partition $\{S_1,\ldots,S_k\}$ of V that minimizes $w(\delta(S_1,\ldots,S_k))$. Let $\lambda_k(H)$ denote the value of the minimum. For a non-negative number b, a lower bound of $\lambda_k(H)$ is

$$l(b)=\min_{p\geq 1}\big\{w(\delta(S_1,\ldots,S_p))-b(p-k)\big\}. \tag{9}$$

Here the minimum is taken over all partitions of V, and b is a fixed non-negative number. The function $l(\cdot)$ is concave and piece-wise linear, and it has at most n break-points. The maximum of l is found at a break-point $\bar{b}$. In what follows we study how to find all break-points $b_i\leq\bar{b}$. For graphs, a similar lower bound was proposed in [3] and independently in [22].

4.1 Break-points of *l*

We start with $b=0$, then the trivial partition $\mathcal{P}=\{V\}$ gives the minimum in (9). The following lemma gives us a way to generate the subsequent break-points.

Lemma 4 *Let $\mathcal{P}'=\{S_1,\ldots,S_p\}$ be a solution of (9) for $b=b'$. Assume that for $b=b''$, $b''\geq b'$, $\mathcal{P}'$ and $\mathcal{P}''=\{T_1,\ldots,T_q\}$ are both solutions of (9), with $q>p$. Then b'' is the strength of one of the sets $\{S_i\}$. Recall that the strength was defined in Sub-section 2.2.*

Proof. Since $q>p$, we can assume that after renumbering, $S_1\subseteq\cup_{i=1}^r T_i$ and $Q_i=T_i\cap S_1\neq\emptyset$, for $i=1,\ldots,r$, $r>1$.

Since $\mathcal{P}'$ is a solution for $b=b''$, we have $w(\delta(Q_1,\ldots,Q_r))\geq b''(r-1)$. If this is not the case, we would have $w(\delta(Q_1,\ldots,Q_r))<b''(r-1)$, and we could improve the solution $\mathcal{P}'$ by removing S_1 and adding the sets $\{Q_i\}$. With the same argument we conclude that a similar inequality holds for any partition of S_1.

Since $\mathcal{P}''$ is also a solution for $b = b''$ we cannot have $w(\delta(Q_1, \ldots, Q_r)) > b''(r-1)$. If that was the case, we could improve the solution $\mathcal{P}''$ replacing the sets T_i, $1 \leq i \leq r$, with their union.

Therefore $w(\delta(Q_1, \ldots, Q_r)) = b''(r-1)$, and $w(\delta(R_1, \ldots, R_t)) \geq b''(t-1)$, for any partition $\{R_1, \ldots, R_t\}$ of S_1. Thus b'' is the strength of S_1.

This suggests the following procedure.

Algorithm 1

Step 0. Start with $\mathcal{P}_0 = \{V\}$, $b = \bar{b} = 0$, $j = 0$.
Step 1. Compute the strength of each set in $\mathcal{P}_j$. Among them, let S_q be a set with minimum strength s_q.
Step 2. Update $\bar{b} \leftarrow s_q$, and to obtain $\mathcal{P}_{j+1}$, replace S_q in $\mathcal{P}_j$ with a partition of S_q giving its strength. Set $j \leftarrow j+1$, if $|\mathcal{P}_j| < k$ go to Step 1, otherwise stop.

The sequence of partitions produced here can be studied in the context of submodular functions as in [11].

4.2 The maximum of l

Consider the last value j, we have $|\mathcal{P}_j| \geq k$. Let $\bar{b}$ be the associated value of the parameter. The corresponding lower bound is

$$\mu = w(\mathcal{P}_{j-1}) - \bar{b}(|\mathcal{P}_{j-1}| - k) = w(\mathcal{P}_j) - \bar{b}(|\mathcal{P}_j| - k). \tag{10}$$

If $|\mathcal{P}_j| = k$, this is a solution of the k-cut problem. Now we treat the case when $|\mathcal{P}_j| > k$.

Since $\bar{b} = \frac{w(\mathcal{P}_j) - w(\mathcal{P}_{j-1})}{|\mathcal{P}_j| - |\mathcal{P}_{j-1}|}$, we obtain the expression below that is needed in the next sub-section.

$$\mu = \frac{|\mathcal{P}_j| - k}{|\mathcal{P}_j| - |\mathcal{P}_{j-1}|} w(\delta(\mathcal{P}_{j-1})) + \frac{k - |\mathcal{P}_{j-1}|}{|\mathcal{P}_j| - |\mathcal{P}_{j-1}|} w(\delta(\mathcal{P}_j)). \tag{11}$$

4.3 An upper bound

Now we produce an approximate solution for the k-cut problem. Let $l = k - |\mathcal{P}_{j-1}|$ and $\gamma = \max\{|e| : e \in E\}$. Let $\{T_1, \ldots, T_r\}$ be the partition of the last set S_q obtained in Step 2 of Algorithm 1. We number the sets $\{T_i\}$ so that $w(\delta(T_i)) \leq w(\delta(T_{i+1}))$, for $i = 1, \ldots, r-1$. We choose $\{T_1, \ldots, T_l\}$, and combine $\{T_{l+1}, \ldots, T_r\}$ into one set. A similar procedure was proposed for graphs in [22].

Theorem 5 *The value of this solution is at most* $\gamma(1 - \frac{1}{n})\lambda_k(H)$.

Proof. We have

$$\sum_{i=1}^{l} w(\delta(T_i)) \leq \frac{l}{r} \sum_{i=1}^{r} w(\delta(T_i)) \leq \gamma \frac{l}{r} w(\delta(T_1, \ldots, T_r)) =$$
$$\gamma \frac{r-1}{r} \frac{l}{r-1} w(\delta(T_1, \ldots, T_r)).$$

Thus the value of this solution is at most

$$w(\mathcal{P}_{j-1}) + \gamma(1-\frac{1}{r})\frac{l}{r-1}w(\delta(T_1,\ldots,T_r)) =$$
$$w(\mathcal{P}_{j-1})\Big(1-\gamma(1-\frac{1}{r})\frac{l}{r-1}\Big) + \gamma(1-\frac{1}{r})\frac{l}{r-1}w(\mathcal{P}_j) \leq$$
$$\gamma(1-\frac{1}{r})\, w(\mathcal{P}_{j-1})(\frac{r-1-l}{r-1}) + \gamma(1-\frac{1}{r})\frac{l}{r-1}w(\mathcal{P}_j) =$$
$$\gamma(1-\frac{1}{r})\,\mu \leq \gamma(1-\frac{1}{r})\,\lambda_k(H) \leq \gamma(1-\frac{1}{n})\,\lambda_k(H).$$

Thus we have a γ-approximation algorithm for hypergraphs of rank γ. Recall that the k-cut problem in hypergraphs is hard to approximate to within large factors under the Exponential Time Hypothesis, see [6], [18]. Also recall that for $\gamma = 2$, under the same hypothesis, a factor of 2 is the best possible approximation, cf. [18].

5 A linear programming relaxation for k-cut

Let $H = (V, E)$ be a connected hypergraph, and $w \in \mathbb{Z}_+^E$. Let $\mathcal{T}(H)$ denote the set of hypertrees of H. We study the linear program

$$\min \sum w(e)x(e) \tag{12}$$
$$\sum_{e\in T} x(e) \geq k-1 \text{ for } T \in \mathcal{T}(H) \tag{13}$$
$$0 \leq x(e) \leq 1 \text{ for } e \in E \tag{14}$$

An integer solution of this gives a solution of the k-cut problem. For graphs a similar linear program was proposed in [20].

Now we extend to hypergraphs the analysis used in [7] for graphs. Let $\mathcal{P}_j$ be the last partition produced by Algorithm 1, and let $\{T_1,\ldots,T_r\}$ be the partition of the last set S_q obtained in Step 2 of Algorithm 1. The vector $\bar{x}$ defined below is a feasible solution of (12)-(14).

- $\bar{x}(e) = 1$ for $e \in \delta(\mathcal{P}_{j-1})$; $\bar{x}(e) = 0$ for $e \in E \setminus \delta(\mathcal{P}_j)$.
- $\bar{x}(e) = \alpha$ for $e \in \delta(\mathcal{P}_j) \setminus \delta(\mathcal{P}_{j-1})$, where $\alpha = \frac{k-|\mathcal{P}_{j-1}|}{|\mathcal{P}_j|-|\mathcal{P}_{j-1}|}$.

Its value is

$$w(\delta(\mathcal{P}_{j-1})) + \frac{k-|\mathcal{P}_{j-1}|}{|\mathcal{P}_j|-|\mathcal{P}_{j-1}|}w(\delta(T_1,\ldots,T_r)) = \tag{15}$$
$$\frac{|\mathcal{P}_j|-k}{|\mathcal{P}_j|-|\mathcal{P}_{j-1}|}w(\delta(\mathcal{P}_{j-1})) + \frac{k-|\mathcal{P}_{j-1}|}{|\mathcal{P}_j|-|\mathcal{P}_{j-1}|}w(\delta(\mathcal{P}_j)).$$

This is the value μ as in (11).

Now consider the dual problem.

$$\max(k-1) \sum_{T \in \mathcal{T}(H)} y_T - \sum_{e \in E} z(e) \tag{16}$$

$$\sum_{T\,:\,e \in T} y_T \leq w(e) + z(e) \forall e \in E \tag{17}$$

$$y \geq 0,\ z \geq 0 \tag{18}$$

To obtain a dual solution define

$$w(e) + \bar{z}(e) = \begin{cases} (b_j/b_i)w(e) & \text{for } e \in \delta(\mathcal{P}_i) - \delta(\mathcal{P}_{i-1}),\ i < j, \\ w(e) & \text{otherwise.} \end{cases}$$

Using $w + \bar{z}$ as capacities leads to a hypergraph whose strength is b_j. Thus this set of capacities yields a maximum (fractional) hypertree packing $\bar{y}$ of value b_j. Now we compute the objective value for the dual vector $(\bar{y}, \bar{z})$. This is

$$(k-1)b_j - \sum_{i=1}^{j-1} (\frac{b_j}{b_i} - 1)(w(\mathcal{P}_i) - w(\mathcal{P}_{i-1})) =$$

$$(k-1)b_j - \sum_{i=1}^{j-1} (\frac{b_j - b_i}{b_i})(w(\mathcal{P}_i) - w(\mathcal{P}_{i-1})) =$$

$$(k-1)b_j - \sum_{i=1}^{j-1} (b_j - b_i)(|\mathcal{P}_i| - |\mathcal{P}_{i-1}|) =$$

$$(k-1)b_j - (|\mathcal{P}_{j-1}| - 1)b_j + w(\mathcal{P}_{j-1}) =$$

$$\frac{k - |\mathcal{P}_{j-1}|}{|\mathcal{P}_j| - |\mathcal{P}_{j-1}|} w(\delta(T_1, \ldots, T_r)) + w(\mathcal{P}_{j-1}).$$

Here we obtained expression (15). Thus $\bar{x}$ and $(\bar{y}, \bar{z})$ have the same value, therefore they are optimal solutions. Hence we have polynomial combinatorial algorithms to produce optimal primal and dual solutions of (12)-(14). For graphs, other authors have suggested the use of the ellipsoid method, or the use of approximate tree packing, see [20], [24], [7], [12]. Having fast algorithms to produce lower and upper bounds gives the possibility of embedding this in a branch and bound procedure.

The optimal value of this linear program is exactly the lower bound μ defined in (10). Hence from Theorem 5 we obtain the following.

Theorem 6 *The integrality gap of this linear program is at most* $\gamma(1 - \frac{1}{n})$.

Notice that as long as the hypergraph is connected, the algorithm from Subsection 3.2 makes fractional copies of the hyperedges to produce a fractional packing of hypertrees. Consider the example in Sub-section 3.2, with $k = 2$. The value of the linear program is 1/3 and the value of a minimum 2-cut is 1. Here we have exactly the gap given by Theorem 6. We can extend this example

to a hypergraph with n nodes, and one hyperedge with weight 1, containing all nodes. Then for $k = 2$ the lower bound is $1/(n-1)$. Again we have exactly the gap given by Theorem 6.

Consider now a non-connected hypergraph. Assuming that the weights are integer, we propose the following. We multiply by n all the weights, and add a minimal set of artificial edges to make the hypergraph connected. We give the weight 1 to each artificial edge. Then minimum k-cuts in the new hypergraph correspond to minimum k-cuts in the original one.

6 A polynomial algorithm for fixed γ and k

Now we show that the algorithm for graphs given in [7] can be extended to hypergraphs. The lemma below was proved in [7] for $\gamma = 2$, the proof for larger values of γ is similar.

Lemma 7 *Let $(\bar{y}, \bar{z})$ be an optimal solution of (16)-(18). Let E' be any set of hyperedges such that $w(E') \leq \alpha\lambda_k(H)$ for some $\alpha \geq 1$. For each hypertree T let $l_T = |E' \cap E(T)|$. Let $\tau = \sum_T \bar{y}_T$ and $p_T = \bar{y}_T/\tau$. For an integer $h \geq (k-1)$, let $q_h = \sum_{T:l_T \leq h} p_T$. Then*

$$q_h \geq 1 - \frac{\gamma\alpha(k-1)(1-\frac{1}{n})}{h+1}$$

Corollary 8 *Let $(\bar{y}, \bar{z})$ be an optimal solution of (16)-(18). Define the support of $\bar{y}$ as $\{T : \bar{y}_T > 0\}$. For every optimum k-cut $A \subseteq E$ there is a hypertree T in the support of $\bar{y}$ such that $|E(T) \cap A| \leq \gamma k - 3$.*

Proof. We apply Lemma 7 with $h = \gamma k - 3$ and $\alpha = 1$. We obtain

$$q_h \geq 1 - \frac{(\gamma k - \gamma)(1-\frac{1}{n})}{\gamma k - 2},$$

and for $\gamma \geq 2$ we have $q_h > 1$.

This suggest the following algorithm: For each hypertree in the support of $\bar{y}$, choose $\gamma k - 3$ hyperedges, contract the remaining hyperedges, and find a minimum k-cut in the resulting hypergraph. This has to be repeated for every choice of $\gamma k - 3$ hyperedges. Recall that the packing algorithm produces $O(m+n)$ hypertrees, so this leads to an $O((m+n)\, n^{\gamma k-3})$ algorithm that enumerates all minimum k-cuts. Fukunaga has given an $O(m^2 n^{\gamma k-1})$ algorithm based on a greedy packing of hypertrees. Using an optimal packing leads to decrease the complexity by a factor of $O(mn^2)$, and to a simpler derivation. Chandrasekaran and Chekuri [4] gave two algorithms with complexities $O(n^{3k(k-1)/2})$ and $O(n^{8k})$, (that are independent of γ). Thus the hypertree packing approach seems to be of interest for hypergraphs of small rank.

References

1. Baiou, M., Barahona, F.: On some algorithmic aspects of hypergraphic matroids. arXiv **2111.05699** (2021)
2. Barahona, F.: Packing spanning trees. Mathematics of Operations Research **20**(1), 104–115 (1995)
3. Barahona, F.: On the k-cut problem. Operations Research Letters **26**(3), 99–105 (2000)
4. Chandrasekaran, K., Chekuri, C.: Hypergraph k-cut for fixed k in deterministic polynomial time. In: 2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS). pp. 810–821. IEEE (2020)
5. Chandrasekaran, K., Xu, C., Yu, X.: Hypergraph k-cut in randomized polynomial time. Mathematical Programming **186**(1), 85–113 (2021)
6. Chekuri, C., Li, S.: A note on the hardness of approximating the k-way hypergraph cut problem. Manuscript, http://chekuri. cs. illinois. edu/papers/hypergraph-kcut. pdf (2015)
7. Chekuri, C., Quanrud, K., Xu, C.: Lp relaxation and tree packing for minimum k-cut. SIAM Journal on Discrete Mathematics **34**(2), 1334–1353 (2020)
8. Cunningham, W.H.: Optimal attack and reinforcement of a network. J. of ACM **32**, 549–561 (1985)
9. Fox, K., Panigrahi, D., Zhang, F.: Minimum cut and minimum k-cut in hypergraphs via branching contractions. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 881–896. SIAM (2019)
10. Frank, A., Király, T., Kriesell, M.: On decomposing a hypergraph into k connected sub-hypergraphs. Discrete Applied Mathematics **131**(2), 373–383 (2003)
11. Fujishige, S.: Theory of principal partitions revisited. In: Research Trends in Combinatorial Optimization, pp. 127–162. Springer (2009)
12. Fukunaga, T.: Computing minimum multiway cuts in hypergraphs. Discrete Optimization **10**(4), 371–382 (2013)
13. Gabow, H.N., Manu, K.: Packing algorithms for arborescences (and spanning trees) in capacitated graphs. Mathematical Programming **82**(1), 83–109 (1998)
14. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. J. Assoc. Comput. Mach. **35**(4), 921–940 (1988)
15. Goldschmidt, O., Hochbaum, D.S.: A polynomial algorithm for the k-cut problem for fixed k. Math. Oper. Res. **19**, 24–37 (1994)
16. Kapoor, S.: On minimum 3-cuts and approximating k-cuts using cut trees. In: International Conference on Integer Programming and Combinatorial Optimization. pp. 132–146. Springer (1996)
17. Lorea, M.: Hypergraphes et matroides. Cahiers Centre Etudes Rech. Oper. **17**, 289–291 (1975)
18. Manurangsi, P.: Inapproximability of maximum edge biclique, maximum balanced biclique and minimum k-cut from the small set expansion hypothesis. In: 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
19. Nagamochi, H., Kamidoi, Y.: Minimum cost subpartitions in graphs. Information processing letters **102**(2-3), 79–84 (2007)
20. Naor, J., Rabani, Y.: Tree packing and approximating k-cuts. In: SODA. vol. 1, pp. 26–27 (2001)
21. Nash-Williams, C.S.J.A.: Edge-disjoint spanning trees of finite graphs. J. London Math. Soc. **36**, 445–450 (1961)

22. Ravi, R., Sinha, A.: Approximating k-cuts using network strength as a lagrangean relaxation. European Journal of Operational Research **186**(1), 77–90 (2008)
23. Saran, H., Vazirani, V.V.: Finding k cuts within twice the optimal. SIAM Journal on Computing **24**(1), 101–108 (1995)
24. Thorup, M.: Minimum k-way cuts via deterministic greedy tree packing. In: Proceedings of the fortieth annual ACM symposium on Theory of computing. pp. 159–166 (2008)
25. Tutte, W.T.: On the problem of decomposing a graph into n connected factors. J. London Math. Soc. **36**, 221–230 (1961)