Hypergraph Connectivity Augmentation in Strongly Polynomial Time

Kristóf Bérczi
MTA-ELTE Matroid Optimization Research Group and HUN-REN-ELTE Egerváry Research Group, Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Karthekeyan Chandrasekaran
University of Illinois, Urbana-Champaign, IL, USA

Tamás Király
MTA-ELTE Matroid Optimization Research Group and HUN-REN-ELTE Egerváry Research Group, Department of Operations Research, Eötvös Loránd University, Budapest, Hungary

Shubhang Kulkarni
University of Illinois, Urbana-Champaign, IL, USA

Abstract

We consider hypergraph network design problems where the goal is to construct a hypergraph that satisfies certain connectivity requirements. For graph network design problems where the goal is to construct a graph that satisfies certain connectivity requirements, the number of edges in every feasible solution is at most quadratic in the number of vertices. In contrast, for hypergraph network design problems, we might have feasible solutions in which the number of hyperedges is exponential in the number of vertices. This presents an additional technical challenge in hypergraph network design problems compared to graph network design problems: in order to solve the problem in polynomial time, we first need to show that there exists a feasible solution in which the number of hyperedges is polynomial in the input size.

The central theme of this work is to overcome this additional technical challenge for certain hypergraph network design problems. We show that these hypergraph network design problems admit solutions in which the number of hyperedges is polynomial in the number of vertices and moreover, can be solved in strongly polynomial time. Our work improves on the previous fastest pseudo-polynomial run-time for these problems. As applications of our results, we derive the first strongly polynomial time algorithms for (i) degree-specified hypergraph connectivity augmentation using hyperedges and (ii) degree-specified hypergraph node-to-area connectivity augmentation using hyperedges.

2012 ACM Subject Classification Mathematics of computing → Hypergraphs; Mathematics of computing → Combinatorial algorithms; Theory of computation → Network optimization

Keywords and phrases Hypergraphs, Hypergraph Connectivity, Submodular Functions, Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.22

Related Version *Full Version*: `https://arxiv.org/abs/2402.10861` [10]

Funding Karthekeyan Chandrasekaran and Shubhang Kulkarni were supported in part by NSF grants CCF-1814613, CCF-1907937, and CCF-2402667. Karthekeyan was supported in part by the Distinguished Guest Scientist Fellowship of the Hungarian Academy of Sciences – grant number VK-6/1/2022. Kristóf and Tamás were supported in part by the Lendület Programme of the Hungarian Academy of Sciences – grant number LP2021-1/2021, by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund – grant number ELTE TKP 2021-NKTA-62 funding scheme, and by the Dynasnet European Research Council Synergy project – grant number ERC-2018-SYG 810115.

Acknowledgements Part of this work was done while Karthekeyan and Shubhang were visiting Eötvös Loránd University.

1 Introduction

In the degree-specified graph connectivity augmentation using edges problem (DS-Graph-CA-using-E), we are given an edge-weighted undirected graph $(G = (V, E_G), c_G : E_G \to \mathbb{Z}_+)$, a degree-requirement function $m : V \to \mathbb{Z}_{\geq 0}$, and a target connectivity function $r : \binom{V}{2} \to \mathbb{Z}_{\geq 0}$. The goal is to verify if there exists an edge-weighted undirected graph $(H = (V, E_H), w_H : E_H \to \mathbb{Z}_+)$ such that the degree of each vertex u in (H, w_H) is $m(u)$ and for every distinct pair of vertices $u, v \in V$, the edge connectivity between u and v in the union of the weighted graphs (G, c_G) and (H, w_H) is at least $r(u, v)$; moreover, the problem asks to construct such a graph (H, w_H) if it exists. Watanabe and Nakamura [52] introduced DS-Graph-CA-using-E for the case of uniform requirement function (i.e., $r(u, v) = k$ for all distinct $u, v \in V$ for some $k \in \mathbb{Z}_+$) and showed that this case is solvable in polynomial time in unweighted graphs. Subsequently, Frank [25] gave a strongly polynomial-time algorithm for DS-Graph-CA-using-E. Since then, designing fast algorithms as well as parallel algorithms for DS-Graph-CA-using-E has been an active area of research [5, 6, 8, 11, 26, 27, 37, 41]. The last couple of years has seen exciting progress for the uniform requirement function culminating in a near-linear time algorithm [12, 13, 14]. In addition to making progress in the algorithmic status of the problem, these works have revealed fundamental structural properties of graph cuts which are of independent interest in graph theory. In this work, we consider generalizations of these connectivity augmentation problems to hypergraphs and design the first strongly polynomial-time algorithms for these generalizations.

We emphasize that DS-Graph-CA-using-E is a feasibility problem, i.e., the goal is to verify if there exists a feasible solution and if so, then find one. There is a closely related optimization variant: the input to the optimization version is a graph $(G = (V, E_G), c_G : E_G \to \mathbb{Z}_+)$ and a target connectivity function $r : \binom{V}{2} \to \mathbb{Z}_+$ and the goal is to find a graph $(H = (V, E_H), w_H : E_H \to \mathbb{Z}_+)$ with minimum total weight $\sum_{e \in E_H} w_H(e)$ such that for every pair of distinct vertices $u, v \in V$, the edge connectivity between u and v in the union of the weighted graphs (G, c_G) and (H, w_H) is at least $r(u, v)$. This optimization version is different from the NP-hard min-cost connectivity augmentation problems (like Steiner tree and tree/cactus/forest augmentation) whose approximability have been improved recently [9, 29, 47, 48, 49]. All algorithms to solve the optimization version [5, 6, 8, 11, 25, 26, 27, 37, 41, 52] reduce it to solving the degree-specified feasibility version, i.e., DS-Graph-CA-using-E, so we focus only on the degree-specified feasibility variant and their generalization to hypergraphs throughout this work. All our results can be extended to an appropriate optimization variant, but we avoid stating them in the interests of brevity.

Hypergraphs. Edges are helpful to model relationships between pairs of entities. Hyperedges are helpful to model relationships between arbitrary number of entities. For this reason, hypergraphs are more accurate models for a rich variety of applications in bioinformatics, statistical physics, and machine learning (e.g., see [21, 22, 23, 35, 39, 42, 44, 50, 51, 53]). These applications have in turn, renewed interests in algorithms for hypergraph optimization problems [1, 2, 3, 4, 15, 16, 17, 18, 19, 24, 28, 30, 32, 33, 34, 36, 38, 43, 45]. A hypergraph $G = (V, E)$ consists of a finite set V of vertices and a set E of hyperedges, where every hyperedge $e \in E$ is a subset of V. Equivalently, a hypergraph is a set system defined over a finite set. We will denote a hypergraph $G = (V, E)$ with hyperedge weights $w : E \to \mathbb{Z}_+$ by the tuple (G, w). Throughout this work, we will be interested only in hypergraphs with positive integral weights and for algorithmic problems where the input/output is a hypergraph, we will require that the weights are represented in binary. If all hyperedges have size at most 2, then the hyperedges are known as edges and we call such a hypergraph as a graph.

We emphasize a subtle but important difference between hypergraphs and graphs: the number of hyperedges in a hypergraph could be exponential in the number of vertices. This is in sharp contrast to graphs where the number of edges is at most the square of the number of vertices. Consequently, in hypergraph network design problems where the goal is to construct a hypergraph with certain properties, we have to be mindful of the number of hyperedges in the solution hypergraph (to be returned by the algorithm). This nuanced issue adds an extra challenging layer to hypergraph network design compared to graph network design problems – e.g., membership in NP becomes non-trivial. Recent works in hypergraph algorithms literature have focused on the number of hyperedges in the context of cut/spectral sparsification of hypergraphs [2, 18, 19, 32, 33, 34, 36, 38, 43, 45]. We will return to the membership in NP issue after we define the relevant problems of interest to this work.

Notation. Let $(G = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph. For $X \subseteq V$, let $\delta_G(X) := \{e \in E : e \cap X \neq \emptyset, e \setminus X \neq \emptyset\}$ and $B_G(X) := \{e \in E : e \cap X \neq \emptyset\}$. We define the cut function $d_{(G,w)} : 2^V \to \mathbb{Z}_{\geq 0}$ by $d_{(G,w)}(X) := \sum_{e \in \delta_G(X)} w(e)$ for every $X \subseteq V$ and the coverage function $b_{(G,w)} : 2^V \to \mathbb{Z}_{\geq 0}$ by $b_{(G,w)}(X) := \sum_{e \in B_G(X)} w(e)$ for every $X \subseteq V$. For a vertex $v \in V$, we use $d_{(G,w)}(v)$ and $b_{(G,w)}(v)$ to denote $d_{(G,w)}(\{v\})$ and $b_{(G,w)}(\{v\})$ respectively. We define the *degree* of a vertex v to be $b_{(G,w)}(v)$ – we note that the degree of a vertex is not necessarily equal to $d_{(G,w)}(v)$ since we could have $\{v\}$ itself as a hyperedge (i.e., a singleton hyperedge that contains only the vertex v). For distinct vertices $u, v \in V$, the connectivity between u and v in (G, w) is $\lambda_{(G,w)}(u, v) := \min\{d_{(G,w)}(X) : u \in X \subseteq V \setminus \{v\}\}$ – i.e., $\lambda_{(G,w)}(u, v)$ is the value of a minimum $\{u, v\}$-cut in the hypergraph. For two hypergraphs $(G = (V, E_G), c_G : E_G \to \mathbb{Z}_+)$ and $(H = (V, E_H), w_H : E_H \to \mathbb{Z}_+)$ on the same vertex set V, we define the hypergraph $(G + H = (V, E_{G+H}), c_G + w_H)$ as the hypergraph with vertex set V and hyperedge set $E_{G+H} := E_G \cup E_H$ with the weight of every hyperedge $e \in E_G \cap E_H$ being $c_G(e) + w_H(e)$, the weight of every hyperedge $e \in E_G \setminus E_H$ being $c_G(e)$, and the weight of every hyperedge $e \in E_H \setminus E_G$ being $w_H(e)$.

1.1 Degree-Specified Hypergraph Connectivity Augmentation

We now define the variant of the DS-Graph-CA-using-E for hypergraphs which will be the focus of this work.

▶ **Definition 1** (DS-Hypergraph-CA-using-H)**.** Degree-specified Hypergraph Connectivity Augmentation using Hyperedges problem *is defined as follows:*

Input: *A hypergraph* $(G = (V, E_G, c_G : E_G \to \mathbb{Z}_+)$*,*
target connectivity function $r : \binom{V}{2} \to \mathbb{Z}_{\geq 0}$*, and*
degree requirement function $m : V \to \mathbb{Z}_{\geq 0}$*.*

Goal: *Verify if there exists a hypergraph* $(H = (V, E_H), w_H : E_H \to \mathbb{Z}_+)$ *such that* $b_{(H,w_H)}(u) = m(u)$ *for every* $u \in V$*,* $\lambda_{(G+H, c_G+w_H)}(u, v) \geq r(u, v)$ *for every distinct* $u, v \in V$*, and if so, then find such a hypergraph.*

Before discussing our results, we emphasize a fundamental difference between DS-Hypergraph-CA-using-H and DS-Graph-CA-using-E. It is clear that DS-Graph-CA-using-E is in NP since a YES instance admits a weighted *graph* (H, w_H) as a feasible solution which serves as a polynomial-time verifiable certificate for the YES instance; in contrast, it is not immediately clear if DS-Hypergraph-CA-using-H is in NP. This is because, the number of hyperedges in the desired hypergraph (H, w_H) could be exponential in the number of vertices, and consequently, exponential in the size of the input. We give a concrete example in Remark 2 below to illustrate this issue.

▶ Remark 2. Suppose that the input instance is given by the empty hypergraph (G, c_G) on n vertices, the target connectivity function r is given by $r(u, v) := 2^{n-1} - 1$ for every pair of distinct vertices $u, v \in V$ and the degree requirement function $m : V \to \mathbb{Z}_{\geq 0}$ is given by $m(u) := 2^{n-1} - 1$ for every vertex $u \in V$. We note that the input specification needs only $n^{O(1)}$ bits. Now, consider the hypergraph $(H = (V, E_H), w_H)$ where $E_H := \{e \subseteq V : |e| \geq 2\}$ with all hyperedge weights being one. The hypergraph (H, w_H) is a feasible solution to the input instance but the number of hyperedges in this hypergraph is $2^n - n - 1$ which is exponential in the number of vertices (and hence, the input size). However, we emphasize that for the given input instance, there is an alternative feasible solution with polynomial number of hyperedges: the hypergraph containing a single hyperedge that contains all vertices with the weight of that hyperedge being $2^{n-1} - 1$ is also a feasible solution.

Thus, in order to design a polynomial-time algorithm for DS-Hypergraph-CA-using-H, a necessary first step is to exhibit the existence of a feasible solution in which the number of hyperedges is polynomial in the input size.

▶ Remark 3. Given that membership in NP is non-trivial, it is tempting to constrain the target hypergraph (H, w_H) to be a graph. This leads to the *degree-specified hypergraph connectivity augmentation using edges* problem (DS-Hypergraph-CA-using-E). Here, the input is the same as that in DS-Hypergraph-CA-using-H, but the goal is to verify if there exists a *graph* with the same desired properties (and if so, find one). Clearly, DS-Hypergraph-CA-using-E is in NP since YES instances admit a weighted *graph* (H, w_H) as a feasible solution which serves as a polynomial-time verifiable certificate of YES instances. However, DS-Hypergraph-CA-using-E is NP-complete [20, 40] (see Table 1).

▶ Remark 4. Showing the existence of a feasible solution hypergraph with small number of hyperedges is a technical challenge in hypergraph network design problems. During first read, we encourage the reader to focus on this issue for all problems that we define and how it is addressed by our results and techniques. The strongly polynomial run-time results that we present are consequences of our techniques to address this issue (using standard algorithmic tools in submodularity).

Keeping the issue of polynomial-sized solutions in mind, we now discuss the status of DS-Hypergraph-CA-using-H. Szigeti [46] showed that DS-Hypergraph-CA-using-H can be solved in pseudo-polynomial time: in particular, if the target connectivity function $r : \binom{V}{2} \to \mathbb{Z}_{\geq 0}$ is given in unary, then the problem can be solved in polynomial time. Moreover, his result implies that if the input instance is feasible, then it admits a solution hypergraph (H, w_H) such that the number of hyperedges in H is at most $\max\{2^{|V|}, \max\{r(u, v) : \{u, v\} \in \binom{V}{2}\}\}$. In this work, we strengthen both the structural and algorithmic results of Szigeti. In particular, we show that feasible instances admit solutions with $O(|V|)$ hyperedges and give a strongly polynomial time algorithm to compute such solutions.

▶ **Theorem 5.** *There exists an algorithm to solve DS-Hypergraph-CA-using-H that runs in time $O(n^7(n + m)^2)$, where n is the number of vertices and m is the number of hypergedges in the input hypergraph. Moreover, if the instance is feasible, then the algorithm returns a solution hypergraph that contains at most $4n$ hyperedges.*

We refer the reader to Table 1 for a list of graph/hypergraph connectivity augmentation problems using edges/hyperedges, previously known results, and our results.

1.2 Degree-Specified Skew-Supermodular Cover Problems

We prove Theorem 5 by focusing on more general function cover problems. These general function cover problems encompass several applications in connectivity augmentation (including DS-Hypergraph-CA-using-H and several others that are discussed in the complete version [10]). The main contribution of this work is the first strongly polynomial time algorithm for these general function cover problems. We recall certain definitions needed to describe the general function cover problems.

▶ **Definition 6.** *Let V be a finite set, $(H = (V, E), w : E \to \mathbb{Z}_+)$ be a hypergraph, and $p : 2^V \to \mathbb{Z}$ be a set function.*

1. *The hypergraph (H, w)* weakly covers *the function p if $b_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$.*
2. *The hypergraph (H, w)* strongly covers *the function p if $d_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$.*

We will be interested in the problem of finding a degree-specified hypergraph that strongly/weakly covers a given function p. In all our applications (including DS-Hypergraph-CA-using-H), the function p of interest will be skew-supermodular and/or symmetric.

▶ **Definition 7.** *Let $p : 2^V \to Z$ be a set function. We will denote the maximum function value of p by K_p, i.e., $K_p := \max\{p(X) : X \subseteq V\}$. The set function p*

1. *is* symmetric *if $p(X) = p(V - X)$ for every $X \subseteq V$, and*
2. *is* skew-supermodular *if for every $X, Y \subseteq V$, at least one of the following inequalities hold:*
   a. *$p(X) + p(Y) \leq p(X \cap Y) + p(X \cup Y)$. If this inequality holds, then we say that p is* locally supermodular *at X, Y.*
   b. *$p(X) + p(Y) \leq p(X - Y) + p(Y - X)$. If this inequality holds, then we say that p is* locally negamodular *at X, Y.*

We will assume access to the skew-supermodular function p via the following oracle.

▶ **Definition 8.** *Let $p : 2^V \to \mathbb{Z}$ be a set function. p-`max-sc-Oracle` $((G_0, c_0), S_0, T_0, y_0)$ takes as input a hypergraph $(G_0 = (V, E_0), c_0 : E_0 \to \mathbb{Z}_+)$, disjoint sets $S_0, T_0 \subseteq V$, and a vector $y_0 \in \mathbb{R}^V$; the oracle returns a tuple $(Z, p(Z))$, where Z is an optimum solution to the following problem:*

$$\max \left\{ p(Z) - d_{(G_0,c_0)}(Z) + y_0(Z) : S_0 \subseteq Z \subseteq V - T_0 \right\}. \qquad (p\text{-}\texttt{max-sc-Oracle})$$

We note that p-`max-sc-Oracle` is strictly stronger than the function evaluation oracle[1]: function evaluation oracle can be implemented using one query to p-`max-sc-Oracle` while it is impossible to maximize a skew-supermodular function using polynomial number of queries to its function evaluation oracle [31]. However, we will see later that p-`max-sc-Oracle` can indeed be implemented in strongly polynomial time for the functions p of interest to our applications. In our algorithmic results, we will ensure that the size of hypergraphs (G_0, c_0) used as inputs to p-`max-sc-Oracle` are polynomial in the input size (in particular, the number of hyperedges in these hypergraphs will be polynomial in the size of the ground set V). We now describe the general function cover problems that will be of interest to this work.

[1] For a function $p : 2^V \to \mathbb{Z}$, the function evaluation oracle takes a subset $X \subseteq V$ as input and returns $p(X)$.

Strong Cover Problem. In all our applications, we will be interested in obtaining a degree-specified strong cover of a function in strongly polynomial time.

▶ **Definition 9** (DS-Sym-Skew-SupMod-StrongCover-using-H)**.** Degree-specified symmetric skew-supermodular strong cover using hyperedges problem *is defined as follows:*

Input:	*A degree requirement function* $m : V \to \mathbb{Z}_{\geq 0}$ *and a* $\underline{\textit{symmeric}}$ *skew-supermodular function* $p : 2^V \to \mathbb{Z}$ *via* p-`max-sc-Oracle`*.*
Goal:	*Verify if there exists a hypergraph* $(H = (V, E), w : E \to \mathbb{Z}_+)$ *such that* $b_{(H,w)}(u) = m(u)$ *for every* $u \in V$ *and* (H, w) $\underline{\textit{strongly}}$ *covers the function* p*, and if so, then find such a hypergraph.*

DS-Sym-Skew-SupMod-StrongCover-using-H was introduced by Bernáth and Király [7] as a generalization of DS-Hypergraph-CA-using-H (and the other applications discussed in the full version [10]). They showed that it is impossible to solve DS-Sym-Skew-SupMod-StrongCover-using-H using polynomial number of queries to the function evaluation oracle. They suggested access to p-`max-sc-Oracle` and we work in the same function access model as Bernáth and Király. We note that it is not immediately clear if feasible instances of DS-Sym-Skew-SupMod-StrongCover-using-H admit solution hypergraphs with polynomial number of hyperedges (see Remark 2), so membership of the problem in NP is not obvious.

Weak Cover Problem. Although our applications will be concerned with degree-specified *strong* cover, our techniques will be concerned with degree-specified *weak* cover problems.

▶ **Definition 10** (DS-Skew-SupMod-WeakCover-using-H)**.** Degree-specified skew-supermodular weak cover using hyperedges problem *is defined as follows:*

Input:	*A degree requirement function* $m : V \to \mathbb{Z}_{\geq 0}$ *and a skew-supermodular function* $p : 2^V \to \mathbb{Z}$ *via* p-`max-sc-Oracle`*.*
Goal:	*Verify if there exists a hypergraph* $(H = (V, E), w : E \to \mathbb{Z}_+)$ *such that* $\sum_{e \in E} w(e) = K_p$*,* $b_{(H,w)}(u) = m(u)$ *for every* $u \in V$*, and* (H, w) $\underline{\textit{weakly}}$ *covers the function* p*, and if so, then find such a hypergraph.*

There is a close relationship between weak cover and strong cover of symmetric skew-supermodular functions: If a hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ strongly covers a function $p : 2^V \to \mathbb{Z}$, then it also weakly covers the function p; the converse statement is false[2]. However, imposing the constraint $\sum_{e \in E} w(e) = K_p$ implies the converse – we elaborate on this now. We note that the requirement $\sum_{e \in E} w(e) = K_p$ is present in DS-Skew-SupMod-WeakCover-using-H but not in DS-Sym-Skew-SupMod-StrongCover-using-H. Firstly, if we drop this constraint from the definition of DS-Skew-SupMod-WeakCover-using-H, then feasible instances of the resulting problem admit trivial solutions[3]. Thus, imposing this constraint makes the problem non-trivial. Secondly, Szigeti [46] showed that if an instance of DS-Skew-SupMod-WeakCover-using-H is feasible, then it admits a solution hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ satisfying the

[2] For example, consider the function $p : 2^V \to \mathbb{Z}$ defined by $p(X) := 1$ for every non-empty proper subset $X \subsetneq V$ and $p(\emptyset) := p(V) := 0$, and the hypergraph $(H = (V, E := \{\{u\} : u \in V\}), w : E \to \{1\})$.

[3] Consider the hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$, where $E := \{\{u\} : u \in V,\ m(u) \geq 1\}$ with $w(\{u\}) := m(u)$ for every $\{u\} \in E$.

constraint $\sum_{e\in E} w(e) = K_p$. Moreover, Bernáth and Király [7] observed that if a hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ with $\sum_{e\in E} w(e) = K_p$ weakly covers a symmetric skew-supermodular function p, then (H, w) also strongly covers p. Thus, the converse of the previously mentioned relationship between weak and strong covers is in fact true after imposing the constraint. The observations of Szigeti [46] and Bernath and Kiraly [7] together imply that in order to solve DS-Sym-Skew-SupMod-StrongCover-using-H, it suffices to solve the DS-Skew-SupMod-WeakCover-using-H problem for the same function p. We will henceforth focus on the latter weak cover problem (and derive results for strong cover problems via the known reduction).

Before stating the main result of the work, we briefly emphasize the technical challenge in addressing the weak cover problem. As before, it is not immediately clear if feasible instances of the DS-Skew-SupMod-WeakCover-using-H problem admits solution hypergraphs in which the number of hyperedges is polynomial in $|V|$. Remark 11 below illustrates the issue with an example that is a modification of the example in Remark 2.

▶ Remark 11. Let $n := |V|$ and consider the degree requirement function $m : V \to \mathbb{Z}_{\geq 0}$ given by $m(u) := 2^{n-1} - 1$ for every $u \in V$ and the function $p : 2^V \to \mathbb{Z}$ given by $p(X) := 2^{n-1} - 1$ for every non-empty proper subset $X \subsetneq V$, $p(V) := 2^n - n - 1$, and $p(\emptyset) := 0$. We note that this function p is skew-supermodular. Consider the hypergraph $(H = (V, E_H), w_H)$, where $E_H := \{e \subseteq V : |e| \geq 2\}$ and all hyperedge weights are one. The hypergraph (H, w_H) is a feasible solution to the input instance but the number of hyperedges in this hypergraph is $2^n - n - 1$ which is exponential in the number of vertices. However, we emphasize that the input instance has an alternative feasible solution with polynomial number of hyperedges: pick an arbitrary vertex $u_0 \in V$ and consider the hypergraph $(H' = (V, E'), w' : E' \to \mathbb{Z}_+)$, where the set of hyperedges is $E' := \{\{u_0\}, V - \{u_0\}, V\}$ and their weights are given by $w'(\{u_0\}) = 2^{n-1} - n = w'(V - \{u_0\})$ and $w'(V) = n - 1$.

Thus, a necessary step in designing a polynomial-time algorithm for DS-Skew-SupMod-WeakCover-using-H is to show that feasible instances admit a solution hypergraph in which the number of hyperedges is polynomial in the input size. Szigeti [46] gave a complete characterization for the existence of a feasible solution to DS-Skew-SupMod-WeakCover-using-H. His proof implies that if a given instance is feasible, then it admits a solution hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ in which the number of hyperedges is K_p. We note that K_p need not be polynomial in $|V|$. His proof also leads to a pseudo-polynomial time algorithm to solve DS-Skew-SupMod-WeakCover-using-H (the algorithm is only pseudo-polynomial time and not polynomial time since the number of hyperedges in the returned hypergraph could be K_p and hence, the run-time depends on K_p). In this work, we show that feasible instances admit a solution with $O(|V|)$ hyperedges and give a strongly polynomial-time algorithm to solve DS-Skew-SupMod-WeakCover-using-H.

▶ **Theorem 12.** *There exists an algorithm to solve DS-Skew-SupMod-WeakCover-using-H that runs in time $O(|V|^5)$ using $O(|V|^4)$ queries to p-`max-sc-Oracle`, where V is the ground set of the input instance. Moreover, if the instance is feasible, then the algorithm returns a solution hypergraph that contains at most $4|V|$ hyperedges. For each query to p-`max-sc-Oracle` made by the algorithm, the hypergraph (G_0, c_0) used as input to the query has $O(|V|)$ vertices and $O(|V|)$ hyperedges.*

Theorem 5 along with the observations of Szigeti [46] and Bernáth and Király [7] together lead to a strongly polynomial-time algorithm for DS-Sym-Skew-SupMod-StrongCover-using-H. We discuss this result for strong cover and its applications in the full version [10] (Theorem 5 is derived as one of the applications).

1.3 Extension to Near-Uniform and Simultaneous Covers

In the complete version [10], we prove several additional results which we briefly describe now. A hypergraph is *uniform* if all hyperedges have the same size; a hypergraph is *near-uniform* if every pair of hyperedges differ in size by at most one. Uniformity/near-uniformity is a natural constraint in network design applications involving hypergraphs – we might be able to create only equal-sized hyperedges in certain applications. Requiring uniform (or near-uniform) hyperedges can also be viewed as a fairness inducing constraint in certain applications. We show that feasible instances of DS-Simul-Skew-SupMod-WeakCover-using-H (and consequently, DS-Hypergraph-CA-using-H) admit a solution with $O(|V|)$ *near-uniform* hyperedges and give a strongly polynomial time algorithm to construct such a solution. We also address *simultaneous* connectivity augmentation problems using hyperedges, where the goal is to *simultaneously* augment two input hypergraphs using the same set of hyperedges to satisfy given connectivity requirements and degree specifications. These algorithms are LP-based in contrast to the combinatorial algorithm presented in this extended abstract. Moreover, the analysis techniques for these LP-based algorithms build on the analysis techniques of this extended abstract. We omit these more general results from this extended abstract in the interests of brevity. For a summary of these results, refer to Tables 1 and 2.

In Table 1 below, we list graph/hypergraph connectivity augmentation problems using edges/hyperedges, previously known results, and our results. In Table 2 below, we list the general function cover problems using hyperedges, previously known results, and our results.

Table 1 Complexity of Graph and Hypergraph Connectivity Augmentation Problems using Edges and Hyperedges. Here, n and m denote the number of vertices and hyperedges respectively in the input hypergraph. Problems having "Near-Uniform" in their title are similar to the corresponding problems without "Near-Uniform" in their title but have the additional requirement that the returned solution hypergraph be *near-uniform*. Results marked with an asterisk are proved in the complete version of the paper [10]. The problems in the last two rows correspond to connectivity augmentation using hyperedges problems where the goal is to simultaneously augment two input hypergraphs using the same set of hyperedges (see the complete version of the paper [10] for their definitions).

Problem	**Complexity Status**
DS-Graph-CA-using-E	Strong Poly [25]
DS-Hypergraph-CA-using-E	NP-comp [20, 40]
DS-Hypergraph-CA-using-H	Psuedo Poly [46] $O(n^7(n+m)^2)$ time (Thm 5)
DS-Hypergraph-CA-using-near-uniform-H	Pseudo Poly [7] Strong Poly*
DS-Simul-Hypergraph-CA-using-H	Pseudo Poly [7] Strong Poly*
DS-Simul-Hypergraph-CA-using-near-uniform-H	Pseudo Poly [7] Strong Poly*

1.4 Techniques: Structural and Algorithmic Result

In this section, we discuss our techniques underlying the proof of Theorem 12. For a function $m : V \to \mathbb{R}$, we denote $m(X) := \sum_{u \in X} m(u)$. Szigeti [46] gave a complete characterization of feasible instances of DS-Skew-SupMod-WeakCover-using-H. He showed that an instance $(m : V \to \mathbb{Z}_{\geq 0}, p : 2^V \to \mathbb{Z})$ of DS-Skew-SupMod-WeakCover-using-H is feasible if and only if $m(X) \geq p(X)$ for every $X \subseteq V$ and $m(u) \leq K_p$ for every $u \in V$. We note that this characterization immediately implies that feasibility of a given instance of DS-

Table 2 Complexity of degree-specified skew-supermodular cover using hyperedges problems. Problems having "Near-Uniform" in their title are similar to the corresponding problems without "Near-Uniform" in their title but have the additional requirement that the returned solution hypergraph be *near-uniform*. Results marked with an asterisk are proved in the complete version of the paper [10]. The problems in the last two rows correspond to cover using hyperedges problems where the goal is to simultaneously cover two different functions using the same set of hyperedges (see the complete version of the paper [10] for their definitions).

Problem	**Complexity Status**
DS-Skew-SupMod-WeakCover-using-H	Pseudo Poly [46] Strong Poly (Thm 12)
DS-Sym-Skew-SupMod-StrongCover-using-H	Pseudo Poly [46] Strong Poly*
DS-Skew-SupMod-WeakCover-using-near-uniform-H	Pseudo Poly [7] Strong Poly*
DS-Sym-Skew-SupMod-StrongCover-using-near-uniform-H	Pseudo Poly [7] Strong Poly*
DS-Simul-Skew-SupMod-WeakCover-using-near-uniform-H	Pseudo Poly [7] Strong Poly*
DS-Simul-Sym-Skew-SupMod-StrongCover-using-near-uniform-H	Pseudo Poly [7] Strong Poly*

Skew-SupMod-WeakCover-using-H can be verified using two calls to p-`max-sc-Oracle`. Our main result, stated in Theorem 13 below, is that feasible instances admit a solution with *linear* number of hyperedges; moreover, such a solution can be found in strongly polynomial time. Theorem 13 immediately implies Theorem 12 via Szigeti's characterization.

▶ **Theorem 13.** *Let $p : 2^V \to \mathbb{Z}$ be a skew-supermodular function and $m : V \to \mathbb{Z}_{\geq 0}$ be a non-negative function such that:*
(a) *$m(X) \geq p(X)$ for every $X \subseteq V$ and*
(b) *$m(u) \leq K_p$ for every $u \in V$.*
Then, there exists a hypergraph $\left(H = (V, E), w : E \to \mathbb{Z}_+\right)$ satisfying the following four properties:
(1) *$b_{(H,w)}(X) \geq p(X)$ for every $X \subseteq V$,*
(2) *$b_{(H,w)}(u) = m(u)$ for every $u \in V$,*
(3) *$\sum_{e\in E} w(e) = K_p$, and*
(4) *$|E| \leq 4|V|$.*
Furthermore, given a function $m : V \to \mathbb{Z}_{\geq 0}$ and access to p-`max-sc-Oracle` of a skew-supermodular function $p : 2^V \to \mathbb{Z}$ where m and p satisfy conditions (a) and (b) above, there exists an algorithm that runs in time $O(|V|^5)$ using $O(|V|^4)$ queries to p-`max-sc-Oracle` and returns a hypergraph satisfying properties 1-4 above. The run-time includes the time to construct the hypergraphs that are used as inputs to p-`max-sc-Oracle`. Moreover, for each query to p-`max-sc-Oracle`, the hypergraph (G_0, c_0) used as input to the query has $O(|V|)$ vertices and $O(|V|)$ hyperedges.

In the rest of this section, we describe our proof technique for Theorem 13. The algorithmic result in Theorem 13 follows from our techniques for the existential result using known tools for submodular functions. So, we focus on describing our proof technique for the existial result here. Let $p : 2^V \to \mathbb{Z}$ be a skew-supermodular function and $m : V \to \mathbb{Z}_{\geq 0}$ be a non-negative function such that $m(X) \geq p(X)$ for every $X \subseteq V$ and $m(u) \leq K_p$ for every $u \in V$. Our proof of the existential result builds on the techniques of Szigeti [46] who proved the existence of a hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ satisfying properties (1)-(3), so we briefly recall his techniques. His proof proceeds by picking a minimal counterexample and arriving at a contradiction. Consequently, the algorithm implicit in the proof is naturally recursive. We present the algorithmic version of his proof since it will be useful for our purposes.

For the purposes of the algorithmic proof of Szigeti's result, we assume that m is a positive-valued function. We show that this assumption is without loss of generality since an arbitrary instance can be reduced to such an instance (for details, see first two paragraphs in Section 2). The main insight underlying Szigeti's proof is the following characterization of hyperedges in a feasible hypergraph.

▶ **Proposition 14.** *Let $p : 2^V \to \mathbb{Z}$ be a skew-supermodular function and $m : V \to \mathbb{Z}_+$ be a positive function such that $m(X) \geq p(X)$ for every $X \subseteq V$ and $m(u) \leq K_p$ for every $u \in V$. Let $A \subseteq V$. Then, there exists a hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$ satisfying properties (1)-(3) such that $A \in E$ if and only if A satisfies the following:*

(i) *A is a transversal for the family of p-maximizers,*
(ii) *A contains the set $\{u \in V : m(u) = K_p\}$,*
(iii) *$m(v) \geq 1$ for each $v \in A$, and*
(iv) *$m(X) - |A \cap X| \geq p(X) - 1$ for every $X \subseteq V$.*

Szigeti's Algorithm [46]. Proposition 14 leads to the following natural recursive strategy to compute a feasible hypergraph. If $K_p = 0$, then the algorithm is in its base case and returns the empty hypergraph (with no vertices) – here, we note that $0 < m(u) \leq K_p = 0$ for every $u \in V$, and consequently, $V = \emptyset$ and thus, the empty hypergraph satisfies properties (1)-(3) as desired. Alternatively, if $K_p > 0$, then the algorithm recurses on appropriately revised versions of the input functions p and m. In particular, the algorithm picks an arbitrary minimal transversal T for the family of p-maximizers and computes the set $A := T \cup \{u \in V : m(u) = K_p\}$. It can be shown that this set A satisfies properties (i)-(iv) of Proposition 14; consequently, there exists a feasible hypergraph containing the hyperedge A. In order to revise the input functions, the algorithm defines (H_0, w_0) to be the hypergraph on vertex set V consisting of the single hyperedge A with weight $w_0(A) = 1$ and constructs the set $\mathcal{Z} := \{u \in A : m(u) = 1\}$. Next, the algorithm defines revised functions $m'' : V - \mathcal{Z} \to \mathbb{Z}$ and $p'' : 2^{V-\mathcal{Z}} \to \mathbb{Z}$ as $m''(u) := m(u) - 1$ if $u \in A - \mathcal{Z}$ and $m''(u) := m(u)$ if $u \in V - A - \mathcal{Z}$ and $p''(X) := \max\{p(X \cup R) - b_{(H_0,w_0)}(X \cup R) : R \subseteq \mathcal{Z}\}$ for every $X \subseteq V - \mathcal{Z}$. The algorithm recurses on the revised input functions m'' and p'' to obtain a hypergraph (H'', w''). Finally, the algorithm obtains the hypergraph (G, c) by adding vertices $\mathcal{Z}$ to (H'', w''), and returns the hypergraph $(G + H_0, c + w_0)$. It can be shown that p'' is a skew-supermodular function and m'' is a positive function such that $m''(X) \geq p''(X)$ for all $X \subseteq V - \mathcal{Z}$ and $m''(u) \leq K_{p''}$ for every $u \in V - \mathcal{Z}$. We note that $K_{p''} = K_p - 1$ by the definition of the function p'' and the choice of set A being a transversal for the family of p-maximizers. Consequently, by induction on K_p and Proposition 1, the algorithm can be shown to terminate in K_p recursive calls and return a hypergraph (H, w) satisfying properties 1-3. Furthermore, we observe that the number of distinct hyperedges added by the algorithm is at most the number of recursive calls since each recursive call adds at most one new hyperedge to (H, w). Thus, the number of distinct hyperedges in (H, w) is also at most K_p. Consequently, in order to reduce the number of distinct hyperedges, it suffices to reduce the recursion depth of Szigeti's algorithm. We note that there exist inputs for which Szigeti's algorithm can indeed witness an execution with exponential recursion depth, and consequently may construct only exponential sized hypergraphs on those inputs (see example in Remark 11). So, we necessarily have to modify his algorithm to reduce the recursion depth.

Our Algorithm. We now describe our modification of Szigeti's algorithm to reduce the recursion depth. We observe that the hyperedge A chosen during a recursive call of Szigeti's algorithm could also be the hyperedge chosen during a subsequent recursive call. In fact, this could repeat for several consecutive recursive calls before the algorithm cannot pick the hyperedge A anymore. We avoid such a sequence of consecutive recursive calls by picking as many copies of the hyperedge A as possible into the hypergraph (H_0, w_0) (i.e., set the weight to be the number of copies picked) and revising the input functions m and p accordingly for recursion. We describe this formally now. Let (p, m) be the input tuple and $A \subseteq V$ be as defined by Szigeti's algorithm. Let

$$\alpha = \min \begin{cases} \alpha^{(1)} := \min\left\{m(u) : u \in A\right\} \\ \alpha^{(2)} := \min\left\{K_p - p(X) : X \subseteq V - A\right\} \\ \alpha^{(3)} := \min\left\{K_p - m(u) : u \in V - A\right\} \end{cases}$$

We construct the hypergraph (H_0, w_0) on vertex set V consisting of a single hyperedge A with weight $w_0(A) = \alpha$. Next, we proceed similar to Szigeti's algorithm as follows: We construct the sets $\mathcal{Z} := \{u \in A : m(u) = \alpha\}$ and $V'' := V - \mathcal{Z}$. Next, we define the set function $p'' : 2^{V''} \to \mathbb{Z}$ as $p''(X) := \max\{p(X \cup R) - b_{(H_0,w_0)}(X \cup R) : R \subseteq \mathcal{Z}\}$ for every $X \subseteq V''$, and the function $m'' : V'' \to \mathbb{Z}$ as $m''(u) := m(u) - \alpha \mathbb{1}_{u \in A}$ for every $u \in V''$, where $\mathbb{1}_{u\in A}$ evaluates to one if $u \in A$ and evaluates to zero otherwise. Next, we recurse on the input tuple (p'', m'') to obtain a hypergraph (H'', w''); obtain the hypergraph (G, c) by adding vertices $\mathcal{Z}$ to (H'', w''), and return $(G + H_0, c + w_0)$.

Recursion Depth Analysis. By induction on K_p (generalizing Szigeti's proof), it can be shown that our algorithm returns a hypergraph satisfying properties 1-3 of Theorem 13 and also terminates within finite number of recursive calls. We now sketch our proof to show a strongly polynomial bound on the recursion depth of our modified algorithm. For this, we consider how the value α is computed. We recall that α is the minimum of the three values $\{\alpha^{(1)}, \alpha^{(2)}, \alpha^{(3)}\}$. Using this, we identify a potential function which strictly increases with recursion depth. For this, we consider three set families that we define now. Let ℓ be the recursion depth of the algorithm on an input instance and let $i \in [\ell]$. Let $\mathcal{Z}_i$ be the set $\mathcal{Z}$ and $\mathcal{D}_i$ be the set $\{u \in V : m(u) = K_p\}$ in the i^{th} recursive call. Let $\mathcal{F}_i$ be the family of inclusionwise minimal p-maximizers where p is the set function input to the i^{th} recursive call (a set X is a p-maximizer if $p(X) = K_p$). Let $\mathcal{Z}_{\leq i} := \cup_{j=1}^{i} \mathcal{Z}_j$ and $\mathcal{F}_{\leq i} := \cup_{j=1}^{i} \mathcal{F}_j$. We consider the potential function $\phi(i) := |\mathcal{Z}_{\leq i}| + |\mathcal{F}_{\leq i}| + |\mathcal{D}_i|$ and show that each of the three terms in the function is non-decreasing with i. Furthermore, if α is determined by $\alpha^{(1)}$, then $|\mathcal{Z}_{\leq i}|$ strictly increases; if α is determined by $\alpha^{(2)}$, then $|\mathcal{F}_{\leq i}|$ strictly increases; and if α is determined by $\alpha^{(3)}$, then $|\mathcal{D}_i|$ strictly increases (see Lemma 20). We note that $\mathcal{Z}_{\leq \ell} \subseteq V$ and $\mathcal{D}_\ell \subseteq V$. Moreover, we examine how the input functions p across recursive calls relate to each other and exploit skew-supermodularity of p to show that the family $\mathcal{F}_{\leq \ell}$ is *laminar* over the ground set V (see Lemma 3.2 of [10]). These facts together imply that the recursion depth is at most $4|V| - 1$.

▶ Remark 15. Our main algorithmic contribution is the modification to Szigeti's algorithm to pick as many copies of a chosen hyperedge as possible during a recursive call, i.e., pick hyperedge A with weight α as opposed to weight 1. Without our modification, there exist inputs for which Szigeti's original algorithm can indeed witness an execution with exponential recursion depth, and consequently may construct only exponential sized hypergraphs on those inputs (see example in Remark 11). We do note that our style of modification is fairly common while converting inductive proofs into efficient algorithms in combinatorial optimization literature. However, there is no standard run-time analysis technique for such modifications – analyzing the run-time of such modifications of inductive proofs to algorithms has required adhoc combinatorial potential functions based on problem structure. Our main analysis contribution here is identifying an appropriate potential function to show that our modified algorithm indeed has linear recursion depth. In the complete version of this work available in arXiv, we build on these analysis ideas to design and analyze LP-based algorithms for the same problems. The LP-based algorithms have two additional advantages: they can return near-uniform hypergraphs as solutions and have the ability to address simultaneous function cover problems.

2 Weak Cover with Linear Number of Hyperedges

In this section, we prove the existential result in Theorem 13. The strongly polynomial-time algorithm follows from our proof for the existential result via standard tools in submodularity and their details are given in the complete version [10].

Notation. For a function $f : 2^V \to \mathbb{Z}$ and a set $\mathcal{Z} \subseteq V$, we denote the contraction of f to $V - \mathcal{Z}$ as $f/_{\mathcal{Z}} : 2^{V-\mathcal{Z}} \to \mathbb{Z}$, where $p/_{\mathcal{Z}}(X) := \max\{p(X \cup R) : R \subseteq \mathcal{Z}\}$ for every $X \subseteq V - \mathcal{Z}$. For a function $m : V \to \mathbb{Z}$ and a set $\mathcal{Z} \subseteq V$, we denote the restriction of m to $V - \mathcal{Z}$ as $m\backslash_{\mathcal{Z}} : V - \mathcal{Z} \to \mathbb{Z}_+$, where $m\backslash_{\mathcal{Z}}(u) = m(u)$ for every $u \in V - \mathcal{Z}$.

We first describe our proof of Theorem 13 under the assumption that the input function $m : V \to \mathbb{Z}_+$ is a *positive* function: In Section 2.1, we present our algorithm (see Algorithm 1). In Section 2.2, we show that our algorithm terminates within a finite (pseudo-polynomial) number of recursive calls and returns a hypergraph satisfying properties (1), (2) and (3) of Theorem 13 (see Lemma 17). In Section 2.3, we give a tighter bound on the number of recursive calls witnessed by our algorithm and show that the hypergraph returned by the algorithm also satisfies property (4) of Theorem 13 (see Lemma 21). In the complete version, we show that our algorithm runs in strongly polynomial time, given the appropriate function evaluation oracle (see Lemma 4.11 of [10]) – we note that this component of the proof is omitted here since it involves standard tools and arguments in submodularity. Lemmas 17, 21, and the polynomial-time implementation details from the full version [10] together complete the proof of Theorem 13 under the assumption that the input function $m : V \to \mathbb{Z}_+$ is a *positive* function. All missing proofs can be found in the complete version of the paper [10].

We now briefly remark on how to circumvent the positivity assumption on the input function m in the above proof. Suppose that the input function m is not a positive function. Let $\mathcal{Z} := \{u \in V : m(u) = 0\} \neq \emptyset$. Then, $p/_{\mathcal{Z}} : 2^{V-\mathcal{Z}} \to \mathbb{Z}$ is a skew-supermodular function and $m\backslash_{\mathcal{Z}} : V - \mathcal{Z} \to \mathbb{Z}_+$ is a *positive* function satisfying the two hypothesis conditions of Theorem 13, i.e. $m\backslash_{\mathcal{Z}}(X) \geq p/_{\mathcal{Z}}(X)$ for every $X \subseteq V - \mathcal{Z}$ and $m\backslash_{\mathcal{Z}}(u) \leq K_{p/_{\mathcal{Z}}}$ for every $u \in V - \mathcal{Z}$. Furthermore, we observe that a hypergraph satisfying properties (1)-(4) for the functions $p/_{\mathcal{Z}}$ and $m\backslash_{\mathcal{Z}}$ also satisfies the four properties for the functions p and m. Finally, $p/_{\mathcal{Z}}$-`max-sc-Oracle` can be implemented using p-`max-sc-Oracle` in strongly polynomial time. Hence, applying our result for the input $(p/_{\mathcal{Z}}, m\backslash_{\mathcal{Z}})$ implies the result for (p, m).

2.1 The Algorithm

Our algorithm takes as input (1) a skew-supermodular function $p : 2^V \to \mathbb{Z}$, and (2) a *positive* function $m : V \to \mathbb{Z}_+$ and returns a hypergraph $\left(H = (V, E), w\right)$. We note that in contrast to the non-negative function m appearing in the statement of Theorem 13, the function m that is input to our algorithm should be positive.

We now give an informal description of the algorithm (refer to Algorithm 1 for a formal description). Our algorithm is recursive. If $V = \emptyset$, then the algorithm is in its base case and returns the empty hypergraph. Otherwise, the algorithm is in its recursive case. First, the algorithm computes an arbitrary minimal transversal $T \subseteq V$ for the family of p-maximizers – i.e., an inclusionwise minimal set $T \subseteq V$ such that $T \cap X \neq \emptyset$ for every $X \subseteq V$ with $p(X) = K_p$. Next, the algorithm computes the set $\mathcal{D} \coloneqq \{u \in V : m(u) = K_p\}$ and defines the set $A \coloneqq T \cup \mathcal{D}$. Next, the algorithm uses the set A to compute the following three intermediate quantities:

$$\begin{aligned}
\alpha^{(1)} &\coloneqq \min\{m(u) : u \in A\},\\
\alpha^{(2)} &\coloneqq \min\left\{K_p - p(X) : X \subseteq V - A\right\},\\
\alpha^{(3)} &\coloneqq \min\left\{K_p - m(u) : u \in V - A\right\},
\end{aligned}$$

and defines $\alpha \coloneqq \min\left\{\alpha^{(1)}, \alpha^{(2)}, \alpha^{(3)}\right\}$. Next, the algorithm computes the set $\mathcal{Z} \coloneqq \{u \in V : m(u) - \alpha = 0\}$ and defines (H_0, w_0) to be the hypergraph on vertex set V consisting of a single hyperedge A with weight $w_0(A) = \alpha$. Next, the algorithm defines the two functions $m' : V \to \mathbb{Z}$ and $m'' : V - \mathcal{Z} \to \mathbb{Z}$ as $m' \coloneqq m - \alpha\chi_A$, and $m'' \coloneqq m'\backslash_{\mathcal{Z}}$. Next, the algorithm defines the two functions $p' : 2^V \to \mathbb{Z}$ and $p'' : 2^{V-\mathcal{Z}} \to \mathbb{Z}$ as $p' \coloneqq p - b_{(H_0,w_0)}$ and $p'' \coloneqq p'/_{\mathcal{Z}}$. The algorithm then recursively calls itself with the input tuple (p'', m'') to obtain a hypergraph $\left(H'' = \left(V'' \coloneqq V - \mathcal{Z}, E''\right), w''\right)$. Finally, the algorithm extends the hypergraph H'' with the set $\mathcal{Z}$ of vertices, adds the set A with weight α as a new hyperedge to the hyperedge set E'', and returns the resulting hypergraph. If the hyperedge A already has non-zero weight in (H'', w''), then the algorithm increases the weight of the hyperedge A by α, and returns the resulting hypergraph.

2.2 Termination and Partial Correctness

In this section, we show that Algorithm 1 terminates within a finite (pseudo-polynomial) number of recursive calls, and moreover, returns a hypergraph satisfying properties (1)-(3) of Theorem 13. The proofs in this section are appropriate generalizations of those by Szigeti [46] and so we defer the details of the proofs to the complete version [10]. The following lemma states useful properties of Algorithm 1.

▶ **Lemma 16.** *Suppose that $V \neq \emptyset$ and the input to Algorithm 1 is a tuple (p, m), where $p : 2^V \to \mathbb{Z}$ is a skew-supermodular function and $m : V \to \mathbb{Z}_+$ is a positive function such that $m(X) \geq p(X)$ for all $X \subseteq V$ and $m(u) \leq K_p$ for all $u \in V$. Let $\alpha, \mathcal{Z}, m', p', m'', p''$ be as defined by Algorithm 1. Then, we have that*

(a) $\alpha \geq 1$,

(b) *the function $m'' : V - \mathcal{Z} \to \mathbb{Z}_+$ is a positive function,*

(c) $p'' = (p - b_{(H_0,w_0)})/_{\mathcal{Z}}$*; moreover, the function p'' is skew-supermodular,*

(d) $m''(u) \leq K_{p''}$ *for all $u \in V - \mathcal{Z}$,*

(e) $m''(X) \geq p''(X)$ *for all $X \subseteq V - \mathcal{Z}$,*

(f) $K_{p''} = K_{p'} = K_p - \alpha$*, and*

(g) $m''(V - \mathcal{Z}) < m(V)$.

Algorithm 1 Weak covering with hyperedges.

```
INPUT: Skew-supermodular function $p : 2^V \to \mathbb{Z}$ and positive function $m : V \to \mathbb{Z}_+$
OUTPUT: Hypergraph $(H = (V, E), w : E \to \mathbb{Z}_+)$

ALGORITHM $(p, m)$ :
 1: if $V = \emptyset$ then return Empty Hypergraph $\left((\emptyset, \emptyset), \emptyset\right)$.
 2: else:
 3:     $\mathcal{D} := \{u \in V : m(u) = K_p\}$
 4:     $T :=$ an arbitrary minimal transversal for the family of $p$-maximizers
 5:     $A := T \cup \mathcal{D}$
 6:     $\alpha := \min \begin{cases} \alpha^{(1)} := \min\{m(u) : u \in A\} \\ \alpha^{(2)} := \min\{K_p - p(X) : X \subseteq V - A\} \\ \alpha^{(3)} := \min\{K_p - m(u) : u \in V - A\} \end{cases}$
 7:     $\mathcal{Z} := \{u \in A : m(u) - \alpha = 0\}$
 8:     Construct $\left(H_0 := (V, E_0 := \{A\}), w_0 : E_0 \to \{\alpha\}\right)$
 9:     $m' := m - \alpha\chi_A$ and $m'' := m'\backslash_{\mathcal{Z}}$
10:     $p' := p - b_{(H_0, w_0)}$ and $p'' := p'/_{\mathcal{Z}}$
11:     $(H'', w'') :=$ ALGORITHM $(p'', m'')$
12:     Obtain hypergraph $(G, c)$ from $(H'', w'')$ by adding vertices $\mathcal{Z}$.
13:     return $(G + H_0, c + w_0)$
```

The above lemma implies that if the input functions p and m to a recursive call of Algorithm 1 satisfy conditions (a) and (b) of Theorem 13, then the input functions to the subsequent recursive call p'', m'' as constructed by Algorithm 1 also satisfy conditions (a) and (b) of Theorem 13. Moreover, $K_{p''} < K_p$ thus guaranteeing that the recursion makes progress. These facts can together be used to prove the following lemma which shows finite termination and partial correctness – partial correctness since it proves only the first three conclusions of Theorem 13.

▶ **Lemma 17.** *Suppose that the input to Algorithm 1 is a tuple (p, m), where $p : 2^V \to \mathbb{Z}$ is a skew-supermodular function and $m : V \to \mathbb{Z}_+$ is a positive function such that $m(X) \geq p(X)$ for all $X \subseteq V$ and $m(u) \leq K_p$ for all $u \in V$. Then, Algorithm 1 terminates within a finite (pseudo-polynomial) number of recursive calls. Furthermore, the hypergraph $\left(H = (V, E), w : E \to \mathbb{Z}_+\right)$ returned by Algorithm 1 satisfies the following three properties:*
1. *$b_{(H,w)}(X) \geq p(X)$ for all $X \subseteq V$,*
2. *$b_{(H,w)}(u) = m(u)$ for all $u \in V$, and*
3. *$\sum_{e \in E} w(e) = K_p$.*

2.3 Recursion Depth and Hypergraph Support Size

In this section, we prove that our algorithm achieves the fourth conclusion of Theorem 13. For this, it suffices to prove an upper bound on the number of recursive calls witnessed by an execution of Algorithm 1. We will prove this in Lemma 21 which is at the end of the section.

Notation. By Lemma 17, the number of recursive calls made by Algorithm 1 is finite. We will use ℓ to denote the depth of recursion. We will refer to the recursive call at depth $i \in [\ell]$ as *recursive call i* or the *i^{th} recursive call*. We let V_i denote the ground set at the start of recursive call i, and $p_i : 2^{V_i} \to \mathbb{Z}$ and $m_i : V_i \to \mathbb{Z}_{\geq 0}$ denote the input functions to recursive call i. Furthermore, for $i \in [\ell - 1]$ we use a subscript i to denote the value of a variable during the i^{th} recursive call of Algorithm 1 – the only exception to this notation is that we use (H_0^i, w_0^i) to denote (H_0, w_0) during the i^{th} recursive call. For convenience, we also define the relevant sets, values and functions during the base case (ℓ^{th} recursive call) as follows: $A_\ell, \mathcal{D}_\ell, \mathcal{Z}_\ell := \emptyset$, $\alpha_\ell, \alpha_\ell^{(1)}, \alpha_\ell^{(2)}, \alpha_\ell^{(3)} := 0$, $m'_\ell, m''_\ell, m_{\ell+1} := m_\ell$, and $p'_\ell, p''_\ell, p_{\ell+1} := p_\ell$. Finally, let $\left(H_i = (V_i, E_i), w_i\right)$ denote the hypergraph returned by the i^{th} recursive call.

We note that Lemma 16 and induction on the recursion depth i immediately imply the following lemma which says that for every $i \in [\ell]$, the input tuple (p_i, m_i) satisfies the hypothesis of Theorem 13.

▶ **Lemma 18.** *Suppose that the input to Algorithm 1 is a tuple (p_1, m_1), where $p_1 : 2^{V_1} \to \mathbb{Z}$ is a skew-supermodular function and $m_1 : V_1 \to \mathbb{Z}_+$ is a positive function such that $m_1(X) \geq p_1(X)$ for all $X \subseteq V_1$ and $m_1(u) \leq K_{p_1}$ for all $u \in V_1$. Let $\ell \in \mathbb{Z}_+$ be the number of recursive calls witnessed by the execution of Algorithm 1 and, for all $i \in [\ell]$, let (p_i, m_i) be the input tuple to the i^{th} recursive call of the execution. Then, for all $i \in [\ell]$ we have that $p_i : 2^{V_i} \to \mathbb{Z}$ is a skew-supermodular function and $m_i : V_i \to \mathbb{Z}_+$ is a positive function such that $m_i(X) \geq p_i(X)$ for all $X \subseteq V_i$ and $m_i(u) \leq K_{p_i}$ for all $u \in V_i$.*

Set Families. To analyze the recursion depth, we will focus on certain set families associated with an execution of Algorithm 1. Let $i \in [\ell]$ be a recursive call of Algorithm 1. We define $\mathcal{Z}_{\leq i} := \cup_{j \in [i]} \mathcal{Z}_i$. We use $\mathcal{F}_i$ and $\mathcal{F}'_i$ to denote the families of *minimal* p_i-maximizers and p'_i-maximizers respectively, i.e., $\mathcal{F}_i$ is the collection of inclusionwise minimal sets in the family $\{X \subseteq V_i : p_i(X) = K_{p_i}\}$ and $\mathcal{F}'_i$ is the collection of inclusionwise minimal sets in the family $\{X \subseteq V_i : p'_i(X) = K_{p'_i}\}$. Lemma 19 below shows the progression of these families across recursive calls of an execution of Algorithm 1. We will also be interested in families of *all* minimal maximizers of the input functions witnessed by the algorithm up to a given recursive call. Formally, we define the family $\mathcal{F}_{\leq i} := \cup_{j \in [i]} \mathcal{F}_j$. Lemma 20 below summarizes useful properties of the stated families.

▶ **Lemma 19.** *Suppose that the input to Algorithm 1 is a tuple (p_1, m_1), where $p_1 : 2^V \to \mathbb{Z}$ is a skew-supermodular function and $m_1 : V \to \mathbb{Z}_+$ is a positive function such that $m_1(X) \geq p_1(X)$ for all $X \subseteq V_1$ and $m_1(u) \leq K_{p_1}$ for all $u \in V_1$. Let $\ell \in \mathbb{Z}_+$ be the number of recursive calls witnessed by the execution of Algorithm 1 and, for all $i \in [\ell]$, let (p_i, m_i) be the input tuple to the i^{th} recursive call of the execution. Then, for all $i \in [\ell]$, we have the following:*

(a) *if $Y \subseteq V_i$ is a p_i-maximizer, then Y is also a p'_i-maximizer, and*
(b) *if $Y \subseteq V_i$ is a p'_i-maximizer such that $Y - \mathcal{Z}_i \neq \emptyset$, then $Y - \mathcal{Z}_i$ is a p_{i+1}-maximizer.*

▶ **Lemma 20.** *Suppose that the input to Algorithm 1 is a tuple (p_1, m_1), where $p_1 : 2^V \to \mathbb{Z}$ is a skew-supermodular function and $m_1 : V \to \mathbb{Z}_+$ is a positive function such that $m_1(X) \geq p_1(X)$ for all $X \subseteq V_1$ and $m_1(u) \leq K_{p_1}$ for all $u \in V_1$. Let $\ell \in \mathbb{Z}_+$ be the number of recursive calls witnessed by the execution of Algorithm 1 and, for all $i \in [\ell]$, let (p_i, m_i) be the input tuple to the i^{th} recursive call of the execution. Then, for all $i \in [\ell - 1]$, we have the following:*

(a) *$\mathcal{Z}_{\leq i} \subseteq \mathcal{Z}_{\leq i+1}$; furthermore, $\alpha_i = \alpha_i^{(1)}$ if and only if $\mathcal{Z}_i \neq \emptyset$ (i.e., $\mathcal{Z}_{\leq i} \subsetneq \mathcal{Z}_{\leq i+1}$),*
(b) *$\mathcal{F}_{\leq i} \subseteq \mathcal{F}_{\leq i+1}$; furthermore, if $\alpha_i = \alpha_i^{(2)} < \alpha_i^{(1)}$, then $\mathcal{F}_{\leq i} \subsetneq \mathcal{F}_{\leq i+1}$,*
(c) *$\mathcal{D}_i \subseteq \mathcal{D}'_i \subseteq \mathcal{D}_{i+1}$; furthermore, if $\alpha_i = \alpha_i^{(3)}$, then $\mathcal{D}_i \subsetneq \mathcal{D}'_i$.*

We now show the main result of the section which says that an execution of Algorithm 1 witnesses at most $4|V|$ recursive calls. Since every recursive call adds at most one new hyperedge to the hypergraph returned by the execution, this also implies that the number of hyperedges in a solution returned by Algorithm 1 is at most $4|V|$. This shows property (4) of Theorem 13.

▶ **Lemma 21.** *Suppose that $V_1 \neq \emptyset$ and the input to Algorithm 1 is a tuple (p_1, m_1), where $p_1 : 2^{V_1} \to \mathbb{Z}$ is a skew-supermodular function and $m_1 : V_1 \to \mathbb{Z}_+$ is a positive function such that $m_1(X) \geq p_1(X)$ for all $X \subseteq V_1$ and $m_1(u) \leq K_{p_1}$ for all $u \in V_1$. Let $\ell \in \mathbb{Z}_+$ be the number of recursive calls witnessed by the execution of Algorithm 1. Then,*
1. *the recursion depth ℓ of Algorithm 1 is at most $4|V_1| - 1$, and*
2. *the number of hyperedges in the hypergraph $(H = (V_1, E), w)$ returned by Algorithm 1 is at most $4|V_1| - 1$.*

Proof. We note that part (2) of the current lemma follows from part (1) since every recursive call adds at most one new hyperedge to the hypergraph returned by the execution in Step 11. Thus, it suffices to show part (1). We define a potential function $\phi : [\ell] \to \mathbb{Z}_{\geq 0}$ as follows: for each $i \in [\ell]$,

$$\phi(i) := |\mathcal{Z}_{\leq i}| + |\mathcal{F}_{\leq i}| + |\mathcal{D}_i|.$$

By Lemma 20, we have that ϕ is a monotone increasing function, since each of the three terms is non-decreasing and at least one of the three terms strictly increases with increasing $i \in [\ell]$. Consequently, the number of recursive calls witnessed by the execution of Algorithm 1 is at most $\phi(\ell) - \phi(0) \leq |\mathcal{Z}_{\leq \ell}| + |\mathcal{F}_{\leq \ell}| + |\mathcal{D}_\ell| \leq 2|V_1| + |\mathcal{F}_{\leq \ell}|$. It remains to bound the size of the family $\mathcal{F}_{\leq \ell}$. By Lemma 16(c) and induction on i, we have that $p_{i+1} = (p_i - b_{(H_0^i, w_0^i)})/\mathcal{Z}_i$ for every recursive call $i \in [\ell - 1]$. In the complete version, we show that for such a sequence $p_1, \ldots, p_\ell$ of skew-supermodular functions, the family $\mathcal{F}_{p_{\leq \ell}}$ is laminar (see Lemma 3.2 of [10]). Consequently, we have that the number of recursive calls witnessed by the execution of Algorithm 1 is at most $2|V_1| + |\mathcal{F}_{\leq \ell}| \leq 4|V_1| - 1$. ◀

3 Conclusion

The theme of this work is showing that certain hypergraph network design problems admit solution hypergraphs with polynomial number of hyperedges and moreover, can be solved in strongly polynomial time. We believe that this is a necessary step to understand hypergraph network design problems further - both in polynomial-time solvable cases as well as NP-hard cases. Our results are for certain abstract hypergraph function cover problems but they have numerous applications; in particular, they enable the first strongly polynomial time algorithms for (i) degree-specified hypergraph connectivity augmentation using hyperedges and (ii) degree-specified hypergraph node-to-area connectivity augmentation using hyperedges. Previous best-known run-time for these problems were pseudo-polynomial. We believe that the abstract hypergraph function cover problems might find more applications in the future. We refer the reader to the complete version of this work for additional results and a discussion of future directions [10].

We note that recent results have shown that DS-Graph-CA-using-E for the case of uniform requirement function can be solved in near-linear time [12, 13, 14]. These results lead to the following natural questions – *Algorithmic question:* Is it possible to solve DS-Hypergraph-CA-using-H in near-linear time? *Structural question:* For feasible instances of DS-Hypergraph-CA-using-H, does there exist a solution hypergraph whose *size* is linear in the number of vertices? We define the size of a hypergraph to be the sum of the sizes of the hyperedges (and not simply the number of hyperedges). Our results show that there exists a solution hypergraph in which the number of hyperedges is linear and hence, the size of such a solution hypergraph is quadratic in the number of vertices. We believe that an affirmative answer to the algorithmic question would also imply an affirmative answer to the structural question. On the other hand, answering the structural question would be a helpful stepping stone towards the algorithmic question.

References

1 O. Alrabiah, V. Guruswami, and R. Li. Randomly punctured Reed-Solomon codes achieve list-decoding capacity over linear-sized fields. Preprint on arXiv: 2304.09445, 2023.

2 N. Bansal, O. Svensson, and L. Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *IEEE 60th Annual Symposium on Foundations of Computer Science*, FOCS, pages 910–928, 2019.

3 C. Beideman, K. Chandrasekaran, and W. Wang. Deterministic enumeration of all minimum k-cut-sets in hypergraphs for fixed k. In *Proceedings of the 33rd annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2208–2228, 2022.

4 C. Beideman, Ka. Chandrasekaran, and W. Wang. Counting and enumerating optimum cut sets for hypergraph k-partitioning problems for fixed k. In *International Colloquium on Automata, Languages and Programming*, ICALP, pages 16:1–16:18, 2022.

5 A. Benczúr. Parallel and fast sequential algorithms for undirected edge connectivity augmentation. *Math. Program.*, 84:595–640, 1999.

6 A. Benczúr and D. Karger. Augmenting Undirected Edge Connectivity in $\tilde{O}(n^2)$ Time. *Journal of Algorithms*, 37(1):2–36, 2000.

7 A. Bernáth and T. Király. Covering skew-supermodular functions by hypergraphs of minimum total size. *Operations Research Letters*, 37(5):345–350, 2009.

8 A. Bhalgat, R. Hariharan, T. Kavitha, and D. Panigrahi. Fast Edge Splitting and Edmonds' Arborescence Construction for Unweighted Graphs. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 455–464, 2008.

9 J. Byrka, F. Grandoni, and A. J. Ameli. Breaching the 2-approximation barrier for connectivity augmentation: a reduction to steiner tree. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 815–825, 2020.

10 K. Bérczi, K. Chandrasekaran, T. Király, and S. Kulkarni. Hypergraph connectivity augmentation in strongly polynomial time, 2024. `arXiv:2402.10861`.

11 G.-R. Cai and Y.-G. Sun. The minimum augmentation of any graph to k-edge-connected graph. *Networks*, 19:151–172, 1989.

12 R. Cen, W. He, J. Li, and Debmalya Panigrahi. Steiner connectivity augmentation and splitting-off in poly-logarithmic maximum flows. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2449–2488, 2023.

13 R. Cen, J. Li, and D. Panigrahi. Augmenting edge connectivity via isolating cuts. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 3237–3252, 2022.

14 R. Cen, J. Li, and D. Panigrahi. Edge connectivity augmentation in near-linear time. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 137–150, 2022.

15 K. Chandrasekaran and C. Chekuri. Hypergraph k-cut for fixed k in deterministic polynomial time. *Mathematics of Operations Research*, 47(4), 2022. Prelim. version in FOCS 2020.

16 K. Chandrasekaran and C. Chekuri. Min-max partitioning of hypergraphs and symmetric submodular functions. *Combinatorica*, 43:455–477, 2023. Prelim. version in SODA 2021.

17 K. Chandrasekaran, C. Xu, and X. Yu. Hypergraph k-Cut in randomized polynomial time. *Mathematical Programming*, 186:85–113, 2021. Prelim. version in SODA 2018.

18 C. Chekuri and C. Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 47(6):2118–2156, 2018.

19 Y. Chen, S. Khanna, and A. Nagda. Near-linear size hypergraph cut sparsifiers. In *IEEE 61st Annual Symposium on Foundations of Computer Science*, FOCS, pages 61–72, 2020.

20 B. Cosh, B. Jackson, and Z. Király. Local edge-connectivity augmentation in hypergraphs is NP-complete. *Discrete Applied Mathematics*, 158(6):723–727, 2010.

21 Q. Dai and Y. Gao. *Hypergraph Modeling*, pages 49–71. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer Nature, Singapore, 2023. `doi:10.1007/978-981-99-0185-2_4`.

22 S. Feng, E. Heath, B. Jefferson, C. Joslyn, H. Kvinge, H. D. Mitchell, B. Praggastis, A. J. Eisfeld, A. C. Sims, L. B. Thackray, S. Fan, K. B. Walters, P. J. Halfmann, D. Westhoff-Smith, Q. Tan, V. D. Menachery, T. P. Sheahan, A. S. Cockrell, J. F. Kocher, K. G. Stratton, N. C. Heller, L. M. Bramer, M. S. Diamond, R. S. Baric, K. M. Waters, Y. Kawaoka, J. E. McDermott, and E. Purvine. Hypergraph models of biological networks to identify genes critical to pathogenic viral response. *BMC Bioinformatics*, 22(1):287, May 2021. doi:10.1186/s12859-021-04197-2.

23 Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao. Hypergraph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3558–3565, July 2019. doi:10.1609/aaai.v33i01.33013558.

24 K. Fox, D. Panigrahi, and F. Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contractions. *ACM Trans. Algorithms*, 19(2), 2023. Prelim. version in SODA 2019.

25 A. Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992.

26 A Frank. Connectivity augmentation problems in network design. *Mathematical Programming: State of the Art 1994*, pages 34–63, 1994.

27 H. N. Gabow. Efficient splitting off algorithms for graphs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC, pages 696–705, 1994.

28 M. Ghaffari, D. Karger, and D. Panigrahi. Random Contractions and Sampling for Hypergraph and Hedge Connectivity. In *Proceedings of the 28th annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1101–1114, 2017.

29 F. Grandoni, A. J. Ameli, and V. Traub. Breaching the 2-approximation barrier for the forest augmentation problem. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1598–1611, 2022.

30 Z. Guo, R. Li, C. Shangguan, I. Tamo, and M. Wootters. Improved List-Decodability and List-Recoverability of Reed-Solomon Codes via Tree Packings. In *IEEE 62nd Annual Symposium on Foundations of Computer Science*, FOCS, pages 708–719, 2022.

31 M. M. Halldórsson, T. Ishii, K. Makino, and K. Takazawa. Posimodular function optimization. *Algorithmica*, 84(4):1107–1131, 2022.

32 A. Jambulapati, Y. P. Liu, and A. Sidford. Chaining, group leverage score overestimates, and fast spectral hypergraph sparsification. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC, pages 196–206, 2023.

33 M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Towards tight bounds for spectral sparsification of hypergraphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC, pages 598–611, 2021.

34 M. Kapralov, R. Krauthgamer, J. Tardos, and Y. Yoshida. Spectral Hypergraph Sparsifiers of Nearly Linear Size. In *IEEE 62nd Annual Symposium on Foundations of Computer Science*, FOCS, pages 1159–1170, 2022.

35 S. Klamt, U.-U. Haus, and F. Theis. Hypergraphs and cellular networks. *PLoS Computational Biology*, 5(5):e1000385, May 2009. doi:10.1371/journal.pcbi.1000385.

36 D. Kogan and R. Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS, pages 367–376, 2015.

37 L. C. Lau and C. K. Yung. Efficient Edge Splitting-Off Algorithms Maintaining All-Pairs Edge-Connectivities. *SIAM Journal on Computing*, 42(3):1185–1200, 2013.

38 J. R. Lee. Spectral hypergraph sparsification via chaining. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC, pages 207–218, 2023.

39 P. Li and O. Milenkovic. Inhomogeneous hypergraph clustering with applications. *Advances in neural information processing systems*, 30, 2017.

40 H. Miwa and H. Ito. NA-edge-connectivity augmentation problems by adding edges. *J. Oper. Res. Soc Japan*, 47:224–243, 2004.

41 D. Naor, D. Gusfield, and C. Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997.

42 S. Ornes. How big data carried graph theory into new dimensions. *Quanta Magazine*, 2021. URL: `https://www.quantamagazine.org/how-big-data-carried-graph-theory-into-new-dimensions-20210819/`.

43 K. Quanrud. Quotient sparsification for submodular functions. Manuscript available at `kentquanrud.com`, November 2022.

44 S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders. High-quality hypergraph partitioning. *ACM Journal of Experimental Algorithmics*, 27:1–39, 2023.

45 T. Soma and Y. Yoshida. Spectral sparsification of hypergraphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 2570–2581, 2019.

46 Z. Szigeti. Hypergraph connectivity augmentation. *Math. Program.*, 84(3):519–527, 1999.

47 V. Traub and R. Zenklusen. A better-than-2 approximation for weighted tree augmentation. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–12, 2022.

48 V. Traub and R. Zenklusen. Local search for weighted tree augmentation and steiner tree. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3253–3272, 2022.

49 V. Traub and R. Zenklusen. A $(1.5+\epsilon)$-Approximation Algorithm for Weighted Connectivity Augmentation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1820–1833, 2023.

50 N. Veldt, A. R. Benson, and J. Kleinberg. Hypergraph cuts with general splitting functions. *SIAM Review*, 64(3):650–685, 2022. `doi:10.1137/20M1321048`.

51 Y. Wang and J. Kleinberg. From graphs to hypergraphs: Hypergraph projection and its remediation. In *(To appear) The Twelfth International Conference on Learning Representations*, 2024. URL: `https://openreview.net/forum?id=qwYKE3VB2h`.

52 T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35(1):96–144, 1987.

53 J.-G. Young, G. Petri, and T. P. Peixoto. Hypergraph reconstruction from network data. *Communications Physics*, 4(11):1–11, June 2021. `doi:10.1038/s42005-021-00637-w`.