Journal of Machine Learning Research 24 (2023) 1-50 Submitted 3/22; Published 5/23

Augmented Sparsifiers for Generalized Hypergraph Cuts*

Nate Veldt NVELDT@TAMU.EDU
Department of Computer Science and Engineering
Texas A&M University
College Station, TX, USA

Austin R. Benson ARB@CS.CORNELL.EDU
Department of Computer Science
Cornell University
Ithaca, NY, USA

Jon Kleinberg KLEINBERG@CORNELL.EDU
Department of Computer Science
Cornell University
Ithaca, NY, USA

Editor: Tina Eliassi-Rad

Abstract

Hypergraph generalizations of many graph cut problems and algorithms have recently been introduced to better model data and systems characterized by multiway relationships. Recent work in machine learning and theoretical computer science uses a generalized cut function for a hypergraph $\mathcal{H} = (V, \mathcal{E})$ that associates each hyperedge $e \in \mathcal{E}$ with a splitting function $\mathbf{w}_e$, which assigns a penalty to each way of separating the nodes of e. When each $\mathbf{w}_e$ satisfies $\mathbf{w}_e(S) = g(|S|)$ for some concave function g, previous work has shown how to reduce the generalized hypergraph cut problem to a directed graph cut problem, although the resulting graph may be very dense. We introduce a framework for sparsifying hypergraph-to-graph reductions, where the hypergraph cut function is $(1+\varepsilon)$-approximated by a cut on a directed graph. For $\varepsilon > 0$ we need at most $O(\varepsilon^{-1}|e| \log |e|)$ edges to reduce any hyperedge e, while only $O(|e|\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ edges are needed to approximate the clique expansion, a widely used heuristic in hypergraph clustering. Our framework leads to improved results for solving cut problems in co-occurrence graphs, decomposable submodular function minimization problems, and localized hypergraph clustering problems.

Keywords: hypergraphs; sparsification; decomposable submodular function minimization; piecewise linear functions; concave function approximation; co-occurrence graphs

1. Introduction

Hypergraphs are a generalization of graphs in which nodes are organized into multiway relationships called hyperedges. Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$ and a set of nodes $S \subseteq V$, a hyperedge $e \in \mathcal{E}$ is said to be *cut* by S if both S and $\bar{S} = V \backslash S$ contain at least one node from e. Developing efficient algorithms for cut problems in hypergraphs is an active

*. This manuscript is an adaptation and extension of a previous conference paper on *Approximate Decomposable Submodular Function Minimization for Cardinality-Based Components*, published at NeurIPS 2021 (Veldt et al., 2021).

area of research in machine learning and theoretical computer science (Chekuri and Xu, 2017, 2018; Kogan and Krauthgamer, 2015; Chandrasekaran et al., 2018), and has been applied to problems in VLSI layout (Alpert and Kahng, 1995; Hadley, 1995; Karypis et al., 1999), sparse matrix partitioning (Akbudak et al., 2013; Ballard et al., 2016), and various clustering and classification tasks (Li and Milenkovic, 2017, 2018b; Veldt et al., 2020a).

Here, we consider recently introduced *generalized* hypergraph cut functions (Li and Milenkovic, 2017, 2018b; Veldt et al., 2022; Yoshida, 2019), which assign different penalties to cut hyperedges based on how the nodes of a hyperedge are split into different sides of the bipartition induced by S. To define a generalized hypergraph cut function, each hyperedge $e \in \mathcal{E}$ is first associated with a *splitting function* $\mathbf{w}_e \colon A \subseteq e \to \mathbb{R}^+$ that maps each node configuration of e (defined by the subset $A = e \cap S$) to a nonnegative penalty. In order to mirror edge cut penalties in graphs, splitting functions are typically assumed to be symmetric ($\mathbf{w}_e(A) = \mathbf{w}_e(e \backslash A)$) and only penalize cut hyperedges (i.e., $\mathbf{w}_e(\emptyset) = 0$). The generalized hypergraph cut function for a set $S \subseteq V$ is then given by

$$\mathbf{cut}_{\mathcal{H}}(S) = \sum_{e \in \mathcal{E}} \mathbf{w}_e(S \cap e)\,. \tag{1}$$

The standard hypergraph cut function is *all-or-nothing*, meaning it assigns the same penalty to a cut hyperedge regardless of how its nodes are separated. Using the splitting function terminology, this means that $\mathbf{w}_e(A) = 0$ if $A \in \{e, \emptyset\}$, and $\mathbf{w}_e(A) = w_e$ otherwise, where w_e is a scalar hyperedge weight. One particularly relevant class of splitting functions are submodular functions, which for all A,$B \subseteq e$ satisfy $\mathbf{w}_e(A) + \mathbf{w}_e(B) \geq \mathbf{w}_e(A \cap B) + \mathbf{w}_e(A \cup B)$. When all hyperedge splitting functions are submodular, solving generalized hypergraph cut problems is closely related to minimizing a decomposable submodular function (Kolmogorov, 2012; Nishihara et al., 2014; Li and Milenkovic, 2018a; Stobbe and Krause, 2010; Ene et al., 2017; Ene and Nguyen, 2015), which in turn is closely related to energy minimization problems often encountered in computer vision (Kohli et al., 2009; Kolmogorov and Zabin, 2004; Freedman and Drineas, 2005). The standard graph cut function is another well-known submodular special case of the cut function given by (1).

One of the most common techniques for solving hypergraph cut problems is to reduce the hypergraph to a graph sharing similar (or in some cases identical) cut properties. Arguably the most widely used reduction technique is clique expansion, which replaces each hyperedge with a (possibly weighted) clique (Benson et al., 2016; Hadley, 1995; Li and Milenkovic, 2017; Zien et al., 1999; Zhou et al., 2006). In the unweighted case this corresponds to applying a splitting function of the form: $\mathbf{w}_e(A) = |A| \cdot |e \backslash A|$. Previous work has also explored other classes of submodular hypergraph cut functions that can be modeled as a graph cut problem on a potentially *augmented* node set (Freedman and Drineas, 2005; Kohli et al., 2009; Kolmogorov and Zabin, 2004; Lawler, 1973; Veldt et al., 2022). This previous research primarily focuses on proving when such a reduction is possible, regardless of the number of edges and auxiliary nodes needed to realize the reduction. However, because hyperedges can be very large and splitting functions may be very general and intricate, many of these techniques lead to large and dense graphs. Therefore, the reduction strategy significantly affects the runtime and practicality of algorithms that run on the reduced graph. This leads to several natural questions. Are the graph sizes resulting from existing techniques inherently necessary for modeling hypergraph cuts? Given a class of functions

that are known to be graph reducible, can one determine more efficient or even the *most* efficient reduction techniques? Finally, is it possible to obtain more efficient reductions and faster algorithms if it is sufficient to just *approximately* model cut penalties?

To answer these questions, we present a novel framework for sparsifying hypergraph-to-graph reductions with provable guarantees on preserving cut properties. Our framework brings together concepts and techniques from several different theoretical domains, including algorithms for solving generalized hypergraph cut problems (Veldt et al., 2022; Li and Milenkovic, 2018b, 2017; Yoshida, 2019), standard graph sparsification techniques (Soma and Yoshida, 2019; Spielman and Teng, 2014; Batson et al., 2014), and tools for approximating functions with piecewise linear curves (Magnanti and Stratila, 2012, 2004). We present sparsification techniques for a large and natural class of submodular splitting functions that are *cardinality-based*, meaning that $\mathbf{w}_e(A) = \mathbf{w}_e(B)$ whenever $|A| = |B|$. These are known to always be graph reducible, and are particularly natural for several downstream applications (Veldt et al., 2022). Our approach leads to graph reductions that are significantly more sparse than previous approaches, and we show that our method is in fact optimally sparse under a certain type of reduction strategy. Our sparsification framework can be directly used to develop faster algorithms for approximately solving hypergraph s-t cut problems (Veldt et al., 2022), and improve runtimes for a large class of cardinality-based decomposable submodular minimization problems (Kohli et al., 2009; Kolmogorov, 2012; Jegelka et al., 2013; Stobbe and Krause, 2010). We also show how our techniques enable us to develop efficient sparsifiers for graphs constructed from co-occurrence data.

1.1 Graph and Hypergraph Sparsification

Our framework and results share numerous connections with existing work on graph sparsification, which we review here. Let $G = (V, E)$ be a graph with a cut function $\mathbf{cut}_G$, which can be viewed as a very restricted case of the generalized hypergraph cut function in Eq. (1). An ε-cut sparsifier for G is a sparse weighted and undirected graph $H = (V, F)$ with cut function $\mathbf{cut}_H$, such that

$$\mathbf{cut}_G(S) \leq \mathbf{cut}_H(S) \leq (1+\varepsilon)\mathbf{cut}_G(S), \tag{2}$$

for every subset $S \subseteq V$. This definition was introduced by Benczúr and Karger (1996), who showed how to obtain a sparsifier with $O(n \log n/\varepsilon^2)$ edges for any graph in $O(m \log^3 n)$ time for an n-node, m-edge graph. The more general notion of *spectral* sparsification, which approximately preserves the Laplacian quadratic form of a graph rather than just the cut function, was later introduced by Spielman and Teng (2011). The best cut and spectral sparsifiers have $O(n/\varepsilon^2)$ edges, which is known to be asymptotically optimal for both spectral and cut sparsifiers (Andoni et al., 2016; Batson et al., 2014). Although studied much less extensively, analogous definitions of cut (Chekuri and Xu, 2018; Kogan and Krauthgamer, 2015) and spectral (Soma and Yoshida, 2019) sparsifiers for hypergraphs have also been developed. However, these apply exclusively to the all-or-nothing cut penalty, and do not preserve generalized cut functions of the form shown in (1). Bansal et al. (2019) also considered a weaker notion of graph and hypergraph sparsification, involving additive approximation terms, but we only consider multiplicative approximations.

1.2 The Present Work: Augmented Sparsifiers for Hypergraph Reductions

In this paper, we introduce an alternative notion of an *augmented* cut sparsifier. We present our results in the context of hypergraph-to-graph reductions, though our framework also provides a new notion of augmented sparsifiers for graphs. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with a generalized cut function $\mathbf{cut}_{\mathcal{H}}$, and let $\hat{G} = (V \cup \mathcal{A}, \hat{E})$ be a directed graph on an augmented node set $V \cup \mathcal{A}$. The graph is equipped with an augmented cut function defined for every $S \subseteq V$ by

$$\mathbf{cut}_{\hat{G}}(S) = \min_{T \subseteq \mathcal{A}} \mathbf{dircut}_{\hat{G}}(S \cup T), \tag{3}$$

where $\mathbf{dircut}_{\hat{G}}$ is the standard *directed* cut function on $\hat{G}$. We say that $\hat{G}$ is an ε-*augmented cut sparsifier* for $\mathcal{H}$ if it is sparse and satisfies

$$\mathbf{cut}_{\mathcal{H}}(S) \leq \mathbf{cut}_{\hat{G}}(S) \leq (1+\varepsilon)\mathbf{cut}_{\mathcal{H}}(S). \tag{4}$$

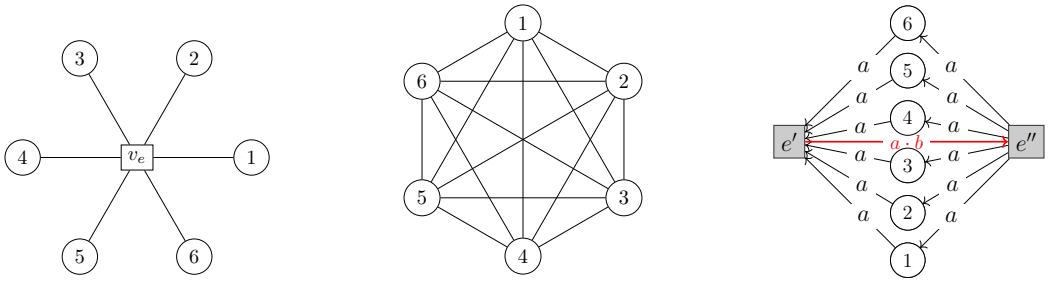
The minimization involved in (3) is especially natural when the goal is to approximate a minimum cut or minimum s-t cut in $\mathcal{H}$. If we solve the corresponding cut problem in $\hat{G}$, nodes from the *auxiliary* node set $\mathcal{A}$ will be automatically arranged in a way that yields the minimum directed cut penalty, as required in (3). If $\hat{S}^*$ is the minimum cut in $\hat{G}$, $S^* = V \cap \hat{S}^*$ will be a $(1+\varepsilon)$-approximate minimum cut in $\mathcal{H}$. Even when solving a minimum cut problem is not the goal, our sparsifiers will be designed in such a way that the augmented cut function in (3) will be easy to evaluate.

Unlike the standard graph sparsification problem, in some cases it may in fact be impossible to find *any* directed graph $\hat{G}$ satisfying the property in (4), independent of the graph's density. In recent work we showed that hypergraphs with non-submodular splitting functions are never graph reducible (Veldt et al., 2022). Although they used different notation and terminology, the work of Živný et al. (2009) implies that even in the case of four-node hyperedges, there exist *submodular* splitting functions (albeit *asymmetric* splitting functions) that are not representable by graph cuts. Nevertheless, there are several special cases in which graph reduction is possible (Kolmogorov and Zabin, 2004; Freedman and Drineas, 2005; Kohli et al., 2009).

Augmented sparsifiers for cardinality-based hypergraph cuts. We specifically consider the class of submodular splitting functions that are cardinality-based, meaning they satisfy $\mathbf{w}_e(A) = \mathbf{w}_e(B)$ whenever $A, B \subseteq e$ satisfy $|A| = |B|$. These are known to be graph reducible (Kohli et al., 2009; Veldt et al., 2022), though existing techniques will reduce a hypergraph $\mathcal{H} = (V, \mathcal{E})$ to a graph with $O(|V| + \sum_{e \in \mathcal{E}} |e|)$ nodes and $O(\sum_{e \in \mathcal{E}} |e|^2)$ edges. We prove the following *sparse* reduction result.

Theorem 1 *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph where each $e \in \mathcal{E}$ is associated with a cardinality-based submodular splitting function. There exists an augmented cut sparsifier $\hat{G}$ for $\mathcal{H}$ with $O(|V| + \frac{1}{\varepsilon}\sum_{e \in \mathcal{E}} \log |e|)$ nodes and $O(\frac{1}{\varepsilon}\sum_{e \in E} |e| \log |e|)$ edges.*

For certain types of splitting functions (e.g., the one corresponding to a clique expansion), we show that our reductions are even sparser.



(a) Star gadget (b) Clique gadget (c) CB-gadget

Figure 1: Three gadgets, each modeling a different hyperedge splitting function.

Augmented sparsifiers for graphs. Another relevant class of augmented sparsifiers to consider is the setting where $\mathcal{H}$ is simply a graph. In this case, if $\mathcal{A}$ is empty and all edges are undirected, condition (4) reduces to the standard definition of a cut sparsifier. A natural question is whether there exist cases where allowing auxiliary nodes and directed edges leads to improved sparsifiers. We show that the answer is yes in the case of dense graphs constructed from co-occurrence data.

Augmented spectral sparsifiers. Just as spectral sparsifiers generalize cut sparsifiers in the standard graph setting, one can define an analogous notion of an augmented *spectral* sparsifier for hypergraph reductions. This can be accomplished using existing hypergraph generalizations of the Laplacian operator (Li and Milenkovic, 2018b; Yoshida, 2019; Chan and Liang, 2018; Louis, 2015). However, although developing augmented spectral sparsifiers constitutes an interesting open direction for future research, it is unclear whether the techniques we develop here can be used or adapted to spectrally approximate generalized hypergraph cut functions. We include further discussion on hypergraph Laplacians and spectral sparsifiers in Section 8, and pose questions for future work. Our primary focus in this manuscript is to develop techniques for augmented *cut* sparsifiers.

1.3 Our Approach: Cut Gadgets and Piecewise Linear Functions

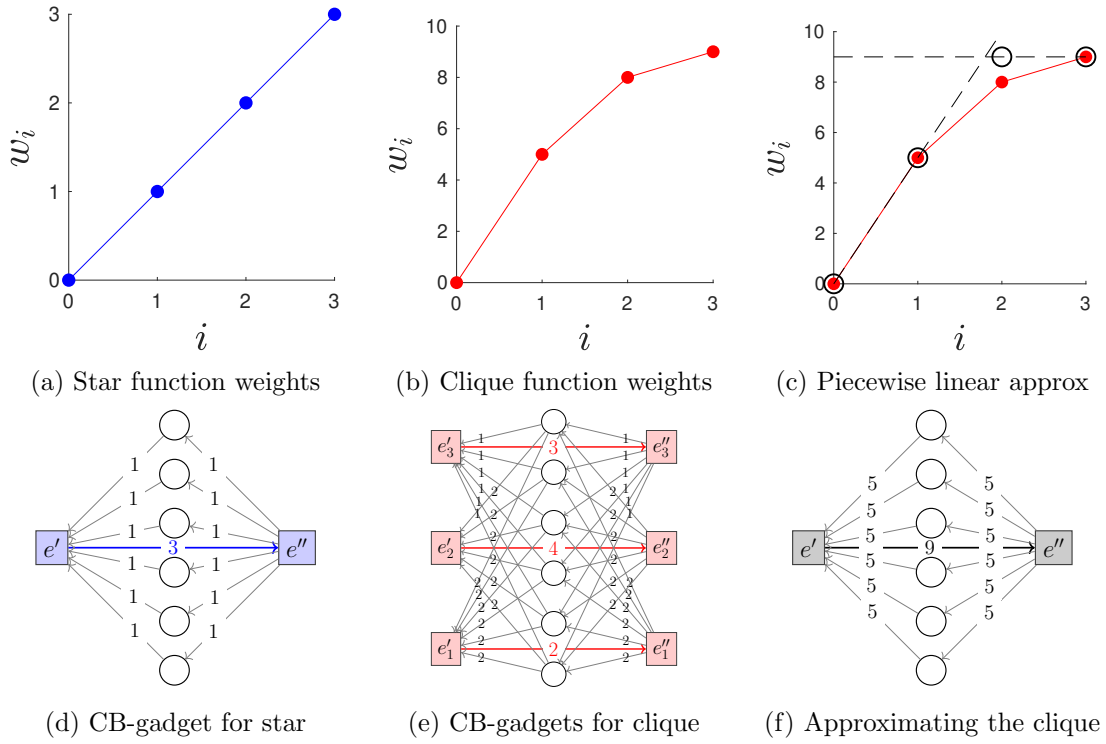
Graph reduction techniques work by replacing a hyperedge with a small graph *gadget* modeling the same cut properties as the hyperedge splitting function. The simplest example of a graph-reducible function is the quadratic splitting function, which we also refer to as the *clique* splitting function:

$$\mathbf{w}_e(S) = |A| \cdot |e \backslash A|, \text{ for } A \subseteq e. \tag{5}$$

This function can be modeled by replacing a hyperedge with a clique (Figure 1b). Another function that can be modeled by a gadget is the linear penalty, which can be modeled by a star gadget (Zien et al., 1999):

$$\mathbf{w}_e(S) = \min\{|A|, |e \backslash A|\}, \text{ for } A \subseteq e. \tag{6}$$

A star gadget (Figure 1a) contains an auxiliary node v_e for each $e \in \mathcal{E}$, which is attached to each $v \in e$ with an undirected edge. In order to model the broader class of submodular



(a) Star function weights (b) Clique function weights (c) Piecewise linear approx

(d) CB-gadget for star (e) CB-gadgets for clique (f) Approximating the clique

Figure 2: (a) The linear splitting function in (6) can be modeled by a sparse gadget (d). (b) The quadratic splitting function in (5) is modeled by a denser gadget (e). (c) A piecewise linear approximation for the quadratic splitting penalties can be modeled by a sparse gadget (f).

cardinality-based splitting functions, we previously introduced the cardinality-based gadget (CB-gadget, Figure 1c) (Veldt et al., 2022). This gadget is parameterized by positive scalars a and b, and includes two auxiliary nodes e' and e''. For each node $v \in e$, there is a directed edge from v to e' and a directed edge from e'' to v, both of weight a. Lastly, there is a directed edge from e' to e'' of weight $a \cdot b$. This CB-gadget corresponds to the following splitting function:

$$\mathbf{w}_e(A) = a \cdot \min\{|A|, |e \backslash A|, b\}. \tag{7}$$

Every submodular, cardinality-based (SCB) splitting function can be modeled by a *combination* of CB-gadgets with different edge weights (Veldt et al., 2022). A different reduction strategy for minimizing submodular energy functions with cardinality-based penalties was also previously developed by Kohli et al. (2009). Both techniques require up to $O(k^2)$ directed edges for a k-node hyperedge.

Sparse combinations of CB-gadgets. Our work introduces a new framework for *approximately* modeling submodular cardinality-based (SCB) splitting functions using a small combinations of CB-gadgets. Figure 2 illustrates our sparsification strategy. We first associate an SCB splitting function with a set of points $\{(i, w_i)\}$, where i represents the number

of nodes on the "small side" of a cut hyperedge, and w_i is the penalty for such a split. We show that when many of these points are collinear, they can be modeled with a smaller number of CB-gadgets. As an example, the star expansion penalties in (6) can be modeled with a single CB-gadget (Figures 2a and 2d), whereas modeling the quadratic penalty in (5) with previous techniques (Veldt et al., 2022) requires many more (Figures 2b and 2e). Given this observation, we design new techniques for ε-approximating the set of points $\{(i, w_i)\}$ with a piecewise linear curve using a small number linear pieces. We then show how to translate the resulting piecewise linear curve back into a smaller combination of CB-gadgets that ε-approximates the original splitting function. Our piecewise linear approximation strategy allows us to find the optimal (i.e., minimum-sized) graph reduction in terms of CB-gadgets. When $\varepsilon = 0$, our approach finds the best way to *exactly* model an SCB splitting function, and requires roughly half the number of gadgets needed by our previous technique (Veldt et al., 2022).[1] More importantly, for larger ε, we prove a sparse approximation result, which says that any submodular cardinality-based splitting function on a k-node hyperedge can be ε-modeled by combining $O(\min\{\frac{1}{\varepsilon}\log k, k\})$ CB-gadgets (Theorem 15). This directly implies the result of Theorem 1. We show that a nearly matching lower bound of $O(\frac{1}{\sqrt{\varepsilon}}\log k)$ CB-gadgets is required for modeling a square root splitting function (Theorem 17). Despite worst case bounds, we prove that only $O(\varepsilon^{-1/2}\log\log\frac{1}{\varepsilon})$ CB-gadgets are needed to approximate the quadratic splitting function, independent of hyperedge size (Theorem 19). This is particularly relevant for approximating the widely used clique expansion technique, as well as for modeling certain types of dense co-occurrence graphs. All of our sparse reduction techniques are combinatorial, deterministic, and very simple to use in practice.

1.4 Augmented Sparsifiers for Co-occurrence Graphs

When $\mathcal{H}$ is just a graph, augmented sparsifiers correspond to a generalization of standard cut sparsifiers that allow directed edges and auxiliary nodes. The auxiliary nodes in this case play a role analogous to Steiner nodes in finding minimum spanning trees. Just as adding Steiner nodes makes it possible to find a smaller weight spanning tree, it is natural to ask whether including an auxiliary node set might lead to better cut sparsifiers for a graph G. We show that the answer is yes for certain classes of dense *co-occurrence* graphs, which are graphs constructed by inducing a clique on a set of nodes that share a certain property or participate in a certain type of group interaction (equivalently, clique expansions of hypergraphs). Steiner nodes have in fact been previously used in constructing certain types of sparsifiers called *vertex* and *flow* sparsifiers (Chuzhoy, 2012). However, these are concerned with preserving certain routing properties between distinguished terminal nodes in a graph, and are therefore distinct from our goal of obtaining ε-cut sparsifiers.

Sparsifying the complete graph. Our ability to sparsify the clique splitting function in (5) implies a new approach for sparsifying a complete graph. Cut sparsifiers for the com-

1. Our previous technique used a set of CB-gadgets that correspond to a piecewise linear function which, in the worst case, has a breakpoint at each of the points (i, w_i), as in Figures 2b. This leads to a simple proof of graph reducibility, but an exact gadget reduction with around half the edges can be obtained by choosing *every other* linear piece from that function. For example, for the function in Figure 2b, we could select the first and third linear piece and we would still have a piecewise linear function that crosses through (i, w_i) for $i \in \{0, 1, 2, 3\}$, but has one breakpoint somewhere between $i = 1$ and $i = 2$.

plete graph provide a simple case study for understanding the differences in sparsification guarantees that can be obtained when we allow auxiliary nodes and directed edges. Furthermore, better sparsifiers for the complete graph can be used to design useful sparsifiers for co-occurrence graphs. We have the following result.

Theorem 2 *Let $G = (V, E)$ be the complete graph on $n = |V|$ nodes. There exists an ε-augmented sparsifier for G with $O(n)$ nodes and $O(n\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ edges.*

By comparison, the best standard cut and spectral sparsifiers for the complete graph have exactly n nodes and $O(n/\varepsilon^2)$ edges. This is tight for spectral sparsifiers (Batson et al., 2014), as well as for degree-regular cut sparsifiers with uniform edge weights (Alon, 1997). Thus, by adding a small number of auxiliary nodes, our sparsifiers enable us to obtain a significantly better dependence on ε when cut-sparsifying a complete graph. Our sparsifier is easily constructed deterministically in $O(n\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ time.

Standard *undirected* sparsifiers for the complete graph have received significant attention as they correspond to expander graphs (Lubotzky, 1988; Alon, 1997; Batson et al., 2014). We remark that the directed augmented cut sparsifiers we produce are very different in nature and should not be viewed as expanders. In particular, unlike for expander graphs, random walks on our complete graph sparsifiers will converge to a very non-uniform distribution. We are interested in augmented sparsifiers for the complete graph simply for their ability to model cut properties in a different way, and the implications this has for sparsifying hypergraph clique expansions and co-occurrence graphs.

Sparsifying co-occurrence graphs. Co-occurrence relationships are inherent in the construction of many types of graphs. Formally, consider a set of $n = |V|$ nodes that are organized into a set of co-occurrence interactions $\mathcal{C} \subseteq 2^V$. Interaction $c \in \mathcal{C}$ is associated with a weight $w_c > 0$, and an edge between nodes i and j is created with weight $w_{ij} = \sum_{c \in \mathcal{C}: i,j \in c} w_c$. When $w_c = 1$ for every $c \in \mathcal{C}$, w_{ij} equals the number of interactions that i and j share. We use d_{avg} to denote the average number of co-occurrence interactions in which nodes in V participate. The cut value in the resulting graph $G = (V, E)$ for a set $S \subseteq V$ is given by the following *co-occurrence cut function*:

$$\mathbf{cut}_G(S) = \sum_{c \in \mathcal{C}} w_c \cdot |S \cap c| \cdot |\bar{S} \cap c|. \tag{8}$$

Graphs with this co-occurrence cut function arise frequently as clique expansions of a hypergraph (Zhou et al., 2006; Benson et al., 2016; Zien et al., 1999; Hadley, 1995), or as projections of a bipartite graph (Li et al., 2007; Neal, 2014; Newman et al., 2001; Ramasco and Morris, 2006; Zhou et al., 2007; Stefano et al., 2013; Watts and Strogatz, 1998). Even when the underlying data set is not first explicitly modeled as a hypergraph or bipartite graph, many approaches implicitly use this approach to generate a graph from data. When enough group interaction sizes are large, G becomes dense, even if $|\mathcal{C}|$ is small. We can significantly sparsify G by applying an efficient sparsifier to each clique induced by a co-occurrence relationship. Importantly, we can do this without ever explicitly forming G. By applying Theorem 2 as a black-box for clique sparsification, we obtain the following result.

Theorem 3 *Let $G = (V, E)$ be the co-occurrence graph for some $\mathcal{C} \subseteq 2^V$ and let $n = |V|$. For $\varepsilon > 0$, there exists an augmented sparsifier $\hat{G}$ with $O(n + |\mathcal{C}| \cdot f(\varepsilon))$ nodes and*

Method	Runtime
Kolmogorov SF (Kolmogorov, 2012)	$\tilde{O}(\mu^2)$
IBFS Strong (Fix et al., 2013)	$O(n^2\theta_{max}\mu_2)$
IBFS Weak (Fix et al., 2013)	$\tilde{O}(n^2\theta_{max} + n\sum_e \lvert e\rvert^4)$
AP (Nishihara et al., 2014)	$\tilde{O}(nR\theta_{avg}\mu)$
RCDM (Ene and Nguyen, 2015)	$\tilde{O}(n^2R\theta_{avg})$
ACDM (Ene and Nguyen, 2015)	$\tilde{O}(nR\theta_{avg})$
Axiotis et al. (Axiotis et al., 2021)	$\tilde{O}(\max_e \lvert e\rvert^2 \cdot \left(\sum_e \lvert e\rvert^2\theta_e + T_{mf}\left(n, n+\mu_2\right)\right)$
This paper (exact solutions)	$\tilde{O}(T_{mf}(\mu,\mu_2)) = \tilde{O}\left(\mu_2 + \mu^{3/2}\right)$
This paper (approximate solutions)	$\tilde{O}\left(T_{mf}(n+\frac{R}{\varepsilon}, \frac{1}{\varepsilon}\mu)\right) = \tilde{O}\left(\frac{\mu}{\varepsilon} + (n+\frac{R}{\varepsilon})^{3/2}\right)$

Table 1: Runtimes for Card-DFSM for various methods, where $\mu = \sum_e |e|$, and $\mu_2 = \sum_e |e|^2$, and where θ_{max} and θ_{avg} denote certain oracle runtimes. These satisfy $\theta_{max} = \Omega(\max |e|)$, and $\theta_{avg} = \Omega(\frac{1}{R}\sum_e |e|)$. $T_{mf}(N, M)$ is the time to solve a max-flow problem with N nodes and M edges. Listed is the original reference for each method. The runtimes listed for IBFS, RCDM, and ACDM are due to updated analysis by Ene et al. (2017). The best runtime analysis for AP is due to Li and Milenkovic (2018a).

$O(n \cdot d_{avg} \cdot f(\varepsilon))$ edges, where $f(\varepsilon) = \varepsilon^{-1/2}\log\log\frac{1}{\varepsilon}$. In particular, if d_{avg} is constant and for some $\delta > 0$ we have $\sum_{c\in\mathcal{C}} |c|^2 = \Omega(n^{1+\delta})$, then forming G explicitly takes $\Omega(n^{1+\delta})$ time, but an augmented sparsifier for G with $O(nf(\varepsilon))$ nodes and $O(nf(\varepsilon))$ edges can be constructed in $O(nf(\varepsilon))$ time.

Importantly, the average co-occurrence degree d_{avg} is not the same as the average node degree in G, which will typically be much larger. Theorem 3 highlights that in regimes where d_{avg} is a constant, our augmented sparsifiers will have fewer edges than the number needed by standard ε-cut sparsifiers. In Section 5, we consider simple graph models that satisfy these assumptions. We also consider tradeoffs between our augmented sparsifiers and standard sparsification techniques for co-occurrence graphs. Independent of the black-box sparsifier we use, implicitly sparsifying G in this way will often lead to significant runtime improvements over forming G explicitly.

1.5 Approximate Cardinality-based DSFM

Typically in hypergraph cut problems it is natural to assume that splitting functions are symmetric and satisfy $\mathbf{w}_e(\emptyset) = \mathbf{w}_e(e) = 0$. However, we show that our sparse reduction techniques apply even when these assumptions do not hold. This allows us to design fast algorithms for approximately solving certain decomposable submodular function minimization (DSFM) problems. Formally a function $f\colon 2^V \to \mathbb{R}^+$ is a decomposable submodular function if it can be written as

$$f(S) = \sum_{e\in\mathcal{E}} \mathbf{f}_e(S \cap e), \tag{9}$$

where each $\mathbf{f}_e$ is a submodular function defined on a set $e \subseteq V$. Following our previous notation and terminology, we say $\mathbf{f}_e$ is cardinality-based if $\mathbf{f}_e(S) = g_e(|S|)$ for some concave function g_e. This special case, which we refer to as Card-DSFM, has been one of the most widely studied and applied variants since the earliest work on DSFM (Kolmogorov, 2012; Stobbe and Krause, 2010). In terms of theory, previous research has addressed specialized runtimes and solution techniques (Kolmogorov, 2012; Stobbe and Krause, 2010; Kohli et al., 2009). In practice, cardinality-based decomposable submodular functions frequently arise as higher-order energy functions in computer vision (Kohli et al., 2009) and set cover functions (Stobbe and Krause, 2010). Even previous research on algorithms for the more general DSFM problem tends to focus on cardinality-based examples in experimental results (Ene et al., 2017; Stobbe and Krause, 2010; Jegelka et al., 2013; Li and Milenkovic, 2018a).

Existing approaches for minimizing these functions focus largely on finding exact solutions. Using our sparse reduction techniques, we develop the first approximation algorithms for the problem. Let $n = |V|$, $R = |\mathcal{E}|$, and $\mu = \sum_{e \in \mathcal{E}} |e|$. In Appendix B, we show that a result similar to Theorem 1 also holds for more general cardinality-based splitting functions. In Section 6, we combine that result with fast recent s-t cut solvers (van den Brand et al., 2021) to prove the following theorem.

Theorem 4 *Let $\varepsilon > 0$. Any cardinality-based decomposable submodular function can be minimized to within a multiplicative $(1+\varepsilon)$ factor in $\tilde{O}\left(\frac{1}{\varepsilon}\sum_{e \in \mathcal{E}} |e| + (n + \frac{R}{\varepsilon})^{3/2}\right)$ time.*

Table 1 compares this runtime against the best previous techniques for Card-DSFM. Our techniques enable us to highlight regimes of the problem where we can obtain significantly faster algorithms in cases where it is sufficient to solve the problem approximately. For example, whenever $n = \Omega(R)$, our algorithms for finding approximate solutions provide a runtime advantage — often a significant one — over approaches for computing an exact solution. We provide a more extensive theoretical runtime comparison in Section 6. In Section 7, we show that implementations of our methods lead to significantly faster results for benchmark image segmentation tasks and localized hypergraph clustering problems.

2. The Sparse Gadget Approximation Problem

A generalized hypergraph cut function is defined as the sum of its splitting functions. Therefore, if we can design a technique for approximately modeling a single hyperedge with a sparse graph, this in turn provides a method for constructing an augmented sparsifier for the entire hypergraph. We now formalize the problem of approximating a submodular cardinality-based (SCB) splitting function using a combination of cardinality-based (CB) gadgets. We abstract this as the task of approximating a certain class of functions with integer inputs (equivalent to SCB splitting functions), using a small number of simpler functions (equivalent to cut properties of the gadgets). Let $[r] = \{1, 2, \ldots, r\}$.

Definition 5 *An r-SCB integer function is a function $\mathbf{w}\colon \{0\} \cup [r] \to \mathbb{R}^+$ satisfying*

$$\mathbf{w}(0) = 0 \tag{10}$$

$$2\mathbf{w}(j) \geq \mathbf{w}(j-1) + \mathbf{w}(j+1) \text{ for } j = 1, \ldots, r-1 \tag{11}$$

$$0 \leq \mathbf{w}(1) \leq \mathbf{w}(2) \leq \ldots \leq \mathbf{w}(r). \tag{12}$$

We denote the set of r-SCB integer functions by $\mathcal{S}_r$.

The value $\mathbf{w}(i)$ represents the splitting penalty for placing i nodes on the small side of a cut hyperedge. In previous work we showed that the inequalities given in Definition 5 are necessary and sufficient conditions for a cardinality-based splitting function to be submodular (Veldt et al., 2022). The r-SCB integer function for a CB-gadget with edge parameters (a, b) (see Equation 7) is

$$\mathbf{w}_{a,b}(i) = a \cdot \min\{i, b\}\,. \tag{13}$$

Combining J CB-gadgets produces a useful notion of a *combined* r-SCB integer function.

Definition 6 *An r-CCB (Combined Cardinality-Based gadget) function of order J, is an r-SCB integer function $\hat{\mathbf{w}}$ with the form*

$$\hat{\mathbf{w}}(i) = \sum_{j=1}^{J} a_j \cdot \min\{i, b_j\}\,, \text{ for } i \in [r]. \tag{14}$$

where the J-dimensional vectors $\mathbf{a} = (a_j)$ and $\mathbf{b} = (v_j)$ parameterizing $\hat{\mathbf{w}}$ satisfy:

$$b_j > 0, a_j > 0 \text{ for all } j \in [J] \tag{15}$$
$$b_j < b_{j+1} \text{ for } j \in [J-1] \tag{16}$$
$$b_J \leq r. \tag{17}$$

We denote the set of r-CCB functions of order J by $\mathcal{C}_r^J$.

The conditions on the vectors $\mathbf{a}$ and $\mathbf{b}$ come from natural observations about combining CB-gadgets. Condition (15) ensures that we do not consider CB-gadgets where all edge weights are zero. The ordering in condition (16) is for convenience; the fact that b_j values are all distinct implies that we cannot collapse two distinct CB-gadgets into a single CB-gadget with new weights. For condition (17), observe that for any $b_J \geq r$, $\min\{i, b_J\} = i$ for all $i \in [r]$. For a helpful visual, note that the r-SCB function in (13) represents splitting penalties for the CB-gadget in Figure 1c. An r-CCB function corresponds to a combination of CB-gadgets, as in Figures 2c and 2f.

In previous work we showed that any combination of CB-gadgets produces a submodular and cardinality-based splitting function, which is equivalent to stating that $\mathcal{C}_r^J \subseteq \mathcal{S}_r$ for all $J \in \mathbb{N}$ (Veldt et al., 2022). Furthermore, $\mathcal{C}_r^r = \mathcal{S}_r$, since any r-SCB splitting function can be modeled by a combination of r CB-gadgets. Our goal here is to determine how to approximate a function $\mathbf{w} \in \mathcal{S}_r$ with some function $\hat{\mathbf{w}} \in \mathcal{C}_r^J$ where $J \ll r$. This corresponds to modeling an SCB splitting function using a *small* combination of CB-gadgets.

Definition 7 *For a fixed $\mathbf{w} \in \mathcal{S}_r$ and an approximation tolerance parameter $\varepsilon \geq 0$, the Sparse Gadget Approximation Problem (Spa-GAP) is the following optimization problem:*

$$\begin{array}{ll} \text{minimize} & \kappa \\ \textit{subject to} & \mathbf{w} \leq \hat{\mathbf{w}} \leq (1+\varepsilon)\mathbf{w} \\ & \hat{\mathbf{w}} \in \mathcal{C}_r^{\kappa}. \end{array} \tag{18}$$

Upper bounding approximations. Problem (18) specifically optimizes over functions $\hat{\mathbf{w}}$ that upper bound $\mathbf{w}$. This simplifies several aspects of our analysis without any practical

consequence. We could instead fix some $\delta \geq 1$ and optimize over functions $\tilde{\mathbf{w}}$ satisfying $\frac{1}{\delta}\mathbf{w} \leq \tilde{\mathbf{w}} \leq \delta\mathbf{w}$. However, this implies that the function $\hat{\mathbf{w}} = \delta\tilde{\mathbf{w}}$ satisfies $\mathbf{w} \leq \hat{\mathbf{w}} \leq (1+\varepsilon)\mathbf{w}$, with $\varepsilon = \delta^2 - 1$. Thus, the problems are equivalent for the correct choice of δ and ε.

Motivation for optimizing over CB-gadgets. A natural question to ask is whether it would be better to search for a sparsest approximating gadget over a broader classes of gadgets. There are several key reasons why we restrict to combinations of CB-gadgets. First of all, we already know these can model *any* SCB splitting function, and thus they provide a very simple building block with broad modeling capabilities. Furthermore, it is clear how to define an optimally sparse combination of CB-gadgets: since all CB-gadgets for a k-node hyperedge have the same number of auxiliary nodes and directed edges, an optimally sparse reduction is one with a minimum number of CB-gadgets. If we instead wish to optimize over all possible gadgets, it is likely that the *best* reduction technique will depend on the splitting function that we wish to approximate. Furthermore, the optimality of a gadget may not even be well-defined, since one must take into account both the number of auxiliary nodes as well as the number of edges that are introduced, and the tradeoff between the two is not always clear. Finally, as we shall see in the next section, by restricting to CB-gadgets, we are able to draw a useful connection between finding sparse gadgets and approximating piecewise linear curves with a smaller number of linear pieces.

3. Sparsification via Piecewise Linear Approximation

We begin by defining the class of piecewise linear functions in which we are interested.

Definition 8 *For $r \in \mathbb{N}$, $\mathcal{F}_r$ is the class of functions $\mathbf{f}\colon [0,\infty] \longrightarrow \mathbb{R}_+$ such that:*

1. $\mathbf{f}(0) = 0$
2. $\mathbf{f}$ *is a constant for all* $x \geq r$
3. $\mathbf{f}$ *is increasing:* $x_1 \leq x_2 \implies \mathbf{f}(x_1) \leq \mathbf{f}(x_2)$
4. $\mathbf{f}$ *is piecewise linear*
5. $\mathbf{f}$ *is concave (and hence, continuous).*

It will be key to keep track of the number of linear pieces that make up a given function $\mathbf{f} \in \mathcal{F}_r$. Let $\mathcal{L}$ be the set of linear functions with nonnegative slopes and intercept terms:

$$\mathcal{L} = \{g(x) = mx + d \,|\, m, d \in \mathbb{R}^+\}. \tag{19}$$

Every $\mathbf{f} \in \mathcal{F}_r$ can be characterized as the lower envelope of some set $L \subset \mathcal{L}$:

$$\mathbf{f}(x) = \min_{g \in L} g(x), \text{ where } L \subset \mathcal{L}. \tag{20}$$

We use $|L|$ to denote the number of linear pieces of $\mathbf{f}$. In order for equation (20) to properly characterize a function in $\mathcal{F}_r$, it must be constant for all $x \geq r$ (property 2 in Definition 8), and thus L must contain exactly one line of slope zero. The *continuous extension* $\hat{\mathbf{f}}$ of an r-CCB function $\hat{\mathbf{w}}$ parameterized by $(\mathbf{a}, \mathbf{b})$ is defined as

$$\hat{\mathbf{f}}(x) = \sum_{j=1}^{J} a_j \cdot \min\{x, b_j\} \text{ for } x \in [0,\infty]. \tag{21}$$

We prove that continuously extending any r-CCB function always produces a function in $\mathcal{F}_r$. Conversely, every $\mathbf{f} \in \mathcal{F}_r$ is the continuous extension of some r-CCB function. Appendix A provides proofs for these results.

Lemma 9 *Let $\hat{\mathbf{f}}$ be the continuous extension for $\hat{\mathbf{w}}$, shown in (21). This function is in the class $\mathcal{F}_r$, and has exactly J positive sloped linear pieces, and one linear piece of slope zero.*

Lemma 10 *Let $\mathbf{f}$ be a function in $\mathcal{F}_r$ with $J+1$ linear pieces. Let b_i denote the ith breakpoint of $\mathbf{f}$, and m_i denote the slope of the ith linear piece of $\mathbf{f}$. Define vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^J$ where $\mathbf{b}(i) = b_i$ and $\mathbf{a}(i) = a_i = m_i - m_{i+1}$ for $i \in [J]$. If $\hat{\mathbf{w}}$ is the r-CCB function parameterized by vectors $(\mathbf{a}, \mathbf{b})$, then $\mathbf{f}$ is the continuous extension of $\hat{\mathbf{w}}$.*

3.1 The Piecewise Linear Approximation Problem

Let $\mathbf{w} \in \mathcal{S}_r$ be an arbitrary SCB integer function. Lemma 10 implies that if we can find a piecewise linear function $\mathbf{f}$ that approximates $\mathbf{w}$ and has few linear pieces, we can extract from it a CCB function $\hat{\mathbf{w}}$ with a small order J that approximates $\mathbf{w}$. Equivalently, we can find a sparse gadget that approximates an SCB splitting function of interest. Our updated goal is therefore to solve the following piecewise linear approximation problem, for a given $\mathbf{w} \in \mathcal{S}_r$ and $\varepsilon \geq 0$:

$$\begin{array}{ll} \text{minimize}_{L \subset \mathcal{L}} & |L| \\ \text{subject to} & \mathbf{w}(i) \leq \mathbf{f}(i) \leq (1+\varepsilon)\mathbf{w}(i) \text{ for } i \in [r] \\ & \mathbf{f} \in \mathcal{F}_r \\ & \mathbf{f}(x) = \min_{g \in L} g(x) \\ & \text{for each } g \in L,\ g(j) = \mathbf{w}(j) \text{ for some } j \in \{0\} \cup [r]. \end{array} \tag{22}$$

The last constraint ensures that each linear piece $g \in L$ we consider crosses through at least one point $(j, \mathbf{w}(j))$. We can add this constraint without loss of generality; if any linear piece g is strictly greater than $\mathbf{w}$ at all integers, we could obtain an improved approximation by scaling g until it is tangent to $\mathbf{w}$ at some point. This constraint, together with the requirement $\mathbf{f} \in \mathcal{F}_r$, implies that the constant function $g^{(r)}(x) = \mathbf{w}(r)$ is contained in every set of linear functions L that is feasible for problem (22). Since all feasible solutions contain this constant linear piece, our focus is on determining the optimal set of positive-sloped linear pieces needed to approximate $\mathbf{w}$.

Optimal linear covers. Given a fixed $\varepsilon \geq 0$ and $i \in \{0\} \cup [r-1]$, we will say a set $L \subset \mathcal{L}$ is a *linear cover* for a function $\mathbf{w} \in \mathcal{S}_r$ over the range $R = \{i, i+1, \ldots, r\}$, if each $g \in L$ upper bounds $\mathbf{w}$ at all points, and if for each $j \in R$ there exists $g \in L$ such that $g(j) \leq (1+\varepsilon)\mathbf{w}(j)$. The set L is an *optimal* linear cover if it contains the minimum number of positive-sloped linear pieces needed to cover R. Thus, an equivalent way of expressing problem (22) is that we wish to find an optimal linear cover for $\mathbf{w}$ over the interval $\{0\} \cup [r]$. There may be many functions in $\mathcal{F}_r$ solving this problem; our goal is to find any one.

3.2 Properties of Linear Pieces in the Cover

We solve problem (22) by iteratively growing a set of linear functions $L \subset \mathcal{L}$ one function at a time, until all of $\mathbf{w}$ is covered. Let $\mathbf{f}$ be the piecewise linear function we construct from

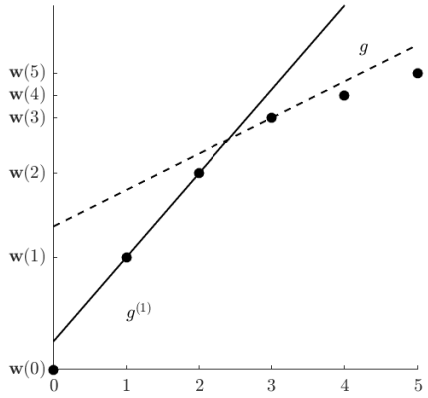


Figure 3: We restrict our attention to lines in $\mathcal{L}$ that coincide with $\mathbf{w}$ at at least one integer value. Thus, every function we consider is incident to two consecutive values of $\mathbf{w}$ (e.g., the solid line, $g^{(1)}$), or, it touches $\mathbf{w}$ at exactly one point (dashed line, g).

linear pieces in L. In order for $\mathbf{f}$ to upper bound $\mathbf{w}$, every function $g \in L$ in problem (22) must upper bound $\mathbf{w}$ at every $i \in \{0\} \cup [r]$. One way to obtain such a linear function is to connect two consecutive points of $\mathbf{w}$. For $i \in \{0\} \cup [r-1]$, we denote the line joining points $(i, \mathbf{w}(i))$ and $(i+1, \mathbf{w}(i+1))$ by

$$g^{(i)}(x) = M_i(x - i) + \mathbf{w}(i), \tag{23}$$

where the slope of the line is $M_i = \mathbf{w}(i+1) - \mathbf{w}(i)$. In order for a line to upper bound $\mathbf{w}$ but only pass through a *single* point $(i, \mathbf{w}(i))$ for some $i \in [r-1]$, it must have the form

$$g(x) = m(x - i) + \mathbf{w}(i)\,, \tag{24}$$

where the slope m satisfies $M_i < m < M_{i-1}$. The existence of such a line g is only possible when the points $(i-1, \mathbf{w}(i-1))$, $(i, \mathbf{w}(i))$, and $(i+1, \mathbf{w}(i+1))$ are not collinear. To understand the strict bounds on m, note that if g passes through $(i, \mathbf{w}(i))$ and has slope exactly M_{i-1}, then g is in fact the line $g^{(i-1)}$ and also passes through $(i-1, \mathbf{w}(i-1))$. If g has slope greater than M_{i-1}, then $g(i-1) < \mathbf{w}(i-1)$ and does not upper bound $\mathbf{w}$ everywhere. We can similarly argue that the slope of g must be strictly greater than M_i so that it does not touch or cross below the point $(i+1, \mathbf{w}(i+1))$.

Figure 3 illustrates both types of functions shown in (23) and in (24). The following observation will help in comparing approximation properties of different functions in $\mathcal{L}$.

Observation 11 *For a fixed $\mathbf{w} \in \mathcal{S}_r$, let $g, h \in \mathcal{L}$ both upper bound $\mathbf{w}$ at all integers $i \in \{0\} \cup [r]$, and assume that for some $j \in \{0\} \cup [r]$, $g(j) = h(j) = \mathbf{w}(j)$. If m_g and m_h are the slopes of g and h respectively, and $m_g \geq m_h \geq 0$, then*

- *For every integer $i \in [0, j]$, $\mathbf{w}(i) \leq g(i) \leq h(i)$*

- *For every integer* $i \in [j, r]$, $\mathbf{w}(i) \leq h(i) \leq g(i)$.

In other words, if g and h are both tangent to $\mathbf{w}$ at the same point j, but g has a larger slope than h, then g provides a better approximation for values smaller than j, while h is the better approximation for values larger than j.

3.3 Building an Optimal Linear Cover

First linear piece. Every set L solving problem (22) must include a linear piece that goes through the origin, so that $\mathbf{f}(0) = 0$. We choose $g^{(0)}(x) = (\mathbf{w}(1) - \mathbf{w}(0))x + \mathbf{w}(0) = \mathbf{w}(1)x$ to be the first linear piece in the set L we construct. Given this first linear piece, we can then compute the largest integer $i \in [r]$ for which $g^{(0)}$ provides a $(1+\varepsilon)$-approximation:

$$p = \max\{i \in [r] \mid g^{(0)}(i) \leq (1+\varepsilon)\mathbf{w}(i)\}.$$

The integer $\ell = p + 1$ therefore is the smallest integer for which we *do not* have a $(1+\varepsilon)$-approximation. If $\ell \leq r$, our task is then to find the smallest number of additional linear pieces in order to cover $\{\ell, \ldots, r\}$ with $(1+\varepsilon)$-approximations. By Observation 11, any other $g \in \mathcal{L}$ with $g(0) = 0$ and $g(1) > \mathbf{w}(1)$ will be a worse approximation to $\mathbf{w}$ at all integer values: $\mathbf{w}(i) \leq g^{(0)}(i) < g(i)$ for all $i \in [r]$. Therefore, as long as we can find a minimum set of additional linear pieces which provides a $(1+\varepsilon)$-approximation for all $\{\ell, \ldots, r\}$, our set of functions L will optimally solve problem (22).

Iteratively finding the next linear piece. Consider now a generic setting in which we are given a left integer endpoint ℓ and we wish to find linear pieces to approximate the function $\mathbf{w}$ from ℓ to r. We first check whether the constant function $g^{(r)}(x) = \mathbf{w}(r)$ provides the desired approximation:

$$g^{(r)}(\ell) \leq (1+\varepsilon)\mathbf{w}(\ell). \tag{25}$$

If so, we augment L to include $g^{(r)}$ and we are done, since this implies that $g^{(r)}$ also provides at least a $(1+\varepsilon)$-approximation at every $i \in \{\ell, \ell+1, \ldots, r\}$. If inequality (25) is not true, we must add another positive-sloped linear function to L in order to get the desired approximation for all $i \in [r]$. We adopt a greedy approach that chooses the next line to be the optimizer of the following objective

$$\begin{array}{ll} \max_{g \in \mathcal{L}} & p' \\ \text{subject to} & \mathbf{w}(j) \leq g(j) \leq (1+\varepsilon)\mathbf{w}(j) \text{ for } j = \ell, \ell+1, \ldots, p'. \end{array} \tag{26}$$

In other words, solving problem (26) means finding a function that provides at least a $(1+\varepsilon)$-approximation from ℓ to as far towards r as possible in order to cover the widest possible contiguous interval with the same approximation guarantee. (There is always a feasible point by adding a line g tangent to $\mathbf{w}(\ell)$.) The following Lemma will help us prove that this greedy scheme produces an optimal cover for $\mathbf{w}$.

Lemma 12 *Let* p^* *the solution to problem* (26) *and* g^* *be the function that achieves it. If* $\hat{L} \subset \mathcal{L}$ *is an optimal cover for* $\mathbf{w}$ *over the integer range* $\{p^*+1, p^*+2, \ldots r\}$*, then* $\{g^*\} \cup \hat{L}$ *is an optimal cover for* $\{\ell, \ell+1, \ldots r\}$.

Proof Let $\tilde{L}$ be an arbitrary optimal linear cover for $\mathbf{w}$ over the range $\{\ell, \ell+1, \ldots, r\}$. This means that $|\hat{L} \cup \{g^*\}| \geq |\tilde{L}|$. We know $\tilde{L}$ must contain a function g such that $g(\ell) \leq$

$(1+\varepsilon)\mathbf{w}(\ell)$. Let p_g be the largest integer satisfying $g(p_g) \leq (1+\varepsilon)\mathbf{w}(p_g)$. By the optimality of p^* and g^*, we know $p^* \geq p_g$. Therefore, the set of functions $\tilde{L} - \{g\}$ must be a cover for the set $\{p_g + 1, p_g + 2, \dots r\} \supseteq \{p^* + 1, p^* + 2, \dots r\}$. Since $\hat{L}$ is an optimal cover for a subset of the integers covered by $\tilde{L} - \{g\}$,

$$|\hat{L}| \leq |\tilde{L} - \{g\}| \implies |\hat{L}| + 1 \leq |\tilde{L}| \implies |\hat{L} \cup \{g^*\}| \leq |\tilde{L}|.$$

Therefore, $|\hat{L} \cup \{g^*\}| = |\tilde{L}|$, so the result follows. ■

We illustrate a simple procedure for solving problem (26) in Figure 4. The function g solving this problem must either join two consecutive points of $\mathbf{w}$ (the form given in (23)), or coincide at exactly one point of $\mathbf{w}$ (form given in (24)). We first identify the integer j^* such that

$$g^{(j^*)}(\ell) \leq (1+\varepsilon)\mathbf{w}(\ell) \text{ and } g^{(j^*+1)}(\ell) > (1+\varepsilon)\mathbf{w}(\ell).$$

In other words, the linear piece connecting $(j^*, \mathbf{w}(j^*))$ and $(j^*+1, \mathbf{w}(j^*+1))$ provides the needed approximation at the left endpoint ℓ, but $g^{(i)}$ for every $i > j^*$ does not. Therefore, the solution to problem (26) has a slope $m \in [M_{j^*}, M_{j^*+1})$, and passes through the point $(j^*, \mathbf{w}(j^*))$. By Observation 11, the line passing through this point with the smallest slope is guaranteed to provide the best approximation for all integers $p \geq j^*$. To minimize the slope of the line while still preserving the needed approximation at $\mathbf{w}(\ell)$, we select the line passing through the points $(\ell, (1+\varepsilon)\mathbf{w}(\ell))$ and $(j^*, \mathbf{w}(j^*))$. This is given by

$$g^*(x) = \frac{\mathbf{w}(j^*) - (1+\varepsilon)\mathbf{w}(\ell)}{(j^* - \ell)}(x - \ell) + (1+\varepsilon)\mathbf{w}(\ell). \tag{27}$$

After adding g^* to L, we find the largest integer $p \leq r$ such that $g^*(p) \leq (1+\varepsilon)\mathbf{w}(p)$. If $p < r$, we still need to find more linear pieces to approximate $\mathbf{w}$, so we begin another iteration. If $p = r$ exactly, we do not need any more positive-sloped linear pieces to approximate $\mathbf{w}$. However, we still add the constant function $g^{(r)}$ to L before terminating. This guarantees that the function $\mathbf{f}(x) = \min_{g \in L}(x)$ we return is in $\mathcal{F}_r$. Furthermore, adding $g^{(r)}$ improves the approximation without affecting the order of the CCB function we obtain from $\mathbf{f}$ by applying Lemma 10. Pseudocode for our procedure for constructing a set of function L is given in Algorithm 1, which relies on Algorithm 2 for solving problem (26).

Theorem 13 *Algorithm 1 runs in $O(r)$ time and returns a function $\mathbf{f}$ that solves problem (22).*

Proof The optimality of the algorithm follows by inductively applying Lemma 12 at each iteration of the algorithm. For the runtime guarantee, note first of all that we can compute and store all slopes and intercepts for linear pieces $g^{(i)}$ (as given in (23)) in $O(r)$ time and space. As the algorithm progresses, we visit each integer $i \in [r]$ once, either to perform a comparison of the form $g^{(i)}(\ell) \leq (1+\varepsilon)\mathbf{w}(\ell)$ for some left endpoint ℓ, or to check whether $g^*(i) \leq (1+\varepsilon)\mathbf{w}(i)$ for some linear piece g^* we added to our linear cover L. Each such g^* can be computed in constant time, and as a loose bound we know we compute at most $O(r)$ such linear pieces for any ε. ■

By combining Algorithm 2 and Lemma 10, we are able to efficiently solve Spa-GAP.

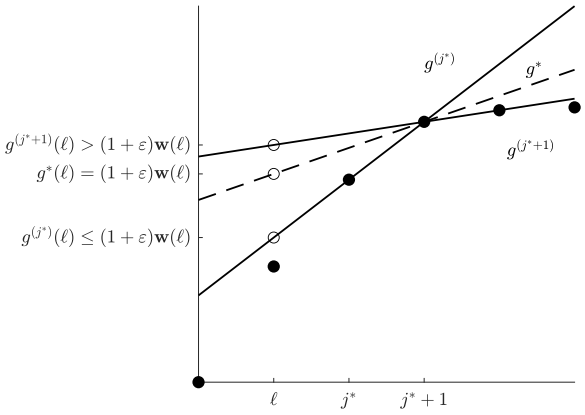


Figure 4: Given a left endpoint ℓ for which we do not have an approximating linear piece, we choose the next linear g^* to provide the desired approximation at ℓ, while also providing a good approximation for the largest possible integer $p > \ell$.

Algorithm 1 FindBest-PL-Approx$(\mathbf{w}, \varepsilon)$ (solves problem (22))

Input: $\mathbf{w} \in \mathcal{S}_r$, $\varepsilon \geq 0$
Output: $\mathbf{f} \in \mathcal{F}_r$ optimizing problem (22)
$L = \{g^{(0)}\}$, where $g^{(0)} = \mathbf{w}(1)x$
$p = \max\{i \in [r] \mid g^{(0)}(i) \leq (1+\varepsilon)\mathbf{w}(i)\}$
$\ell = p + 1$
while $\ell \leq r$ **do**
  $(g^*, p) =$ FindNext$(\mathbf{w}, \varepsilon, \ell)$
  $\ell \leftarrow p + 1$
  $L \leftarrow L \cup \{g^*\}$
  if $p = r$ **then**
    $L \leftarrow L \cup \{g^{(r)}\}$, where $g^{(r)}(x) = \mathbf{w}(r)$
  end if
end while
Return $\mathbf{f}$ defined by $\mathbf{f}(x) = \min_{g \in L} g(x)$

Theorem 14 *Let* $\mathbf{f}$ *be the solution to problem* (22)*, and* $\hat{\mathbf{w}}$ *be the CCB function obtained from Lemma 10 based on* $\mathbf{f}$*. Then* $\hat{\mathbf{w}}$ *optimally solves* Spa-GAP *(problem* (18)*).*

Proof Since $\mathbf{f}$ and $\hat{\mathbf{w}}$ coincide at integer values, and $\mathbf{f}$ approximates $\mathbf{w}$ at integer values, we know $\mathbf{w}(i) \leq \hat{\mathbf{w}}(i) \leq (1+\varepsilon)\mathbf{w}(i)$ for $i \in [r]$. Thus, $\hat{\mathbf{w}}$ is feasible for problem (18). If κ^* is the number of positive-sloped linear pieces of $\mathbf{f}$, then the order of $\hat{\mathbf{w}}$ is κ^* by Lemma 10, and this must be optimal for problem (18). Otherwise, there would exist some upper bounding CCB function $\mathbf{w}'$ of order $\kappa' < \kappa^*$ that approximates $\mathbf{w}$ to within $1 + \varepsilon$. But by Lemma 9, this would imply that the continuous extension of $\mathbf{w}'$ is some $\mathbf{f}' \in \mathcal{F}_r$ with exactly κ' positive-sloped linear pieces that is feasible for problem (22), contradicting the optimality of $\mathbf{f}$. ■

Algorithm 2 FindNext$(\mathbf{w}, \varepsilon, \ell)$ (solves problem (26))

Input: $\mathbf{w} \in \mathcal{S}_r$, $\varepsilon \geq 0$, $\ell \in [r]$
Output: $g \in \mathcal{L}$ optimizing problem (26)
if $\mathbf{w}(r) \leq (1+\varepsilon)\mathbf{w}(\ell)$ **then**
  Return $(g^{(r)}, r+1)$, where $g^{(r)}(x) = \mathbf{w}(r)$
else
  $j^* = \ell$
  while $g^{(j^*+1)}(\ell) \leq (1+\varepsilon)\mathbf{w}(\ell)$ **do**
    $j^* = j^* + 1$
  end while
  $g^*(x) = \frac{\mathbf{w}(j^*) - (1+\varepsilon)\mathbf{w}(\ell)}{(j^* - \ell)}(x - \ell) + (1+\varepsilon)\mathbf{w}(\ell)$
  $p = \max\{i \in [r] \mid g^*(p) \leq (1+\varepsilon)\mathbf{w}(p)\}$
  Return (g^*, p)
end if

4. Bounding the Size of the Optimal Reduction

The previous section showed an efficient strategy for finding the minimum number of linear pieces needed to approximate an SCB integer function. We now bound the number of linear pieces needed in different cases, and highlight implications for sparsifying hyperedges with SCB splitting functions. In the worst case, we need $O(\frac{1}{\varepsilon}\log k)$ gadgets, where k is the hyperedge size, and this is nearly tight for the square root splitting function. Finally, we show that $O(\varepsilon^{-1/2}\log\log\frac{1}{\varepsilon})$ gadgets suffice to approximate the clique splitting function, which is useful for sparsifying co-occurrence graphs and clique expansions of hypergraphs.

4.1 The $O(\frac{1}{\varepsilon}\log k)$ Upper Bound

We begin by showing that a logarithmic number of CB-gadgets is sufficient to approximate any SCB splitting function.

Theorem 15 *Let $\varepsilon \geq 0$ and $\mathbf{w}_e$ be an SCB splitting function on a k-node hyperedge. There exists a set of $O(\log_{1+\varepsilon} k)$ CB-gadgets, which can be constructed in $O(k\log_{1+\varepsilon} k)$ time, whose splitting function $\hat{\mathbf{w}}_e$ satisfies $\mathbf{w}_e(A) \leq \hat{\mathbf{w}}_e(A) \leq (1+\varepsilon)\mathbf{w}_e(A)$ for all $A \subseteq e$.*

Proof Let $r = \lfloor k/2 \rfloor$, and let $\mathbf{w} \in \mathcal{S}_r$ be the SCB integer function corresponding to $\mathbf{w}_e$, i.e., $\mathbf{w}(i) = \mathbf{w}_e(A)$ for $A \subseteq e$ such that $|A| \in \{i, k-i\}$. If we join all points of the form $(i, \mathbf{w}(i))$ for $i \in [r]$ by a line, this results in a piecewise linear function $\mathbf{f} \in \mathcal{F}_r$ that is concave and increasing on the interval $[0, r]$. We first show that there exists a set of $O(\log_{1+\varepsilon} r)$ linear pieces that approximates $\mathbf{f}$ on the entire interval $[1, r]$ to within a factor $(1+\varepsilon)$. Our argument follows similar previous results for approximating a concave function with a logarithmic number of linear pieces (Magnanti and Stratila, 2012; Gan et al., 2020).

For any value $y \in [1, r]$, not necessarily an integer, $\mathbf{f}(y)$ lies on a linear piece of $\mathbf{f}$ which we will denote by $g^{(y)}(x) = M_y \cdot x + B_y$, where $M_y \geq 0$ is the slope and $B_y \geq 0$ is the intercept. When $y = i$ is an integer, it may be the breakpoint between two distinct linear pieces, in which case we use the rightmost line so that $g^{(y)} = g^{(i)}$ as in (23), so $g^{(i)}(x) = M_i \cdot x + B_i$

where $M_i = \mathbf{w}(i+1) - \mathbf{w}(i)$ and $B_i = \mathbf{w}(i) - M_i \cdot i$. For any $z \in (y, r)$, the line $g^{(y)}$ provides a z/y approximation to $\mathbf{f}(z) = g^{(z)}(z)$, since

$$g^{(y)}(z) = M_y \cdot z + B_y \leq \frac{z}{y}(M_y \cdot y + B_y) = \frac{z}{y}\mathbf{f}(y) \leq \frac{z}{y}\mathbf{f}(z).$$

Equivalently, the line $g^{(y)}$ provides a $(1+\varepsilon)$-approximation for every $z \in [y, (1+\varepsilon)y]$. Thus, it takes J linear pieces to cover the set of intervals $[1, (1+\varepsilon)], [(1+\varepsilon), (1+\varepsilon)^2], \ldots, [(1+\varepsilon)^{J-1}, (1+\varepsilon)^J]$, and overall at most $1 + \lceil \log_{1+\varepsilon} r \rceil$ linear pieces to cover all of $[0, r]$.

Since Algorithm 1 finds the *smallest* set of linear pieces to $(1+\varepsilon)$-cover the splitting penalties, this smallest set must also have at most $O(\log_{1+\varepsilon} r)$ linear pieces. Given this piecewise linear approximation, we can use Lemma 10 to extract a CCB function $\hat{\mathbf{w}}$ of order $J = O(\log_{1+\varepsilon} r)$ satisfying $\mathbf{w}(i) \leq \hat{\mathbf{w}}(i) \leq (1+\varepsilon)\mathbf{w}(i)$ for $i \in \{0\} \cup [r]$. This $\hat{\mathbf{w}}$ in turn corresponds to a set of J CB-gadgets that $(1+\varepsilon)$-approximates the splitting function $\mathbf{w}_e$. Computing edge weights for the CB-gadgets using Algorithm 1 and Lemma 10 takes only $O(r)$ time, so the total runtime for constructing the combined gadgets is equal to the number of individual edges that must be placed, which is $O(k \log_{1+\varepsilon} k)$. ■

Theorem 1 on augmented sparsifiers follows as a corollary of Theorem 15. Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$ where each hyperedge has an SCB splitting function, we can use Theorem 15 to expand each $e \in \mathcal{E}$ into a gadget that has $O(\log_{1+\varepsilon} |e|)$ auxiliary nodes and $O(|e| \log_{1+\varepsilon} |e|)$ edges. Since $\log_{1+\varepsilon} n$ behaves as $\frac{1}{\varepsilon} \log n$ as $\varepsilon \to 0$, Theorem 1 follows.

In Appendix B, we show that using a slightly different reduction, we can prove that Theorem 15 holds even when we do not require splitting functions to be symmetric or satisfy $\mathbf{w}_e(\emptyset) = \mathbf{w}_e(e) = 0$. In Section 6 we use this fact to develop approximation algorithms for cardinality-based decomposable submodular function minimization.

4.2 Near Tightness on the Square Root Function

Our upper bound is nearly tight for the square root r-SCB integer function,

$$\mathbf{w}(i) = \sqrt{i} \text{ for } i \in \{0\} \cup [r]. \tag{28}$$

To prove this, we rely on a result of Magnanti and Stratila (2012) on the number of linear pieces needed to approximate the square root function over a continuous interval.

Lemma 16 *(Lemma 3 in (Magnanti and Stratila, 2012)) Let $\varepsilon > 0$ and $\phi(x) = \sqrt{x}$. Let ψ be a piecewise linear function whose linear pieces are all tangent lines to ϕ, satisfying $\psi(x) \leq (1+\varepsilon)\phi(x)$ for all $x \in [l, u]$ for $0 < l < u$. Then ψ contains at least $\lceil \log_{\gamma(\varepsilon)} \frac{u}{l} \rceil$ linear pieces, where $\gamma(\varepsilon) = (1 + 2\varepsilon(2+\varepsilon) + 2(1+\varepsilon)\sqrt{\varepsilon(2+\varepsilon)})^2$. There exists a piecewise linear function ψ^* of this form with exactly $\lceil \log_{\gamma(\varepsilon)} \frac{u}{l} \rceil$ linear pieces.[2] As $\varepsilon \to 0$, this values behaves as $\frac{1}{\sqrt{32\varepsilon}} \log \frac{u}{l}$.*

Lemma 16 is concerned with approximating the square root function for *all* values on a *continuous* interval. Therefore, it does not immediately imply any bounds on approximating

2. This additional statement is not included explicitly in the statement of Lemma 3 in the work of Magnanti and Stratila (2012), but it follows directly from the proof of the lemma, which shows how to construct such an optimal function ψ^*.

a discrete set of splitting penalties. In fact, we know that when lower bounding the number of linear pieces needed to approximate any $\mathbf{w} \in \mathcal{S}_r$, there is no lower bound of the form $q(\varepsilon)f(r)$ that holds for *all* $\varepsilon > 0$, if q is a function such that $q(\varepsilon) \to \infty$ as $\varepsilon \to 0$. This is simply because we can approximate $\mathbf{w}$ by piecewise linear interpolation, leading to an upper bound of $O(r)$ linear pieces even when $\varepsilon = 0$. Therefore, the best we can expect is a lower bound that holds for ε values that may still go to zero as $r \to \infty$, but are bounded in such a way that we do not contradict the $O(r)$ upper bound that holds for all SCB integer functions. We prove such a result for the square root splitting function, using Lemma 16 as a black box. When ε falls below the bound we assume in the following theorem statement, forming $O(r)$ linear pieces will be nearly optimal.

Theorem 17 *Let $\varepsilon > 0$ and $\mathbf{w}(i) = \sqrt{i}$. If $\varepsilon \geq r^{-\delta}$ for some constant $\delta \in (0, 2)$, then any piecewise linear function providing a $(1+\varepsilon)$-approximation for $\mathbf{w}$ contains $\Omega(\log_{\gamma(\varepsilon)} r)$ linear pieces. This behaves as $\Omega(\varepsilon^{-1/2} \log r)$ as $\varepsilon \to 0$.*

Proof Let L^* be the optimal set of linear pieces returned by running Algorithm 1. In order to show $|L^*| = \Omega(\log_{\gamma(\varepsilon)} r)$, we will construct a new set of linear pieces L that has asymptotically the same number of linear pieces as L^*, but also provides a $(1+\varepsilon)$-approximation for all x in an interval $[r^\beta, r]$ for some constant $\beta < 1$. Invoking Lemma 16 will then guarantee the final result.

Recall that L^* includes only two types of linear pieces: either linear pieces g satisfying $g(j) = \sqrt{j}$ for exactly one integer j (see Equation 24), or linear pieces formed by joining two points of $\mathbf{w}$ (see Equation 23). For the square root splitting function, the latter type of linear piece is of the form

$$g^{(t)}(i) = (\sqrt{t+1} - \sqrt{t})(i - t) + \sqrt{t}, \tag{29}$$

for some positive integer t less than r. This is the linear interpolation of the points $(t, \sqrt{t})$ and $(t+1, \sqrt{t+1})$. Both types of linear pieces bound $\phi(x) = \sqrt{x}$ above at *integer* points, but they may cross below ϕ at non-integer values of x. To apply Lemma 16, we would like to obtain a set of linear pieces that are all tangent lines to ϕ. We accomplish this by replacing each linear piece in L^* with two or three linear pieces that are tangent to ϕ at some point. For a positive integer j, let g_j denote the line tangent to $\phi(x) = \sqrt{x}$ at $x = j$, which is given by

$$g_j(x) = \frac{1}{2\sqrt{j}}(x - j) + \sqrt{j}. \tag{30}$$

We form a new set of linear pieces L made up of lines tangent to ϕ using the following replacements:

- If L^* contains a linear piece g that satisfies $g(j) = \sqrt{j}$ for exactly one integer j, add lines g_{j-1}, g_j, and g_{j+1} to L.
- If for an integer t, L^* contains $g^{(t)}$ (see Equation 29), add lines g_t and g_{t+1} to L.

By Observation 11, this replacement can only improve the approximation guarantee at integer points. Thus, L provides a $(1+\varepsilon)$-approximation at integer values, is made up of lines that are tangent to ϕ, and contains at most three times the number of lines in L^*.

Due to the concavity of ϕ, if a single line $g \in L$ provides a $(1+\varepsilon)$-approximation at consecutive integers i and $i+1$, then g provides the same approximation guarantee for all $x \in [i, i+1]$. However, if two integers i and $i+1$ are not *both* covered by the *same* line in L, then L does not necessarily provide a $(1+\varepsilon)$-approximation for every $x \in [i, i+1]$. There can be at most $|L|$ intervals of this form, since each interval defines an "intersection" at which one line $g \in L$ ceases to be a $(1+\varepsilon)$-approximation, and another line $g' \in L$ "takes over" as the line providing the approximation.

By Lemma 16, we can cover an entire interval $[i, i+1]$ for any integer i using a set of $\lceil \log_{\gamma(\varepsilon)} \left(1+\frac{1}{i}\right) \rceil$ linear pieces that are tangent to ϕ somewhere in $[i, i+1]$. Since $1+\sqrt{\varepsilon} \leq \gamma(\varepsilon)$, it in fact takes only one linear piece to cover $[i, i+1]$ as long as $1 + 1/i \leq 1 + \sqrt{\varepsilon} \implies i \geq 1/\sqrt{\varepsilon}$. Since $\varepsilon \geq r^{-\delta}$, interval $[i, i+1]$ can be covered by a single linear piece if $i \geq r^{\delta/2}$. Therefore, for each interval $[i, i+1]$, with $i \geq r^{\delta/2}$, that is not already covered by a single linear piece in L, we add one more linear piece to L to cover this interval. This at most doubles the size of L.

The resulting set L will have at most 6 times as many linear pieces as L^*, and is guaranteed to provide a $(1+\varepsilon)$-approximation for all integers, as well as the entire continuous interval $[r^{\delta/2}, r]$. Since δ is a fixed constant strictly less than 2, applying Lemma 16 shows that L has at least

$$\left\lceil \log_{\gamma(\varepsilon)} \frac{r}{r^{\delta/2}} \right\rceil = \Omega(\log_{\gamma(\varepsilon)} r^{1-\delta/2}) = \Omega(\log_{\gamma(\varepsilon)} r)$$

linear pieces. Therefore, $|L^*| = \Omega(\log_{\gamma(\varepsilon)} r)$ as well. ■

4.3 Improved Bound for the Clique Function

When approximating the clique expansion splitting function, Algorithm 1 finds a piecewise linear curve with at most $O(\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ linear pieces. We prove this by highlighting a different approach for constructing a piecewise linear curve with this many linear pieces, which upper bounds the number of linear pieces in the *optimal* curve found by Algorithm 1.

Clique splitting penalties for a k-node hyperedge correspond to nonnegative integer values of the continuous function $\zeta(x) = x \cdot (k-x)$. As we did in Section 3.3, we want to build a set of linear pieces L that provides and upper bounding $(1+\varepsilon)$-cover of ζ at integer values in $[0, r]$, where $r = \lfloor k/2 \rfloor$. We start by adding the line $g^{(0)}(x) = (\mathbf{w}(1) - \mathbf{w}(0))x + \mathbf{w}(0) = (k-1) \cdot x$ to L, which perfectly covers the first two splitting penalties $\mathbf{w}(0) = 0$ and $\mathbf{w}(1) = k-1$. In the remainder of our new procedure we will find a set of linear pieces to $(1+\varepsilon)$-cover ζ at *every* value of $x \in [1, k/2]$, even non-integer x.

We apply a greedy procedure similar to Algorithm 1. At each iteration we consider a leftmost endpoint z_i which is the largest value in $[1, k/2]$ for which we already have a $(1+\varepsilon)$-approximation. In the first iteration, we have $z_1 = 1$. We then would like to find a new linear piece that provides a $(1+\varepsilon)$-approximation for all values from z_i to some z_{i+1}, where the value of z_{i+1} is maximized. We restrict to linear pieces that are tangent to ζ. The line tangent to ζ at $t \in [1, k/2]$ is given by

$$g_t(x) = kx - 2tx + t^2 \,. \tag{31}$$

We find z_{i+1} in two steps:

1. **Step 1:** Find the maximum value t such that $g_t(z_i) = (1+\varepsilon)\zeta(z_i)$.
2. **Step 2:** Given t, find the maximum z_{i+1} such that $g_t(z_{i+1}) = (1+\varepsilon)\zeta(z_{i+1})$.

After completing these two steps, we add the linear piece g_t to L, knowing that it covers all values in $[z_i, z_{i+1}]$ with a $(1+\varepsilon)$-approximation. At this point, we will have a cover for all values in $[0, z_{i+1}]$, and we begin a new iteration with z_{i+1} being the largest value covered. We continue until we have covered all values up until $z_{i+1} \geq k/2$. If $t > k/2$ in Step 1 of the last iteration, we adjust the last linear piece to instead be the line tangent to ζ at $x = k/2$, so that we only include lines that have a nonnegative slope.

Lemma 18 *For any $z_i \in [1, k/2]$, the values of t and z_{i+1} in steps 1 and 2 are given by*

$$t = z_i + \sqrt{z_i(k-z_i)\varepsilon} \tag{32}$$

$$z_{i+1} = \frac{t}{1+\varepsilon} + \frac{k\varepsilon}{2(1+\varepsilon)} + \frac{1}{2(1+\varepsilon)}\left(k^2\varepsilon^2 + 4\varepsilon t(k-t)\right)^{1/2} \tag{33}$$

Proof The proof simply requires solving two different quadratic equations. For Step 1:

$$\begin{aligned} g_t(z_i) = (1+\varepsilon)\zeta(z_i) &\iff kz_i - 2tz_i + t^2 = (1+\varepsilon)(z_i k - z_i^2) \\ &\iff t^2 - 2z_i t - \varepsilon z_i k + (1+\varepsilon)z_i^2 = 0 \end{aligned}$$

Taking the larger solution to maximize t:

$$t = \frac{1}{2}\left(2z_i + \sqrt{4z_i^2 - 4(1+\varepsilon)z_i^2 + 4\varepsilon k z_i}\right) = z_i + \sqrt{z_i(k-z_i)\varepsilon}.$$

For Step 2:

$$\begin{aligned} g_t(z_{i+1}) = (1+\varepsilon)\zeta(z_{i+1}) &\iff kz_{i+1} - 2tz_{i+1} + t^2 = (1+\varepsilon)(z_{i+1}k - z_{i+1}^2) \\ &\iff (1+\varepsilon)z_{i+1}^2 + z_{i+1}(-\varepsilon k - 2t) + t^2 = 0. \end{aligned}$$

We again take the larger solution since we want to maximize z_{i+1}:

$$\begin{aligned} z_{i+1} &= \frac{1}{2(1+\varepsilon)}\left(\varepsilon k + 2t + \sqrt{\varepsilon^2 k^2 + 4t\varepsilon k + 4t^2 - 4(1+\varepsilon)t^2}\right) \\ &= \frac{1}{2(1+\varepsilon)}\left(\varepsilon k + 2t + \sqrt{\varepsilon^2 k^2 + 4t\varepsilon(k-t)}\right). \end{aligned}$$

■

Algorithm 3 summarizes the new procedure for covering the clique splitting function. Since $z_1 = 1$, if $\varepsilon \geq 1$, then

$$z_2 \geq \frac{1}{2(1+\varepsilon)}(2k\varepsilon) = \frac{k\varepsilon}{1+\varepsilon} \geq \frac{k}{2},$$

so after one step we have covered $[1, k/2]$. We can therefore focus on $\varepsilon < 1$.

Theorem 19 *For $\varepsilon < 1$, if L is the output from Algorithm 3, then $|L| = O(\varepsilon^{-1/2}\log\log\frac{1}{\varepsilon})$.*

Algorithm 3 Find a $(1+\varepsilon)$-cover L for the clique splitting function.

Input: Hyperedge size k, $\varepsilon \geq 0$
Output: $(1+\varepsilon)$ cover for clique splitting function.
$L = \{g^{(0)}\}$, where $g^{(0)}(x) = (k-1)x$
$z = 1$
do
  $t \leftarrow z + \sqrt{z(k-z)\varepsilon}$
  $z \leftarrow \frac{t}{1+\varepsilon} + \frac{k\varepsilon}{2(1+\varepsilon)} + \frac{1}{2(1+\varepsilon)}\left(k^2\varepsilon^2 + 4\varepsilon t(k-t)\right)^{1/2}$
  if $t > k/2$ **then**
    $L \leftarrow L \cup \{g_{k/2}\}$, where $g_{k/2}(x) = k/2$
  else
    $L \leftarrow L \cup \{g_t\}$, where $g_t(x) = kx - 2tx + t^2$
  end if
while $z_{i+1} < k/2$
Return $\mathbf{f}$ defined by $\mathbf{f}(x) = \min_{g \in L} g(x)$

Proof We get a loose bound for the value of t in Lemma 18 by noting that $(k - z_i) \geq k/2 \geq z_i$:

$$t = z_i + \sqrt{z_i\varepsilon(k-z_i)} \geq z_i + \sqrt{z_i^2\varepsilon} = z_i(1+\sqrt{\varepsilon}). \tag{34}$$

Since we assumed $\varepsilon < 1$, we know that

$$\frac{t}{1+\varepsilon} \geq \frac{z_i(1+\sqrt{\varepsilon})}{1+\varepsilon} > z_i. \tag{35}$$

Therefore, from (33) we see that

$$z_{i+1} > z_i + \frac{k\varepsilon}{2(1+\varepsilon)} + \frac{1}{2(1+\varepsilon)}\left(k^2\varepsilon^2 + 4\varepsilon t(k-t)\right)^{1/2} \tag{36}$$

$$> z_i + \frac{k\varepsilon}{2(1+\varepsilon)} + \frac{1}{2(1+\varepsilon)}\left(k^2\varepsilon^2\right)^{1/2} = z_i + \frac{k\varepsilon}{1+\varepsilon}. \tag{37}$$

From this we see that at each iteration, we cover an additional interval of length $z_{i+1} - z_i > \frac{k\varepsilon}{1+\varepsilon}$, and therefore we know it will take at most $O(1/\varepsilon)$ iterations to cover all of $[1, k/2]$. This upper bound is loose, however. The value of $z_{i+1} - z_i$ in fact increases significantly with each iteration, allowing the algorithm to cover larger and larger intervals as it progresses.

Since $z_1 = 1$ and $z_{i+1} - z_i \geq \frac{k\varepsilon}{1+\varepsilon}$, we see that $z_j \geq k\varepsilon$ for all $j \geq 3$. For the remainder of the proof, we focus on bounding the number of iterations it takes to cover the interval $[k\varepsilon, k/2]$. We separate the progress made by Algorithm 3 into different rounds. Round j refers to the set of iterations that the algorithm spends to cover the interval

$$R_j = \left[k\varepsilon^{\left(\frac{1}{2}\right)^{j-1}}, k\varepsilon^{\left(\frac{1}{2}\right)^{j}}\right]. \tag{38}$$

For example, Round 1 starts with the iteration i such that $z_i \geq k\varepsilon$, and terminates when the algorithm reaches an iteration i' where $z_{i'} \geq k\varepsilon^{1/2}$. A key observation is that it takes

less than $4/\sqrt{\varepsilon}$ iterations for the algorithm to finish Round j for any value of j. To see why, observe that from the bound in (36) we have

$$\begin{aligned} z_{i+1} - z_i &> \frac{k\varepsilon}{2(1+\varepsilon)} + \frac{1}{2(1+\varepsilon)}\left(k^2\varepsilon^2 + 4\varepsilon t(k-t)\right)^{1/2} \\ &> \frac{1}{2(1+\varepsilon)}\left(4\varepsilon t(k-t)\right)^{1/2} \geq \frac{1}{2(1+\varepsilon)}\left(4\varepsilon z_i \frac{k}{2}\right)^{1/2} > \frac{\sqrt{2}}{2}\frac{\sqrt{k\varepsilon}}{(1+\varepsilon)}\sqrt{z_i}. \end{aligned}$$

For each iteration i in Round j, we know that $z_i \geq k\varepsilon^{\left(\frac{1}{2}\right)^{j-1}}$, so that

$$z_{i+1} - z_i > \frac{\sqrt{2}}{2}\frac{\sqrt{k\varepsilon}}{(1+\varepsilon)}\sqrt{k\varepsilon^{\left(\frac{1}{2}\right)^{j-1}}} \geq \frac{\sqrt{2}}{2}\frac{k\varepsilon^{\frac{1}{2}+\left(\frac{1}{2}\right)^j}}{1+\varepsilon} = C \cdot k \cdot \varepsilon^{\frac{1}{2}+\left(\frac{1}{2}\right)^j}, \tag{39}$$

where $C = \sqrt{2}/(2(1+\varepsilon))$ is a constant larger than $1/4$. Since each iteration of Round j covers an interval of length at least $C \cdot k \cdot \varepsilon^{\frac{1}{2}+\left(\frac{1}{2}\right)^j}$, and the right endpoint for Round j is $k\varepsilon^{\left(\frac{1}{2}\right)^j}$, the maximum number of iterations needed to complete Round j is

$$\frac{k\varepsilon^{\left(\frac{1}{2}\right)^j}}{C \cdot k \cdot \varepsilon^{\frac{1}{2}+\left(\frac{1}{2}\right)^j}} = \frac{1}{C\sqrt{\varepsilon}}. \tag{40}$$

Therefore, after p rounds, the algorithm will have performed $O(p \cdot \varepsilon^{-1/2})$ iterations, to cover the interval $[1, k\varepsilon^{\left(\frac{1}{2}\right)^p}]$. Since we set out to cover the interval $[1, k/2]$, this will be accomplished as soon as p satisfies $\varepsilon^{\left(\frac{1}{2}\right)^p} \geq 1/2$, which holds as long as $p \geq \log_2 \log_2 \frac{1}{\varepsilon}$:

$$\varepsilon^{\left(\frac{1}{2}\right)^p} \geq 1/2 \iff \left(\frac{1}{2}\right)^p \log_2 \varepsilon \geq -1 \iff \log_2 \varepsilon \geq -2^p \iff \log_2 \log_2 \frac{1}{\varepsilon} \leq p.$$

This means that the number of iteration of Algorithm 3, and therefore the number of linear pieces in L, is bounded above by $O(\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$. ■

We obtain a proof of Theorem 2 on sparsifying the complete graph as a corollary.

Proof of Theorem 2. A complete graph on n nodes can be viewed as a hypergraph with a single n-node hyperedge with a clique expansion splitting function. Theorem 2 says that the clique expansion integer function $\mathbf{w}(i) = i \cdot (n-i)$ can be covered with $O(\varepsilon^{-1/2} \log\log \varepsilon^{-1})$ linear pieces, which is equivalent to saying the clique expansion splitting function can be modeled using this many CB-gadgets. Each CB-gadget has two auxiliary nodes and $(2n+1)$ directed edges. This results in an augmented sparsifier for the complete graph with $O(n\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ edges. This is only meaningful if ε is small enough so that $O(\varepsilon^{-1/2} \log\log \frac{1}{\varepsilon})$ is asymptotically less than n, so our sparsifier has $O(n + \varepsilon^{-1/2} \log\log \frac{1}{\varepsilon}) = O(n)$ nodes.

□

5. Sparsifying Co-occurrence Graphs

Recall from the introduction that a co-occurrence graph is formally defined by a set of nodes V and a set of subsets $\mathcal{C} \subseteq 2^V$. In practice, each $c \in \mathcal{C}$ could represent some type of group

interaction involving nodes in c or a set of nodes sharing the same attribute. We define the co-occurrence graph $G = (V, E)$ on $\mathcal{C}$ to be the graph where nodes i and j share an edge with weight $w_{ij} = \sum_{c \in \mathcal{C}} w_c$, where $w_c \geq 0$ is a weight associated with co-occurrence set $c \in \mathcal{C}$. The case when $w_c = 1$ is standard and is an example of a common practice of "one-mode projections" of bipartite graphs or affiliation networks (Lattanzi and Sivakumar, 2009; Li et al., 2007; Neal, 2014; Newman et al., 2001; Ramasco and Morris, 2006; Zhou et al., 2007; Benson et al., 2020) — a graph is formed on the nodes from one side of a bipartite graph by connecting two nodes whenever they share a common neighbor on the other side, where edges are weighted based on the number of shared neighbors.

A co-occurrence graph G has the following *co-occurrence* cut function:

$$\mathbf{cut}_G(S) = \sum_{c \in \mathcal{C}} w_c \cdot |S \cap c| \cdot |\bar{S} \cap c|. \tag{41}$$

In this sense, the co-occurrence graph is naturally interpreted as a weighted clique expansion of a hypergraph $\mathcal{H} = (V, \mathcal{C})$, which itself is a special case of reducing a submodular, cardinality-based hypergraph to a graph. However, this type of graph construction is by no means restricted to literature on hypergraph clustering. In many applications, the first step in a larger experimental pipeline is to construct a graph of this type from a large data set. The resulting graph is often quite dense, as numerous domains involve large hyperedges (Veldt et al., 2020a; Purkait et al., 2016). This makes it expensive to form, store, and compute over co-occurrence graphs in practice.

Solving cut problems on these dense co-occurrence graphs arises naturally in many settings. For example, any hypergraph clustering application that relies on a clique expansion involves a graph with a co-occurrence cut function (Agarwal et al., 2006; Hadley, 1995; Hein et al., 2013; Huang et al., 2015; Zhou et al., 2006; Li and Milenkovic, 2017; Vannelli and Hadley, 1990; Zien et al., 1999; Rodríguez, 2009; Veldt et al., 2020b,a). Clustering social networks is another use case, as online platforms have many ways to create groups of users (e.g., events, special interest groups, businesses, organizations, etc.), that can be large in practice. Furthermore, cuts in co-occurrence graphs of students on a university campus (based on, e.g., common classes, living arrangements, or physical proximity) are relevant to preventing the spread of infectious diseases such as COVID-19.

In these cases, it would be more efficient to sparsify the graph *without ever forming it explicitly*, by sparsifying large cliques induced by co-occurrence relationships. Although this strategy seems intuitive, it is often ignored in practice. We therefore present several theoretical results that highlight the benefits of this implicit approach to sparsification. Our focus is on results that can be achieved using augmented sparsifiers for cliques, though many of the same benefits could also be achieved with standard sparsification techniques.

5.1 Proof of Theorem 3

Let $\mathcal{C}$ be a set of nonempty co-occurrence groups on a set of n nodes, V, and let $G = (V, E)$ be the corresponding co-occurrence graph on $\mathcal{C}$. For $c \in \mathcal{C}$, let $k_c = |c|$ be the number of nodes in c. For $v \in V$, let d_v be the co-occurrence degree of v: the number of sets c containing v. Let $d_{avg} = \frac{1}{n} \sum_{v \in V} d_v$ be the average co-occurrence degree. We re-state and prove Theorem 3, first presented in the introduction. The proof holds independent of the

weight w_c we associate with each $c \in \mathcal{C}$, since we can always scale our graph reduction techniques by an arbitrary positive weight.

Restatement of Theorem 3. Let $\varepsilon > 0$ and $f(\varepsilon) = \varepsilon^{-1/2} \log\log \frac{1}{\varepsilon}$. There exists an augmented sparsifier for G with $O(n + |\mathcal{C}| \cdot f(\varepsilon))$ nodes and $O(n \cdot d_{avg} \cdot f(\varepsilon))$ edges. In particular, if d_{avg} is constant and for some $\delta > 0$ we have $\sum_{c \in \mathcal{C}} |c|^2 = \Omega(n^{1+\delta})$, then forming G explicitly takes $\Omega(n^{1+\delta})$ time, but an augmented sparsifier for G with $O(nf(\varepsilon))$ nodes and $O(nf(\varepsilon))$ edges can be constructed in $O(nf(\varepsilon))$ time.

Proof The set c induces a clique in the co-occurrence graph with $O(k_c^2)$ edges. Therefore, the runtime for explicitly forming $G = (V, E)$ by expanding cliques and placing all edges equals $O(\sum_{c \in \mathcal{C}} k_c^2) = \Omega(n^{1+\delta})$. By Theorem 2, for each $c \in \mathcal{C}$ we can produce an augmented sparsifier with $O(k_c f(\varepsilon))$ directed edges and $O(f(\varepsilon))$ new auxiliary nodes. Sparsifying each clique in this way will produce an augmented sparsifier $\hat{G} = (\hat{V}, \hat{E})$ where

$$|\hat{E}| = \sum_{c \in \mathcal{C}} O(k_c f(\varepsilon)) = O(f(\varepsilon) \cdot n \cdot d_{avg}) \tag{42}$$

$$|\hat{V}| = n + \sum_{c \in \mathcal{C}} O(f(\varepsilon)) = O(n + |\mathcal{C}| f(\varepsilon)). \tag{43}$$

Observe that $n \cdot d_{avg} = \sum_{v \in V} d_v = \sum_{c \in \mathcal{C}} k_c$. If d_{avg} is a constant, this implies that $\sum_{c \in \mathcal{C}} k_c = O(n)$, and furthermore that $|\mathcal{C}| = O(n)$, since each $k_c \geq 1$. Therefore $|\hat{E}|$ and $|\hat{V}|$ are both $O(nf(\varepsilon))$. Only $O(f(\varepsilon))$ edge weights need to be computed for the clique, so the overall runtime is just the time it takes to explicitly place the $O(nf(\varepsilon))$ edges. ■

The above theorem includes the case where $|\mathcal{C}| = o(n)$, meaning that $\mathcal{C}$ is made up of a sublinear number of large co-occurrence interactions. In this case, our augmented sparsifier will have fewer than $O(nf(\varepsilon))$ nodes. When $|\mathcal{C}| = \omega(n)$, the average degree will no longer be a constant and therefore it becomes theoretically beneficial to sparsify each clique in $\mathcal{C}$ using standard undirected sparsifiers. For each $c \in \mathcal{C}$, standard cut sparsification techniques will produce an ε-cut sparsifier of c with $O(k_c \varepsilon^{-2})$ undirected edges and exactly k_c nodes. If two nodes appear in multiple co-occurrence relationships, the resulting edges can be collapsed into a weighted edge between the nodes, meaning that the number of edges in the resulting sparsifier does not depend on d_{avg}. We discuss tradeoffs between different sparsification techniques in depth in a later subsection. Regardless of the sparsification technique we apply, implicitly sparsifying a co-occurrence graph will often lead to a significant decrease in runtime compared to forming the entire graph prior to sparsifying it.

5.2 A Simple Co-occurrence Model

We now consider a simple model for co-occurrence graphs with a power-law group size distribution, that produces graphs satisfying the conditions of Theorem 3 in a range of different parameter settings. Such distributions have been observed for many types of co-occurrence graphs constructed from real-world data (Clauset et al., 2009; Benson et al., 2020). More formally, let V be a set of n nodes, and assume a co-occurrence set c is randomly generated by sampling a set of size K from a discrete power-law distribution where for $k \in [1, n]$:

$$\mathbb{P}[K = k] = Ck^{-\gamma}.$$

Here, C is a normalizing constant for the distribution, and γ is a parameter of the model. Once K is drawn from this model, a co-occurrence set c is generated with a set of K nodes from V chosen uniformly at random. This procedure can be repeated an arbitrary number of times (drawing all sizes K independently) to produce a set of co-occurrence sets $\mathcal{C}$. This $\mathcal{C}$ can then be used to generate a co-occurrence graph $G = (V, E)$. (The end result of this procedure is a type of *random intersection graph* (Bloznelis et al., 2013).) We first consider a parameter regime where set sizes are constant on average but large enough to produce a dense co-occurrence graph that is inefficient to explicitly form in practice. The regime has an exponent $\gamma \in (2, 3)$, which is common in real-world data (Clauset et al., 2009).

Theorem 20 *Let $\mathcal{C}$ be a set of $O(n)$ co-occurrence sets from the power-law model with $\gamma \in (2, 3)$. The expected degree of each node is constant and $\mathbb{E}\left[\sum_{c\in\mathcal{C}} |c|^2\right] = O(n^{4-\gamma})$.*

Proof Let K be the size of a randomly generated co-occurrence set. We compute:

$$\mathbb{E}[K^2] = \sum_{k=1}^{n} k^2 \cdot \mathbb{P}[K = k] = C \cdot \sum_{k=1}^{n} k^{2-\gamma} \leq C \cdot \left[1 + \int_1^n x^{2-\gamma} dx\right] = C + \frac{Cn^{3-\gamma}}{3-\gamma} - \frac{C}{3-\gamma} = O(n^{3-\gamma}).$$

Therefore,

$$\mathbb{E}\left[\sum_{c\in\mathcal{C}} |c|^2\right] = \sum_{c\in\mathcal{C}} \mathbb{E}[K^2] = O(n^{4-\gamma}).$$

For a node $v \in V$ and a randomly generated set c, the probability that v will be selected to be in c is

$$\mathbb{P}[v \in c] = \sum_{k=1}^{n} \mathbb{P}[|c| = k] \cdot \frac{\binom{n-1}{k-1}}{\binom{n}{k}} = C \cdot \sum_{k=1}^{n} k^{-\gamma} \cdot \frac{k}{n} = \frac{C}{n} \cdot \left[1 + \int_1^n x^{1-\gamma} dx\right] = O(n^{-1}).$$

Since there are $O(n)$ co-occurrence sets in $\mathcal{C}$ and they each are generated independently, in expectation, v will have a constant degree. ■

We similarly consider another regime of co-occurrence graphs where the number of co-occurrence sets is $o(n)$, but the co-occurrence sets are larger on average.

Theorem 21 *Let $\mathcal{C}$ be a set of $O(n^\beta)$ co-occurrence sets, where $\beta \in (0, 1)$, obtained from the power-law co-occurrence model with $\gamma = 1 + \beta$. Then the expected degree of each node will be a constant and $\mathbb{E}\left[\sum_{c\in\mathcal{C}} |c|^2\right] = O(n^3)$.*

Proof Again let K be a random variable representing the co-occurrence set size. We have

$$\mathbb{E}[K^2] = C \cdot \sum_{k=1}^{n} k^{2-\gamma} = O(n^{3-\gamma}) \implies \mathbb{E}\left[\sum_{c\in\mathcal{C}} |c|^2\right] = O(n^{\beta+4-\gamma}) = O(n^3).$$

For a node $v \in V$ and a randomly generated set c, the probability that v will be in c is

$$\mathbb{P}[v \in c] = \sum_{k=1}^{n} \mathbb{P}[|c| = k] \cdot \frac{\binom{n-1}{k-1}}{\binom{n}{k}} = \frac{C}{n} \sum_{k=1}^{n} k^{1-\gamma} = O(n^{2-\gamma-1}) = O(n^{-\beta})$$

Since there are $O(n^\beta)$ co-occurrence sets in $\mathcal{C}$, the expected degree of v is a constant. ■

In Theorem 21, the exponent of the power-law distribution is assumed to be directly related to the number of co-occurrence sets in $\mathcal{C}$. This assumption is included simply to ensure that we are in fact considering co-occurrence graphs with $O(n)$ nodes. We could alternatively consider a power-law distribution with exponent $\gamma \in (1, 2)$ and generate $O(n^\beta)$ co-occurrence sets for any $\beta < 1 - \gamma$. We simply note that in this regime, the expected average degree will be $o(1)$. Assuming we exclude isolated nodes, this will produce a co-occurrence graph with $o(n)$ nodes in expectation. Our techniques still apply in this setting, and we can produce augmented sparsifiers with $O(|\mathcal{C}| \cdot f(\varepsilon))$ nodes and $O(n \cdot d_{avg} \cdot f(\varepsilon)) = o(n \cdot f(\varepsilon))$ edges. When $|\mathcal{C}| = \Omega(n)$, then $d_{avg} = \Omega(1)$ and the number of edges in our augmented sparsifiers will have worse than linear dependence on n. However, in this regime we can still quickly obtain sparsifiers with $O(n\varepsilon^{-2})$ edges via implicit sparsification by using standard undirected sparsifiers.

More sophisticated models for generating co-occurrence graphs can be derived from existing models for projections of bipartite graphs (Benson et al., 2020; Bloznelis et al., 2013; Bloznelis and Petuchovas, 2017). These make it possible to set different distributions for node degrees in V and highlight other classes of co-occurrence graphs satisfying the assumptions of Theorem 3. We have focused on the simplest model for illustrating classes of power-law co-occurrence graphs that satisfy the assumptions of the theorem.

5.3 Tradeoffs in sparsification techniques

There are several tradeoffs to consider when using different black-box sparsifiers for implicit co-occurrence sparsification. Standard sparsification techniques involve no auxiliary nodes, and have undirected edges, which is beneficial in numerous applications. Also, the number of edges they require is independent of d_{avg}. Therefore, in cases where the average co-occurrence degree is larger than a constant, we obtain better theoretical improvements using standard sparsifiers.

On the other hand, in many settings, it is natural to assume the number of co-occurrences each node belongs to is a constant, even if some co-occurrences are very large. In these regimes, our augmented sparsifiers will have fewer edges that traditional sparsifiers due to a better dependence on ε. Our techniques are also deterministic and our sparsifiers are very easy to construct in practice. Edge weights for our sparsifiers are easy to determine in $O(f(\varepsilon))$ time for each co-occurrence group using Algorithm 1 (or Algorithm 3) coupled with Lemma 10. The bottleneck in our construction is simply visiting each node in a set c to place edges between it and the auxiliary nodes. Even in cases where there are no asymptotic reductions in theoretical runtime, our techniques provide a simple and highly practical tool for solving cut problems on co-occurrence data.

6. Approximate Cardinality-based DSFM

Appendix B shows how our reduction techniques can be adjusted to apply to splitting functions that are asymmetric and do not necessarily satisfy $\mathbf{w}_e(\emptyset) = \mathbf{w}_e(e) = 0$. Section 3 addresses the special case of symmetric and non-cut ignoring functions, as these are natural for hypergraph cut problems (Li and Milenkovic, 2017, 2018b; Veldt et al., 2022), and pro-

vide the clearest exposition of our main techniques and results. Furthermore, the reduction from Section 3 requires roughly half the number of edges that the generalized reduction strategy in Appendix B would use for symmetric splitting functions. Nevertheless, the same asymptotic upper bound of $O(\frac{1}{\varepsilon}\log k)$ edges holds for approximately modeling the more general splitting function on a k-node hyperedge. Our techniques then lead to the first approximation algorithms for the more general problem of minimizing cardinality-based decomposable submodular functions.

6.1 Decomposable Submodular Function Minimization

Any submodular function can be minimized in polynomial time (Orlin, 2009; Iwata et al., 2001; Iwata and Orlin, 2009), but the runtimes for general submodular functions are impractical in most cases. A number of recent papers have developed faster algorithms for minimizing submodular functions that are sums of simpler submodular functions (Ene et al., 2017; Kolmogorov, 2012; Stobbe and Krause, 2010; Li and Milenkovic, 2018a; Nishihara et al., 2014; Ene and Nguyen, 2015; Jegelka et al., 2011, 2013). This is also know as decomposable submodular function minimization (DSFM). Many energy minimization problems from computer vision correspond to DSFM problems (Kohli et al., 2009; Kolmogorov and Zabin, 2004; Freedman and Drineas, 2005).

Let $f\colon 2^V \to \mathbb{R}^+$ be a submodular function, such that for $S \subseteq V$,

$$f(S) = \sum_{e\in\mathcal{E}} \mathbf{f}_e(S \cap e), \tag{44}$$

where for each $e \in \mathcal{E}$, $\mathbf{f}_e$ is a simpler submodular function with support only on a subset $e \subseteq V$. We can assume without loss of generality that every $\mathbf{f}_e$ is a non-negative function. The goal of DSFM is to find $\arg\min_S f(S)$. The terminology used for problems of this form differs depending on the context. We will continue referring to $\mathcal{E}$ as a hyperedge set, V as a node set, $\mathbf{f}_e$ as generalized splitting functions, and f as a hypergraph cut function.

Cardinality-based decomposable submodular function minimization (Card-DSFM) has received significant special attention in previous work (Kohli et al., 2009; Kolmogorov, 2012; Jegelka et al., 2011, 2013; Stobbe and Krause, 2010). This variant of the problem assumes that each function $\mathbf{f}_e$ satisfies $\mathbf{f}_e(S) = g_e(|S|)$ for some concave function g_e Unlike most existing work on generalized hypergraph cut functions (Li and Milenkovic, 2017, 2018b; Veldt et al., 2022), research on DSFM does not typically assume that the functions $\mathbf{f}_e$ are symmetric, and also do not assume that $\mathbf{f}_e(\emptyset) = \mathbf{f}_e(e) = 0$. In the appendix we demonstrate that our sparse reduction techniques can also be adapted to apply to this more general setting as well. This leads to a proof of Theorem 4.

Proof of Theorem 4. Lemma 28 shows that every cardinality-based submodular function $\mathbf{f}_e$ can be approximately modeled by a combination of $O(\frac{1}{\varepsilon}\log k)$ asymmetric CB-gadgets. This implies that any instance of Card-DSFM can be reduced to an s-t cut problem on a graph with $N = O(n+\frac{1}{\varepsilon}\sum_{e\in\mathcal{E}}\log|e|)$ nodes and $M = O(n+\frac{1}{\varepsilon}\sum_{e\in\mathcal{E}}\log|e|)$ edges. Applying a recent $\tilde{O}(M+N^{1.5})$-time algorithm for the maximum s-t flow problem of van den Brand et al. (2021) yields the overall runtime guarantee of $\tilde{O}\left(\frac{1}{\varepsilon}\sum_{e\in\mathcal{E}}|e| + (n+\frac{R}{\varepsilon})^{3/2}\right)$. If we wish to find the optimal solution to an instance of Card-DSFM, we can also set $\varepsilon = 0$ and obtain

a reduced graph with $O(\sum_{e\in\mathcal{E}}|e|)$ nodes and $O(\sum_{e\in\mathcal{E}}|e|^2)$ edges. The resulting runtime will then be $\tilde{O}\left(\sum_{e\in\mathcal{E}}|e|^2+\left(\sum_{e\in\mathcal{E}}|e|\right)^{3/2}\right)$.

6.2 Runtime Comparison for Cardinality-Based DSFM

We refer to our approximation algorithm for Card-DSFM as SparseCard, since it depends on sparse reduction techniques. In the remainder of the section, we provide a careful runtime comparison between SparseCard and competing runtimes for Card-DSFM. We focus on each runtime's dependence on $n=|V|$, $R=|E|$, and support sizes $|e|$, and use $\tilde{O}$ notation to hide logarithmic factors of n, R, and $1/\varepsilon$. To easily compare weakly polynomial runtimes, we assume that each $\mathbf{f}_e$ has integer outputs, and assume that $\log(\max_S f(S))$ is small enough that it can also be absorbed by $\tilde{O}$ notation. Our primary goal is to highlight the runtime improvements that are possible when an approximate solution suffices. Among algorithms for DSFM, SparseCard is unique in its ability to quickly find solutions with a priori multiplicative approximation guarantees. Previous approaches for DSFM focus on either obtaining exact solutions, or finding a solution to within an *additive* approximation error $\epsilon>0$ (Axiotis et al., 2021; Ene et al., 2017; Li and Milenkovic, 2018a; Jegelka et al., 2013). In the latter case, setting ϵ small enough will guarantee an optimal solution in the case of integer output functions. However, these results provide no a priori multiplicative approximation guarantee. Furthermore, using a larger value of ϵ for these additive approximations only improves runtimes in logarithmic terms. In constrast, setting $\varepsilon>0$ will often lead to substantial runtime decreases for SparseCard.

Competing runtime guarantees. Table 1 lists runtimes for existing methods for DSFM. We have listed the asymptotic runtime for SparseCard when applying the recent maximum flow algorithm of van den Brand et al. (2021). While this leads to the best theoretical guarantees for our method, asymptotic runtime improvements over competing methods can also be shown using alternative fast algorithms for maximum flow (Gao et al., 2021; Lee and Sidford, 2014; Goldberg and Rao, 1998). For the submodular flow algorithm of Kolmogorov (2012), we have reported the runtime guarantee provided specifically for Card-DSFM. While other approaches have frequently been applied to Card-DSFM (Ene et al., 2017; Stobbe and Krause, 2010; Li and Milenkovic, 2018a; Jegelka et al., 2013), runtimes guarantees for this case have not been presented explicitly and are more challenging to exactly pinpoint. Runtimes for most algorithms depend on certain oracles for solving smaller minimization problems at functions $\mathbf{f}_e$ in an inner loop. For $e\in E$, let $\mathcal{O}_e$ be a *quadratic minimization oracle*, which for an arbitrary vector w solves $\min_{y\in B(\mathbf{f}_e)}\|y+w\|$ where $B(\mathbf{f}_e)$ is the base polytope of the submodular function $\mathbf{f}_e$ (see Ene et al. 2017; Bach 2013; Jegelka et al. 2013 for details). Let θ_e be the time it take to evaluate the oracle at $e\in E$, and define $\theta_{max}=\max_{e\in E}\theta_e$ and $\theta_{avg}=\frac{1}{R}\sum_{e\in E}\theta_e$. Although these oracles admit faster implementations in the case of concave cardinality functions, it is not immediately clear from previous work what is the best possible runtime. When $w=0$, solving $\min_{y\in B(\mathbf{f}_e)}\|y+w\|$ takes $O(|e|\log|e|)$ time (Jegelka et al., 2013), so this serves as a best case runtime we can expect for the more general oracle $\mathcal{O}_e$ based on previous results. We note also that in the case of the function function $\mathbf{f}_e(A)=|A||e\backslash A|$, Ene et al. (2017) highlight that an $O(|e|\log|e|+|e|\tau_e)$ algorithm can be used, where τ_e denotes the time it take to evaluate $\mathbf{f}_e(S\cap e)$ for any $S\subseteq e$. In our

runtime comparisons will use the bound $\theta_e = \Omega(|e|)$, as it is reasonable to expect that any meaningful submodular function we consider should take a least linear time to minimize.

Fast approximate solutions ($\varepsilon > 0$). Barring the regime where support sizes $|e|$ are all very small, the accelerated coordinate descent method (ACDM) of Ene and Nguyen (2015) has the fastest previous runtime. For a simple parameterized runtime analysis, consider a DSFM problem where the average support size is $(1/R)\sum_e |e| = \Theta(n^\alpha)$ for $\alpha \in [0, 1]$, and $R = \Theta(n^\beta)$, where $\beta \geq 1 - \alpha$ must hold if we assume each $v \in V$ is in the support for at least one function $\mathbf{f}_e$. An exact runtime comparison depends on the best runtime for the oracle $\mathcal{O}_e$ for concave cardinality functions. If an $O(|e| \log |e|)$ oracle is possible, the overall runtime guarantee for ACDM would be $\Omega(n^{1+\alpha+\beta})$. Meanwhile, for a small constant $\varepsilon > 0$, SparseCard provides a $(1+\varepsilon)$-approximate solution in time $\tilde{O}(n^{\alpha+\beta} + \max\{n^{3/2}, n^{3\beta/2}\})$, which will faster by at least a factor $\tilde{O}(\sqrt{n})$ whenever $\beta \leq 1$. When $\beta > 1$, finding an approximation with SparseCard is guaranteed to be faster whenever $R = o(n^{2+2\alpha})$. If the best case oracle $\mathcal{O}_e$ for concave cardinality functions is $\omega(|e| \log |e|)$, the runtime improvement of our method is even more significant.

Guarantees for exact solutions ($\varepsilon = 0$). As an added bonus, running SparseCard with $\varepsilon = 0$ leads to the fastest runtime for finding *exact* solutions in many regimes. In this case, we can guarantee SparseCard will be faster than ACDM when the average support size is $\Theta(n^\alpha)$ and $R = o(n^{2-\alpha})$. SparseCard can also find exact solutions faster than other discrete optimization methods (Kolmogorov, 2012; Axiotis et al., 2021; Fix et al., 2013) in wide parameter regimes. Our method has a faster runtime guarantee than the submodular flow algorithm of Kolmogorov (2012) in all parameter regimes, and has a better runtime guarantee than the strongly-polynomial IBFS algorithm (Fix et al., 2013) when $\sum_e |e| = o(n^4)$, which includes all cases where $R = o(n^3)$. Both variants of IBFS as well as the recent method of Axiotis et al. (2021) become impractical if even a *single* function $\mathbf{f}_e$ has support size $|e| = O(n)$ (even when the average support size is much smaller). In this case, runtimes for these methods are $\Omega(n^5)$. Even using the very loose bounds $\sum_e |e|^2 \leq Rn^2$ and $\sum_e |e| \leq nR$, our method is guaranteed to find exact solutions faster as long as $R = o(n^{7/3})$, with significant additional improvements when approximate solutions suffice. In the extreme case where $\max_e |e| = O(1)$, the algorithm of Axiotis et al. (2021) obtains the best theoretical guarantees, as it has the same asymptotic runtime as a single maximum flow computation on an n-node, R-edge graph. It is worth noting that in this regime, running SparseCard with $\varepsilon = 0$ can exactly solve Card-DSFM with the same asymptotic runtime guarantee as long as $R = \Theta(n)$, and has the added practical advantage that it requires only one call to a maximum flow oracle.

The runtime guarantee for SparseCard when $\varepsilon = 0$ can be matched asymptotically by combining existing exact reduction techniques (Kohli et al., 2009; Stobbe and Krause, 2010; Veldt et al., 2022) with fast maximum flow algorithms. However, our method has the practical advantage of finding the *sparsest* exact reduction in terms of CB-gadgets. For example, this results in a reduced graph with roughly half the number of edges required if we were to apply our previous exact reduction techniques (Veldt et al., 2022). Analogously, Stobbe and Krause (2010) showed that a concave cardinality function can be decomposed as a sum of modular functions plus a combination of $|e| - 1$ threshold potentials, our approximation technique will find a linear combination with $\lfloor |e|/2 \rfloor$ threshold potentials. This amounts to the observation that any $k + 1$ points $\{i, g(i)\}$ can be joined by $\lfloor k/2 \rfloor + 1$ lines instead of

using k. Overall, the most significant advantage of SparseCard over existing reduction methods is its ability to find fast approximate solutions.

7. Experiments

In addition to its strong theoretical guarantees, SparseCard is very practical and leads to substantial improvements in benchmark image segmentation problems and localized hypergraph clustering tasks. We focus on image segmentation and localized hypergraph clustering tasks that simultaneously include component functions of large and small support, which are common in practice (Shanu et al., 2016; Ene et al., 2017; Veldt et al., 2022; Liu et al., 2021; Purkait et al., 2016). Image segmentation experiments were run on a laptop with a 2.2 GHz Intel Core i7 processor and 8GB of RAM. For our local hypergraph clustering experiments, in order to run a larger number of experiments we used a machine with 4 x 18-core, 3.10 GHz Intel Xeon gold processors with 1.5 TB RAM. We consider public data sets previously made available for academic research, and use existing open source software for competing methods. Image data sets are available at `http://people.csail.mit.edu/stefje/code.html` and hypergraph clustering data sets are available at `www.cs.cornell.edu/~arb/data/`. Our code is available at `https://github.com/nveldt/SparseCardDSFM`.

7.1 Benchmark Image Segmentation Tasks

SparseCard provides faster approximate solutions for standard image segmentation tasks previously used as benchmarks for DSFM (Jegelka et al., 2013; Li and Milenkovic, 2018a; Ene et al., 2017). We consider the *smallplant* and *octopus* segmentation tasks from Jegelka and Bilmes (2011). These amount to minimizing a decomposable submodular function on a ground set of size $|V| = 427 \cdot 640 = 273280$, where each $v \in V$ is a pixel from a 427×640 pixel image and there are three types of component functions. The first type are unary potentials for each pixel/node, i.e., functions of support size 1 representing each node's bias to be in the output set. The second type are pairwise potentials from a 4-neighbor grid graph; pixels i and j share an edge if they are directly adjacent vertically or horizontally. The third type are region potentials of the form $\mathbf{f}_e(A) = |A||e \backslash A|$ for $A \subseteq e$, where e represents a superpixel region. The problem can be solved via maximum flow even without sophisticated reduction techniques for cardinality functions, as a region potential function on e can be modeled by placing a clique of edges on e. We compute an optimal solution using this reduction.

Comparison against exact reductions. Compared with the exact reduction method, running SparseCard with $\varepsilon > 0$ leads to much sparser graphs, much faster runtimes, and a posteriori approximation factors that are significantly better than $(1 + \varepsilon)$. For example, on the *smallplant* data set, when $\varepsilon = 1.0$, the returned solution is within a factor 1.004 of optimality, even though the a priori guarantee is a 2-approximation. The sparse reduced graph has only 0.013 times the number of edges in the exact reduced graph, and solving the sparse flow problem is over two orders of magnitude faster than solving the problem on the exact reduced graph (which takes roughly 20 minutes on a laptop). In Table 2 we list the sparsity, runtime, and a posteriori guarantee obtained for a range of ε values on the *smallplant* data set using the superpixel segmentation with 500 regions.

ε	1.0	0.2336	0.0546	0.0127	0.003	0.0007	0.0002
Approx.-1	$4 \cdot 10^{-3}$	$2 \cdot 10^{-3}$	$6 \cdot 10^{-4}$	$6 \cdot 10^{-5}$	$3 \cdot 10^{-5}$	$7 \cdot 10^{-6}$	$7 \cdot 10^{-7}$
Sparsity	0.013	0.017	0.02	0.035	0.06	0.108	0.196
Runtime	4.1	5.6	6.7	11.5	24.3	41.4	74.3

Table 2: Results from SparseCard for different $\varepsilon > 0$ on the *smallplant* instance with 500 superpixels. Sparsity is the fraction of edges in the approximate graph reduction compared with the exact reduction. Finding the exact solution on the dense exact reduced graph took roughly 20 minutes.

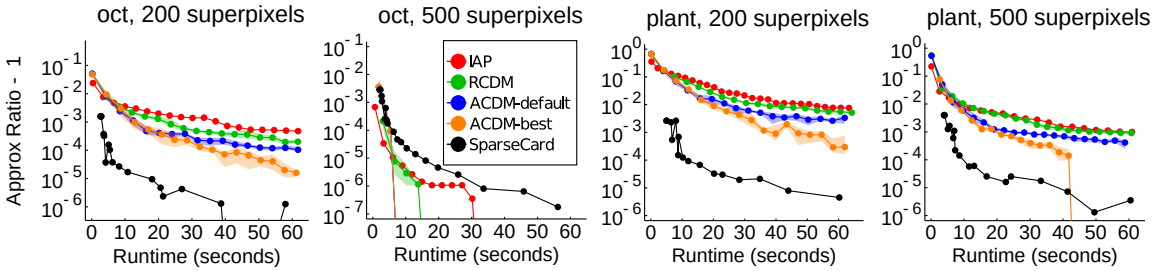


Figure 5: Approximation factor minus 1 vs. runtime for solutions returned by SparseCard and competing methods on four image segmentation tasks. SparseCard obtains faster approximate solutions on all but the easiest instance, where all methods obtain good results within a few seconds. We display the average of 5 runs for competing methods, with lighter colored region showing upper and lower bounds from these runs. SparseCard is deterministic and was run once for each ε on a decreasing logarithmic scale. Our method maintains an advantage even against post-hoc best case parameters for competing approaches: ACDM-best is the best result obtained by running ACDM for a range of empirical parameters c for each data set and reporting the best result. The default is $c = 10$ (blue curve). Best post-hoc results for the plots from left to right were $c = 25, 10, 50, 25$. It is unclear how to determine the best c in advance.

Comparison against continuous optimization algorithms. We also compare against recent C++ implementations of ACDM, RCDM, and Incidence Relation AP, the last of which is an improved version of the standard AP method (Nishihara et al., 2014) provided by Li and Milenkovic (2018a). These use the divide-and-conquer method of Jegelka et al. (2013), implemented specifically for concave cardinality functions, to solve the quadratic minimization oracle $\mathcal{O}_e$ for region potential functions. Although these continuous optimization methods come with no a priori approximation guarantees, we can compare them against SparseCard by computing a posteriori approximations obtained using intermediate solutions returned after every few hundred iterations. In more detail, we extract the best level set $S_\lambda = \{i \colon x_i > \lambda\}$ from the vector of dual variables $\mathbf{x} = (x_i)$ at various steps, and compute the approximation ratio $\operatorname{argmin}_\lambda f(S_\lambda)/f(S^*)$ where S^* is the optimal solution determined via the exact max-flow solution. Figure 5 displays approximation ratio versus

runtime for four DSFM instances (two data sets $\times$ two superpixel segmentations). SparseCard was run for a range of ε values on a decreasing logarithmic scale from 1 to 10^{-4}, and obtains significantly better results on all but the *octopus* with 500 superpixels instance. This is the easiest instance; all methods obtain a solution within a factor 1.001 of optimality within a few seconds. ACDM depends on a hyperparameter c controlling the number of iterations in an outer loop. Even when we choose the best post-hoc c value for each data set, SparseCard maintains its overall advantage. Appendix C provides additional details regarding the competing algorithms and their parameter settings.

We focus on comparisons with continuous optimization methods rather than other discrete optimization methods, as the former are better equipped for our goal of finding approximate solutions to DSFM problems involving functions of large support. To our knowledge, no implementations for the methods of Kolmogorov (2012) or Axiotis et al. (2021) exist. Meanwhile, IBFS (Fix et al., 2013) is designed for finding exact solutions when all support sizes are small. Recent empirical results (Ene et al., 2017) confirm that this method is not designed to handle the large region potential functions we consider.

7.2 Hypergraph local clustering

Graph reduction techniques have been frequently and successfully used as subroutines for hypergraph local clustering and semi-supervised learning methods (Liu et al., 2021; Veldt et al., 2020a; Li and Milenkovic, 2017; Yin et al., 2017). Replacing exact reductions with our approximate reductions can lead to significant runtime improvements without sacrificing on accuracy, and opens the door to running local clustering algorithms on problems where exact graph reduction would be infeasible. We illustrate this by using SparseCard as a subroutine for a method we previously designed called HyperLocal (Veldt et al., 2020a). This algorithm finds local clusters in a hypergraph by repeatedly solving hypergraph minimum s-t cut problems, which could also be viewed as instances of Card-DSFM. HyperLocal was originally designed to handle only the δ-*linear* penalty $\mathbf{f}_e(A) = \min\{|A|, |e\backslash A|, \delta\}$, for parameter $\delta \geq 1$, which can already be modeled sparsely with a single CB-gadget. SparseCard makes it possible to sparsely model any concave cardinality penalty. We specifically use approximate reductions for the weighted clique penalty $\mathbf{f}_e(A) = (|e|-1)^{-1}|A||e\backslash A|$, the square root penalty $\mathbf{f}_e(A) = \sqrt{\min\{|A|, |e\backslash A|\}}$, and the sublinear power function penalty $\mathbf{f}_e(A) = (\min\{|A|, |e\backslash A|\})^{0.9}$, all of which require $O(|e|^2)$ edges to model exactly using previous reduction techniques. Weighted clique penalties in particular have been used extensively in hypergraph clustering (Agarwal et al., 2006; Yin et al., 2017; Kumar et al., 2020; Zien et al., 1999), including by methods specifically designed for local clustering and semi-supervised learning (Li et al., 2018; Yin et al., 2017; Zhou et al., 2006).

Stackoverflow hypergraph. We consider a hypergraph clustering problem where nodes are 15.2M questions on `stackoverflow.com` and each of the 1.1M hyperedges defines a set of questions answered by the same user. The mean hyperedge size is 23.7, the maximum size is over 60k, and there are 2165 hyperedges with at least 1000 nodes. Questions with the same topic tag (e.g., "common-lisp") constitute small labeled clusters in the data set. We previously showed that HyperLocal can detect clusters quickly with the δ-linear penalty by solving localized s-t cut problems near a seed set. Applying exact graph reductions for other concave cut penalties is infeasible, due to the extremely large hyperedge sizes, and

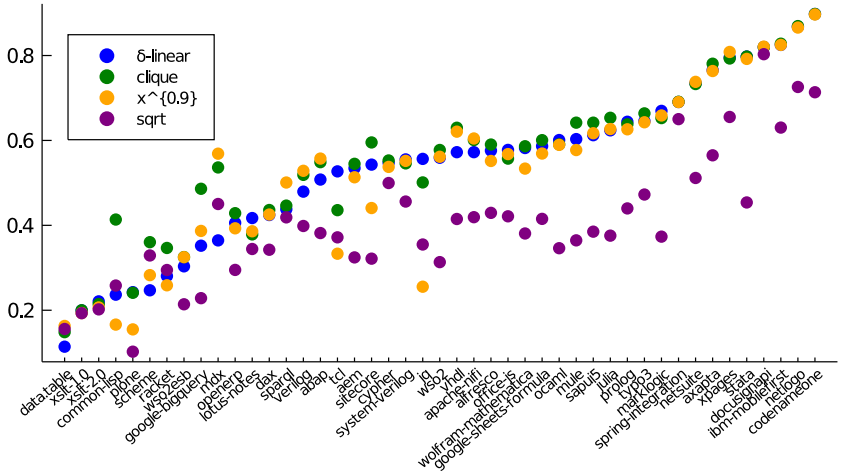


Figure 6: Average F1 score obtained for each localized clustering using 4 different hyperedge cut penalties. For clique, $x^{0.9}$, and the square root penalties, we used an approximate reduction with $\varepsilon = 1.0$. The clique penalty had the highest average F1 score on 26 clusters, the δ-linear had the highest on 10 clusters, and $x^{0.9}$ had the highest average score on the remaining 9 clusters.

we found that using a clique expansion after simply removing large hyperedges performed poorly (Veldt et al., 2020a). Using SparseCard as a subroutine opens up new possibilities.

Experimental setup and parameter settings. We follow our previously used experimental setup (Veldt et al., 2020a) for detecting localized clusters in the Stackoverflow hypergraph. We focus on a set of 45 local clusters, all of which are question topics involving between 2,000 and 10,000 nodes. For each cluster, we generate a random seed set by selecting 5% of the nodes in the target cluster uniformly at random, and then add neighboring nodes to the seed set to grow it into a larger input set Z to use for HyperLocal (see Veldt et al. 2020a for details). We set $\delta = 5000$ for the δ-linear hyperedge cut function and set a locality parameter for HyperLocal to be 1.0 for all experiments. With this setup, using HyperLocal with the δ-linear penalty will then reproduce our original experimental settings (Veldt et al., 2020a). Our goal is to show how using SparseCard leads to fast and often improved results for alternative penalties that could not previously been used.

Experimental results. In Figure 6 we show the mean F1 score for each cluster (across the ten random seed sets) obtained by the δ-linear penalty and the three alternative penalties when $\varepsilon = 1.0$. The clique penalty obtained the highest average F1 score on 26 clusters, the δ-linear obtained the highest average score on 10 clusters, and the sublinear penalty obtained the best average score on the remaining 9 clusters. Importantly, cut penalties that previously could not be used on this data set (*clique*, $x^{0.9}$) obtain the best results for most clusters. The square root penalty does not perform particularly well on this data set, but it is instructive to consider its runtime (Figure 7f). Theorem 17 shows that asymptotically this function has a worst-case behavior in terms of the number of CB-gadgets needed to approximate

it. We nevertheless obtain reasonably fast results for this penalty function, indicating that our techniques can provide effective sparse reductions for any concave cardinality function of interest. We also ran experiments with $\varepsilon = 0.1$ and $\varepsilon = 0.01$, which led to noticeable increases in runtime but only very slight changes in F1 scores (Figure 7). This indicates why exact reductions are not possible in general, while also showing that our sparse approximate reductions serve as fast and very good proxies for exact reductions.

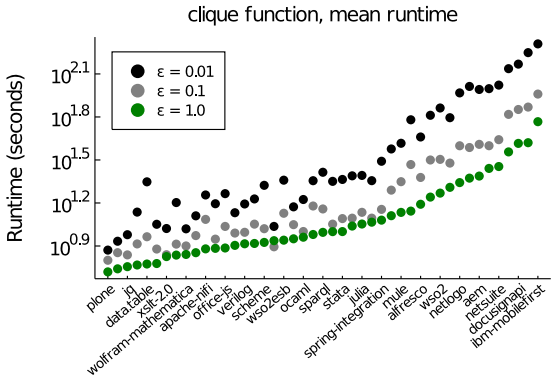
8. Conclusion and Discussion

We have introduced the notion of an augmented cut sparsifier, which approximates a generalized hypergraph cut function with a sparse directed graph on an augmented node set. Our approach relies on a connection we highlight between graph reduction strategies and piecewise linear approximations to concave functions. Our framework leads to more efficient techniques for approximating hypergraph s-t cut problems via graph reduction, improved sparsifiers for co-occurrence graphs, and fast algorithms for approximately minimizing cardinality-based decomposable submodular functions.

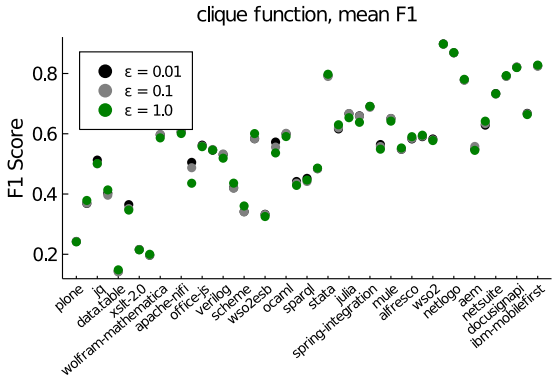
A natural question is how to choose an appropriate approximation parameter ε in practice. Our experimental results first of all indicate that choosing even a fairly large value (e.g., $\varepsilon = 1$) leads to a significant runtime savings at an extremely small cost to performance. The approximation guarantees we observed in practice are far better than the a priori $(1+\varepsilon)$ approximation guarantee (Table 2). In our clustering experiments, choosing $\varepsilon = 1$ led to no meaningful difference from choosing $\varepsilon = 0.01$ in terms of detecting ground truth clusters (Figure 7). More generally, if practitioners do not have a specific requirement for the approximation factor but have specific constraints on computational resources, our theoretical results can be used to check how many edges are needed to reduce the hypergraph to a graph for a range of different tolerance values ε. Doing so can provide estimates for runtime and memory costs without having to explicitly form the reduced graph. One can then choose the smallest ε that ensures a reasonable size for the reduced graph.

As noted in Section 1.2, an interesting open question is to establish and study analogous notions of augmented *spectral* sparsification, given that spectral sparsifiers provide a useful generalization of cut sparsifiers in graphs (Spielman and Teng, 2011). One way to define such a notion is to apply existing definitions of submodular hypergraph Laplacians (Li and Milenkovic, 2018b; Yoshida, 2019) to both the original hypergraph and its sparsifier. This requires viewing our augmented sparsifier as a hypergraph with splitting functions of the form $\mathbf{w}_e(A) = a \cdot \min\{|A|, |e \backslash A|, b\}$, corresponding to hyperedges with cut properties that can be modeled by a cardinality-based gadget. From this perspective, augmented spectral sparsification means approximating a generalized hypergraph cut function with another hypergraph cut function involving simplified splitting functions. While this provides one possible definition for augmented spectral sparsification, it is not clear whether the techniques we have developed can be used to satisfy this definition. Furthermore, it is not clear whether obtaining such a sparsifier would imply any immediate runtime benefits for approximating the spectra of generalized hypergraph Laplacians, or for solving generalized Laplacian systems (Fujii et al., 2021; Li et al., 2020). We leave these for future work.

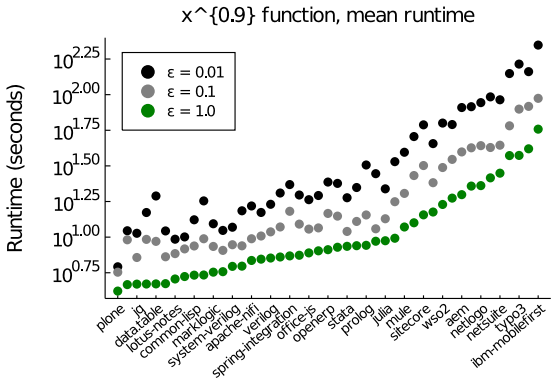
While our work provides the optimal reduction strategy in terms of cardinality-based gadgets, this is more restrictive than optimizing over all possible gadgets for approximately



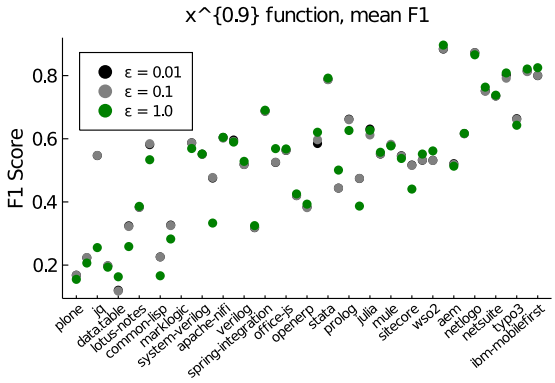
(a) Clique Runtime



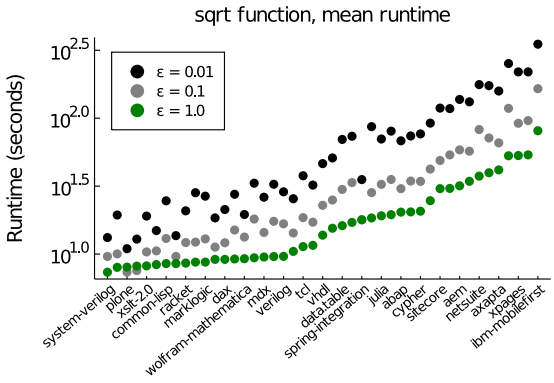
(b) Clique F1 Scores



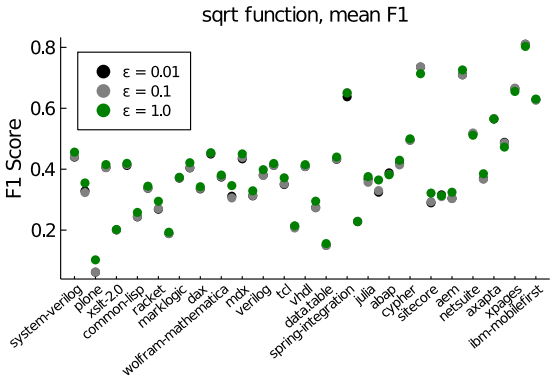
(c) $x^{0.9}$ Runtime



(d) $x^{0.9}$ F1 Scores



(e) Square Root Runtime



(f) Square Root F1 Scores

Figure 7: We display differences in runtimes and F1 scores when using different ε values when approximating three hyperedge cut penalties. Using a larger ε provides a significant runtime savings with virtually no affect on F1 scores. Clusters have been re-arranged in the horizontal axis for each hyperedge cut penalty for easier visualization.

modeling hyperedge cut penalties. Optimizing over a broader space of gadgets poses another interesting direction for future work, but is more challenging in several ways. First of all, it is unclear how to even define an optimal reduction when optimizing over arbitrary gadgets, since it is preferable to avoid both adding new nodes and adding new edges, but the tradeoff between these two goals is not clear. Another challenge is that the best reduction may depend heavily on the splitting function we wish to reduce, which makes developing a general approach difficult. A natural next step would be to at least better understand lower bounds on the number of edges and auxiliary nodes needed to model different cardinality-based splitting functions. While we do not have any concrete results, there are several indications that cardinality-based gadgets may be nearly optimal in many settings. For example, star expansions and clique expansions provide a more efficient way to model linear and quadratic splitting functions respectively, but modeling these functions with cardinality-based gadgets only increases the number of edges by roughly a factor two.

Finally, we find it interesting that using auxiliary nodes and directed edges makes it possible to sparsify the complete graph using only $O(n\varepsilon^{-1/2}\log\log\frac{1}{\varepsilon})$ edges, whereas standard sparsifiers require $O(n\varepsilon^{-2})$. We would like to better understand whether both directed edges *and* auxiliary nodes are necessary for making this possible, or whether improved approximations are possible using only one or the other.

Acknowledgments

This research was supported by NSF Award DMS-1830274, ARO Award W911NF19-1-0057, ARO MURI, JPMorgan Chase & Co., a Simons Investigator Award, a Vannevar Bush Faculty Fellowship, and a grant from the AFOSR. The authors thank Pan Li for helpful conversations about decomposable submodular function minimization.

Appendix A. Proofs of Lemmas in Section 3

A.1 Proof of Lemma 9

Proof Define $b_0 = 0$ for notational convenience. The first three conditions in Definition 8 can be seen by inspection, recalling that $0 < a_j$ and $0 < b_j \leq r$ for all $j \in [J]$. Observe that $\hat{\mathbf{f}}$ is linear over the interval $[b_{i-1}, b_i)$ for $i \in [J]$, since for $x \in [b_{i-1}, b_i)$,

$$\hat{\mathbf{f}}(x) = \sum_{j=1}^{J} a_j \cdot \min\{x, b_j\} = \sum_{j=1}^{i-1} a_j b_j + x \cdot \sum_{j=i}^{J} a_j \,.$$

In other words, the ith linear piece of $\hat{\mathbf{f}}$, defined over $x \in [b_{i-1}, b_i)$ is given by $\hat{\mathbf{f}}^{(i)}(x) = I_i + S_i x$, where the intercept and slope terms are given by $I_i = \sum_{j=1}^{i-1} a_j b_j$ and $S_i = \sum_{j=i}^{J} a_j$. For the first J intervals of the form $[b_{i-1}, b_i)$, the slopes are always positive but strictly decreasing. Thus, there are exactly J positive-sloped linear pieces. The final linear piece is a flat line, since $\hat{\mathbf{f}}(x) = \sum_{j=1}^{J} a_j b_j$ for all $x \geq b_J$. The concavity of $\hat{\mathbf{f}}$ follows directly from the fact that it is a continuous and piecewise linear function with decreasing slopes. ■

A.2 Proof of Lemma 10

Proof Since $\mathbf{f}$ is in $\mathcal{F}_r$, it has J positive-sloped linear pieces and one flat linear piece, and therefore it has exactly J breakpoints: $0 < b_1 < b_2 < \ldots < b_J$. Let $\mathbf{b} = (b_j)$ be the vector storing these breakpoints. For convenience we define $b_0 = 0$, though b_0 is not stored in $\mathbf{b}$. By definition, $\mathbf{f}$ is constant for all $x \geq r$, which implies that $b_J \leq r$.

Let $f_i = \mathbf{f}(b_i)$. For $i \in [J]$, the positive slope of the ith linear piece of $\mathbf{f}$, which occurs in the range $[b_{i-1}, b_i]$, is given by

$$m_i = \frac{f_i - f_{i-1}}{b_i - b_{i-1}}. \tag{45}$$

The ith linear piece of $\mathbf{f}$ is given by

$$\mathbf{f}^{(i)}(x) = m_i(x - b_{i-1}) + f_{i-1} \qquad \text{for } x \in [b_{i-1}, b_i]. \tag{46}$$

The last linear piece of $\mathbf{f}$ is a flat line over the interval $x \in [b_J, \infty)$, i.e., $m_{J+1} = 0$. Since $\mathbf{f}$ has positive and strictly decreasing slopes, we can see that $a_i = m_i - m_{i+1} > 0$ for all $i \in [J]$.

Let $\hat{\mathbf{w}}$ be the order-J CCB function constructed from vectors $(\mathbf{a}, \mathbf{b})$, and let $\hat{\mathbf{f}}$ be its resulting continuous extension:

$$\hat{\mathbf{f}} = \sum_{j=1}^{J} a_j \cdot \min\{x, b_j\}. \tag{47}$$

We must check that $\hat{\mathbf{f}} = \mathbf{f}$. By Lemma 9, we know that $\hat{\mathbf{f}}$ is in $\mathcal{F}_r$ and has exactly $J + 1$ linear pieces. The functions will be the same, therefore, if they share the same values at breakpoints. Evaluating $\hat{\mathbf{f}}$ at an arbitrary breakpoint b_i gives:

$$\hat{\mathbf{f}}(b_i) = \left(\sum_{j=1}^{i-1} a_j \cdot b_j\right) + b_i \cdot \left(\sum_{j=i}^{J} a_j\right) = \left(\sum_{j=1}^{i-1} a_j \cdot b_j\right) + b_i \cdot m_i. \tag{48}$$

We first confirm that the functions coincide at the first breakpoint:

$$\hat{\mathbf{f}}(b_1) = b_1 \cdot m_1 = b_1 \cdot \frac{f_1 - f_0}{b_1 - b_0} = b_1 \frac{f_1}{b_1} = f_1.$$

For any fixed $i \in \{2, 3, \ldots, J\}$,

$$\begin{aligned}
\hat{\mathbf{f}}(b_i) - \hat{\mathbf{f}}(b_{i-1}) &= \left(\sum_{j=1}^{i-1} a_j b_j\right) + b_i m_i - \left(\sum_{j=1}^{i-2} a_j b_j\right) - b_{i-1} m_{i-1} \\
&= a_{i-1} b_{i-1} + b_i m_i - b_{i-1} m_{i-1} = (m_{i-1} - m_i) b_{i-1} + b_i m_i - b_{i-1} m_{i-1} \\
&= m_i(b_i - b_{i-1}) = f_i - f_{i-1}.
\end{aligned}$$

Since $\mathbf{f}(b_1) = \hat{\mathbf{f}}(b_1)$ and $\mathbf{f}(b_i) - \mathbf{f}(b_{i-1}) = \hat{\mathbf{f}}(b_i) - \hat{\mathbf{f}}(b_{i-1})$ for $i \in \{2, 3, \ldots, t\}$, we have $\mathbf{f}(b_i) = \hat{\mathbf{f}}(b_i)$ for $i \in [J]$. Therefore, $\mathbf{f}$ and $\hat{\mathbf{f}}$ are the same piecewise linear function. ■

Appendix B. Sparsification for Generalized Splitting Functions

In Sections 2 and 3 we focused on sparsification techniques for representing splitting functions that are symmetric and penalize only *cut* hyperedges:

$$\mathbf{w}_e(S) = \mathbf{w}_e(e \backslash S) \text{ for all } S \subseteq e$$
$$\mathbf{w}_e(e) = \mathbf{w}_e(\emptyset) = 0.$$

These assumptions are standard for generalized hypergraph cut problems (Veldt et al., 2022; Li and Milenkovic, 2017, 2018b), and lead to the clearest exposition of our main results. In this appendix, we extend our sparse approximation techniques so that they apply even if we remove these restrictions. This will allow us to obtain improved techniques for approximately solving a certain class of decomposable submodular functions (see Section 6). Formally, our goal is to minimize

$$\underset{S \subseteq V}{\text{minimize}} \, f(S) = \sum_{e \in \mathcal{E}} \mathbf{w}_e(S \cap e), \tag{49}$$

where each $\mathbf{w}_e$ is a submodular cardinality-based function, that is not necessarily symmetric and does not need to equal zero when the hyperedge e is uncut. Our proof strategy for reducing this more general problem to a graph s-t cut problem closely follows the same basic set of steps used in Section 3 for the special case.

B.1 Submodularity Constraints for Cardinality-Based Functions

We first provide a convenient characterization of general cardinality-based submodular functions. By *general* we mean the splitting function does not need to be symmetric nor does it need to have a zero penalty when the hyperedge is uncut.

Lemma 22 *Let $\mathbf{w}_e$ be a general submodular cardinality-based splitting function on a k-node hyperedge e, and let w_i denote the penalty for any $A \subseteq e$ with $|A| = i$. Then for $i \in \{1, 2, \ldots, k-1\}$, we have the inequality $2w_i \geq w_{i-1} + w_{i+1}$.*

Proof Let $v_1, v_2, \ldots, v_k$ denote the nodes in the hyperedge. Submodularity means that for all $A, B \subseteq e$, $\mathbf{w}(A) + \mathbf{w}(B) \geq \mathbf{w}(A \cup B) + \mathbf{w}(A \cap B)$. In order to show the desired inequality, simply set $A = \{v_1, v_2, \ldots, v_i\}$ and $B = \{v_2, v_2, \ldots, v_{i+1}\}$ and the result follows. ■

To simplify our analysis, as we did for the symmetric case, we will define a set of functions that is virtually identical to these splitting functions on k-node hyperedges, but are defined over integers from 0 to k rather than on subsets of a hyperedge.

Definition 23 *A k-GSCB (Generalized Submodular Cardinality-Based) integer function is a function $\mathbf{w} : \{0\} \cup [k] \to \mathbb{R}^+$ satisfying $2\mathbf{w}(i) \geq \mathbf{w}(i-1) + \mathbf{w}(i+1)$ for all $i \in [k-1]$.*

B.2 Combining Gadgets for Generalized SCB Functions

Our goal is to show how to approximate k-GSCB integer functions using piecewise linear functions with few linear pieces. This in turn corresponds to approximating a hyperedge splitting function with a sparse gadget. In order for this to work for our more general class of splitting functions, we use a slight generalization of an asymmetric gadget we introduced in previous work (Veldt et al., 2022).

Definition 24 *The asymmetric cardinality-based gadget (ACB-gadget) for a k-node hyperedge e is parameterized by scalars a and b and constructed as follows:*

- *Introduce an auxiliary vertex v_e.*
- *For each $v \in e$, introduce a directed edge from v to v_e with weight $a \cdot (k-b)$, and a directed edge from v_e to v with weight $a \cdot b$.*

The ACB-gadget models the following k-GSCB integer function:

$$\mathbf{w}_{a,b}(i) = a \cdot \min\{i \cdot (k-b), (k-i) \cdot b\}. \tag{50}$$

To see why, consider where we must place the auxiliary node v_e when solving a minimum s-t cut problem involving the ACB-gadget. If we place i nodes on the s-side, then placing v_e on the s-side has a cut penalty of $ab(k-i)$, whereas placing v_e on the t-side gives a penalty of $ai(k-b)$. To minimize the cut, we choose the smaller of the two options.

Previously we showed that asymmetric splitting functions can be modeled exactly by a combination of $k-1$ ACB-gadgets (Veldt et al., 2022). As we did in Section 3 for symmetric splitting functions, we will show here that a much smaller number of gadgets suffices if we are content to approximate the cut penalties of an asymmetric splitting function. In our previous work we enforced the constraint $\mathbf{w}_e(\emptyset) = \mathbf{w}_e(0) = 0$ even for asymmetric splitting functions, but we remove this constraint here. In order to model the cut properties of an arbitrary GSCB splitting function, we define a combined gadget involving multiple ACB-gadgets, as well as edges from each node $v \in e$ to the source and sink nodes of the graph. The augmented cut function for the resulting directed graph $\hat{G} = (V \cup \mathcal{A} \cup \{s,t\}, \hat{E})$ will then be given by $\mathbf{cut}_{\hat{G}}(S) = \min_{T \subseteq \mathcal{A}} \mathbf{dircut}_{\hat{G}}(\{s\} \cup S \cup T)$ for a set $S \subseteq V$, where **dircut** is the directed cut function on $\hat{G}$. Finding a minimum s-t cut in $\hat{G}$ will solve objective (49), or equivalently, the cardinality-based DSFM problem.

Definition 25 *A k-CG function (k-node, combined gadget function) $\hat{\mathbf{w}}$ of order J is a k-GSCB integer function that is parameterized by scalars z_0, z_k, and (a_j, b_j) for $j \in [J]$. The function has the form:*

$$\hat{\mathbf{w}}(i) = z_0 \cdot (k-i) + z_k \cdot i + \sum_{j=1}^{J} a_j \min\{i \cdot (k-b_j), (k-i) \cdot b_j\}. \tag{51}$$

The scalars parameterizing $\hat{\mathbf{w}}$ satisfy

$$\begin{aligned} &b_j > 0, a_j > 0 \text{ for all } j \in [J] \\ &b_j < b_{j+1} \text{ for all } j \in [J-1] \\ &b_J < k \text{ and } z_0 \geq 0 \text{ and } z_k \geq 0. \end{aligned}$$

Conceptually, the function shown in (51) represents a combination of J ACB-gadgets for a hyperedge e, where additionally for each node $v \in e$ we have place a directed edge from a source node s to v of weight z_0, and an edge from v to a sink node t with weight z_k.

The continuous extension of the k-CG function in (51) is defined to be:

$$\hat{\mathbf{f}}(x) = z_0 \cdot (k - x) + z_k \cdot x + \sum_{j=1}^{J} a_j \min\{x \cdot (k - b_j), (k - x) \cdot b_j\} \text{ for } x \in [0, k]. \tag{52}$$

Lemmas 26 and 27 are analogous to Lemmas 9 and 10 for symmetric splitting functions.

Lemma 26 *The continuous extension $\hat{\mathbf{f}}$ of $\hat{\mathbf{w}}$ is nonnegative over the interval $[0, k]$, piecewise linear, concave, and has exactly $J + 1$ linear pieces.*

Proof Nonnegativity follows quickly from the positivity of z_0, z_k, and (a_i, b_i) for $i \in [J]$, and $b_J < k$. For other properties, we begin by re-writing the function as

$$\hat{\mathbf{f}}(x) = z_0 \cdot (k - x) + z_k \cdot x + \sum_{j=1}^{J} a_j \min\{x \cdot (k - b_j), (k - x) \cdot b_j\} \tag{53}$$

$$= kz_0 + x(z_k - z_0) + k \cdot \sum_{j=1}^{J} a_j \min\{x, b_j\} - x \cdot \sum_{j=1}^{J} a_j b_j \tag{54}$$

$$= kz_0 + x(z_k - z_0) + kx \cdot \sum_{j: x < b_j} a_j + k \cdot \sum_{j: x \geq b_j} a_j b_j - x \cdot \sum_{j=1}^{J} a_j b_j. \tag{55}$$

Define

$$\beta = \sum_{j=1}^{J} a_j b_j, \qquad \beta_t = \sum_{j=1}^{t} a_j b_j, \qquad \alpha_t = \sum_{j=t+1}^{J} a_j,$$

and observe that β_t is strictly increasing with t, and α_t is strictly *decreasing* with t. Define $b_0 = 0$ and $b_{J+1} = k$ for notational convenience. For any $t \in \{0\} \cup [J]$, the function is linear over the interval $[b_t, b_{t+1})$, since for $x \in [b_t, b_{t+1})$, we have

$$\begin{aligned}
\hat{\mathbf{f}}(x) &= kz_0 + x(z_k - z_0) + kx \cdot \sum_{j: x < b_j} a_j + k \cdot \sum_{j: x \geq b_j} a_j b_j - x \cdot \sum_{j=1}^{J} a_j b_j \\
&= kz_0 + x(z_k - z_0) + kx \sum_{j=t+1}^{J} a_j + k \cdot \sum_{j=1}^{t} a_j b_j - x \sum_{j=1}^{J} a_j b_j \\
&= kz_0 + x(z_k - z_0) + kx\alpha_t + k\beta_t - x\beta.
\end{aligned}$$

Thus, $\hat{\mathbf{f}}$ is piecewise linear. Furthermore, the slope of the line over the interval $[b_t, b_{t+1})$ is $(z_k - z_0 - \beta + k\alpha_t)$, which strictly decreases as t increases. The fact that all slopes are distinct means that there are exactly $J + 1$ linear pieces, and the fact that these slopes are decreasing means that the function is concave over the interval $[0, k]$. ■

Lemma 27 *For every function $\mathbf{f}$ that is nonnegative, piecewise linear with $J + 1$ linear pieces, and concave over the interval $[0, k]$, there exists some k-CG function $\hat{\mathbf{w}}$ of order J such that $\mathbf{f}$ is the continuous extension of $\hat{\mathbf{w}}$.*

Proof The function $\hat{\mathbf{w}}$ will be defined by choosing parameters z_0, z_k, and (a_j, b_j) for $j \in [J]$. Let $\hat{\mathbf{f}}$ denote the continuous extension of the function $\hat{\mathbf{w}}$ that we will build. From the proof of Lemma 26, we know that the parameter b_j will correspond to the jth breakpoint of $\hat{\mathbf{f}}$. Therefore, given $\mathbf{f}$, we set b_j to be the jth breakpoint of the function $\mathbf{f}$, so that the functions match at breakpoints. For convenience, we also set $b_0 = 0$ and $b_{J+1} = k$. We then set $z_0 = \mathbf{f}(0)/k$ and $z_k = \mathbf{f}(k)/k$, to guarantee that $\hat{\mathbf{f}}(0) = \mathbf{f}(0)$ and $\hat{\mathbf{f}}(k) = \mathbf{f}(k)$. In order to set the a_j values, we first compute the slopes of each line of $\mathbf{f}$. Let $f_j = \mathbf{f}(b_j)$ for $j \in \{0\} \cup [J+1]$. The jth linear piece of $\mathbf{f}$ has the slope:

$$m_i = \frac{f_i - f_{i-1}}{b_i - b_{i-1}}.$$

Finally, for $j \in [J]$ we set $a_j = \frac{1}{k}(m_j - m_{j+1})$. All of our chosen parameters satisfy the conditions of Definition 25, so it simply remains to check that $\mathbf{f}$ and $\hat{\mathbf{f}}$ coincide at breakpoints.

Let $t \in [J]$. Using Equation (55) to evaluate $\hat{\mathbf{f}}$ at breakpoint b_t, we get

$$\hat{\mathbf{f}}(b_t) = f_0 + \frac{b_t}{k}(f_k - f_0) + kb_t \sum_{j=t+1}^{J} a_j + k\sum_{j=1}^{t} a_j b_j - b_t \sum_{j=1}^{J} a_j b_j. \tag{56}$$

We can simplify several terms using the fact that $a_j = \frac{1}{k}(m_j - m_{j+1})$. First of all,

$$k\sum_{j=t+1}^{J} a_j = \sum_{j=t+1}^{J} [m_j - m_{j+1}] = m_{t+1} - m_{J+1}.$$

Furthermore,

$$\begin{aligned} k\sum_{j=1}^{t} a_j b_j &= \sum_{j=1}^{t}(m_j - m_{j+1})b_j = m_1 b_1 - m_{t+1}b_t + \sum_{j=2}^{t} m_j(b_j - b_{j-1}) \\ &= (f_1 - f_0) - m_{t+1}b_t + \sum_{j=2}^{t}[f_j - f_{j-1}] = (f_1 - f_0) - m_{t+1}b_t + f_t - f_1 \\ &= f_t - f_0 - m_{t+1}b_t. \end{aligned}$$

Similarly, we see that $\sum_{j=1}^{J} a_j b_j = \frac{1}{k}\left(f_J - f_0 - m_{J+1}b_J\right)$. Plugging this into Equation (56):

$$\begin{aligned} \hat{\mathbf{f}}(b_t) &= f_0 + \frac{b_t}{k}(f_{J+1} - f_0) + b_t(m_{t+1} - m_{J+1}) + f_t - f_0 - m_{t+1}b_t - \frac{b_t}{k}\left(f_J - f_0 - m_{J+1}b_J\right) \\ &= \frac{b_t}{k} f_{J+1} - b_t m_{J+1} + f_t - \frac{b_t}{k}\left(f_J - m_{J+1}b_J\right) \\ &= f_t + \frac{b_t}{k}(f_{J+1} - f_J) - b_t m_{J+1}\left(1 - \frac{b_J}{k}\right) \\ &= f_t + \frac{b_t}{k}(f_{J+1} - f_J) - b_t\left(\frac{f_{J+1} - f_J}{k - b_J}\right)\left(\frac{k - b_J}{k}\right) = f_t = \mathbf{f}(b_t). \end{aligned}$$

So we see that $\mathbf{f} = \hat{\mathbf{f}}$ at breakpoints, and therefore these be the same piecewise linear function. ■

B.3 Finding the Best Piecewise Approximation

As we did for symmetric splitting functions, we can quickly find the best piecewise linear $(1+\varepsilon)$-approximation to a k-GSCB integer function $\mathbf{w}$ using a greedy approach. We omit proof details, as they exactly mirror arguments provided for the symmetric case. The submodularity constraint $2\mathbf{w}(i) \geq \mathbf{w}(i+1) + \mathbf{w}(i-1)$ for $i \in \{0\} \cup [k]$ can be viewed as a discrete version of concavity, and will ensure that the piecewise linear function returned by such a procedure will also be nonnegative and concave. After obtaining the piecewise linear approximation, we can apply Lemma 27 to reverse engineer a k-CG function of a small order that approximates $\mathbf{w}$. We obtain the same asymptotic upper bound on the number of linear pieces needed to approximate $\mathbf{w}$.

Lemma 28 *Let $\mathbf{w}$ be a k-GSCB integer function and $\varepsilon \geq 0$. There exists a k-CG function $\hat{\mathbf{w}}$ of order $J = O(\frac{1}{\varepsilon}\log k)$ that satisfies $\mathbf{w}(i) \leq \hat{\mathbf{w}}(i) \leq (1+\varepsilon)\mathbf{w}(i)$ for any $i \in \{0\} \cup [k]$.*

B.4 Approximating Cardinality-Based Sum of Submodular Functions

Recall that k-CG functions correspond to combinations of ACB-gadgets for a hyperedge e as well as directed edges between nodes in e and the source and sink nodes in some minimum s-t cut problem. Each ACB-gadget involves one new auxiliary node and $2|e|$ directed edges, and the number of ACB-gadgets is equal to the order of the k-CG function (the number of linear pieces minus one). Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with $n = |V|$ nodes, where each splitting function is submodular, cardinality-based, and is not required to be symmetric or penalize only cut hyperedges. Finding the minimum cut in $\mathcal{H}$ corresponds to solving the sum of submodular splitting functions given in (49). For $\varepsilon \geq 0$, we can preserve cuts in $\mathcal{H}$ to within a factor $(1+\varepsilon)$ by introducing a source and sink node s and t and applying our sparse reduction techniques to each hyperedge to obtain a directed graph $\hat{G} = (V \cup \mathcal{A} \cup \{s,t\}, \hat{E})$, where $\mathcal{A}$ is the set of auxiliary nodes, with $N = O(n + \frac{1}{\varepsilon}\sum_{e\in\mathcal{E}} \log|e|)$ nodes and $M = O(n + \frac{1}{\varepsilon}\sum_{e\in\mathcal{E}} \log|e|)$ edges. Even if the size of each $e \in \mathcal{E}$ is $O(n)$, we have $N = O(n + \varepsilon^{-1}|\mathcal{E}|\log n)$ and $M = O(\varepsilon^{-1}\log n \sum_{e\in\mathcal{E}}|e|)$.

Appendix C. Parameter Settings for Image Segmentation Experiments

The continuous optimization methods for DSFM that we compare against are implemented in C++ with a MATLAB front end. The Incidence Relation AP (IAP) method is an improved version of the AP method (Nishihara et al., 2014). Li and Milenkovic (2018a) showed that the runtime of the method can be significantly faster if one accounts for so-called *incidence relations*, which describe sets of nodes that define the support of a component function. In our experiments we also ran the standard AP algorithm as implemented by Li and Milenkovic, but this always performed noticeably worse that IAP in practice, so we only report results for IAP. Neither of these methods require setting any hyperparameters.

Li and Milenkovic (2018a) also showed that accounting for incidence relations leads to improved *parallel* runtimes for ACDM and RCDM, but this does not improve serial runtimes. To simulate improved parallel runtimes, the implementations ACDM and RCDM of these authors include a parallelization parameter $\alpha = K/R$, where K is the number of projections performed in an inner loop of these methods, and R is the number of component functions. In theory, the K projections could be performed in parallel, leading to faster

overall runtimes. The comparative parallel performance between methods can be simulated by seeing how quickly the methods converge in terms of the number of total projections performed. Note however that the implementations themselves are serial, and only simulate what could happen in a parallel setting. Li and Milenkovic (2018a) demonstrated that the minimum number of total projections needed to achieve convergence to within a small tolerance is typically achieved when α is quite small. When projections are performed in parallel, choosing a larger α may still be advantageous. However, since our goal is to obtain the fast serial runtimes, we chose a small value $\alpha = 0.01$, based on the results of Li and Milenkovic. We also tried larger and smaller values of α in post-hoc experiments on all four instances of DSFM, though this led to little variation in performance.

In addition to α, ACDM relies on an empirical parameter c controlling the number of iterations in an outer loop. We used the recommended default parameter $c = 10$. In general it is unclear how to set this parameter a priori to obtain better than default behavior on a given DSFM instance. In order to highlight the strength of SparseCard relative to ACDM, we additionally tried post-hoc tuning of c on each data set to see how much this could affect results. We ran ACDM on each of the four instances of DSFM (2 image data sets $\times$ 2 superpixel segmentations each) for all $c \in \{10, 25, 50, 100, 200\}$, three different times, for $50R$ projections (i.e., on average we visit and perform a projection step at each component function 50 times). This took roughly 30-45 seconds for each run. We then computed the average duality gap for each c and each instance over the three trials, and re-ran the algorithm for even longer using the best value of c for each instance. The result is shown as ACDM-best in Figure 5. SparseCard still maintains a clear advantage over this method on the instances that involve 200 superpixels (i.e., very large region potentials). Our method also obtains better approximations for the *smallplant* data set with 500 superpixels for the first 40 seconds, after which point ACDM-best converges to the optimal solution. Nevertheless, we would not have seen this improved behavior without post-hoc tuning of the hyperparameter c for ACDM. Meanwhile, SparseCard only relies on the parameter ε, and it is very easy to understand how setting this parameter affects the algorithm as it directly controls the sparsity and a priori approximation guarantee for our method.

We remark finally that Li and Milenkovic (2018a) also considered and implemented an alternate version of RCDM (RCDM-greedy) with a greedy sampling strategy for visiting component functions in the method's inner loop. Despite being advantageous for parallel implementations, we found that in practice that this method had worse serial runtimes, so we did not report results for it.

References

Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. In *ICML '06*, 2006.

Kadir. Akbudak, Enver. Kayaaslan, and Cevdet. Aykanat. Hypergraph partitioning based models and methods for exploiting cache locality in sparse matrix-vector multiplication. *SIAM Journal on Scientific Computing*, 2013.

Noga Alon. On the edge-expansion of graphs. *Comb. Probab. Comput.*, 1997.

Charles J Alpert and Andrew B Kahng. Recent directions in netlist partitioning: a survey. *Integration*, 1995.

Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P. Woodruff, and Qin Zhang. On sketching quadratic forms. In *ITCS '16*, 2016.

Kyriakos Axiotis, Adam Karczmarz, Anish Mukherjee, Piotr Sankowski, and Adrian Vladu. Decomposable submodular function minimization via maximum flow. In *ICML '21*, 2021.

Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 2013.

Grey Ballard, Alex Druinsky, Nicholas Knight, and Oded Schwartz. Hypergraph partitioning for sparse matrix-matrix multiplication. *ACM Trans. Parallel Comput.*, 2016.

N. Bansal, O. Svensson, and L. Trevisan. New notions and constructions of sparsification for graphs and hypergraphs. In *FOCS '19*, 2019.

Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Review*, 2014.

András A Benczúr and David R Karger. Approximating s–t minimum cuts in $\tilde{O}(n^2)$ time. In *STOC '96*, 1996.

Austin R. Benson, David F. Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 2016.

Austin R. Benson, Paul Liu, and Hao Yin. A simple bipartite graph projection model for clustering in networks. *arXiv preprint: https://arxiv.org/abs/2007.00761*, 2020.

Mindaugas Bloznelis and Justinas Petuchovas. Correlation between clustering and degree in affiliation networks. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2017.

Mindaugas Bloznelis et al. Degree and clustering coefficient in sparse random intersection graphs. *The Annals of Applied Probability*, 2013.

T. H. Hubert Chan and Zhibin Liang. Generalizing the hypergraph laplacian via a diffusion process with mediators. In *Computing and Combinatorics*, 2018.

Karthekeyan Chandrasekaran, Chao Xu, and Xilin Yu. Hypergraph k-cut in randomized polynomial time. In *SODA '18*, 2018.

Chandra Chekuri and Chao Xu. Computing minimum cuts in hypergraphs. In *SODA '17*, 2017.

Chandra Chekuri and Chao Xu. Minimum cuts and sparsification in hypergraphs. *SIAM Journal on Computing*, 2018.

Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *STOC '12*, 2012.

Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 2009.

Alina Ene and Huy L. Nguyen. Random coordinate descent methods for minimizing decomposable submodular functions. In *ICML '15*, 2015.

Alina Ene, Huy Nguyen, and László A Végh. Decomposable submodular function minimization: discrete and continuous. In *NeurIPS '17*, 2017.

A. Fix, T. Joachims, S. M. Park, and R. Zabih. Structured learning of sum-of-submodular higher order energy functions. In *ICCV '13*, 2013.

D. Freedman and P. Drineas. Energy minimization via graph cuts: settling what is possible. In *CVPR '05*, 2005.

Kaito Fujii, Tasuku Soma, and Yuichi Yoshida. Polynomial-time algorithms for submodular laplacian systems. *Theoretical Computer Science*, 2021.

Junhao Gan, David F. Gleich, Nate Veldt, Anthony Wirth, and Xin Zhang. Graph clustering in all parameter regimes. In *MFCS '20*, 2020.

Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than Goldberg-Rao. In *FOCS '21*, 2021.

Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 1998.

Scott W. Hadley. Approximation techniques for hypergraph partitioning problems. *Discrete Applied Mathematics*, 1995.

Matthias Hein, Simon Setzer, Leonardo Jost, and Syama Sundar Rangapuram. The total variation on hypergraphs - learning on hypergraphs revisited. In *NeurIPS '13*, 2013.

Jin Huang, Rui Zhang, and Jeffrey Xu Yu. Scalable hypergraph learning and processing. In *ICDM '15*, 2015.

Satoru Iwata and James B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *SODA '09*, 2009.

Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 2001.

Stefanie Jegelka and Jeff Bilmes. Submodularity beyond submodular energies: Coupling edges in graph cuts. In *CVPR '11*, 2011.

Stefanie Jegelka, Hui Lin, and Jeff A Bilmes. On fast approximate submodular minimization. In *NeurIPS '11*, 2011.

Stefanie Jegelka, Francis Bach, and Suvrit Sra. Reflection methods for user-friendly submodular optimization. In *NeurIPS '13*, 2013.

G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: applications in VLSI domain. *IEEE Transactions on Very Large Scale Integration Systems*, 1999.

Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *ITCS '15*, 2015.

Pushmeet Kohli, Philip HS Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 2009.

V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

Vladimir Kolmogorov. Minimizing a sum of submodular functions. *Discrete Appl. Math.*, 2012.

Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. Hypergraph clustering by iteratively reweighted modularity maximization. *Applied Network Science*, 2020.

Silvio Lattanzi and D Sivakumar. Affiliation networks. In *STOC '09*, 2009.

E. L. Lawler. Cutsets and partitions of hypergraphs. *Networks*, 1973.

Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}\sqrt{rank}$ iterations and faster algorithms for maximum flow. In *FOCS '14*, 2014.

Jianbo Li, Jingrui He, and Yada Zhu. E-tail product return prediction via hypergraph-based local graph cut. In *KDD '18*, 2018.

Menghui Li, Jinshan Wu, Dahui Wang, Tao Zhou, Zengru Di, and Ying Fan. Evolving model of weighted networks inspired by scientific collaboration networks. *Physica A: Statistical Mechanics and its Applications*, 2007.

Pan Li and Olgica Milenkovic. Inhomogeneous hypergraph clustering with applications. In *NeurIPS '17*, 2017.

Pan Li and Olgica Milenkovic. Revisiting decomposable submodular function minimization with incidence relations. In *NeurIPS '18*, 2018a.

Pan Li and Olgica Milenkovic. Submodular hypergraphs: p-laplacians, Cheeger inequalities and spectral clustering. In *ICML '18*, 2018b.

Pan Li, Niao He, and Olgica Milenkovic. Quadratic decomposable submodular function minimization: Theory and practice. *Journal of Machine Learning Research*, 2020.

Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. Strongly local hypergraph diffusions for clustering and semi-supervised learning. In *WWW '21*, 2021.

Anand Louis. Hypergraph markov operators, eigenvalues and approximation algorithms. In *STOC '15*, 2015.

A. Lubotzky. Ramanujan graphs. *Combinatorica*, 1988.

Thomas L. Magnanti and Dan Stratila. Separable concave optimization approximately equals piecewise linear optimization. In *IPCO '04*, 2004.

Thomas L Magnanti and Dan Stratila. Separable concave optimization approximately equals piecewise-linear optimization. *arXiv preprint arXiv:1201.3148*, 2012.

Zachary Neal. The backbone of bipartite projections: Inferring relationships from co-authorship, co-sponsorship, co-attendance and other co-behaviors. *Social Networks*, 2014.

M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 2001.

Robert Nishihara, Stefanie Jegelka, and Michael I. Jordan. On the convergence rate of decomposable submodular function minimization. In *NeurIPS '14*, 2014.

James B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 2009.

Pulak Purkait, Tat-Jun Chin, Alireza Sadri, and David Suter. Clustering with hypergraphs: the case for large hyperedges. *IEEE transactions on pattern analysis and machine intelligence*, 2016.

José J. Ramasco and Steven A. Morris. Social inertia in collaboration networks. *Phys. Rev. E*, 2006.

J.A. Rodríguez. Laplacian eigenvalues and partition problems in hypergraphs. *Applied Mathematics Letters*, 2009.

I. Shanu, C. Arora, and P. Singla. Min norm point algorithm for higher order mrf-map inference. In *CVPR '16*, 2016.

Tasuku Soma and Yuichi Yoshida. Spectral sparsification of hypergraphs. In *SODA '19*, 2019.

Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 2011.

Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 2014.

Domenico De Stefano, Vittorio Fuccella, Maria Prosperina Vitale, and Susanna Zaccarin. The use of different data sources in the analysis of co-authorship networks and scientific performance. *Social Networks*, 2013.

Peter Stobbe and Andreas Krause. Efficient minimization of decomposable submodular functions. In *NeurIPS '10*, 2010.

Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *STOC '21*, 2021.

A. Vannelli and S. W. Hadley. A gomory-hu cut tree representation of a netlist partitioning problem. *IEEE Transactions on Circuits and Systems*, 1990.

Nate Veldt, Austin R. Benson, and Jon Kleinberg. Minimizing localized ratio cut objectives in hypergraphs. In *KDD '20*, 2020a.

Nate Veldt, Anthony Wirth, and David F. Gleich. Parameterized correlation clustering in hypergraphs and bipartite graphs. In *KDD '20*, 2020b.

Nate Veldt, Austin R Benson, and Jon Kleinberg. Approximate decomposable submodular function minimization for cardinality-based components. *NeurIPs '21*, 2021.

Nate Veldt, Austin R. Benson, and Jon Kleinberg. Hypergraph cuts with general splitting functions. *SIAM Review*, 2022.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 1998.

Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *KDD '17*, 2017.

Yuichi Yoshida. Cheeger inequalities for submodular transformations. In *SODA '19*, 2019.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *NeurIPS '06*, 2006.

Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Phys. Rev. E*, 2007.

J. Y. Zien, M. D. F. Schlag, and P. K. Chan. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1999.

Stanislav Živný, David A. Cohen, and Peter G. Jeavons. The expressive power of binary submodular functions. In *MFCS '09*, 2009.