Strongly Local Hypergraph Diffusions for Clustering and Semi-supervised Learning

Meng Liu
liu1740@purdue.edu
Purdue University
United States

Nate Veldt
nveldt@cornell.edu
Cornell University
United States

Haoyu Song
song522@purdue.edu
Purdue University
United States

Pan Li
panli@purdue.edu
Purdue University
United States

David F. Gleich
dgleich@purdue.edu
Purdue University
United States

ABSTRACT

Hypergraph-based machine learning methods are now widely recognized as important for modeling and using higher-order and multiway relationships between data objects. Local hypergraph clustering and semi-supervised learning specifically involve finding a well-connected set of nodes near a given set of labeled vertices. Although many methods for local graph clustering exist, there are relatively few for localized clustering in hypergraphs. Moreover, those that exist often lack flexibility to model a general class of hypergraph cut functions or cannot scale to large problems. To tackle these issues, this paper proposes a new diffusion-based hypergraph clustering algorithm that solves a quadratic hypergraph cut based objective akin to a hypergraph analog of Andersen-Chung-Lang personalized PageRank clustering for graphs. We prove that, for graphs with fixed maximum hyperedge size, this method is strongly local, meaning that its runtime only depends on the size of the output instead of the size of the hypergraph and is highly scalable. Moreover, our method enables us to compute with a wide variety of cardinality-based hypergraph cut functions. We also prove that the clusters found by solving the new objective function satisfy a Cheeger-like quality guarantee. We demonstrate that on large real-world hypergraphs our new method finds better clusters and runs much faster than existing approaches. Specifically, it runs in a few seconds for hypergraphs with a few million hyperedges compared with minutes for a flow-based technique. We furthermore show that our framework is general enough that can also be used to solve other p-norm based cut objectives on hypergraphs.

CCS CONCEPTS

• **Mathematics of computing** → **Hypergraphs**; • **Theory of computation** → **Semi-supervised learning**; • **Computing methodologies** → **Cluster analysis**.

KEYWORDS

hypergraph, local clustering, community detection, PageRank

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449887

ACM Reference Format:
Meng Liu, Nate Veldt, Haoyu Song, Pan Li, and David F. Gleich. 2021. Strongly Local Hypergraph Diffusions for Clustering and Semi-supervised Learning. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449887

1 INTRODUCTION

Two common scenarios in graph-based data analysis are: (i) What are the clusters, groups, modules, or communities in a graph? and (ii) Given some limited label information on the nodes of the graph, what can be inferred about missing labels? These statements correspond to the clustering and semi-supervised learning problems respectively, and while there exists a strong state of the art in algorithms for these problems on graphs [3, 13, 19, 29, 34, 35, 37, 42], research on these problems is currently highly active for hypergraphs [8, 17, 27, 33, 38, 40, 41] building on new types of results [15, 26, 32] compared to prior approaches [1, 20, 43]. The lack of flexible, diverse, and scalable hypergraph algorithms for these problems limits the opportunities to investigate rich structure in data. For example, clusters can be relevant treatment groups for statistical testing on networks [10] or identify common structure across many types of sparse networks [24]. Likewise, semi-supervised learning helps to characterize subtle structure in the emissions spectra of galaxies in astronomy data through characterizations in terms of biased eigenvectors [23]. The current set of hypergraph algorithms are insufficient for such advanced scenarios.

Hypergraphs, indeed, enable a flexible and rich data model that has the potential to capture subtle insights that are difficult or impossible to find with traditional graph-based analysis [5, 27, 33, 38, 40]. But, hypergraph generalizations of graph-based algorithms often struggle with scalability and interpretation [1, 15] with ongoing questions of whether particular models *capture* the higher-order information in hypergraphs. Regarding scalablility, an important special case for that is a *strongly local algorithm.* Strongly local algorithms are those whose runtime depends on the size of the output rather than the size of the graph. This was only recently addressed for various hypergraph clustering and semi-supervised learning frameworks [17, 33]. This property enables fast (seconds to minutes) evaluation even for massive graphs with hundreds of millions of nodes and edges [3] (compared with hours). For graphs,

perhaps the best known strongly local algorithm is the Andersen-Chung-Lang (henceforth, ACL) approximation for personalized PageRank [3] with applications to local community detection and semi-supervised learning [42]. *The specific problem we address is a mincut-inspired hypergraph generalization of personalized PageRank along with a strongly local algorithm to rapidly approximate solutions.* Our formulation differs in a number of important ways from existing Laplacian [43] and quadratic function-based hypergraph PageRank generalizations [25, 27, 31].

Although our localized hypergraph PageRank is reasonably simple to state formally (§3.2), there are a variety of subtle aspects to both the problem statement and the algorithmic solution. First, we wish to have a formulation that enables the richness of possible hypergraph cut functions. These hypergraph cut functions are an essential component to rich hypergraph models because they determine when a group of nodes ought to belong to the same cluster or obtain a potential new label for semi-supervised learning. Existing techniques based on star expansions (akin to treating the hypergraph as a bipartite graph) or a clique expansion (creating a weighted graph by adding edges from a clique to the graph for each hyperedge) only model a limited set of cut functions [2, 32]. More general techniques based on Lovász extensions [25, 27, 40] pose substantial computational difficulties. Second, we need a problem framework that gives sparse solutions such that they can be computed in a strongly local fashion and then we need an algorithm that is actually able to compute these—the mere existence of solutions is insufficient for deploying these ideas in practice as we wish to do. Finally, we need an understanding of the relationship between the results of this algorithm and various graph quantities, such as minimal conductance sets as in the original ACL method.

To address these challenges, we extend and employ a number of recently proposed hypergraph frameworks. First, we show a new result on a class of hypergraph to graph transformations [32]. These transformations employ carefully constructed directed graph gadgets, along with a set of auxiliary nodes, to encode the properties of a general class of cardinality based hypergraph cut functions. Our simple new result highlights how these transformations not only preserve *cut* values, but preserve the *hypergraph conductance* values as well (§3.1). Then we localize the computation in the reduced graph using a general strategy to build strongly local computations. This involves a particular modification often called a "localized cut" graph or hypergraph [4, 11, 28, 33]. We then use a squared 2-norm (i.e. a *quadratic* function) instead of a 1-norm that arises in the mincut-graph to produce the hypergraph analogue to strongly local personalized PageRank. Put another way, applying all of these steps on a graph (instead of a hypergraph) is equivalent to a characterization of personalized PageRank vector [12].

Once we have the framework in place (§3.1,§3.2), we are able to show that an adaptation of the *push* method for personalized PageRank (§3.3) will compute an approximate solution in time that depends only on the localization parameters and is independent of the size of a hypergraph with fixed maximum hyperedge size (Theorem 3.5). Consequently, the algorithms are strongly local.

The final algorithm we produce is extremely efficient. It is a small factor (2-5x) slower than running the ACL algorithm for graphs on the star expansion of the hypergraph. It is also a small factor (2-5x) faster than running an optimized implementation of the ACL algorithm on the clique expansion of the hypergraph. Nevertheless, for many instances of semi-supervised learning problems, it produces results with much larger F1 scores than alternative methods. In particular, it is much faster and performs much better with extremely limited label information than a recently proposed flow-based method [33].

Summary of additional contributions. In addition to providing a strongly local algorithm for the squared 2-norm (i.e. a *quadratic* function) in §3.2, which gives better and faster empirical performance (§7), we also discuss how to use a p-norm (§6) instead. Finally, we also show a Cheeger inequality that relates our results to the hypergraph conductance of a nearby set (§4).

Our method is the first algorithm for hypergraph clustering that includes *all* of the following features: it is (1) strongly-local, (2) can grow a cluster from a small seed set, (3) models flexible hyperedge cut penalties, and (4) comes with a conductance guarantee.

A motivating case study with Yelp reviews. We begin by illustrating the need and utility for the methods instead with a simple example of the benefit to these spectral or PageRank-style hypergraph approaches. For this purpose we consider a hypothetical use case with an answer that is easy to understand in order to compare our algorithm to a variety of other approaches. We build a hypergraph from the Yelp review dataset (https://www.yelp.com/dataset). Each restaurant is a vertex and each user is a hyperedge. This model enables users, i.e. hyperedges, to capture subtle socioeconomic status information as well as culinary preferences in terms of which types of restaurants they visit and review. The task we seek to understand is either an instance of local clustering or semi-supervised learning. Simply put, given a random sample of 10 restaurants in Las Vegas Nevada, we seek to find other restaurants in Las Vegas. The overall hypergraph has around 64k vertices and 616k hyperedges with a maximum hyperedge size of 2566. Las Vegas, with around 7.3k restaurants, constitutes a small localized cluster.

We investigate a series of different algorithms that will identify a cluster nearby a seed node in a hypergraph: (1) Andersen-Chung-Lang PageRank on the star and clique expansion of the hypergraph (ACL-Star, ACL-Clique, respectively), these algorithms are closely related to ideas proposed in [1, 43]; (2) HyperLocal, a recent maximum flow-based hypergraph clustering algorithm [33]; (3) quadratic hypergraph PageRank [25, 31] (which is also closely related to [15]), and (4) our Local Hypergraph-PageRank (LHPR). These are all strongly local except for (3), which we include because our algorithm LHPR is essentially the strongly local analogue of (3).

The results are shown in Figure 1. The flow-based HyperLocal method has difficulty finding the entire cluster. Flow-based methods are known to have trouble expanding small seed sets [11, 28, 34] and this experiment shows that same behavior. Our strongly local hypergraph PageRank (LHPR) slightly improves on the performance of a quadratic hypergraph PageRank (QHPR) that is not strongly local. In particular, it has 10k non-zero entries (of 64k) in its solution.

This experiment shows the opportunities with our approach for large hypergraphs. We are able to model a flexible family of hypergraph cut functions beyond those that use clique and star expansions and we equal or outperform all the other methods. For instance, another more complicated method [17] designed for small hyperedge sizes showed similar performance to ACL-Clique (F1 around 0.85) and took much longer.

Figure 1: This figure shows locations of the ~7,300 restaurants of Las Vegas that are reviewed on Yelp and how often algorithms recover them from a set of 10 random seeds; our hypergraph PageRank (LHPR) methods has the highest accuracy and finds the result by exploring only 10000 vertices total compared with a fully dense vector for QHPR giving a boost to scalability on larger graphs. The colors show the regions that are missed (red or orange) or found (blue) by each algorithm over 15 trials. HyperLocal is a flow-based method that is known to have trouble growing small seed sets as in this experiment. (The parameters for HyperLocal were chosen in consultation its authors; other parameters were hand tuned for best case performance.)

2 NOTATION AND PRELIMINARIES

Let $G = (V, E, w)$ be a directed graph with $|V| = n$ and $|E| = m$. For simplicity, we require weights $w_{ij} \geq 1$ for each directed edge $(i, j) \in E$. We interpret an undirected graph as having two directed edges (i, j) and (j, i). For simplicity, we assume the vertices are labeled with indices 1 to n, so that we may use these labels to index matrices and vectors. For instance, we define $\mathbf{d}$ as the length-n out-degree vector where its ith component $d_i = \sum_{j \in V} w_{ij}$. The incidence matrix $\mathbf{B} \in \{0, -1, 1\}^{m \times n}$ measures the difference of adjacent nodes. The kth row of $\mathbf{B}$ corresponds to an edge, say (i, j), and has exactly two nonzero values, 1 for the node i and -1 for the node j. (Recall that we have directed edges, so the origin of the edge is always 1 and the destination is always -1.)

Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph where each hyperedge $e \in \mathcal{E}$ is a subset of V. Let $\zeta = \max_{e \in \mathcal{E}} |e|$ be the maximum hyperedge size. With each hyperedge, we associate a *splitting function* f_e that we use to assess an appropriate penalty for splitting the hyperedge among two labels or splitting the hyperedge between two clusters. Formally, let S be a cluster and let $A = e \cap S$ be the hyperedge's nodes inside S, then $f_e(A)$ penalizes splitting e. A common choice in early hypergraph literature was the *all-or-nothing* split, which assigns a fixed value if a hyperedge is split or zero if all nodes in the hyperedge lie in the same cluster [14, 18, 22]: $f_e(A) = 0$ if $A = e$ or $A = \emptyset$ and $f_e(A) = 1$ otherwise (or an alternative constant). More recently, there have been a variety of alternative splitting functions proposed [26, 27, 32] that provide more flexibility. We discuss more choices in the next section (§3.1). With a splitting function identified, the cut value of any given set S can be written as $\text{cut}_{\mathcal{H}}(S) = \sum_{e \in E} f_e(e \cap S)$. The node degree in this case can be defined as $d_i = \sum_{e: i \in e} f_e(\{i\})$ [26, 33], though other types of degree vectors can also be used in both the graph and hypergraph case. This gives rise to a definition of conductance on a hypergraph

$$\phi_{\mathcal{H}}(S) = \frac{\mathbf{cut}_{\mathcal{H}}(S)}{\min(\mathbf{vol}(S), \mathbf{vol}(\bar{S}))} \tag{1}$$

where $\mathbf{vol}(S) = \sum_{i \in S} d_i$. This reduces to the standard definition of graph conductance when each edge has only two nodes ($\zeta = 2$) and we use the *all-or-nothing penality*.

Diffusion algorithms for semi-supervised learning and local clustering. Given a set of seeds, or what we commonly think of as a reference, set R, a diffusion is any method that produces a real-valued vector $\mathbf{x}$ over all the other vertices. For instance, the personalized PageRank method uses R to define the *personalization vector* or *restart vector* underlying the process [3]. The PageRank solution or the sparse Andersen-Chung-Lang approximation [3] are then the diffusion $\mathbf{x}$. Given a diffusion vector $\mathbf{x}$, we *round* it back to a set S by performing a procedure called a sweepcut. This involves sorting $\mathbf{x}$ from largest to smallest and then evaluating the hypergraph conductance of each prefix set $S_j = \{[1], [2], \ldots, [k]\}$, where $[i]$ is the id of the ith largest vertex. The set returned by sweepcut picks the minimum conductance set S_j. Since the sweepcut procedures are general and standardized, we focus on the computation of $\mathbf{x}$. When these algorithms are used for semi-supervised learning, the returned set S is presumed to share the label as the reference (seed) set R; alternatively, its value or rank information may be used to disambiguate multiple labels [13, 42].

3 METHOD

Our overall goal is to compute a hypergraph diffusion that will help us perform a sweepcut to identify a set with *reasonably small* conductance nearby a reference set of vertices in the graph. We explain our method: *localized hypergraph quadratic diffusions* (LHQD) or also *localized hypergraph PageRank* (LHPR) through two transformations before we formally state the problem and algorithm. We adopted this strategy so that the final proposal is well justified because some of the transformations require additional context to appreciate. Computing the final sweepcut is straightforward for hypergraph conductance, and so we do not focus on that step.

3.1 Hypergraph-to-graph reductions

Minimizing conductance is NP-hard even in the case of simple graphs, though numerous techniques have been designed to approximate the objective in theory and practice [3, 4, 9]. A common strategy for searching for low-conductance sets in hypergraphs is to first reduce a hypergraph to a graph, and then apply existing graph-based techniques. This sounds "hacky" or least "ad-hoc" but this idea is both principled and rigorous. The most common approach is to apply a clique expansion [1, 5, 26, 43, 46], which explicitly models splitting functions of the form $f_e(A) \propto |A||e \backslash A|$.

For instance Benson et al. [5] showed that clique expansion can be used to convert a 3-uniform hypergraph into a graph that preserves the *all-or-nothing* conductance values. For larger hyperedge sizes, *all-or-nothing* conductance is preserved to within a distortion factor depending on the size of the hyperedge. Later, Li et al. [26] were the first to introduce more generalized notions of hyperedge splitting functions, focusing specifically on submodular functions.

Definition 3.1. A splitting function f_e is *submodular* if

$$f_e(A) + f_e(B) \geq f_e(A \cup B) + f_e(A \cap B) \quad \forall A, B \subseteq e. \tag{2}$$

These authors showed that for this submodular case, clique expansion could be used to define a graph preserving conductance to within a factor $O(\zeta)$ (ζ is the largest hyperedge size).

More recently, Veldt et al. [32] introduced graph reduction techniques that *exactly* preserve submodular hypergraph *cut* functions which are cardinality-based.

Definition 3.2. A splitting function f_e is cardinality-based if

$$f_e(A) = f_e(B) \quad \text{whenever } |A| = |B|. \tag{3}$$

Cardinality-based splitting functions are a natural choice for many applications, since node identification is typically irrelevant in practice, and the cardinality-based model produces a cut function that is invariant to node permutation. Furthermore, most previous research on applying generalized hypergraph cut penalties implicitly focused on cut functions are that are naturally cardinality-based [5, 15, 20, 25, 27, 46]. Because of their ubiquity and flexibility, in this work we also focus on hypergraph cut functions that are submodular and cardinality-based. We briefly review the associated graph transformation and then we build on previous work by showing that these hypergraph reductions can be used to preserve the hypergraph *conductance* objective, and not just hypergraph cuts.

Reduction for Cardinality-Based Cuts. Veldt et al. [32] gave results that show the cut properties of a submodular, cardinality-based hypergraph could be preserved by replacing each hyperedge with a set of directed graph *gadgets*. Each gadget for a hyperedge e is constructed by introducing a pair of auxiliary nodes a and b, along with a directed edge (a, b) with weight $\delta_e > 0$. For each $v \in e$, two unit-weight directed edges are introduced: (v, a) and (b, v). The entire gadget is then scaled by a weight $c_e \geq 0$. The resulting gadget represents a simplified splitting function of the following form:

$$f_e(A) = c_e \cdot \min\{|A|, |e \backslash A|, \delta_e\}. \tag{4}$$

Figure 2(b) illustrates the process of replacing a hyperedge with a gadget. The cut properties of any submodular cardinality-based splitting function can be exactly modeled by introducing a set of $O(|e|)$ or fewer such splitting functions [32]. If an approximation suffices, only $O(\log |e|)$ gadgets are required [6].

An important consequence of these reduction results is that in order to develop reduction techniques for *any* submodular cardinality-based splitting functions, it suffices to consider hyperedges with splitting functions of the simplified form given in (4). In the remainder of the text, we focus on splitting functions of this form, with the understanding that all other cardinality-based submodular splitting functions can be modeled by introducing multiple hyperedges on the same set of nodes with different edge weights.

In Figure 2, we illustrate the procedure of reducing a small hypergraph to a directed graph, where we introduce a single gadget per hyperedge. Formally, for a hypergraph $\mathcal{H} = (V, E)$, this procedure produces a directed graph $G = (\hat{V}, \hat{E})$, with directed edge set $\hat{E}$, and node set $\hat{V} = V \cup V_a \cup V_b$, where V is the set of original hypergraph nodes. Sets V_a, V_b store auxiliary nodes, in such a way that for each pair of auxiliary nodes a, b where (a, b) is a directed edge, we have $a \in V_a$ and $b \in V_b$. This reduction technique was previously developed as a way of preserving minimum cuts and minimum s-t cuts for the original hypergraph. Here, we extend this result to show that for a certain choice for node degree, this reduction also preserves hypergraph conductance.

Theorem 3.3. *Define a degree vector* $\mathbf{d}$ *for the reduced graph* $G = (\hat{V}, \hat{E})$ *such that* $\mathbf{d}(v) = d_v$ *is the out-degree for each node* $v \in V$, *and* $\mathbf{d}(u) = d_u = 0$ *for every auxiliary node* $u \in V_a \cup V_b$. *If* T^* *is the minimum conductance set in* G *for this degree vector, then* $S^* = T^* \cap V$ *is the minimum hypergraph conductance set in* $\mathcal{H} = (V, E)$.

Proof. From previous work on these reduction techniques [6, 32], we know that the cut penalty for a set $S \subseteq V$ in $\mathcal{H}$ equals the cut penalty in the directed graph, as long as auxiliary nodes are arranged in a way that produces the smallest cut penalty subject to the choice of node set $S \subseteq V$. Formally, for $S \subseteq V$,

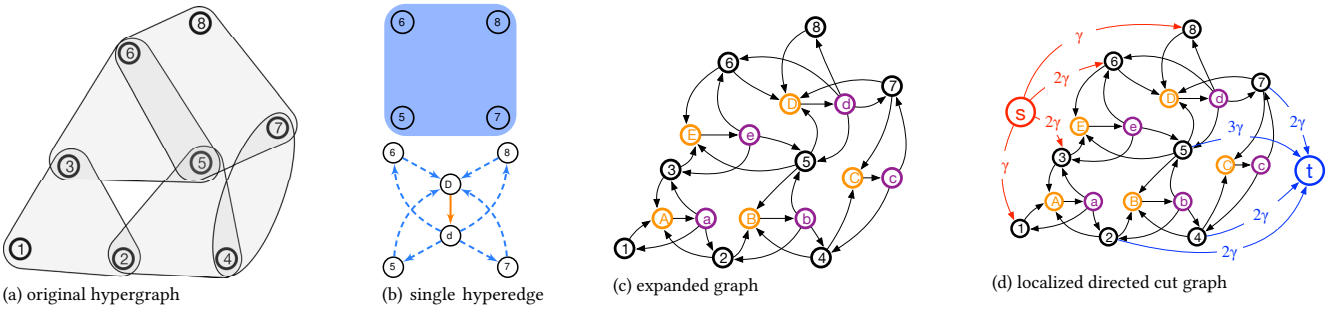
$$\mathbf{cut}_{\mathcal{H}}(S) = \underset{T \subset \hat{V}:\, S = T \cap V}{\text{minimize}} \ \mathbf{cut}_G(T), \tag{5}$$

where $\mathbf{cut}_G$ denotes the weight of directed out-edges originating inside S that are cut in G. By our choice of degree vector, the volume of nodes in G equals the volume of the non-auxiliary nodes in $\mathcal{H}$. That is, for all $T \subseteq \hat{V}$, $\mathbf{vol}_G(T) = \sum_{v \in V} d_v + \sum_{u \in V_a \cup V_b} d_u = \mathbf{vol}_G(T \cap V) = \mathbf{vol}_{\mathcal{H}}(T \cap V)$. Let $T^* \subseteq \hat{V}$ be the minimum conductance set in G, and $S^* = T^* \cap V$. Without loss of generality we can assume that $\mathbf{vol}_G(T^*) \leq \mathbf{vol}_G(\bar{T}^*)$. Since T^* minimizes conductance, and auxiliary nodes have no effect on the volume of this set, $\mathbf{cut}_G(T^*) = \text{minimize}_{T \subset \hat{V}:\, T \cap S^*} \mathbf{cut}_G(T) = \mathbf{cut}_{\mathcal{H}}(S^*)$, and so $\mathbf{cut}_G(T^*)/\mathbf{vol}_G(T^*) = \mathbf{cut}_{\mathcal{H}}(S^*)/\mathbf{vol}_{\mathcal{H}}(S^*)$. Thus, minimizing conductance in G minimizes conductance in $\mathcal{H}$. □

3.2 Localized Quadratic Hypergraph Diffusions

Having established a conductance-preserving reduction from a hypergraph to a directed graph, we now present a framework for detecting localized clusters in the reduced graph G. To accomplish this, we first define a *localized directed cut graph*, involving a source and sink nodes and new weighted edges. This approach is closely related to previously defined localized cut graphs for local graph clustering and semi-supervised learning [4, 7, 12, 28, 34, 44], and a similar localized cut hypergraph used for flow-based hypergraph clustering [33]. The key conceptual difference is that we apply this construction directly to the reduced graph G, which by Theorem 3.3 preserves conductance of the original hypergraph $\mathcal{H}$. Formally, we assume we are given a set of nodes $R \subseteq V$ around which we wish to find low-conductance clusters, and a parameter $\gamma > 0$. The localized directed cut graph is defined by applying the following steps to G:

- Introduce a source node s, and for each $r \in R$ define a directed edge (s, r) of weight γd_r
- Introduce a sink node t, and for each $v \in \bar{R}$ define a directed edge (v, t) with weight γd_v.



(a) original hypergraph (b) single hyperedge reduction gadget (c) expanded graph (d) localized directed cut graph

Figure 2: A simple illustration of hypergraph reduction (Section 3.1) and localization (Section 3.2). (a) A hypergraph with 8 nodes and 5 hyperedges. (b) An illustration of the hyperedge transformation gadget for δ-linear splitting function. (c) The hypergraph is reduced to a directed graph by adding a pair of auxiliary nodes for each hyperedge and this preserves hypergraph conductance computations (Theorem 3.3). (d) The localized directed cut graph is created by adding a source node s, a sink node t and edges from s to hypergraph nodes or from hypergraph nodes to t to *localize* a solution.

We do not connect auxiliary nodes to the source or sink, which is consistent with the fact that their degree is defined to be zero in order for Theorem 3.3 to hold. We illustrate the construction of the localized directed cut graph in Figure 2(d). It is important to note that in practice we do not in fact form this graph and store it in memory. Rather, this provides a conceptual framework for finding localized low-conductance sets in G, which in turn correspond to good clusters in $\mathcal{H}$.

Definition: Local hypergraph quadratic diffusions. Let $\mathbf{B}$ and $\mathbf{w}$ be the incidence matrix and edge weight vector of the localized directed cut graph with γ. The objective function for our hypergraph clustering diffusion, which we call *local hypergraph quadratic diffusion* or simply *local hypergraph PageRank*, is

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \frac{1}{2}\mathbf{w}^T(\mathbf{B}\mathbf{x})_+^2 + \kappa\gamma\sum_{i\in V} x_i d_i \\ \text{subject to} & x_s = 1, x_t = 0, \mathbf{x} \geq 0. \end{array} \tag{6}$$

We use the function $(x)_+ = \max\{x, 0\}$, applied element-wise to $\mathbf{Bx}$, to indicate we only keep the positive elements of this product. This is analogous to the fact that we only view a directed edge as being cut if it crosses from the source to the sink side; this is similar to previous directed cut minorants on graphs and hypergraphs [39]. The first term in the objective corresponds to a 2-norm minorant of the minimum s-t cut objective on the localized directed cut graph. (In an undirected regular graph, the term $\mathbf{w}^T(\mathbf{Bx})_+$ turns into an expression with the Laplacian, which can in turn be formally related to PageRank [12]). If instead, we replace exponent 2 with a 1 and ignore the second term, this would amount to finding a minimum s-t cut (which can be solved via a maximum flow). The second term in the objective is included to encourage sparsity in the solution, where $\kappa \geq 0$ controls the desired level of sparsity. With $\kappa > 0$ we are able to show in the next section that we can compute solutions in time that depends only on κ, γ, and $\mathbf{vol}(R)$, which allows us to evaluate solutions to (6) in a strongly local fashion.

3.3 A strongly local solver for LHQD (6)

In this section, we will provide a strongly local algorithm to approximately satisfy the optimality conditions of (6). We first state the optimality conditions in Theorem 3.4, and then present the algorithm to solve them. The simplest way to understand this algorithm is as a generalization of the Andersen-Chung-Lang push procedure for PageRank [3], which we will call ACL as well as the more recent nonlinear push procedure [28]. Two new challenges about this new algorithm are: (1) the new algorithm operates on a directed graph, which means unlike ACL there is no single closed form update at each iteration and (2) there is no sparsity regularization for auxiliary nodes, which will break the strongly local guarantees for existing analyses of the push procedure.

We begin with the optimality conditions for (6).

Theorem 3.4. *Fix a seed set R, $\gamma > 0$, $\kappa > 0$, define a residual function $\mathbf{r}(\mathbf{x}) = -\frac{1}{\gamma}\mathbf{B}^T diag((\mathbf{Bx})_+)\mathbf{w}$. A necessary and sufficient condition to satisfy the KKT conditions of (6) is to find $\mathbf{x}^*$ where $\mathbf{x}^* \geq 0$, $\mathbf{r}(\mathbf{x}^*) = [r_s, \mathbf{g}^T, r_t]^T$ with $g_i \leq \kappa d_i$ (where $\mathbf{d}$ reflects the graph before adding s and t but does include the 0 degree nodes), $(\kappa d_i - g_i)^T x_i^* = 0$ for $i \in V$ and $g_i = 0$ for all auxilary nodes added.*

It is a straightforward application of determining optimality conditions for convex programs. Detailed proof of this would be included in a longer version of this material. We further note that solutions $\mathbf{x}^*$ are unique because the problem is strongly convex due to the quadratic.

In §3.1, we have shown that the reduction technique of any cardinality submodular-based splitting function suffices to introduce multiple directed graph gadgets with different δ_e and c_e. In order to simplify our exposition, we assume that each hyperedge has a δ-linear threshold splitting function [33] $f_e = \min\{|A|, |e\backslash A|, \delta\}$ with $\delta \geq 1$ to be a tunable parameter. This splitting function can be exactly modeled by replacing each hyperedge with one directed graph gadget with $c_e = 1$ and $\delta_e = \delta$. (This is what is illustrated in Figure 2.) Also when $\delta = 1$, it models the standard unweighted *all-or-nothing* cut [14, 18, 22] and when δ goes to infinity, it models star expansion [46]. Thus this splitting function can interpolate these two common cut objectives on hypergraphs by varying δ.

By assuming that we have a δ-linear threshold splitting function, this means we can associate exactly *two* auxiliary nodes with each hyperedge. We call these a and b for simplicity. We also let V_a be the set of all a auxilary nodes and V_b be the set of all b nodes.

At a high level, the algorithm to solve this proceeds as follows: whenever there exists a graph node $i \in V$ that violates optimality, i.e. $r_i > \kappa d_i$, we first perform a `hyperpush` at i to increase x_i so that the optimality condition is approximately satisfied, i.e., $r_i =$

Algorithm 1 LHQD(γ, κ, ρ) for set R and hypergraph H with δ-linear penalty where $0 < \rho < 1$ determines accuracy

1: Let $\mathbf{x} = 0$ except for $x_s = 1$ and set $\mathbf{r} = -\gamma^{-1}\mathbf{B}^T \operatorname{diag}((\mathbf{B}\mathbf{x})_+)\mathbf{w}(\delta, \gamma)$.
2: While there is any vertex $i \in V$ where $r_i > \kappa d_i$, or stop if none exists *(find an optimality violation)*
3: Perform LHQD-hyperpush at vertex i so that $r_i = \rho\kappa d_i$, updating $\mathbf{x}$ and $\mathbf{r}$. *(satisfy optimality at i)*
4: For each pair of adjacent auxiliary nodes a, b where $a \in V_a$, $b \in V_b$ and $a \to b$, perform LHQD-auxpush at a and b so that $r_a = r_b = 0$, then update $\mathbf{x}$ and $\mathbf{r}$ after each auxpush.
5: Return $\mathbf{x}$

Algorithm 2 LQHD-hyperpush($i, \gamma, \kappa, \mathbf{x}, \mathbf{r}, \rho$)

1: Solve Δx_i with $s_a^{(i)}$, $s_b^{(i)}$, $a_{min}^{(i)}$ and $b_{min}^{(i)}$ using (9). *(assume the order of i doesn't change among its adjacent nodes)*
2: **if** (10) doesn't hold *(adding Δx_i changed the order of i)* **then**
3: Binary search on Δx_i until we find the smallest interval among all adjacent nodes of i that will include $x_i + \Delta x_i$, update $s_a^{(i)}$, $s_b^{(i)}$, $a_{min}^{(i)}$ and $b_{min}^{(i)}$.
4: Solve Δx_i with the found interval by setting $r_i = \rho\kappa d_i$ in (7).
5: **end if**
6: Update $\mathbf{x}$ and $\mathbf{r}$, $x_i \leftarrow x_i + \Delta x_i$, $r_i \leftarrow \rho\kappa d_i$

$\rho\kappa d_i$ where $0 < \rho < 1$ is a given parameter that influences the approximation. This changes the solution $\mathbf{x}$ only at the current node i and residuals at adjacent auxiliary nodes. Then we immediately *push* on adjacent auxiliary nodes, which means we increase their value so that the residuals remain zero. After pushing each pair (a, b) of associated auxiliary nodes, we then update residuals for *adjacent* nodes in V. Then we search for another optimality violation. (See Algorithm 1 for a formalization of this strategy.) When $\rho < 1$, we only approximately satisfy the optimality conditions; and this approximation strategy has been repeatedly and successfully used in existing local graph clustering algorithms [3, 12, 28].

Notes on optimizing the procedure. Algorithm 1 formalizes a general strategy to approximately solve these diffusions. We now note a number of optimizations that we have found to greatly accelerate this strategy. First, note that $\mathbf{x}$ and $\mathbf{r}$ can be kept as sparse vectors with only a small set of entries stored. Second, note that we can maintain a list of optimality violations because each update to $\mathbf{x}$ only causes $\mathbf{r}$ to increase, so we can simply check if each coordinate increase creates a new violation and add it to a queue. Third, to find the value that needs to be "pushed" to each node, a general strategy is to use a binary search procedure as we will use for the p-norm generalization in §6. However, if the tolerance of the binary search is too small, it will slow down each iteration. If the tolerance is too large, the solution will be too far away from the true solution to be useful. In the remaining of this section, we will show that in the case of quadratic objective (6), we can (i) often avoid binary search and (ii) when it is still required, make the binary search procedure unrelated to the choice of tolerance in those iterations where we do need it. These detailed techniques will not change the time complexity of the overall algorithm, but make a large difference in practice.

We will start by looking at the expanded formulations of the residual vector. When $i \in V$, r_i expands as:

$$r_i = \frac{1}{\gamma}\sum_{b\in V_b} w_{bi}(x_b - x_i)_+ - \frac{1}{\gamma}\sum_{a\in V_a} w_{ia}(x_i - x_a)_+ + d_i[\mathrm{Ind}(i \in R) - x_i]. \tag{7}$$

Similarly, for each $a \in V_a$, $b \in V_b$ where $a \to b$, they will share the same set of original nodes and their residuals can be expanded as:

$$\begin{aligned} r_a &= -w_{ab}(x_a - x_b) + \textstyle\sum_{i\in V} w_{ia}(x_i - x_a)_+ \\ r_b &= w_{ab}(x_a - x_b) - \textstyle\sum_{i\in V} w_{bi}(x_b - x_i)_+ \end{aligned} \tag{8}$$

Note here we use a result that $x_a \geq x_b$ (Lemma A.1).

The goal in each hyperpush is to first find Δx_i such that $r_i' = \rho\kappa d_i$ and then in auxpush, for each pair of adjacent auxiliary nodes (a, b), find Δx_a and Δx_b such that r_a' and r_b' remain zero. (Δx_i, Δx_a and Δx_b are unique because the quadratic is strongly convex.) Observe that r_i, r_a and r_b are all piecewise linear functions, which means we can derive a closed form solution once the relative ordering of adjacent nodes is determined. Also, in most cases, the relative ordering won't change after a few initial iterations. So we can first reuse the ordering information from last iteration to directly solve Δx_i, Δx_a and Δx_b and then check if the ordering is changed.

Given these observations, we will record and update the following information for each pushed node. Again, this information can be recorded in a *sparse* fashion. When the pushed node i is a original node, for its adjacent $a \in V_a$ and $b \in V_b$, we record:

- $s_a^{(i)}$: the sum of edge weights w_{ia} where $x_a < x_i$
- $s_b^{(i)}$: the sum of edge weights w_{bi} where $x_b > x_i$
- $a_{min}^{(i)}$: the minimum x_a where $x_a \geq x_i$
- $b_{min}^{(i)}$: the minimum x_b where $x_b > x_i$

Now assume the ordering is the same, r_i' can be written as $r_i' = r_i - \frac{1}{\gamma}(s_a^{(i)} + s_b^{(i)})\Delta x_i = \rho\kappa d_i$, so

$$\Delta x_i = \gamma(r_i - \rho\kappa d_i)/(s_a^{(i)} + s_b^{(i)}). \tag{9}$$

Then we need to check whether the assumption holds by checking

$$x_i + \Delta x_i \leq \min\left(a_{min}^{(i)}, b_{min}^{(i)}\right) \tag{10}$$

If not, we need to use binary search to find the new location of $x_i + \Delta x_i$ (Note Δx_i here is the true value that is still unknown), update $s_a^{(i)}$, $s_b^{(i)}$, $a_{min}^{(i)}$ and $b_{min}^{(i)}$ and recompute Δx_i. This process is summarized in LQHD-hyperpush.

Similarly, when the pushed nodes $a \in V_a$, $b \in V_b$ where $a \to b$, are a pair of auxiliary nodes, for its adjacent nodes $i \in V$, we record:

- z_a: the sum of edge weights w_{ia} where $x_a < x_i$
- z_b: the sum of edge weights w_{bi} where $x_b > x_i$
- $x_{min}^{(a)}$: the minimum x_i where $x_a < x_i$
- $x_{min}^{(b)}$: the minimum x_i where $x_b < x_i$

Algorithm 3 LQHD-auxpush$(i, a, b, \gamma, \mathbf{x}, \mathbf{r}, \Delta x_i)$

1: Solve Δx_a, Δx_b with z_a, z_b, $x^{(a)}_{min}$ and $x^{(b)}_{min}$ using (11).
2: **if** (12) doesn't hold. *(adding $\Delta x_a, \Delta x_b$ altered the order)* **then**
3: Binary search on Δx_a until we find the smallest interval among all adjacent original nodes of a that will include $x_a + \Delta x_a$, update z_a, $x^{(a)}_{min}$, similarly for z_b, $x^{(b)}_{min}$.
4: Solve $\Delta x_a, \Delta x_b$ with the found intervals by setting $r_a = r_b = 0$ in (8).
5: **end if**
6: Change the following entries in $\mathbf{x}$ and $\mathbf{r}$ to update the solution and the residual
7: (a) $x_a \leftarrow x_a + \Delta x_a$ and $x_b \leftarrow x_b + \Delta x_b$
8: (b) For each neighboring node $i \to a$ where $i \in V$, $r_i \leftarrow r_i + \frac{1}{\gamma} w_{ia}(x_i - x_a)_+ - \frac{1}{\gamma} w_{ia}(x_i - x_a - \Delta x_a)_+ - \frac{1}{\gamma} w_{bi}(x_b - x_i)_+ + \frac{1}{\gamma} w_{bi}(x_b + \Delta x_b - x_i)_+$

Then we solve Δx_a, Δx_b by solving the following linear system (here we assume $x_b \geq x_i$)

$$\begin{cases} -w_{ab}(\Delta x_a - \Delta x_b) + \frac{w_{ia}}{\gamma}((x'_i - x_a)_+ - (x_i - x_a)_+) - z_a \Delta x_a = 0 \\ w_{ab}(\Delta x_a - \Delta x_b) - \frac{w_{bi}}{\gamma}((x_b - x'_i)_+ - (x_b - x_i)_+) + z_b \Delta x_b = 0 \end{cases} \tag{11}$$

where x'_i refers to the updated x_i after applying LQHD-hyperpush at node i. And the assumption will hold if and only if the following inequalities are all satisfied:

$$x'_i \leq x_b, \qquad x_a + \Delta x_a \leq x^{(a)}_{min}, \qquad x_b + \Delta x_b \leq x^{(b)}_{min} \tag{12}$$

If not, we also need to use binary search to update the locations of $x_a + \Delta x_a$ and $x_b + \Delta x_b$, update z_a, z_b, $x^{(a)}_{min}$, $x^{(b)}_{min}$ and recompute Δx_a and Δx_b.

Establishing a runtime bound. The key to understand the strong locality of the algorithm is that after each LQHD-hyperpush, the decrease of $\|vg\|_1$ can be lower bounded by a value that is independent of the total size of the hypergraph, while LHQD-auxpush doesn't change $\|\mathbf{g}\|_1$. Formally, we have the following theorem:

Theorem 3.5. *Given $\gamma > 0$, $\kappa > 0$, $\delta > 0$ and $0 < \rho < 1$. Suppose the splitting function f_e is submodular, cardinality-based and satisfies $1 \leq f_e(\{i\}) \leq \delta$ for any $i \in e$. Then calling* LQHD-auxpush *doesn't change $\|\mathbf{g}\|_1$ while calling* LQHD-hyperpush *on node $i \in V$ will decrease $\|\mathbf{g}\|_1$ by at least $\gamma\kappa(1-\rho)d_i/(\gamma\kappa + \delta)$.*

Suppose LHQD *stops after T iterations and d_i is the degree of the original node updated at the i-th iteration, then T must satisfy:*

$$\textstyle\sum_{i=1}^{T} d_i \leq (\gamma\kappa + \delta)vol(R)/\gamma\kappa(1-\rho) = O(vol(R)).$$

The proof is in the appendix. This theorem only upper bounds the number of iterations Algorithm 1 requires. Each iteration will also take $O(\sum_{e \in \mathcal{E}, i \in e} |e|)$ amount of work. This ignores the binary search, which only scales it by $\log(\max\{d_i, \max_{e \in \mathcal{E}, i \in e}\{|e|\}\})$ factor in the worst case. Putting these pieces together shows that if we have a hypergraph with *independently bounded* maximum hyperedge size, then we can treat this additional work as a constant. Consequently, our solver is strongly local for graphs with bounded maximum hyperedge size; this matches the interpretation in [33].

4 LOCAL CONDUCTANCE APPROXIMATION

We give a local conductance guarantee that results from solving (6). Because of space, we focus on the case $\kappa = 0$. We prove that a sweepcut on the solution $\mathbf{x}$ of (6) leads to a Cheeger-type guarantee for conductance of the hypergraph $\mathcal{H}$ even when the seed-set size $|R|$ is 1. It is extremely difficult to guarantee a good approximation property with an arbitrary seed node, and so we first introduce a seed sampling strategy $\mathbb{P}$ with respect to a set S^* that we wish to find. Informally, the seed selection strategy says that the expected solution mass outside S^* is not too large, and more specifically, not too much larger than if you had seeded on the entire target set S^*.

Definition 4.1. Denote $\mathbf{x}(\gamma, R)$ as the solution to (6) with $\kappa = 0$. A *good sampling* strategy $\mathbb{P}$ for a target set S^* is

$$\mathbb{E}_{v \in \mathbb{P}}\left[\frac{1}{d_v}\sum_{u \in V \setminus S^*} d_u x_u(\gamma, \{v\})\right] \leq \frac{c}{\mathbf{vol}(S^*)}\sum_{u \in V \setminus S^*} d_u x_u(\gamma, S^*)$$

for some positive constant c.

Note that $\mathbf{vol}(S^*)$ is just to normalize the effect of using different numbers of seeds. For an arbitrary S^*, a good sampling strategy $\mathbb{P}$ for the standard graph case with $c = 1$ is to sample nodes from S^* proportional to their degree. Now, we provide our main theorem and show its proof in Appendix B .

Theorem 4.2. *Given a set S^* of vertices s.t. $\mathbf{vol}(S^*) \leq \frac{\mathbf{vol}(\mathcal{H})}{2}$ and $\phi_{\mathcal{H}}(S^*) \leq \frac{\gamma}{8c}$ for some positive constant γ, c. If we have a seed sampling strategy $\mathbb{P}$ that satisfies Def. 4.1, then with probability at least $\frac{1}{2}$, sweepcut on (6) with find $S_{\mathbf{x}}$ with*

$$\phi(S_{\mathbf{x}}) \leq \sqrt{32\gamma\bar{\delta}\ln(100\,\mathbf{vol}(S^*)/d_v)},$$

where $\bar{\delta} = \max_{e \in \partial S_{\mathbf{x}}} \min\{\delta_e, |e|/2\}$ where $\partial S_{\mathbf{x}} = \{e \in \mathcal{E} | e \cap S_{\mathbf{x}} \neq \emptyset, e \cap \bar{S}_{\mathbf{x}} \neq \emptyset\}$ and v is the seeded node.

The proof is in the appendix. This implies that for any set S^*, if we have a sampling strategy that matches S^*, our method can find a node set with conductance $O(\sqrt{\phi_{\mathcal{H}}(S^*)\bar{\delta}\log(\mathbf{vol}(S^*))})$ after tuning γ. The term $\bar{\delta}$ is the additional cost that we pay for performing graph reduction. The dependence on $\bar{\delta}$ essentially generalizes the previous works that analyzed the conductance with only *all-or-nothing* penalty [25, 31], as our result matches these when $\bar{\delta} = 1$. But our method gives the flexibility to choose other values δ_e and while $\bar{\delta}$ in the worst case could be as large as $|e|/2$, in practice, $\bar{\delta}$ can be chosen much smaller (See §7). Also, although we reduce $\mathcal{H}$ into a directed graph G, the previous conductance analysis for directed graphs [25, 39] is not applicable as we have degree zero nodes in G. Those degree zero nodes introduce challenges.

5 DIRECTLY RELATED WORK

We have discussed most related work in-situ throughout the paper. Here, we address a few related hypergraph PageRank vectors directly. First, Li et al. [25] defined a quadratic hypergraph PageRank by directly using Lovász extension of the splitting function f_e to control the diffusion instead of a reduction. Both Li et al. [25] and Takai et al. [31] simultaneously proved that this PageRank can be used to partition hypergraphs with an *all-or-nothing* penalty and a

Cheeger-type guarantee. Neither approach gives a strongly local algorithm and they have complexity $O(|\mathcal{E}||V|\min\{|\mathcal{E}|,|V|\}\text{poly}(\zeta))$ or in terms of Euler integration or subgradient descent.

6 GENERALIZATION TO P-NORMS

In the context of the local graph clustering, the quadratic cut objective can sometimes "over-expand" or "bleed out" over natural boundaries in the data. (This is the opposite problem to the maxflow-based clustering.) To solve this issue, [28] proposed a more general p-norm based cut objective, where $1 < p \leq 2$. The corresponding p-norm diffusion algorithm can not only grow from small seed set, but also capture the boundary better than 2-norm cut objective. Moreover, [37] proposed a related p-norm flow objective that shares similar characteristics. Our *hypergraph diffusion* framework easily adapts to such a generalization.

Definition: p-norm local hypergraph diffusions. Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, seeds R, and values γ, κ. Let $\mathbf{B}, \mathbf{w}$ again be the incidence matrix and weight vector of the localized reduced directed cut graph. A p-norm local hypergraph diffusion is:

$$\begin{array}{ll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{w}^T \ell((\mathbf{B}\mathbf{x})_+) + \kappa\gamma \sum_{i\in V} x_i d_i \\ \text{subject to} & x_s = 1, x_t = 0, \mathbf{x} \geq 0. \end{array} \tag{13}$$

Here $\ell(x) = \frac{1}{p}x^p$, $1 < p \leq 2$. And the corresponding residual function is $\mathbf{r}(\mathbf{x}) = -\frac{1}{\gamma}\mathbf{B}^T \text{diag}(\ell'((\mathbf{B}\mathbf{x})_+))\mathbf{w}$.

The idea of solving (13) is similar to the quadratic case, where the goal is to iteratively push values to x_i as long as node i violates the optimality condition, i.e. $r_i > \kappa d_i$. The challenge of solving a more general p-norm cut objective is that we no longer have a closed form solution even if the ordering of adjacent nodes is known. Thus, we need to use binary search to find Δx_i, Δx_a and Δx_b up to ε accuracy at every iteration. This means that in the worst case, the general push process can be slower than 2-norm based push process by a factor of $O(\log(1/\varepsilon))$. We defer the details of the algorithm to a longer version of the paper, but we note that a similar analysis shows that this algorithm is strongly local.

7 EXPERIMENTS

In the experiments, we will investigate both the LHQD (2-norm) and 1.4-norm cut objectives with the δ-linear threshold as the splitting function (more details about this function in §3.3). Our focus in the experiments is on the use of the methods for semi-supervised learning. Consequently, we consider how well the algorithms identify "ground truth" clusters that represent various known labels in the datasets when given a small set of seeds. (We leave detailed comparisons of the conductances to a longer version.)

In the plots and tables, we use LH-2.0 to represent our LHQD or LHPR method and LH-1.4 to represent the 1.4 norm version from §6. The other four methods we compare are:
(i) ACL [3], which is initially designed to compute approximated PageRank on graphs. Here we transform each hypergraph to a graph using three different techniques, which are star expansion (star+ACL), unweighted clique expansion (UCE+ACL) and weighted clique expansion (WCE+ACL) where a hyperedge e is replaced by a clique where each edge has weight $1/|e|$ [43]. ACL is known as one of the fastest and most successful local graph clustering algorithm in several benchmarks [28, 34] and has a similar quadratic guarantee on local graph clustering [3, 45].
(ii) flow [33], which is the maxflow-mincut based local method designed for hypergraphs. Since the flow method has difficulty growing from small seed set as illustrated in the yelp experiment in §1, we will first use the one hop neighborhood to grow the seed set. (OneHop+flow) To limit the number of neighbors included, we will order the neighbors using the *BestNeighbors* as introduced in [33] and only keep at most 1000 neighbors. (Given a seedset R, *BestNeighbors* orders nodes based on the fraction of hyperedges incident to v that are also incident to at least one node from R.)
(iii) LH-2.0+flow, this is a combination of LH-2.0 and flow where we use the output of LH-2.0 as the input to the flow method to refine.
(iv) HGCRD [17], this is a hypergraph generalization of CRD [36], which is a hybrid diffusion and flow.[1]

In order to select an appropriate δ for different datasets, Veldt et al. found that the optimal δ is usually consistent among different clusters in the same dataset [33]. Thus, the optimal δ can be visually approximated by varying δ for a handful of clusters if one has access to a subset of ground truth clusters in a hypergraph. We adapt the same procedure in our experiments and report the results in App. C. Other parameters are in the reproduction details footnote.[2]

7.1 Detecting Amazon Product Categories

In this experiment, we use different methods to detect Amazon product categories [30]. The hypergraph is constructed from Amazon product review data where each node represents a product and each hyperedge is set of products reviewed by the same person. It has 2,268,264 nodes and 4,285,363 hyperedges. The average size of hyperedges is around 17. We select 6 different categories with size between 100 and 10000 as ground truth clusters used in [33]. We set $\delta = 1$ for this dataset (more details about this choice in §C). We select 1% nodes (at least 5) as seed set for each cluster and report median F1 scores and median runtime over 30 trials in Table 1 and 2. Overall, LH-1.4 has the best F1 scores and LH-2.0 has the second best F1. The two fastest methods are LH-2.0 and star+ACL. While achieving better F1 scores, LH-2.0 is 20x faster than HyperLocal (flow) and 2-5x faster than clique expansion based methods.

7.2 Detecting Stack Overflow Question Topics

In the Stack Overflow dataset, we have a hypergraph with each node representing a question on "stackoverflow.com" and each hyperedge representing questions answered by the same user [33]. Each question is associated with a set of tags. The goal is to find questions having the same tag when seeding on some nodes with a given target tag. This hypergraph is much larger with 15,211,989

[1] Another highly active topic for clustering and semi-supervised learning involves graph neural networks (GNN). Prior comparisons between GNNs and diffusions shows mixed results in the *small seed set* regime we consider [16, 28] and complicates doing a fair comparison. As such, we focus on comparing with the most directly related work.
[2] *Reproduction details.* The full algorithm and evaluation codes can be found here https://github.com/MengLiuPurdue/LHQD. We fix the LH locality parameter γ to be 0.1, approximation parameter ρ to be 0.5 in all experiments. We set $\kappa = 0.00025$ for Amazon and $\kappa = 0.0025$ for Stack Overflow based on cluster size. For ACL, we use the same set of parameters as LH. For LH-2.0+flow, we set the flow method's locality parameter to be 0.1. For OneHop+flow, we set the locality parameter to be 0.05, 0.0025 on Amazon and Stack Overflow accordingly. For HGCRD, we set $U = 3$ (maximum flow that can be send out of a node), $h = 3$ (maximum flow that an edge can handle), $w = 2$ (multiplicative factor for increasing the capacity of the nodes at each iteration), $\alpha = 1$ (controls the eligibility of hyperedge), $\tau = 0.5$ and 6 maximum iterations.

Table 1: Median F1 scores on detecting Amazon product categories over 30 trials, the small violin plots show variance.

Alg	12	18	17	25	15	24
	F1 & Med.	F1 & Med.	F1 & Med.	F1 & Med.	F1 & Med.	F1 & Med.
LH-2.0	0.77	0.65	0.25	0.19	0.22	0.62
LH-1.4	0.9	0.79	0.32	0.22	0.27	0.77
LH-2.0+flow	0.95	0.82	0.15	0.16	0.16	0.87
star+ACL	0.64	0.51	0.19	0.15	0.2	0.49
WCE+ACL	0.64	0.51	0.2	0.14	0.21	0.51
UCE+ACL	0.27	0.09	0.06	0.05	0.11	0.14
OneHop+flow	0.52	0.6	0.16	0.12	0.09	0.22
HGCRD	0.56	0.4	0.05	0.06	0.07	0.17

Table 2: Median runtime in seconds on detecting Amazon product categories

Alg	12	18	17	25	15	24
LH-2.0	0.9	0.7	2.8	1.0	5.6	13.3
LH-1.4	8.0	6.3	32.3	9.8	53.8	127.3
LH-2.0+flow	3.5	5.1	421.1	17.8	34.9	151.5
star+ACL	0.2	0.2	0.3	0.2	0.5	0.8
WCE+ACL	18.6	17.2	19.0	16.5	21.5	20.1
UCE+ACL	9.8	10.9	11.2	10.7	13.3	15.5
OneHop+flow	308.8	141.7	359.2	224.9	81.5	82.4
HGCRD	120.3	56.4	78.1	21.2	239.4	541.3

Table 3: This table summarizes the median of median runtimes in seconds for the Stack Overflow experiments as well as median Precision, Recall and F1 over the 40 clusters.

Alg.	LH2	LH1.4	LH2	ACL	ACL	ACL	Flow	HG-
			+flow	+star	+WCE	+UCE	+1Hop	CRD
Time	3.69	39.89	43.84	1.54	15.25	13.71	48.28	72.31
Pr	0.65	0.66	0.74	0.66	0.65	0.66	0.83	0.46
Rc	0.67	0.67	0.59	0.6	0.66	0.65	0.11	0.01
F1	0.66	0.66	0.66	0.63	0.65	0.65	0.19	0.02

nodes and 1,103,243 edges. The average hyperedge size is around 24. We select 40 clusters with 2,000 to 10,000 nodes and a conductance score below 0.2 using the *all-or-nothing* penalty. (There are 45 clusters satisfying these conditions, 5 of them are used to select δ.) In this dataset, a large δ can give better results. For diffusion based methods, we set the δ-linear threshold to be 1000 (more details about this choice in App. §C), while for flow based method, we set $\delta = 5000$ based on Figure 3 of [33]. In Table 3, we summarize some recovery statistics of different methods on this dataset. In Figure 3, we report the performance of different methods on each cluster. Overall, LH-2.0 achieves the best balance between speed and accuracy, although all the diffusion based methods (LH, ACL) have extremely similar F1 scores (which is different from the last experiment). The flow based method still has difficulty growing from small seed set as we can see from the low recall in Table 3.

7.3 Varying Number of Seeds

In this section, we vary the ratio of seed set from 0.1% to 10%. At each seed ratio, denoted as r, we set $\kappa = 0.025r$. And for each of

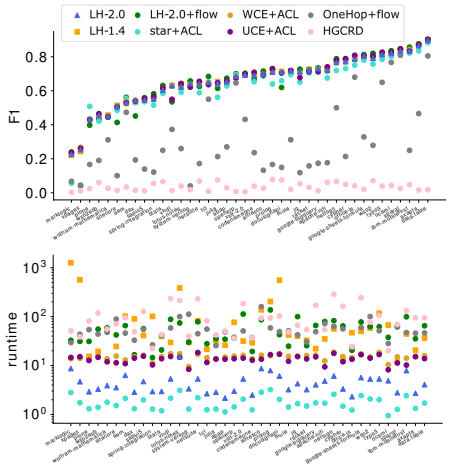


Figure 3: The upper plot shows median F1 scores of different methods over 40 clusters from the Stack Overflow dataset. The lower plot shows median running time. LH-2.0 achieves the best balance between speed and accuracy; LH-1.4 can sometimes be slower than the flow method when the target cluster contains many large hyperedges.

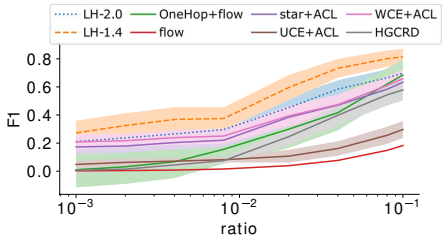


Figure 4: This plot shows the median of median F1 scores on detecting those 6 clusters in the Amazon data when varying the seed size. The envelope represents 1 standard error over the 6 median F1 scores.

the 6 clusters, we take the median F1 score over 10 trials. To have a global idea of how different methods perform on this dataset, we take another median over the 6 median F1 scores. For the flow-based method, we also consider removing the OneHop growing procedure. The results are summarized in Figure 4. We can see our hypergraph diffusion based method (LH-1.4, LH-2.0) performs better than alternatives for all seed sizes especially for small seed sets, although flow dramatically improves for large seed sizes.

8 DISCUSSION

This paper studies the opportunities for strongly local quadratic and p-norm diffusions in the context of local clustering and semi-supervised learning.

One of the distinct challenges we encountered in preparing this manuscript was comparing against the ideas of others. Clique expansions are often problematic because they can involve quadratic memory for each hyperedge if used simplistically. For running the baseline ACL PageRank diffusion on the clique expansion, we were able to use the *linear* nature of this algorithm to implicitly model the clique expansion without realizing the actual graph in memory. (We lack space to describe this though.) For others the results

were less encouraging. We, for instance, were unable to set integration parameters for the Euler scheme employed by QHPR [31] to produce meaningful results in §7.

Consequently, we wish to discuss the parameters of our method and why they are reasonably easy to set. The values γ and κ both control the size of the result. Roughly, γ corresponds to how much the diffusion is allowed to spread from the seed vertices and κ controls how aggressively we *sparsify* the diffusion. To get a bigger result, then, set γ or κ a little bit smaller. The value of ρ corresponds only to how much one of our solutions can differ from the *unique* solution of (6). Fixing $\rho = 0.5$ is fine for empirical studies unless the goal is to compare against other strategies to solve that same equation. The final parameter is δ, which interpolates between the *all-or-nothing* penalty and the *cardinality* penalty as discussed in §3.2. This can be chosen based on an experiment as we did here, or by exploring a few small choices between 1 and half the largest hyperedge size.

In closing, *flow-based* algorithms have often been explained or used as *refinement* operations to the clusters produced by spectral methods [21] as in LH-2.0+flow. As a final demonstration of this usage on the Yelp experiment from §1, we have precision, recall, and F1 result of $0.87, 0.998, 0.93$, respectively, which demonstrates how these techniques may be easily combined to even more accurately find target clusters.

REFERENCES

[1] Sameer Agarwal, Kristin Branson, and Serge Belongie. 2006. Higher Order Learning with Graphs. In *ICML*. 17–24.
[2] Sameer Agarwal, Jongwoo Lim, Lihi Zelnik-Manor, Pietro Perona, David Kriegman, and Serge Belongie. 2005. Beyond Pairwise Clustering. In *CVPR*. 838–845.
[3] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local graph partitioning using pagerank vectors. In *FOCS*. 475–486.
[4] Reid Andersen and Kevin J. Lang. 2008. An Algorithm for Improving Graph Partitions. In *SODA*. 651–660.
[5] Austin Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353 (2016), 163–166.
[6] Austin R Benson, Jon Kleinberg, and Nate Veldt. 2020. Augmented Sparsifiers for Generalized Hypergraph Cuts. *arXiv preprint arXiv:2007.08075* (2020).
[7] Avrim Blum and Shuchi Chawla. 2001. Learning from Labeled and Unlabeled Data Using Graph Mincuts. In *ICML*. 19–26.
[8] Uthsav Chitra and Benjamin J. Raphael. 2019. Random Walks on Hypergraphs with Edge-Dependent Vertex Weights. In *ICML*. 1172–1181.
[9] Fan R. L. Chung. 1992. *Spectral Graph Theory*. American Mathematical Society.
[10] D. Eckles, B. Karrer, and J. Ugander. 2017. Design and Analysis of Experiments in Networks: Reducing Bias from Interference. *J. Causal Inference* 5 (2017).
[11] K. Fountoulakis, M. Liu, D. F. Gleich, and M. W. Mahoney. 2020. Flow-based Algorithms for Improving Clusters: A Unifying Framework, Software, and Performance. *arXiv* cs.LG (2020), 2004.09608.
[12] David Gleich and Michael Mahoney. 2014. Anti-differentiating approximation algorithms: A case study with min-cuts, spectral, and flow. In *ICML*. 1018–1025.
[13] David F. Gleich and Michael W. Mahoney. 2015. Using Local Spectral Methods to Robustify Graph-Based Learning Algorithms. In *SIGKDD*. 359–368.
[14] Scott W. Hadley. 1995. Approximation techniques for hypergraph partitioning problems. *Discrete Applied Mathematics* 59, 2 (1995), 115 – 127.
[15] M. Hein, S. Setzer, L. Jost, and S. S. Rangapuram. 2013. The Total Variation on Hypergraphs - Learning on Hypergraphs Revisited. In *NeurIPS*. 2427–2435.
[16] Rania Ibrahim and David F. Gleich. 2019. Nonlinear Diffusion for Community Detection and Semi-Supervised Learning. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. ACM, New York, NY, USA, 739–750.
[17] Rania Ibrahim and David F. Gleich. 2020. Local Hypergraph Clustering using Capacity Releasing Diffusion. *arXiv* cs.SI (2020), 2003.04213.
[18] E. Ihler, D. Wagner, and F. Wagner. 1993. Modeling hypergraphs by graphs with the same mincut properties. *Inform. Process. Lett.* 45 (1993), 171–175.
[19] T. Joachims. 2003. Transductive learning via spectral graph partitioning. In *ICML*. 290–297.
[20] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. 1999. Multilevel hypergraph partitioning: applications in VLSI domain. *VLSI* 7, 1 (March 1999), 69–79.
[21] K. Lang. 2005. Fixing two weaknesses of the spectral method. In *NeurIPS*. 715–722.
[22] E. L. Lawler. 1973. Cutsets and partitions of hypergraphs. *Networks* 3, 3 (1973), 275–285.
[23] D. Lawlor, T. Budavári, and M. W Mahoney. 2016. Mapping the Similarities of Spectra: Global and Locally-biased Approaches to SDSS Galaxies. *Astrophys. J.* 833, 1 (2016), 26.
[24] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. 2009. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Math.* 6, 1 (2009), 29–123.
[25] Pan Li, Niao He, and Olgica Milenkovic. 2020. Quadratic Decomposable Submodular Function Minimization: Theory and Practice. *JMLR* 21 (2020), 1–49.
[26] Pan Li and Olgica Milenkovic. 2017. Inhomogeneous Hypergraph Clustering with Applications. In *NeurIPS*. 2308–2318.
[27] Pan Li and Olgica Milenkovic. 2018. Submodular Hypergraphs: p-Laplacians, Cheeger Inequalities and Spectral Clustering. In *ICML*, Vol. 80. 3014–3023.
[28] Meng Liu and David F. Gleich. 2020. Strongly local p-norm-cut algorithms for semi-supervised learning and local graph clustering. arXiv:2006.08569 [cs.SI]
[29] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi. 2012. A Local Spectral Method for Graphs: With Applications to Improving Graph Partitions and Exploring Data Graphs Locally. *JMLR* 13 (2012), 2339–2365.
[30] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *EMNLP-IJCNLP*.
[31] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. 2020. Hypergraph Clustering Based on PageRank. In *KDD*. 1970–1978.
[32] Nate Veldt, Austin R. Benson, and Jon Kleinberg. 2020. Hypergraph Cuts with General Splitting Functions. arXiv:2001.02817 [cs.DS]
[33] Nate Veldt, Austin R Benson, and Jon Kleinberg. 2020. Minimizing Localized Ratio Cut Objectives in Hypergraphs. In *KDD*. 1708–1718.
[34] Nate Veldt, David F. Gleich, and Michael W. Mahoney. 2016. A Simple and Strongly-Local Flow-Based Method for Cut Improvement. In *ICML*. 1938–1947.
[35] Nate Veldt, Christine Klymko, and David F. Gleich. 2019. Flow-Based Local Graph Clustering with Better Seed Set Inclusion. In *SDM*. 378–386.
[36] D. Wang, K. Fountoulakis, M. Henzinger, M. W. Mahoney, and S. Rao. 2017. Capacity releasing diffusion for speed and locality. In *ICML*. 3598–3607.
[37] Shenghao Yang, Di Wang, and Kimon Fountoulakis. 2020. p-Norm Flow Diffusion for Local Graph Clustering. *arXiv preprint arXiv:2005.09810* (2020).
[38] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *KDD*. 555–564.
[39] Yuichi Yoshida. 2016. Nonlinear Laplacian for digraphs and its applications to network analysis. In *WSDM*. 483–492.
[40] Yuichi Yoshida. 2019. Cheeger Inequalities for Submodular Transformations. In *SODA*. 2582–2601.
[41] Chenzi Zhang, Shuguang Hu, Zhihao Gavin Tang, and T-H. Hubert Chan. 2017. Re-Revisiting Learning on Hypergraphs: Confidence Interval and Subgradient Method. In *ICML*. 4026–4034.
[42] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. 2003. Learning with Local and Global Consistency. In *NIPS*.
[43] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *NeurIPS*. 1601–1608.
[44] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. 2003. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML*. 912–919.
[45] Zeyuan Allen Zhu, Silvio Lattanzi, and Vahab S Mirrokni. 2013. A Local Algorithm for Finding Well-Connected Clusters.. In *ICML (3)*. 396–404.
[46] J. Y. Zien, M. D. F. Schlag, and P. K. Chan. 1999. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE TCAD* 18, 9 (1999), 1389–1399.

A PROOF OF THEOREM 3.5

First we have the following observations on $\mathbf{r}$ and $\mathbf{x}$.

Lemma A.1. *At any iteration of Algorithm 1, for each pair of auxiliary nodes, $a \in V_a$, $b \in V_b$ and $a \to b$, $x_a \geq x_b$.*

Lemma A.2. *At any iteration of Algorithm 1, for any $i \in V \cup V_a \cup V_b$, g_i will stay nonnegative and $0 \leq x_i \leq 1$.*

Both lemmas can be easily proved by contradiction. More detailed proof would be included in a longer version of this material.

Proof of Theorem 3.5. By using Lemma A.2, $\|\mathbf{g}\|_1$ becomes

$$\|\mathbf{g}\|_1 = \sum_{i \in V \cup V_a \cup V_b} g_i = \sum_{i \in R} d_i(1 - x_i) - \sum_{i \in \bar{R}} d_i x_i$$

This implies that any change to the auxiliary nodes will not affect $||\mathbf{g}||_1$. Thus calling `LHQD-auxpush` doesn't change $||\mathbf{g}||_1$. When there exists $i \in V$ such that $g_i > \kappa d_i$, then `hyper-push` will find Δx_i such

that $g_i' = \rho\kappa d_i$. Then the new g_i' can be written as

$$g_i' = \frac{1}{\gamma}\sum_{b\in V_b} w_{bi}(x_b - x_i - \Delta x_i)_+ - \frac{1}{\gamma}\sum_{a\in V_a} w_{ia}(x_i + \Delta x_i - x_a)_+ + \kappa d_i(\text{Ind}[i \in R] - x_i - \Delta x_i) \quad (14)$$

Note g_i' is a decreasing function of Δx_i and $g_i' > 0$ when $\Delta x_i = 0$, $g_i' < 0$ when $\Delta x_i = 1$ by using Lemma A.2. This suggests that there exists a unique Δx_i that satisfies $g_i' = \rho\kappa d_i$. Moreover, $(x_b - x_i - \Delta x_i)_+ \geq (x_b - x_i)_+ - \Delta x_i$ and $(x_i + \Delta x_i - x_a)_+ \leq (x_i - x_a)_+ + \Delta x_i$, thus we have

$$\rho\kappa d_i = g_i' \geq g_i - \tfrac{1}{\gamma}(\textstyle\sum_{b\in V_b} w_{bi} + \sum_{a\in V_a} w_{ia})\Delta x_i - \kappa d_i \Delta x_i$$

From equation (4.9) of [32],

$$\textstyle\sum_{b\in V_b} w_{bi} = \sum_{a\in V_a} w_{ai} = \sum_{e\in\mathcal{E}, i\in e} f_e(\{i\}) \leq \delta d_i$$

Thus, we have $d_i\Delta x_i \geq \frac{g_i - g_i'}{\kappa + \delta/\gamma} > \frac{\gamma\kappa(1-\rho)}{\gamma\kappa+\delta} d_i$ So the decrease of $||\mathbf{g}||_1$ will be at least $\gamma\kappa(1-\rho)d_i/(\gamma\kappa+\delta)$. Since $||\mathbf{g}||_1 = \text{vol}(R)$ initially, we have $\sum_{i=1}^{T} d_i \leq (\gamma\kappa+\delta)\text{vol}(R)/\gamma\kappa(1-\rho) = O(\text{vol}(R))$.

B PROOF OF THEOREM 4.2

We first introduce some simplifying notation. We use 1_v to denote the canonical basis with vth component as 1 and others as 0 and $1_S = \sum_{v\in S} 1_v$. Without loss of generality, we assume each gadget reduced from a hyperedge has weight $c_e = 1$. Otherwise one can simply add c_e as the coefficients and nothing needs to change. We omit the degree regularization term in (6). Furthermore, we normalize the seeds to guarantee $\sum_{v\in V} d_v x_v = 1$ and group terms in (6) to remove the source $x_s = 1$ and sink $x_t = 0$

$$\underset{\mathbf{x}}{\text{minimize}} \quad Q(\mathbf{x}) \triangleq \gamma \sum_{v\in V} d_v \left(x_v - \frac{1}{\mathbf{vol}(S)} 1_S\right)^2 + \sum_{e\in\mathcal{E}} Q_e(\mathbf{x}) \quad (15)$$

where

$$Q_e(\mathbf{x}) \triangleq \min_{x_a^{(e)}, x_b^{(e)}} \sum_{v\in e}\left[(x_v - x_a^{(e)})_+^2 + (x_b^{(e)} - x_v)_+^2\right] + \delta_e(x_a^{(e)} - x_b^{(e)})_+^2. \quad (16)$$

We denote $M = \mathbf{vol}(\mathcal{H}) = \sum_{v\in V} d_v$. We denote the solution $\mathbf{x}$ with the parameter γ and the seed set S of the optimization (15) as $\mathbf{x}(\gamma, S)$ and its component for node v as $x_v(\gamma, S)$. We also define another degree weighted vector $p = D\mathbf{x}$, where D is the diagonal degree matrix. For a vector p and a node set S', we use $p(S')$ to denote $p(S') = \sum_{v\in S'} p_v$. It is easy to check that $p(\gamma, S)(V) = 1$ for any S. For a node set S, define $\partial S = \{e \in \mathcal{E} | e \cap S \neq \emptyset, e \cap V\backslash S \neq \emptyset\}$.

We now define our main tool: the Lovász-Simonovits Curve.

Definition B.1 (Lovász-Simonovits Curve (LSC)). Given an $\mathbf{x}$, we order its components from large to small by breaking equality arbitrarily, say $x_1, x_2, ..., x_n$ w.l.o.g, and define $S_j^{\mathbf{x}} = \{x_1, ..., x_j\}$. LSC defines a corresponding piece-wise function $I_{\mathbf{x}} : [0, M] \to R$ s.t $I_{\mathbf{x}}(0) = 0$, $I_{\mathbf{x}}(\mathbf{vol}(G)) = 1$ and $I(\mathbf{vol}(S_j^{\mathbf{x}})) = p(S_j^{\mathbf{x}})$. And for any $k \in [\mathbf{vol}(S_j^{\mathbf{x}}), \mathbf{vol}(S_{j+1}^{\mathbf{x}})]$,

$$I_{\mathbf{x}}(k) = I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}})) + (k - \mathbf{vol}(S_j^{\mathbf{x}}))x_j.$$

Our proof depends on the following two Lemma B.2 and B.3 that will characterize the upper and lower bound of $I_{\mathbf{x}}$, which finally leads to the main theorem.

Lemma B.2. *For a set S, given an $\mathbf{x} = \mathbf{x}(\gamma, S)$, let $\phi_{\mathbf{x}}$ and $S_{\mathbf{x}}$ be the minimal conductance and the node set obtained through a sweep-cut over $\mathbf{x}$. For any integer $t > 0$ and $k \in [0, M]$, the following bound holds*

$$I_{\mathbf{x}}(k) \leq \frac{k}{M} + \frac{\gamma t}{2+\gamma} + \sqrt{\frac{\min(k, M-k)}{\min_{i\in S} d_i}}(1 - \frac{\sigma_{\mathbf{x}}^2\phi_{\mathbf{x}}^2}{8})^t$$

where $\sigma_{\mathbf{x}} = (2\max_{e\in\partial S_{\mathbf{x}}} \min\{\delta_e, |e|/2\} + 1)^{-1}$.

Lemma B.3. *For a set S, if a node $v \in S$ is sampled according to a distribution $\mathbb{P}$ s.t*

$$\mathbb{E}_{v\sim\mathbb{P}}[p(\gamma, \{v\})(\bar{S})] \leq cp(\gamma, S)(\bar{S}), \quad (17)$$

where c is a constant, then with probability at least $\frac{1}{2}$, one has

$$p(\gamma, \{v\})(S) \geq 1 - 2c\phi(S)/\gamma.$$

Lemma B.3 gives the lower bound $I_{\mathbf{x}(\gamma,\{v\})}(\mathbf{vol}(S))$ as this value is no less than $p(\gamma, \{v\})(S)$. Note that the sampling assumption of Lemma B.3 is natural in the standard graph case, when $\mathbb{P}$ samples each node proportionally to its degree, we have an equality with $c = 1$ in (17). Combining Lemma B.2 and B.3, we arrive at

Theorem B.4. *Given a set S^* of vertices s.t. $\boldsymbol{vol}(S^*) \leq \frac{M}{2}$ and $\phi(S^*) \leq \frac{\gamma}{8c}$ for some positive constants γ, c. If there exists a distribution $\mathbb{P}$ s.t. $\mathbb{E}_{v\sim\mathbb{P}}[p(\gamma, \{v\})](\bar{S}^*) \leq cp(\gamma, S^*)(\bar{S}^*)$, then with probability at least $\frac{1}{2}$, the obtained conductance satisfies*

$$\phi_{\mathbf{x}} \leq \sqrt{32\gamma \max_{e\in\partial S_{\mathbf{x}}} \min\left\{\delta_e, \frac{|e|}{2}\right\} ln\left(100\frac{\boldsymbol{vol}(S^*)}{d_v}\right)},$$

where $\mathbf{x} = \mathbf{x}(\gamma, \{v\})$ and v is sampled from $\mathbb{P}$. $S_{\mathbf{x}}$ is the node set obtained via the sweep-cut over $\mathbf{x}$.

Proof. We combine Lemma B.3 and B.2 and use the same technique as Thm. 17 in [25] (Sec. 7.7.3). □

By removing the normalizing on the number of seeded nodes, Thm. B.4 becomes Thm. 4.2.

B.1 Proof of Lemma B.2

Define $L_e(\mathbf{x}) \triangleq \nabla_{\mathbf{x}} \frac{1}{2} Q_e(\mathbf{x})$ (16) and with some algebra, we have

$$L_e(\mathbf{x}) = \sum_{v\in e}\left[(x_v - x_a^{(e)*})_+ - (x_b^{(e)*} - x_v)_+\right] 1_v,$$

where $x_a^{(e)*}$ and $x_b^{(e)*}$ are the optimal values in (16). In the following, we will first prove Lemma B.5 and further use it to prove Lemma B.6. The same proof in Thm. 16 in [25] (Sec. 7.7.2) can be used to leverage Lemma B.6 to prove Lemma B.2.

Lemma B.5. *Given an $\mathbf{x}$, we order its components from large to small, say $x_1, x_2, ..., x_n$ w.l.o.g., and define $S_j^{\mathbf{x}} = \{x_1, ..., x_j\}$. Define $\sigma_j^{\mathbf{x}} = (1 + 2\max_{e\in\partial S_j^{\mathbf{x}}} \min\{\delta_e, |e|/2\})^{-1}$, and we have*

$$2I_{\mathbf{x}}(\boldsymbol{vol}(S_j^{\mathbf{x}})) - \langle \sum_{e\in E} L_e(x), 1_{S_j^{\mathbf{x}}} \rangle \leq I_{\mathbf{x}}(\boldsymbol{vol}(S_j^{\mathbf{x}}) - \sigma_j^{\mathbf{x}} cut(S_j^{\mathbf{x}})) + I_{\mathbf{x}}(\boldsymbol{vol}(S_j^{\mathbf{x}}) + \sigma_j^{\mathbf{x}} cut(S_j^{\mathbf{x}})).$$

Lemma B.6. *Suppose* $\mathbf{x} = \mathbf{x}(\gamma, S)$, $\mathbf{x}_0 = 1_S/\mathbf{vol}(S)$ *and* $\sigma_j^{\mathbf{x}} = (1 + 2\max_{e\in\partial S_j^{\mathbf{x}}} \min\{\delta_e, |e|/2\})^{-1}$. *We have*

$$I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}})) \le \frac{\gamma}{2+\gamma} I_{\mathbf{x}_0}(S_j^{\mathbf{x}_0}) + \frac{1}{2+\gamma}(I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}}) - \sigma_j^{\mathbf{x}} cut(S_j^{\mathbf{x}}))+ I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}}) + \sigma_j^{\mathbf{x}} cut(S_j^{\mathbf{x}}))). \quad (18)$$

Furthermore, for $k \in [0, M]$, $I_{\mathbf{x}}(k) \le I_{\mathbf{x}_0}(k)$.

Proof of Lemma B.5. Given a hyperedge e, we order $\{x_v \mid v \in e\}$ from large to small by breaking equality arbitrarily and obtain $x_1^{(e)}, x_2^{(e)}, ..., x_e^{(e)}$. Suppose $x_k^{(e)} \in S_j^{\mathbf{x}}$ and $x_{k+1}^{(e)} \notin S_j^{\mathbf{x}}$. Then,

$$\langle L_e(x), 1_{S_j^{\mathbf{x}}}\rangle = \sum_{i=1}^{k}\left[(x_i^{(e)} - x_a^{(e)*})_+ - (x_b^{(e)*} - x_i^{(e)})_+\right] \quad (19)$$

Next, we will bound (19) by analyzing three cases. We only focus on $x_a^{(e)*} > x_b^{(e)*}$ and otherwise x_v's are constant for all $v \in e$. Also denote $k_+ = \max\{i \mid x_i^{(e)} > x_a^{(e)*}\}$, $k_- = \min\{e + 1 - i \mid x_i^{(e)} < x_b^{(e)*}\}$, and $k^- = |e| + 1 - k_-$. By the optimality of $x_a^{(e)*}$, $x_b^{(e)*}$, we have

$$x_a^{(e)*} = (k_- + \delta_e)X_1^{k_+} + \delta_e X_{k^-}^{|e|}, \quad x_b^{(e)*} = \delta_e X_1^{k_+} + (k_+ + \delta_e)X_{k^-}^{|e|},$$

where $X_{i_1}^{i_2} = (k_+k_- + \delta_e(k_+ + k_-))^{-1}\sum_{i=i_1}^{i_2} x_i^{(e)}$.

Thus, we have

$$\langle L_e(x), 1_{S_j^{\mathbf{x}}}\rangle = \begin{cases} [(k_+ - k)(k_- + \delta_e) + k_-\delta_e]X_1^k & \text{if } k \le k_+ \\ \quad -k(k_- + \delta_e)X_{k+1}^{k_+} - k\delta_e X_{k^-}^{|e|}, & \\ k_-\delta_e X_1^{k_+} - k_+\delta_e X_{k^-}^{|e|}, & \text{if } k_+ \le k \le k^- \\ (|e| - k)\delta_e X_1^{k_+} + (|e| - k)(k_+ + \delta_e)X_{k^-}^{k} & \text{if } k \ge k^-. \\ \quad -[k_+\delta_e + (k - k^-)(k_+ + \delta_e)]X_k^{|e|} & \end{cases} \quad (20)$$

By using the definition of $X_{i_1}^{i_2}$, noticing that all coefficients on $x_i^{(e)} \le 1$ in the left hand side of (20), and a good deal of algebra, we can further show

$$\begin{cases} \frac{k(k_+-k)(k_-+\delta_e)+k_-\delta_e}{k_+k_-+\delta_e(k_++k_-)} \ge \frac{2}{\delta_e'+2}\min\{k, |e|-k, \delta_e\} & \text{if } k \le k_+ \\ \frac{k_+k_-\delta_e}{k_+k_-+\delta_e(k_++k_-)} \ge \frac{1}{2\delta_e'+1}\min\{k, |e|-k, \delta_e\}, & \text{if } k_+ \le k \le k^- \\ \frac{k_-[k_+\delta_e+(k-k^-)(k_++\delta_e)]}{k_+k_-+\delta_e(k_++k_-)} \ge \frac{2}{\delta_e'+2}\min\{k, |e|-k, \delta_e\}, & \text{if } k \ge k^-. \end{cases}$$

where $\delta_e' = \min\{\delta_e, |e|/2\}$. In each case of (20), the sum of positive coefficient before each $x_i^{(e)}$ equals to the sum of negative coefficients, which are both lower bounded by $\frac{1}{2\delta_e'+1}$ times the splitting cost of e. Therefore,

$$2I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}})) - \sum_{e\in E}\langle L_e(x), 1_{S_j^{\mathbf{x}}}\rangle \le I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}}) - \sigma_j^{\mathbf{x}} cut(S_j^{\mathbf{x}})) + I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}}) + \sigma_j^{\mathbf{x}} cut(S_j^{\mathbf{x}})),$$

where $\sigma_j^{\mathbf{x}} = 1/(2\max_{e\in\partial S_j^{\mathbf{x}}}\delta_e' + 1)$, which concludes the proof.

Proof of Lemma B.6. Compute the derivative of $Q(\mathbf{x})$ (15) w.r.t. $\mathbf{x}$ and use the optimality of $\mathbf{x} = \mathbf{x}(\gamma, S)$ to get $0 = \gamma D(\mathbf{x} - \mathbf{x}_0) + L_e(\mathbf{x})$. Therefore,the inner product with $1_{S_j^{\mathbf{x}}}$

$$0 = \gamma(I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}})) - I_{\mathbf{x}_0}(\mathbf{vol}(S_j^{\mathbf{x}_0}))) + \langle L_e(\mathbf{x}), 1_{S_j^{\mathbf{x}}}\rangle.$$

Plug in Lemma B.5 and we achieve (18). By using the concavity of $I_{\mathbf{x}}$, we obtain $I_{\mathbf{x}}(\mathbf{vol}(S_j^{\mathbf{x}})) \le I_{\mathbf{x}_0}(\mathbf{vol}(S_j^{\mathbf{x}_0}))$ and therefore $I_{\mathbf{x}}(k) \le I_{\mathbf{x}_0}(k)$ for any $k \in [0, M]$.

B.2 Proof of Lemma B.3

If the following Lemma B.7 is true, then we have $\mathbb{E}_{v\sim\mathbb{P}}[p(\gamma, \{v\})(\bar{S})] \le cp(\gamma, S)(\bar{S}) = c(1 - p(\gamma, S)(S)) \le c\phi(S)/\gamma$. By using Markov's inequality, with probability $\frac{1}{2}$, we will sample a node v such that $p(\gamma, \{v\})(\bar{S}) \le 2c\phi(S)/\gamma$, which concludes the proof.

Lemma B.7. *For any* $S \subset V$, $p(\gamma, S)(S) \ge 1 - \phi(S)/\gamma$.

Proof. This mass from the nodes in S to the nodes in $\bar{S}$ naturally diffuses from the auxiliary nodes $v_a^{(e)}$ to $v_b^{(e)}$ for $e \in \partial S$. As need to lower bound $p(\gamma, S)(S)$, we may consider fixing $x_v = 0, \forall v \in \bar{S}$ of $Q(\mathbf{x})$ and the obtained solution $\tilde{\mathbf{x}}$ naturally satisfies

$$p(\gamma, S)(S) \ge \sum\nolimits_{v\in S} d_v\tilde{x}_v, \text{ where } \tilde{\mathbf{x}} \triangleq \arg\min_{\mathbf{x}} Q(\mathbf{x})|_{x_v=0,\forall v\in\bar{S}}.$$

The optimality of $\tilde{x}_a^{(e)}, \tilde{x}_b^{(e)}$ for $e \in \partial S$ in this case implies

$$-\textstyle\sum_{v\in e}(\tilde{x}_v - \tilde{x}_a^{(e)})_+ + \delta_e(\tilde{x}_a^{(e)} - \tilde{x}_b^{(e)})_+ = 0,$$
$$\textstyle\sum_{v\in e}(-\tilde{x}_v + \tilde{x}_b^{(e)})_+ - \delta_e(\tilde{x}_a^{(e)} - \tilde{x}_b^{(e)})_+ = 0$$

As $\tilde{x}_v = 0$ for $v \in e\backslash S$ and $\tilde{x}_v \le \frac{1}{\mathbf{vol}(S)}$ for $v \in e \cap S$, we have

$$\tilde{x}_b^{(e)} \ge \frac{\delta_e}{\delta_e + |e\backslash S|}\tilde{x}_a^{(e)}, \ \tilde{x}_a^{(e)} \le \frac{|e\cap S|}{\delta_e + |e\cap S|}\frac{1}{\mathbf{vol}(S)},$$

and further, $\forall e \in \partial S$,

$$\textstyle\sum_{v\in e\cap S}(\tilde{x}_v - \tilde{x}_a^{(e)})_+ = \delta_e(\tilde{x}_a^{(e)} - \tilde{x}_b^{(e)})_+ \quad (21)$$
$$\le \frac{|e\cap S||e\backslash S|\delta_e}{(\delta_e + |e\cap S|)(\delta_e + |e\backslash S|)}\frac{1}{\mathbf{vol}(S)} \le \frac{\min\{|e\cap S|, |e\backslash S|, \delta_e\}}{\mathbf{vol}(S)}.$$

The optimality of $\tilde{\mathbf{x}}$ implies

$$\sum_{v\in S}\gamma d_v\left(\tilde{x}_v - \frac{1}{\mathbf{vol}(S)}\right) + \sum_{e\in\partial S}\sum_{v\in e\cap S}(\tilde{x}_v - \tilde{x}_a^{(e)})_+ = 0. \quad (22)$$

Here we use that $\sum_{e\subset S}\sum_{v\in e}[(\tilde{x}_v - \tilde{x}_a^{(e)})_+ + (\tilde{x}_b^{(e)} - \tilde{x}_v)_+] = 0$. Plug (21) into (22) and we have which concludes the proof.

$$0 \le \gamma\left(\sum_{v\in S} d_v\tilde{x}_v - 1\right) + \sum_{e\in\partial S}\frac{\min\{|e\cap S|, |e\backslash S|, \delta_e\}}{\mathbf{vol}(S)}, \quad \phi(S)$$

C SELECTING δ.

To select δ for each dataset, we run LH-2.0 on a handful of alternative clusters as we vary δ. Below, we show F1 scores on those clusters and pick $\delta = 1$ for Amazon and $\delta = 1000$ for Stack Overflow.