Cut-matching Games for Generalized Hypergraph Ratio Cuts

Nate Veldt
nveldt@tamu.edu
Texas A&M University
College Station, Texas, USA

ABSTRACT

Many social networks and web-based datasets are characterized by multiway interactions (e.g., groups of co-purchased online retail products or group conversations in Q&A forums) and hypergraph clustering is a fundamental primitive for analyzing these types of interactions. We present an $O(\log n)$-approximation algorithm for a broad class of hypergraph *ratio cut* objectives. This includes objectives involving generalized hypergraph cut functions, which allow a user to penalize cut hyperedges differently depending on the number of nodes in each cluster. Our method generalizes the cut-matching framework for graph ratio cuts, and relies only on solving maximum s-t flow problems in a special reduced graph. It is significantly faster than existing hypergraph ratio cut algorithms, while also solving a more general problem. In numerical experiments on various web-based hypergraphs, we show that it quickly finds ratio cut solutions within a small factor of optimality.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; **Approximation algorithms**; **Hypergraphs**.

KEYWORDS

hypergraphs, clustering, ratio cuts, cut-matching games

ACM Reference Format:
Nate Veldt. 2023. Cut-matching Games for Generalized Hypergraph Ratio Cuts. In *Proceedings of the ACM Web Conference 2023 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583539

1 INTRODUCTION

Graphs are a popular way to model social networks and web-based datasets, but there has been a growing realization that many complex systems of interactions on the web are characterized by *multiway* relationships that cannot be directly encoded using graph edges. For example, users on social media join interest groups and participate in group events, online shoppers purchase multiple products at once, discussions in Q&A forums typically involve multiple participants, and email communication often involves multiple receivers. These higher-order interactions can be modeled by a *hypergraph*: a set of nodes that share multiway relationships called hyperedges. A standard primitive for analyzing group interaction data is to perform clustering, i.e., identifying groups of related nodes in a hypergraph. This is a direct generalization of the well-studied *graph* clustering problem. Hypergraph clustering methods have already been used to address numerous web-based data analysis tasks, such as detecting different types of Amazon products based on groups of co-reviewed products [43], identifying related vacation rentals on Trivago from user browsing behavior [10], clustering restaurants based on Yelp reviews [28], and finding related posts in online Q&A forums from group discussion data [44].

A "good" cluster in a (hyper)graph is a set of nodes with many internal (hyper)edges, but few (hyper)edges connecting different clusters. One of the most standard ways to formalize this goal is to minimize a ratio cut objective, which measures the ratio between the cluster's *cut* and some notion of the cluster's size. In the graph setting, the cut simply counts the number of edges across the cluster boundary. The standard hypergraph cut function similarly counts the number of hyperedges on the boundary. However, unlike the graph setting, there is more than one way to partition a hyperedge across two clusters. This has led to the notion of a generalized hypergraph cut function, which assigns different penalties to different ways of splitting a hyperedge. The performance of downstream hypergraph clustering applications can strongly depend on the type of generalized cut penalty that is used [13, 26–28, 43–45].

Previous work and limitations. Common ratio cut objectives include conductance and expansion, which have been defined both for graphs and hypergraphs (see Section 2). These are NP-hard even in the graph case, but can be approximated using spectral methods, which often come with so-called *Cheeger* inequality guarantees [11], or by using various types of multicommodity-flow and expander embedding methods [5, 21, 24, 36]. Several techniques for approximating graph ratio cuts have already been extended to hypergraphs, but there are many new challenges in this setting. First of all, many hypergraph Laplacian operators are nonlinear, which leads to additional challenges in computing or approximating eigenvectors to use for clustering [7, 27, 30, 51]. Secondly, there are many challenges associated with obtaining guarantees for generalized hypergraph cut functions, which can be much more general than graph cut functions. For these and other reasons, there is currently a wide gap between theoretical algorithms and practical techniques for minimizing hypergraph ratio cut objectives. The best approximation algorithms rely on expensive convex relaxations (e.g., memory intensive relaxations with $O(n^3)$ constraints [20, 31]) and complicated rounding techniques that are not implemented in practice [7, 20, 27, 31, 51]. Another downside specifically for spectral and random-walk based techniques [6, 7, 25–27, 40, 51] is that their Cheeger-like approximation guarantees are $O(n)$ even in the graph setting. In the hypergraph setting, their approximation factors also often get worse as the maximum hyperedge size

WWW '23, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583539

grows [6, 7, 26]. A limitation in another direction is that many existing hypergraph ratio cut algorithms are designed only for the simplest notion of a hypergraph cut function rather than generalized cuts [6, 7, 20, 31, 40, 52]. Finally, although a number of recent techniques apply to generalized hypergraph cut functions and come with practical implementations, these do not provide theoretical guarantees for global ratio cuts because they are either heuristics [27] or because they focus on localized objectives that are biased to a specific region of a large hypergraph [13, 17, 28, 43, 44].

The present work: practical approximation algorithms. We present the first algorithm for hypergraph ratio cuts that simultaneously (a) comes with a nontrivial approximation guarantee (i.e., better than $O(n)$) and (b) applies to generalized hypergraph cuts. Additionally, compared with approximation algorithms that apply only to the standard hypergraph cut function, our method comes with a substantially faster runtime guarantee and a practical implementation. In more detail, our algorithm has an $O(\log n)$ approximation guarantee and applies to any cardinality-based submodular hypergraph cut function, which is a general cut function that captures many popular hypergraph cut functions as special cases [45]. It relies only on solving a sequence of maximum s-t flow problems in a special type of reduced graph, which can be made efficient using an off-the-shelf max-flow subroutine.

Our method generalizes the *cut-matching* framework for graph ratio cuts [21, 36], which alternates between solving max-flows and finding bisections in order to embed an expander into the graph. Although maximum flow problems and expander embeddings are well-understood in the graph setting, these concepts are more nuanced and not as well understood in hypergraphs, especially in the case of generalized hypergraph cuts. We overcome these technical challenges by providing a new approach for embedding an expander *graph* inside a hypergraph, which is accomplished by solving maximum flow problems in an reduced directed graph that models the generalized cut properties of the hypergraph. To summarize:

- We present a new framework for embedding expander graphs into hypergraphs in order to obtain lower bounds for NP-hard hypergraph ratio cut objectives (Lemma 3.2).
- We present an $O(\log n)$-approximation algorithm that applies to any hypergraph ratio cut objective with a submodular cardinality-based cut function and any positive node weight function, and relies simply on solving a sequence of graph max-flow problems.
- We provide additional practical techniques for making our method more efficient in practice and for obtaining improved lower bounds and a posteriori approximation guarantees (Theorem 5.3).
- We implement our method and show that it is orders of magnitude more scalable than previous approximation algorithms and allows us to detect high quality ratio cuts in large hypergraphs arising in various types of web-based data analysis tasks.

An extended version of the paper is available online [42].

2 PRELIMINARIES AND RELATED WORK

Let $G = (V, E)$ denote a graph with $n = |V|$ nodes, and an edge set E. We denote edges in undirected graphs using bracket notation $e = \{u, v\} \in E$; an edge may additionally be associated with a nonnegative edge weight $w_G(e) = w_G(u, v) \geq 0$. Given a subset of nodes $S \subseteq V$, let $\bar{S} = V \backslash S$. The boundary of S is

$$\partial_G(S) = \partial_G(\bar{S}) = \{e \in E \colon |e \cap S| = 1 \text{ and } |e \cap \bar{S}| = 1\}. \quad (1)$$

This is the set of edges with one endpoint in S and the other endpoint in $\bar{S}$. The *cut* of S is the weight of edges on the boundary:

$$\mathbf{cut}_G(S) = \mathbf{cut}_G(\bar{S}) = \textstyle\sum_{e \in \partial_G(S)} w_G(e). \quad (2)$$

Given a positive node weight function $\pi \colon V \to \mathbb{R}_{>0}$, the ratio cut objective with respect to π for a set S is denoted by

$$\phi_G(S, \pi) = \mathbf{cut}_G(S) \big/ \min\{\pi(S), \pi(\bar{S})\}, \quad (3)$$

where $\pi(S) = \sum_{u \in S} \pi(u)$. This captures the well-known special case of *conductance* (when $\pi(u)$ equals node degree d_u), and *expansion* (when $\pi(u) = 1$). We will refer to $\phi_G(S, \pi)$ as the π-expansion of S. The π-expansion of a graph G is then

$$\phi_{G,\pi} = \min_{S \subset V} \phi_G(S, \pi). \quad (4)$$

We may drop G and π from subscripts when clear from context. For a given node weight function π, a graph $G = (V, E)$ is a π-expander if there exists a constant $c > 0$ such that for every $S \subseteq V$, $\phi(S) > c$.

Directed graphs. Many of our results apply to directed graphs. In this case, we will use parenthesis notation to denote edge directions, i.e. $(u, v) \in E$ indicates an edge from node u to node v. For a set $S \subseteq V$ the directed cut function is

$$\mathbf{cut}_G(S) = \textstyle\sum_{(u,v) \in E \colon u \in S, v \in \bar{S}} w_G(u, v).$$

Expansion in a directed graph is defined by using a directed cut function in (3). We can treat an undirected graph G as a directed graph by replacing each undirected edge with two directed edges, in which case the two notions of π-expansion match.

2.1 Graph flows

A flow function $f \colon E \to \mathbb{R}_{\geq 0}$ on a directed graph $G = (V, E)$ assigns a nonnegative flow value $f_{ij} \geq 0$ to each edge $(i, j) \in E$. If $f_{ij} \leq w_G(i, j)$ for each $(i, j) \in E$, then f satisfies *capacity constraints*. In general, the *congestion* of f is the maximum capacity violation:

$$\mathbf{congestion}(f) = \max_{(i,j) \in E} f_{ij} / w_G(i, j). \quad (5)$$

We say that f satisfies flow constraints at a node $v \in V$ if the flow into v equals the flow out of v:

$$\textstyle\sum_{i \colon (i,v) \in E} f_{iv} = \sum_{j \colon (v,j) \in E} f_{vj}. \quad (6)$$

If the flow into v is greater than the flow out of v then v is an *excess* node. If the flow out of v is more than the flow into v then v is a *deficit* node. Given two flow functions $f^{(1)}$ and $f^{(2)}$, the sum $f' = f^{(1)} + f^{(2)}$ is the flow obtaining by defining $f'_{ij} = f^{(1)}_{ij} + f^{(2)}_{ij}$ for each $(i, j) \in E$. This flow satisfies $\mathbf{congestion}(f') \leq \mathbf{congestion}(f^{(1)}) + \mathbf{congestion}(f^{(2)})$. If $f^{(1)}$ and $f^{(2)}$ both satisfy flow constraints at a node v, then f' will as well. We will consider two special types of flows, and make use of a standard flow decomposition result (Lemma 2.1).

Definition 2.1 (s-t flow). *Given $\{s, t\} \subseteq V$, an s-t flow on G is a flow that satisfies flow constraints on each $v \in V - \{s, t\}$. We say that f routes flow from s to t, and has a flow value*

$$|f| = \textstyle\sum_{i \colon (s,i) \in E} f_{si} - \sum_{j \colon (j,s) \in E} f_{js}. \quad (7)$$

Definition 2.2 (multicommodity flow). *A multicommodity flow problem in G is defined by a set $\mathcal{D}$ of demand pairs $(u, v) \in V \times V$ and corresponding weights $w_{uv} \geq 0$. The flow f is a feasible multicommodity flow for $\mathcal{D}$ if it can be written as $f = \sum_{(u,v)\in\mathcal{D}} f^{(uv)}$ where $f^{(uv)}$ is a u-v flow that routes w_{uv} units of flow from u to v.*

Lemma 2.1. *(Theorem 3.5 in [1] or Lemma 2.20 in [49].) Let f be a flow function on a graph $G = (V, E)$ with n nodes and m edges. The flow can be decomposed as $f = \sum_{i=1}^{\ell} f_i$ where $\ell \leq m + n$, and where for each i, the edges with positive flow in f_i either form a simple directed path from a deficit node to an excess node or form a cycle.*

2.2 General hypergraphs cuts and expansion

A hypergraph $\mathcal{H} = (V, \mathcal{E})$ is a generalization of a graph where V denotes a node set and $\mathcal{E}$ is a *hyperedge* set where each $e \in \mathcal{E}$ is a subset of nodes in V of arbitrary size. The degree of a node v is denoted by d_v. The boundary of $S \subseteq V$ is denoted by

$$\partial_{\mathcal{H}}(S) = \partial_{\mathcal{H}}(\bar{S}) = \{e \in \mathcal{E} \colon |e \cap S| > 0 \text{ and } |e \cap \bar{S}| > 0\}. \quad (8)$$

If each hyperedge $e \in \mathcal{E}$ has a nonnegative scalar weight $w_{\mathcal{H}}(e)$, the standard *all-or-nothing* hypergraph cut function is given by

$$\mathbf{cut}_{\mathcal{H}}(S) = \textstyle\sum_{e\in\partial_{\mathcal{H}}(S)} w_{\mathcal{H}}(e). \quad (9)$$

It is useful in many settings to assign different cut penalties for separating the nodes of a hyperedge in different ways. This has led to the concept of generalized hypergraph cut functions [13, 26–28, 43–45, 51, 53]. Formally, each hyperedge $e \in \mathcal{E}$ is associated with a *splitting function* $\mathbf{w}_e \colon A \subseteq e \to \mathbb{R}$ that assigns a penalty for each way of separating the nodes of a hyperedge. The generalized hypergraph cut is then given by

$$\mathbf{cut}_{\mathcal{H}}(S) = \textstyle\sum_{e\in\partial_{\mathcal{H}}(S)} \mathbf{w}_e(S \cap e). \quad (10)$$

Following recent work [27, 43–45], we focus on *cardinality-based submodular hypergraph cut function*, which means all splitting functions satisfying the following properties for all $A, B \subseteq e$:

$$\begin{aligned}
&\text{(nonnegative) } \mathbf{w}_e(A) \geq 0; \quad \text{(uncut-ignoring) } \mathbf{w}_e(\emptyset) = 0\\
&\text{(symmetric) } \mathbf{w}_e(A) = \mathbf{w}(e\backslash A)\\
&\text{(submodular) } \mathbf{w}_e(A) + \mathbf{w}_e(B) \geq \mathbf{w}_e(A \cap B) + \mathbf{w}_e(A \cup B)\\
&\text{(cardinality-based) } \mathbf{w}_e(A) = \mathbf{w}_e(B) \text{ if } |A| = |B|.
\end{aligned}$$

Our objective: generalized hypergraph cut expansion. Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$ with a cardinality-based submodular hypergraph cut function $\mathbf{cut}_{\mathcal{H}}$ and node weight function $\pi \colon V \to \mathbb{R}_{>0}$, the hypergraph π-expansion of a set $S \subseteq V$ is defined to be

$$\phi_{\mathcal{H}}(S, \pi) = \frac{\mathbf{cut}_{\mathcal{H}}(S)}{\min\{\pi(S), \pi(\bar{S})\}}. \quad (11)$$

The minimum value of this objective over all S is denoted by $\phi_{\mathcal{H},\pi}$. Our focus is to develop an approximation algorithm for $\phi_{\mathcal{H},\pi}$.

2.3 Related work

Expansion ($\pi(v) = 1$) and conductance ($\pi(v) = d_v$) are two of the most widely-studied graph ratio cut objectives. For many years, the best approximation for graph expansion and conductance was $O(\log n)$, based on solving a multicommodity flow problem [24] that can also be viewed as a linear programming relaxation. The current best approximation factor is $O(\sqrt{\log n})$, based on a semidefinite programming relaxation [5]. Khandekar et al. [21] later introduced the *cut-matching* framework, providing an $O(\log^2 n)$ approximation based on faster maximum s-t flow computations, rather than multicommodity flows. The same framework was later used to design improved $O(\log n)$-approximation algorithms [34, 36].

Many variants of the hypergraph ratio-cut objective (11) have been considered [26–28, 30, 45, 51, 52], for different cut functions and node weight functions π. A number of Cheeger-style approximation guarantees have been developed for these objectives based on eigendecompositions of nonlinear hypergraph Laplacians [7, 27, 51] or PageRank-based methods [25, 40], but their worst-case approximation factors are never better than $O(n)$. Other techniques rely on clique expansions of hypergraphs [6, 26, 52], which also have poor worst case approximation factors that also scale poorly with the maximum hyperedge size. The $O(\log n)$ multicommodity flow algorithm [24] and the SDP-based $O(\sqrt{\log n})$ approximation for expansion [5] have been generalized to the hypergraph setting [20, 31]. However, these only apply to the standard all-or-nothing hypergraph cut function (9). They are also more restrictive in terms of node weights. We consider a general nonnegative node weight function π, while these previous methods focus on either expansion node weights ($\pi(v) = 1$) [31] or conductance node weights ($\pi(v) = d_v$) [20]. There are also numerous results on minimizing locally-biased ratio cut objectives in graphs [3, 4, 14, 22, 32, 37, 47, 48] and hypergraphs [13, 17, 28, 43, 44], but these do not provide guarantees for global ratio cut objectives.

Concurrent work. Ameranis et al. [2] independently and concurrently developed an $O(\log n)$-approximation algorithm for generalized hypergraph ratio cuts based on cut-matching. Their algorithm applies to a more general class of monotone submodular cut functions, though the authors focused on theoretical results and therefore did not provide an implementation. See the extended version for a more detailed comparison [42].

3 HYPERGRAPH EXPANDER EMBEDDINGS

Expander embeddings are common techniques for lower bounding expansion in undirected graphs [5, 21, 24, 38]. We generalize this basic approach in order to develop a strategy for lower bounding hypergraph ratio cut objectives, by embedding an expander graph into a special type of directed graph that models a generalized hypergraph cut function. See the appendix for all proofs.

3.1 Hypergraph cut preservers

Many hypergraph clustering methods rely on reducing a hypergraph to a graph and then applying an existing graph technique [6, 18, 26, 45, 50]. We employ a precise notion of hypergraph reduction for modeling generalized hypergraph cuts [44, 45].

Definition 3.1 (augmented cut preserver). *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with generalized cut function. The directed graph $G(\mathcal{H}) = (\hat{V}, \hat{E})$ is an* augmented cut preserver *for $\mathcal{H}$ if $\hat{V} = V \cup \mathcal{A}$ where $\mathcal{A}$ is an auxiliary node set and if $G(\mathcal{H})$ preserves cuts in $\mathcal{H}$ in the sense that for every $S \subseteq V$:*

$$\boldsymbol{cut}_{\mathcal{H}}(S) = \min_{U\subseteq\mathcal{A}} \boldsymbol{cut}_{G(\mathcal{H})}(S \cup U). \quad (12)$$

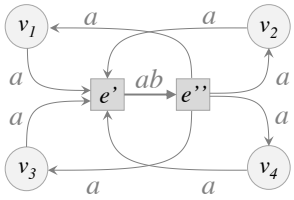


Figure 1: CB-gadget parameterized by weights a and b for a four-node hyperedge $e = \{v_1, v_2, v_3, v_4\}$.

The cut preserving property in (12) essentially says that for any fixed $S \subseteq V$, we can arrange nodes from $\mathcal{A}$ into two sides of a cut in a way that minimizes the overall (directed) cut penalty in $G(\mathcal{H})$. There is more than one way to construct an augmented cut preserver $G(\mathcal{H})$ for a cardinality-based submodular hypergraph cut function [23, 44, 45]. The general strategy is to replace each hyperedge $e \in \mathcal{E}$ with a small *gadget* involving directed weighted edges and auxiliary nodes, in a way that models the splitting function of the hyperedge. We specifically use reductions based on symmetric cardinality-based gadgets (CB-gadgets) [44, 45] (Figure 1). For a hyperedge $e \in \mathcal{E}$ with k nodes, a CB-gadget introduces two new nodes e' and e'', which are added to the augmented node set $\mathcal{A}$. For each $v \in e$, the gadget has two directed edges (v, e') and (e'', v) that are both given weight a, and a directed edge (e', e'') with weight ab, where a and b are nonnegative parameters for the CB-gadget. Figure 2(b) is an illustration of an augmented cut preserver for a small hypergraph, where each hyperedge is replaced by a single CB-gadget and edge-weights are omitted. Each hyperedge could also be replaced by a combination of CB-gadgets with different weights, in order to model more complicated hypergraph cut functions. It is possible to model any cardinality-based submodular splitting function using a combination of $\lfloor |e|/2 \rfloor$ CB-gadgets, by carefully choosing parameters (a, b) for each gadget [45]. In cases where it is enough to approximately model the hypergraph cut function, one can instead use an augmented sparsifier [44] to approximate the cut function while using fewer CB-gadgets.

3.2 Expander embeddings in hypergraphs

Let $G = (V, E_G)$ be a directed graph with edge weight $w_G(u, v)$ for each $(u, v) \in E_G$, and let $H = (V, E_H)$ be an undirected graph on the same set of nodes with weight $w_H(i, j) \geq 0$ for edge $\{i, j\} \in E_H$. For every set $S \subseteq V$, let $\mathcal{D}(S) = \{(u, v) \in S \times \bar{S} \colon \{u, v\} \in E_H\}$ denote a set of directed pairs of nodes that share an edge in H.

Definition 3.2 (embedding H in G). *Graph H can be embedded in G with congestion γ if for each bisection $\{S, \bar{S}\}$:*

- *For each $(u, v) \in \mathcal{D}(S)$ there is a u-v flow function $f^{(uv)}$ that routes $w_H(u, v)$ units of flow from u to v via edges in G.*
- *The flow $f = \sum_{(u,v)\in\mathcal{D}(S)} f^{(uv)}$ can be routed through G with congestion at most γ, i.e., for each $(i, j) \in E_G$,*

$$\sum_{(u,v)\in\mathcal{D}(S)} f_{ij}^{(uv)} \leq \gamma w_G(i, j). \tag{13}$$

Lemma 3.1. *If H is embedded in G with congestion γ, then for every node weight function π, $\phi_{G,\pi} \geq \frac{1}{\gamma}\phi_{H,\pi}$.*

The immediate implication of Lemma 3.1 is that if H is an expander, then the expansion of G is $\Omega(\frac{1}{\gamma})$. Definition 3.2 differs slightly from other notions of embeddings that are used when approximating graph ratio cuts. This definition is chosen in a way that is easier to generalize to the hypergraph setting.

Combining hypergraph cut preservers [44, 45] with Definition 3.2 provides a new strategy for bounding hypergraph π-expansion.

Lemma 3.2. *Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with a generalized hypergraph cut function $\boldsymbol{cut}_{\mathcal{H}}$ and let $G(\mathcal{H}) = (\hat{V} = V \cup \mathcal{A}, \hat{E})$ be an augmented cut preserver for $\mathcal{H}$. If graph $H = (V, E_H)$ can be embedded in $G(\mathcal{H})$ with congestion γ, then for every node weight function π, $\phi_{\mathcal{H},\pi} \geq \frac{1}{\gamma}\phi_{H,\pi}$.*

This assumes fixed node weights π and does not directly relate conductance in $\mathcal{H}$ to conductance in H (as conductance depends on node degrees). However, this provides an important step in lower bounding arbitrary ratio-cut objectives, including conductance.

4 HYPERGRAPH FLOW EMBEDDING

To apply Lemma 3.2, we would like to embed an expander into $G(\mathcal{H})$ with a small congestion, and then find a set S that is not too far from the resulting lower bound. As an important step in this direction, in this section we will show how to embed a special type of bipartite graph into $\mathcal{H}$ whose congestion is related to a small expansion set. Given a partition $\{R, \bar{R}\}$ of the node set V, we will design a procedure that takes in a parameter $\alpha > 0$ and using a single maximum flow computation either (1) returns a set $S \subseteq V$ such that $\phi_{\mathcal{H}}(S, \pi) < \alpha$, or (2) produces a bipartite graph between R and $\bar{R}$ that can be embedded in $G(\mathcal{H})$ with congestion $1/\alpha$. In Section 5, we will show how to combine these bipartite graphs to embed an expander into $G(\mathcal{H})$. Proofs are provided in the appendix.

4.1 Maximum flows in an auxiliary graph

Fix $\alpha > 0$ and a partition $\{R, \bar{R}\}$ satisfying $\pi(R) \leq \pi(\bar{R})$, and set $\eta = \pi(R)/\pi(\bar{R})$. We will solve a maximum s-t flow problem on an auxiliary graph $G(\mathcal{H}, R, \alpha)$ (see Figure 2(c)) constructed as follows:

- Construct a CB-gadget cut preserver $G(\mathcal{H})$ for $\mathcal{H}$ [44, 45], and scale all edge weights by $\frac{1}{\alpha}$.
- Add an extra source node s and a sink node t.
- For each $r \in R$, add a directed edge (s, r) with weight $\pi(r)$.
- For each $v \in \bar{R}$, add a directed edge (v, t) with weight $\eta\pi(v)$.

We will use a solution to the maximum s-t flow problem to either find a set S with small expansion (by considering the dual minimum s-t cut problem), or find an embeddable bipartite graph between R and $\bar{R}$ with congestion $\frac{1}{\alpha}$ (by considering a flow decomposition). This mirrors the same strategy that is used in cut-matching games for graph expansion [21], though the construction and proofs are more involved for generalized hypergraph cut functions.

Finding a small π-expansion set. The minimum s-t cut problem in $G(\mathcal{H}, R, \alpha)$ is equivalent to solving the following optimization problem over the hypergraph $\mathcal{H}$:

$$\text{minimize}_{S\subseteq V} \ \frac{1}{\alpha}\mathbf{cut}_{\mathcal{H}}(S) + \pi(R \cap \bar{S}) + \eta\pi(\bar{R} \cap S). \tag{14}$$

If we set $S = \emptyset$, this corresponds to separating node s from all other nodes in $G(\mathcal{H}, R, \alpha)$, which is a valid s-t cut with value $\pi(R)$. If we can find an s-t cut value that is strictly smaller than $\pi(R)$ in $G(\mathcal{H}, R, \alpha)$, it will provide a set S with small expansion in $\mathcal{H}$.

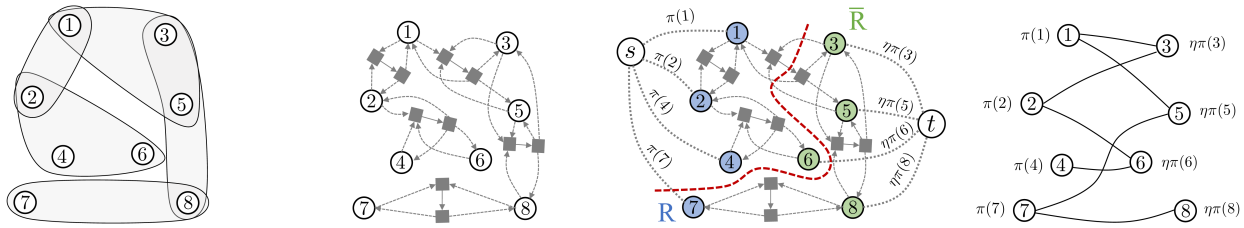


Figure 2: (a) An example hypergraph $\mathcal{H} = (V, \mathcal{E})$. (b) An *augmented cut preserver* $G(\mathcal{H})$ (Definition 3.1) for $\mathcal{H}$. (c) The auxiliary graph $G(\mathcal{H}, R, \alpha)$ and minimum s-t cut (red line). (d) Toy example of an embedded π-regular bipartite graph M_R.

Lemma 4.1. *If the minimum s-t cut value in $G(\mathcal{H}, R, \alpha)$ is less than $\pi(R)$, then the set S minimizing objective* (14) *satisfies $\phi_{\mathcal{H}}(S, \pi) < \alpha$.*

Max-flow subroutines for solving an objective closely related to (14) have already been used to minimize localized ratio cut objectives in hypergraphs [43]. This generalizes earlier work on localized ratio cuts in graphs [4, 37, 47, 48]. The present work differs in that we use max-flow solutions to approximate *global* ratio cut objectives. A key step in doing so is showing how to embed a bipartite graph into $G(\mathcal{H})$ when Lemma 4.1 does not apply.

Embedding a bipartite graph. If the min s-t cut in $G(\mathcal{H}, R, \alpha)$ is $\pi(R)$, then the max s-t flow solution in $G(\mathcal{H}, R, \alpha)$ saturates every edge touching s or t. We can then define a bipartite graph M_R between R and $\bar{R}$ that can be embedded in $G(\mathcal{H})$ with congestion $\frac{1}{\alpha}$. Let f be a maximum s-t flow on $G(\mathcal{H}, R, \alpha)$ with value $|f| = \pi(R)$. Letting $n = |V|$, define a matrix $\mathbf{M}_R \in \mathbb{R}^{n \times n}$ that is initialized to be the all zeros matrix. Using Lemma 2.1, we can decompose the flow f into $f = \sum_{i=1}^{\ell} f_i$, where for each i the edges that have a positive flow in f_i form either a cycle or a simple s-t path. For our purposes we can ignore the cycles and focus on the s-t paths. An s-t flow path f_i always starts by sending $|f_i|$ units of flow from s to some node $r \in R$, and eventually ends by sending the same amount of flow through an edge (v, t) where $v \in \bar{R}$. For each such flow path, we perform the update $\mathbf{M}_R(r, v) \leftarrow \mathbf{M}_R(r, v) + |f_i|$. After iterating through all ℓ flow paths, we define M_R to be the bipartite graph whose adjacency matrix is $\mathbf{M}_R$ (see Figure 2(d)). The construction of $G(\mathcal{H}, R, \alpha)$ and the fact that f saturates all edges touching s and t implies the following degree properties for M_R:

- For each $r \in R$, the weighted degree of r in M_R is $\pi(r)$.
- For each $u \in \bar{R}$, the weighted degree of u in M_R is $\eta\pi(u)$.

Following previous terminology [35], we refer to a bipartite graph satisfying these properties as a π-regular bipartite graph. To relate this to previous cut-*matching* games for graph expansion, note that if $|R| = |\bar{R}|$ and $\pi(u) = 1$ for all $u \in V$, then M_R will be a fractional *matching* and $\mathbf{M}_R$ will be a doubly stochastic matrix.

Proving a bound on congestion. When $|f| = \pi(R)$, the maximum s-t flow in $G(\mathcal{H}, R, \alpha)$ can be viewed as a directed multicommodity flow that is routed through a $\frac{1}{\alpha}$-scaled copy of $G(\mathcal{H})$. However, there is one subtle issue we must overcome in order to confirm that M_R can be embedded in $G(\mathcal{H})$ with congestion $\frac{1}{\alpha}$. Definition 3.2 requires that for every bisection $\{S, \bar{S}\}$, there must be a way to route $\mathbf{M}_R(u, v)$ units of flow from u to v if $u \in S$ and $v \in \bar{S}$. However, f only routes flow from R to $\bar{R}$ in the *directed* graph $G(\mathcal{H}, R, \alpha)$. This issue does not arise in cut-matching games for undirected graphs, as each undirected edge $\{u, v\}$ can be viewed as a pair of directed edges (u, v) and (v, u), making it easy to send flow in two directions simultaneously. However, in the directed graph $G(\mathcal{H}, R, \alpha)$, it is possible that for a given bipartition $\{S, \bar{S}\}$, the flow f will send flow from a node $r \in R \cap \bar{S}$ to a node $u \in \bar{R} \cap S$, which does not directly satisfy the requirement in Definition 3.2. The following lemma confirms that we can overcome this. Its proof relies on carefully considering the edge structure in $G(\mathcal{H})$ and showing how to use an implicit flow-reversing procedure when necessary in order to satisfy Definition 3.2. A proof is included in the appendix.

Lemma 4.2. *If $|f| = \pi(R)$, the graph M_R can be embedded in $G(\mathcal{H})$ with congestion $\frac{1}{\alpha}$ in the sense of Definition 3.2.*

4.2 The flow-embedding algorithm

We combine Lemmas 4.1 and 4.2 into a method HyperCutOrEmbed (Algorithm 1) for obtaining both a good cut and an embeddable bipartite graph for any input partition $\{R, \bar{R}\}$ with $\pi(R) \leq \pi(\bar{R})$. We assume that the hypergraph $\mathcal{H}$ is connected, and that the hypergraph weights are scaled so that there is a minimum penalty of 1 when cutting any hyperedge. This implies a lower bound of $2/\pi(V)$ on the minimum π-expansion, which is achieved if there is a set S with $\phi_{\mathcal{H}}(S) = \pi(V)/2$ and $\mathbf{cut}_{\mathcal{H}}(S) = 1$. HyperCutOrEmbed repeatedly solves maximum s-t flow problems to find either a bipartite graph or a cut with bounded π-expansion. The algorithm uses black-box subroutines for maximum s-t flows and minimum s-t cuts, and a procedure FlowEmbed that decomposes a flow into a π-regular bipartite graph as outlined in Section 4.1.

Theorem 4.3. *HyperCutOrEmbed returns a π-regular bipartite graph M_R that can be embedded in $G(\mathcal{H})$ with congestion $1/\alpha$ and a set S with $\phi_{\mathcal{H}}(S, \pi) < 2\alpha$ for some $\alpha \geq 2/\pi(V)$. The algorithm terminates in $O(\log |E| + \log U + \log \pi(V))$ iterations where U is the maximum hyperedge cut penalty.*

5 HYPERGRAPH RATIO CUT ALGORITHMS

HyperCutOrEmbed finds a node set whose π-expansion is related to a graph that can be embedded in $\mathcal{H}$. Applying Lemma 3.2 directly does not imply a useful lower bound or approximation algorithm for π-expansion in $\mathcal{H}$, as the bipartite graph itself does not have a large π-expansion. In this section we show how to use existing strategies for building an expander graph in order to design an approximation algorithm for hypergraph π-expansion.

5.1 Expander building subroutines

The standard cut-matching procedure for expansion in an undirected graph $G = (V, E)$ can be described as a two-player game

Algorithm 1 HyperCutOrEmbed($\mathcal{H}, R, \pi$)

Input: $\mathcal{H} = (V, \mathcal{E})$, node weights π, bisection $\{R, \bar{R}\}$
Output: M_R with congestion $1/\alpha$; S with $\phi_{\mathcal{H}}(S, \pi) \leq 2\alpha$
Set $\alpha = 2/\pi(V)$, NoCutFound = true
while NoCutFound **do**
  f = MaxSTFlow($G(\mathcal{H}, R, \alpha)$).
  if $|f| = \pi(R)$ **then**
    $M_R \leftarrow$ FlowEmbed($G(\mathcal{H}), R, f$); $\alpha \leftarrow 2\alpha$
  else
    S = MinSTCut($G(\mathcal{H}, R, \alpha), f$)
    NoCutFound = false
Return M_R, S, α

between a cut player and a matching player. At the start of iteration i, the cut player produces a bisection $\{R_i, \bar{R}_i\}$, and the matching player produces a fractional perfect matching M_i between R_i and $\bar{R}_i$, encoded by a doubly stochastic matrix $\mathbf{M}_i \in [0,1]^{|V|\times|V|}$. After t iterations, the union of matchings defines a graph $H_t = \bigcup_{j=1}^{t} M_j$ with adjacency matrix $\mathbf{A}_t = \sum_{j=1}^{t} \mathbf{M}_j$. The goal of the cut player is to choose bisections in a way that minimizes the number of rounds it takes before H_t is an expander, while the goal of the matching player is to choose matchings that maximize the number of rounds. Khandekar et al. [21] provided a strategy for the cut player which, for any matching player subroutine, will force H_t to have expansion at least $1/2$ for some $t = O(\log^2 n)$ with high probability. This was used to show an $O(\log^2 n)$ approximation for graph expansion.

In follow-up work, the cut-matching framework has been generalized in many different ways [29, 34–36, 38]. Although most results focus on expansion node weights ($\mu(v) = 1$), Orecchia et al. [35] recently introduced a setting where the cut player produces a set R_i satisfying $\pi(R_i) \leq \pi(\bar{R}_i)$ at each iteration, and the matching player produces a π-regular bipartite graph M_i on $\{R_i, \bar{R}_i\}$. Taking the union of all bipartite graphs up through the ith iteration produces a graph $H_i = (V, E_{H_i})$. Lemma 5.1 summarizes a cut player strategy that applies to this setting and leads to an $O(\log n)$-approximation for graph ratio cut objectives.

Lemma 5.1 (Theorem 7 in [35]). *There exists a cut player strategy such that, for any matching player strategy, H_t satisfies $\phi_{H_t,\pi} = \Omega(\log n)$ with high probability for some $t = O(\log^2 n)$. In round i, the cut player can compute R_i in time $O(|E_{H_i}| \cdot polylog(\pi(V)))$.*

This strategy applies a heat-kernel random walk in H_t, so we refer to it as HeatKernelPartition. The approach was first used for expansion weights ($\mu(v) = 1$) by Orecchia et al. [36] and considered in more depth in Orecchia's PhD thesis [34]. Lemma 5.1 generalizes this to general node weights and was presented in recent work on overlapping graph clustering [35]. More technical details on cut-matching games with general node weights were presented recently by Amerinas et al. [2]. In the first iteration, H_0 is empty, so the cut player starts with any balanced bipartition $\{R, \bar{R}\}$.

5.2 The approximation algorithm

The notion of expander embedding that we have introduced involves embedding an expander graph into a hypergraph $\mathcal{H}$. Therefore, we can use existing cut player strategies to build a π-expander, which we embed into $\mathcal{H}$ in the sense of Definition 3.2 using algorithm HyperCutOrEmbed. Our cut-matching approximation algorithm for generalized hypergraph cut expansion (Algorithm 2) is obtained by using HyperCutOrEmbed for the matching player and HeatKernelPartition for the cut player. See the appendix for a detailed runtime analysis.

Algorithm 2 HCM: $O(\log n)$ approximation for π-expansion

Input: $\mathcal{H} = (V, \mathcal{E})$, generalized $\mathbf{cut}_{\mathcal{H}}$, bisection $\{R, \bar{R}\}$
Output: Set S with small π-expansion
$M_0 = \emptyset$
for $i = 1$ to t **do**
  R_i = HeatKernelPartition($M_0, M_1, M_2, \ldots, M_{i-1}$)
  M_i, S_i, α_i = HyperCutOrEmbed($\mathcal{H}, R_i$)
Return $S^* = \text{argmin}_{i=1,2,\ldots t}\, \phi_{\mathcal{H}}(S_i, \pi)$

Theorem 5.2. *For some $t = O(\log^2 n)$, Algorithm 2 is an $O(\log n)$-approximation algorithm for hypergraph π-expansion where $\mathbf{cut}_{\mathcal{H}}$ is any submodular cardinality-based hypergraph cut function.*

Proof. At iteration i, HyperCutOrEmbed produces a π-regular bipartite graph M_i that can be embedded in $G(\mathcal{H})$ with expansion $\frac{1}{\alpha_i}$ and a set S_i with expansion $\phi_{\mathcal{H}}(S_i, \pi) \leq 2\alpha_i$. Define $i^* = \text{argmin}_i\, \alpha_i$ and note that $\phi_{\mathcal{H}}(S^*, \pi) \leq \phi_{\mathcal{H}}(S_{i^*}, \pi)$. The union of bipartite graphs H_t can be embedded in $G(\mathcal{H})$ with congestion $\sum_{i=1}^{t} \frac{1}{\alpha_i} \leq \frac{t}{\alpha_{i^*}}$. From Lemma 5.1, we have $t \leq c_1 \log^2 n$ and $\phi_{H_t,\pi} \geq c_2 \log n$ with high probability, where c_1 and c_2 are positive constants. Combining this with Lemma 3.2 we have

$$\frac{\alpha_i^*}{\log n} \leq \frac{c_1 \alpha_i^*}{t} \log n \leq \frac{c_1 \alpha_i^*}{c_2 t} \phi_{H_t,\pi} \leq \frac{c_1}{c_2} \phi_{\mathcal{H},\pi}.$$

So $\phi_{\mathcal{H},\pi} = \Omega\left(\alpha_i^*/\log n\right)$, and the algorithm returns a set S^* with $\phi_{\mathcal{H}}(S^*, \pi) \leq 2\alpha_i^*$, proving the $O(\log n)$ approximation guarantee. □

5.3 Practical improvements

We incorporate a number of practical updates to the algorithm to simplify its implementation and improve approximation guarantees in practice. First of all, our implementation uses the push-relabel max-flow algorithm [9], which has a worse (but still fast) theoretical runtime and comes with various heuristics that make it very fast in practice. For the flow decomposition, we use a standard decomposition technique (see Theorem 3.5 in [1]) that does not require dynamic trees. We also use an altered version of HyperFlowEmbed that returns a bipartite graph M_R that can be embedded with congestion $1/\alpha$ as well as a set S that has π-expansion equal to α, rather than just a set S with $\phi_{\mathcal{H}}(S, \pi) < 2\alpha$. This improves the approximation by a factor of 2, which does not change the theoretical $O(\log n)$ approximation but can make a substantial difference in practice. See the appendix for details.

Finally, we establish sharper lower bounds on the approximation guarantee satisfied by the algorithm in each iteration. This allows us to obtain improved a posteriori approximation guarantees in practice. For node weight function π, let $\mathbf{D}_\pi$ be the diagonal matrix where $\mathbf{D}_\pi(i,i) = \pi(i)$ and define $\mathcal{L}_t = \mathbf{D}_\pi^{-1/2} \mathbf{L}_t \mathbf{D}_\pi^{-1/2}$. Let $\lambda_2(\mathcal{L}_t)$ be the 2nd smallest eigenvalue of this matrix. The following theorem presents a precise and easy-to-compute lower bound on the

Table 1: HCM results on 4 larger hypergraphs.

	n	m	avg $\lvert e\rvert$	Approx.	Run. (s)
Amazon9	13138	31502	8.1	2.78 ±0.012	254.8 ±10.4
Mathoverfl	73851	5446	24.2	3.08 ±0.014	643.8 ±31.1
Tripadvisor	8929	130568	4.1	2.71 ±0.025	699.3 ±29.8
Trivago	172738	233202	3.1	2.78 ±0.038	2372.6 ±50.8

approximation factor achieved by our approximation algorithm at each iteration. For this result we use the updated version of HyperFlowEmbed that returns a set S with $\phi_{\mathcal{H}}(S,\pi) = \alpha$.

Theorem 5.3. *After t iterations, Algorithm 2 returns a set S_t^* satisfying $\phi_{\mathcal{H}}(S_t^*,\pi) \le \rho_t \phi_{\mathcal{H},\pi}$, where $\rho_t = \frac{2\gamma_t \phi_{\mathcal{H}}(S_t^*)}{\lambda_2(\mathcal{L}_t)} \le \frac{2t}{\lambda_2(\mathcal{L}_t)}$ and $\gamma_t = \sum_{i=1}^{t} \frac{1}{\alpha_i}$.*

This theorem suggests another alternative for the cut player strategy: choose a partition $\{R, \bar{R}\}$ by thresholding entries in the second smallest eigenvector of $\mathcal{L}_t$. Embedding a π-regular bipartite graph across this partition increases the value of $\lambda_2(\mathcal{L}_t)$ in subsequent iterations. There exist extreme cases where greedily choosing a bipartition based on this eigenvector makes slow progress (see discussion on page 35 of [34]). However, this tends to produce good results in practice, and we can use Theorem 5.3 to compute concrete lower bounds on π-expansion using this strategy.

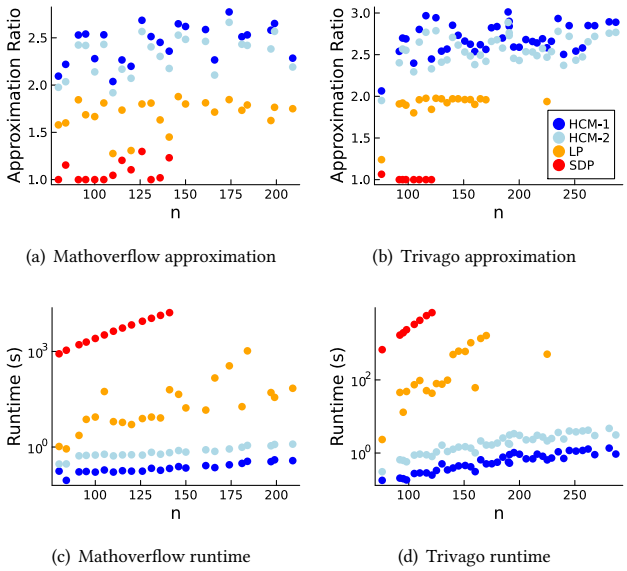
6 EXPERIMENTS

We implement our hypergraph cut matching algorithm (HCM) in Julia using all of the practical improvements from Section 5.3. We compare it against other hypergraph ratio-cut algorithms in minimizing global ratio cut objectives on various hypergraphs encoding common types of higher-order multiway interactions on the web. All experiments were run on an Macbook Air with an Apple M1 Chip and 16GB of RAM. Code and data are provided at https://github.com/nveldt/HyperCutMatch. The full version of the paper contains additional details and experiments [42].

Web-based hypergraphs. *Trivago* encodes sets of vacation rentals on `Trivago.com` that are visited during a browsing session [10]. *Mathoverflow* encodes sets of `mathoverflow.com` posts that are answered by the same user [43]. *Amazon9* encodes sets of retail products (from 9 product categories of a larger dataset [33]) that are reviewed by the same user. *TripAdvisor* encodes sets of co-reviewed `Tripadvisor.com` accommodations [46].

6.1 Comparison against convex relaxations

Our method is orders of magnitude more scalable than previous theoretical approximation algorithms for hypergraph ratio cuts and also applies to a much broader class of problems. We compare against the $O(\sqrt{\log n})$ approximation based on semidefinite programming [31] (SDP) and the $O(\log n)$-approximation algorithm (LP) based on rounding a linear programming relaxation [20]. We focus on the expansion objective ($\pi(v) = 1$ for all v) with the standard hypergraph cut function, since these methods do not apply to generalized hypergraph cuts. We solve the SDP relaxation with Mosek software, using CVX [15] in Matlab as a front end. We use Gurobi software with Julia as a front end to solve the LP relaxation. We select a range of small *Mathoverflow* and *Trivago* hypergraphs



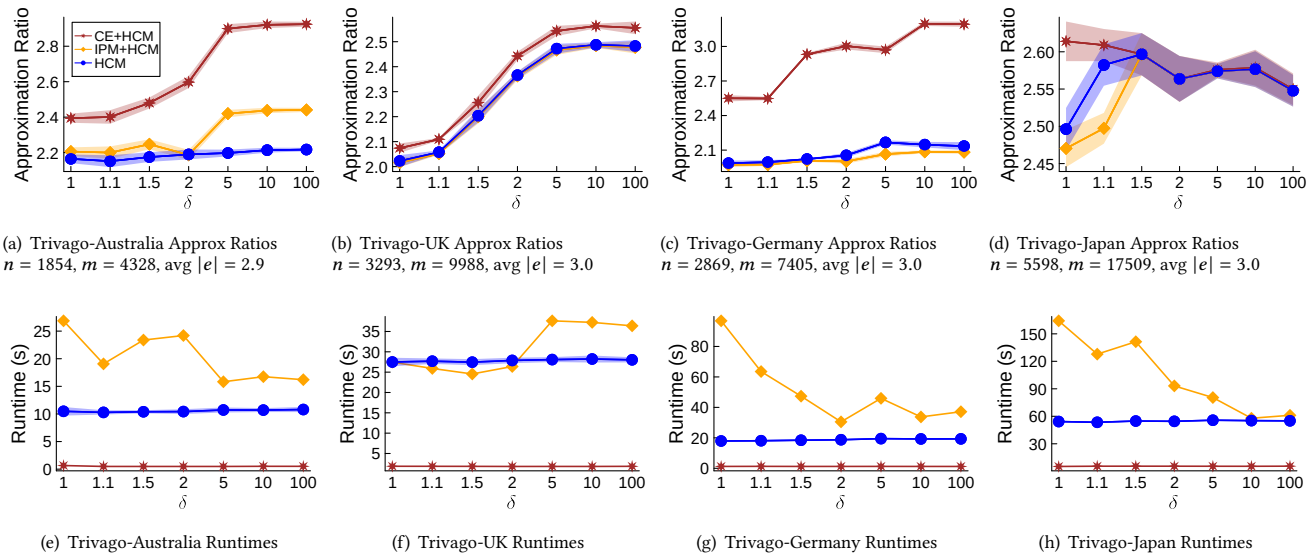
(a) Mathoverflow approximation (b) Trivago approximation

(c) Mathoverflow runtime (d) Trivago runtime

Figure 3: Results for three approximation algorithms on small hypergraphs. LP and SDP do not scale and often fail even for these small hypergraphs. HCM-1 uses $10\log_2 n$ rounds of cut-matching, and HCM-2 uses $30\log_2 n$ rounds.

(subhypergraphs of the hypergraphs in Table 1) to use as benchmarks. We chose 23 *Mathoverflow* hypergraphs, each corresponding to a set of posts (nodes) with a certain topic tag (e.g., *graph-theory*). These hypergraphs have between $n = 80$ and $n = 209$ nodes. The number of hyperedges m tends to be between $n/3$ and $n/2$ for these hypergraphs, and the average hyperedge size ranges from 3 to 7. We also consider 41 *Trivago* hypergraphs (each corresponding to a different city tag, e.g., *Austin, USA*), most of which have between 1-3 times as many hyperedges as nodes, and all of which have average hyperedge sizes between 2 and 4.

Figure 3 reports results for LP, SDP, and our method HCM, all of which compute an explicit lower bound on the optimal expansion which can be used to compute an a posteriori approximation guarantee. We run HCM with two different iteration numbers to illustrate the tradeoff between runtime and approximation guarantee. When SDP and LP converge, they produce very good lower bounds for hypergraph expansion and can be rounded to produce very good a posteriori approximation guarantees. The issue is that these methods do not scale well even to very small instances. We are able to obtain results for all *Mathoverflow* datasets using LP, but the method times out (> 30 minutes) on all but 19 of the 41 *Trivago* hypergraphs. The SDP method is even less scalable and would not converge for almost any of the small hypergraphs if we set a 30-minute time limit. It took 4.5 hours to run this method for a 141-node *Mathoverflow* hypergraph. Given scalability issues, we did not attempt to use this method on larger datasets. Meanwhile, HCM obtains high quality solutions extremely quickly, typically within a matter of a few seconds. Table 1 lists additional results (averages and standard deviations over 5 runs) for $5\log_2 n$ iterations of HCM on four larger hypergraphs, which are far too large for LP and SDP.



(a) Trivago-Australia Approx Ratios
$n = 1854$, $m = 4328$, avg $|e| = 2.9$

(b) Trivago-UK Approx Ratios
$n = 3293$, $m = 9988$, avg $|e| = 3.0$

(c) Trivago-Germany Approx Ratios
$n = 2869$, $m = 7405$, avg $|e| = 3.0$

(d) Trivago-Japan Approx Ratios
$n = 5598$, $m = 17509$, avg $|e| = 3.0$

(e) Trivago-Australia Runtimes

(f) Trivago-UK Runtimes

(g) Trivago-Germany Runtimes

(h) Trivago-Japan Runtimes

Figure 4: Top row: approximations obtained by comparing the best conductance set found by each method against the conductance lower bound computed by HCM. Bottom row: runtimes in seconds. The runtimes for IPM and CE do not include the time it takes to compute the HCM lower bound. We plot mean over 5 runs of HCM; shaded region indicates standard deviation.

6.2 Trivago hypergraphs and generalized cuts

One distinct advantage of HCM is that it computes explicit lower bounds on hypergraph π-expansion (via Theorem 5.3) that can be used to check a posteriori approximation guarantees in practice. To illustrate the power of this feature, we compare HCM against the inverse power method for submodular hypergraphs (IPM) [27] and the clique expansion method for inhomogeneous hypergraphs (CE) [26]. These methods constitute the current state-of-the-art in minimizing ratio cuts in hypergraphs with generalized cut functions. IPM is a generalization of a previous method that applied only to the standard hypergraph cut [16]. CE generalizes previous clique expansion techniques [6, 52], which only applied to more restrictive hypergraph cut functions. Both methods are more practical than LP and SDP and apply to generalized hypergraph cuts, but they have weaker theoretical guarantees. IPM is a heuristic with no approximation guarantees, while the approximation guarantee for CE scales poorly with the maximum hyperedge size, and can be $O(n)$ in the worst case even for graphs. The performance of IPM depends heavily on which vector it is given as a warm start. We tried several options and found that the best results were obtained by using the eigenvector computed by CE as a warm start.

Figure 4 displays runtimes and a posteriori approximation guarantees for HCM, IPM, and CE on four Trivago hypergraphs, corresponding to vacation rentals in Australia, United Kingdom, Germany, and Japan. We specifically consider the 2-core of each hypergraph, as these 2-cores have more interesting and varied cut structure and therefore serve as better case studies for comparing algorithms for generalized hypergraph ratio cuts. For these experiments we are minimizing the conductance objective ($\pi(v) = d_v$), and we use a generalized hypergraph cut function that applies a δ-linear splitting function $\mathbf{w}_e(S) = \min\{|S \cap e|, |\bar{S} \cap e|, \delta\}$ at each hyperedge. We choose this splitting function as previous research has shown that the choice of δ can significantly affect the size and structure of the output set and influence performance in downstream clustering applications [28, 43, 44]. The four *Trivago* hypergraphs exhibit a range of different cuts that are found by varying δ, and therefore provide a good case study for how well these algorithms find different types of ratio cuts for generalized splitting functions.

In terms of finding small conductance sets, HCM trades off in performance with IPM, though they return very similar results. However, HCM is significantly faster, and unlike IPM it is additionally computing lower bounds that allow it to certify how close its solution is to optimality. In Figure 4, we are in fact using the HCM lower bound to obtain a posteriori approximations for IPM and CE. These methods are unable to provide such strong approximation guarantees on their own. Hence, even in cases where IPM finds better conductance sets, the lower bounds computed by HCM provide new information that can be used to prove an approximation guarantee. Since the performance of IPM also relies on using CE as a warm start, we see that in many ways it is a combination of all three algorithms that leads to the best results.

7 DISCUSSION

We have presented the first algorithm for minimizing hypergraph ratio cuts that simultaneously (1) has an $O(\log n)$ approximation guarantee, (2) applies to generalized hypergraph cut functions, and (3) comes with a practical implementation. This algorithm is very successful at finding ratio cuts within a small factor (around 2-3) of a lower bound on the optimal solution. One open question is to explore how to choose the best generalized hypergraph cut functions to use in different applications of interest. Another open direction is finding improved approximation algorithms for hypergraph ratio cuts that apply to general submodular splitting functions, even those that are not *cardinality-based*.

REFERENCES

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. 1988. *Network flows*. Prentice-Hall Inc.

[2] Konstantinos Ameranis, Antares Chen, Lorenzo Orecchia, and Erasmo Tani. 2023. Efficient Flow-based Approximation Algorithms for Submodular Hypergraph Partitioning via a Generalized Cut-Matching Game. *arXiv preprint arXiv:2301.08920* (2023).

[3] Reid Andersen, Fan Chung, and Kevin Lang. 2006. Local Graph Partitioning using PageRank Vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '06)*.

[4] Reid Andersen and Kevin J. Lang. 2008. An Algorithm for Improving Graph Partitions. In *Proceedings of the 2008 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*. Society for Industrial and Applied Mathematics, 651–660.

[5] Sanjeev Arora, Satish Rao, and Umesh Vazirani. 2009. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)* 56, 2 (2009).

[6] Austin R. Benson, David F. Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[7] T-H Hubert Chan, Anand Louis, Zhihao Gavin Tang, and Chenzi Zhang. 2018. Spectral properties of hypergraph laplacian and approximation algorithms. *Journal of the ACM (JACM)* 65, 3 (2018), 1–48.

[8] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. 2022. Maximum Flow and Minimum-Cost Flow in Almost-Linear Time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS '22)*. 612–623. https://doi.org/10.1109/FOCS54457.2022.00064

[9] Boris V Cherkassky and Andrew V Goldberg. 1997. On implementing the push-relabel method for the maximum flow problem. *Algorithmica* 19, 4 (1997).

[10] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. 2021. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances* 7, 28 (2021), eabh1303. https://doi.org/10.1126/sciadv.abh1303

[11] Fan R. K. Chung. 1997. *Spectral Graph Theory*. Vol. 92. American Mathematical Society. https://doi.org/10.1090/cbms/092

[12] Michael B Cohen, Yin Tat Lee, and Zhao Song. 2021. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)* 68, 1 (2021).

[13] Kimon Fountoulakis, Pan Li, and Shenghao Yang. 2021. Local hyper-flow diffusion. In *Advances in Neural Information Processing Systems (NeurIPS '21, Vol. 34)*.

[14] K. Fountoulakis, M. Liu, D. F. Gleich, and M. W. Mahoney. 2023. Flow-based Algorithms for Improving Clusters: A Unifying Framework, Software, and Performance. *SIAM Review (to appear)* (2023).

[15] Michael Grant and Stephen Boyd. 2014. CVX: Matlab Software for Disciplined Convex Programming, version 2.1. http://cvxr.com/cvx.

[16] Matthias Hein, Simon Setzer, Leonardo Jost, and Syama Sundar Rangapuram. 2013. The total variation on hypergraphs-learning on hypergraphs revisited. In *Advances in Neural Information Processing Systems (NeurIPS '13, Vol. 26)*.

[17] Rania Ibrahim and David F Gleich. 2020. Local hypergraph clustering using capacity releasing diffusion. *Plos one* 15, 12 (2020), e0243485.

[18] Edmund Ihler, Dorothea Wagner, and Frank Wagner. 1993. Modeling hypergraphs by graphs with the same mincut properties. *Inform. Process. Lett.* 45, 4 (1993), 171 – 175. https://doi.org/10.1016/0020-0190(93)90115-P

[19] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2021. A Faster Algorithm for Solving General LPs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21)*. Association for Computing Machinery, New York, NY, USA, 823–832. https://doi.org/10.1145/3406325.3451058

[20] Michael Kapralov, Robert Krauthgamer, Jakab Tardos, and Yuichi Yoshida. 2020. Towards Tight Bounds for Spectral Sparsification of Hypergraphs. *arXiv preprint arXiv:2011.06530* (2020).

[21] Rohit Khandekar, Satish Rao, and Umesh Vazirani. 2009. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)* 56, 4 (2009), 1–15.

[22] Kyle Kloster and David F Gleich. 2014. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '14)*. 1386–1395.

[23] Pushmeet Kohli, L'ubor Ladický, and Philip H. S. Torr. 2009. Robust Higher Order Potentials for Enforcing Label Consistency. *International Journal of Computer Vision* 82, 3 (2009), 302–324. https://doi.org/10.1007/s11263-008-0202-0

[24] Tom Leighton and Satish Rao. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)* 46, 6 (1999), 787–832.

[25] Pan Li, Niao He, and Olgica Milenkovic. 2020. Quadratic Decomposable Submodular Function Minimization: Theory and Practice. *Journal of Machine Learning Research* 21, 106 (2020), 1–49. http://jmlr.org/papers/v21/18-790.html

[26] Pan Li and Olgica Milenkovic. 2017. Inhomogeneous Hypergraph Clustering with Applications. In *Advances in Neural Information Processing Systems 30 (NeurIPS '17)*. 2308–2318.

[27] Pan Li and Olgica Milenkovic. 2018. Submodular Hypergraphs: p-Laplacians, Cheeger Inequalities and Spectral Clustering. In *Proceedings of the 35th International Conference on Machine Learning (ICML '18)*. 3014–3023.

[28] 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)*. 2092–2103.

[29] Anand Louis. 2010. Cut-matching games on directed graphs. *arXiv preprint arXiv:1010.1047* (2010).

[30] Anand Louis. 2015. Hypergraph Markov Operators, Eigenvalues and Approximation Algorithms. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing (STOC '15)*. 713–722.

[31] Anand Louis and Yury Makarychev. 2016. Approximation algorithms for hypergraph small-set expansion and small-set vertex expansion. *Theory of Computing* 12, 1 (2016), 1–25.

[32] Michael W Mahoney, Lorenzo Orecchia, and Nisheeth K Vishnoi. 2012. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *The Journal of Machine Learning Research* 13, 1 (2012), 2339–2365.

[33] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP '19)*. 188–197.

[34] Lorenzo Orecchia. 2011. *Fast approximation algorithms for graph partitioning using spectral and semidefinite-programming techniques*. Ph. D. Dissertation. University of California, Berkeley.

[35] Lorenzo Orecchia, Konstantinos Ameranis, Charalampos Tsourakakis, and Kunal Talwar. 2022. Practical Almost-Linear-Time Approximation Algorithms for Hybrid and Overlapping Graph Clustering. In *International Conference on Machine Learning (ICML '22)*. 17071–17093.

[36] Lorenzo Orecchia, Leonard J Schulman, Umesh V Vazirani, and Nisheeth K Vishnoi. 2008. On partitioning graphs via single commodity flows. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC '08)*. 461–470.

[37] Lorenzo Orecchia and Zeyuan Allen Zhu. 2014. Flow-based algorithms for local graph clustering. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA '14)*. SIAM, 1267–1286.

[38] Jonah Sherman. 2009. Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '09)*. 363–372.

[39] Daniel D Sleator and Robert Endre Tarjan. 1983. A data structure for dynamic trees. *Journal of computer and system sciences* 26, 3 (1983), 362–391.

[40] Yuuki Takai, Atsushi Miyauchi, Masahiro Ikeda, and Yuichi Yoshida. 2020. Hypergraph clustering based on pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1970–1978.

[41] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2021. Minimum Cost Flows, MDPs, and ℓ_1-Regression in Nearly Linear Time for Dense Instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*. 859–869.

[42] Nate Veldt. 2023. Cut-matching Games for Generalized Hypergraph Ratio Cuts. *arXiv preprint arXiv:2301.12274* (2023).

[43] Nate Veldt, Austin R. Benson, and Jon Kleinberg. 2020. Minimizing Localized Ratio Cut Objectives in Hypergraphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. 1708–1718.

[44] Nate Veldt, Austin R Benson, and Jon Kleinberg. 2021. Approximate Decomposable Submodular Function Minimization for Cardinality-Based Components. In *Advances in Neural Information Processing Systems (NeurIPS '21, Vol. 34)*.

[45] Nate Veldt, Austin R. Benson, and Jon Kleinberg. 2022. Hypergraph Cuts with General Splitting Functions. *SIAM Rev.* 64, 3 (2022), 650–685.

[46] Nate Veldt, Austin R Benson, and Jon Kleinberg. 2023. Combinatorial characterizations and impossibilities for higher-order homophily. *Science Advances* 9, 1 (2023), eabq3200.

[47] Nate Veldt, David Gleich, and Michael Mahoney. 2016. A Simple and Strongly-Local Flow-Based Method for Cut Improvement. In *Proceedings of The 33rd International Conference on Machine Learning (ICML '16)*. 1938–1947.

[48] Nate Veldt, Christine Klymko, and David F. Gleich. 2019. Flow-Based Local Graph Clustering with Better Seed Set Inclusion. In *Proceedings of the 2019 SIAM International Conference on Data Mining (SDM '19)*.

[49] David P Williamson. 2019. *Network flow algorithms*. Cambridge University Press.

[50] Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. 2017. Local Higher-Order Graph Clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. 555–564.

[51] Yuichi Yoshida. 2019. Cheeger Inequalities for Submodular Transformations. In *Proceedings of the 2019 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '19)*. 2582–2601. https://doi.org/10.1137/1.9781611975482.160

[52] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. 2006. Learning with Hypergraphs: Clustering, Classification, and Embedding. In *Advances in Neural Information Processing Systems (NeurIPS '06)*. 1601–1608.

[53] Yu Zhu and Santiago Segarra. 2022. Hypergraph cuts with edge-dependent vertex weights. *Applied Network Science* 7, 1 (2022), 1–20.

A PROOFS

Proof of Lemma 3.1. Let $S \subseteq V$ be an arbitrary set of nodes satisfying $\pi(S) \leq \pi(\bar{S})$. For each pair $(u,v) \in \mathcal{D}(S)$ there is a flow $f^{(uv)}$ over edges in E_G with flow value $w_H(u,v)$, and these flows can simultaneously be routed in E_G with congestion γ. All of these flows must pass through the cut edges in E_G, since each such pair (u,v) crosses the bipartition $\{S, \bar{S}\}$. Thus, we have

$$\begin{aligned}\textbf{cut}_H(S) &= \sum_{(u,v)\in\mathcal{D}(S)} w_H(u,v) \leq \sum_{(u,v)\in\mathcal{D}(S)} \sum_{(i,j)\in\partial_G S} f_{ij}^{(uv)} \\ &\leq \gamma \sum_{(i,j)\in\partial_G S} w_G(i,j) = \gamma \textbf{cut}_G(S).\end{aligned}$$

Therefore, $\phi_G(S) = \frac{\textbf{cut}_G(S)}{\pi(S)} \geq \frac{1}{\gamma}\frac{\textbf{cut}_H(S)}{\pi(S)} = \frac{1}{\gamma}\phi_H(S)$.

Proof of Lemma 3.2. Let $S \subseteq V$ be an arbitrary set of nodes satisfying $\pi(S) \leq \pi(\bar{S})$. Define $\hat{S} = S \cup U$ where $U \subseteq \mathcal{A}$ is chosen in such a way that $\textbf{cut}_{\mathcal{H}}(S) = \textbf{cut}_{G(\mathcal{H})}(\hat{S})$. Recall that $\mathcal{D}(S)$ denotes the set of pairs (u,v) where $\{u,v\} \in E_H$ with $u \in S$ and $v \in V - S$. Since H is embedded in $G(\mathcal{H})$ with congestion γ, for each pair $(u,v) \in \mathcal{D}(S)$ we have routed $w_H(u,v)$ flow from u to v. Summing up all of the flows crossing the cut gives:

$$\begin{aligned}\textbf{cut}_H(S) &= \sum_{(u,v)\in\mathcal{D}(S)} w_H(u,v) \leq \gamma \sum_{(i,j)\in\partial_{G(\mathcal{H})}\hat{S}} w_{G(\mathcal{H})}(i,j) \\ &= \gamma \textbf{cut}_{G(\mathcal{H})}(\hat{S}) = \gamma \textbf{cut}_{\mathcal{H}}(S).\end{aligned}$$

And so we have $\phi_{\mathcal{H}}(S) = \frac{\textbf{cut}_{\mathcal{H}}(S)}{\pi(S)} \geq \frac{1}{\gamma}\frac{\textbf{cut}_H(S)}{\pi(S)} = \frac{1}{\gamma}\phi_H(S)$.

Proof of Lemma 4.1. The minimum s-t cut set in $G(\mathcal{H}, R, \alpha)$ is some set of nodes $\{s\} \cup S \cup \mathcal{A}_S$ where $S \subseteq V$ is a set of nodes from the original hypergraph $\mathcal{H}$ and $\mathcal{A}_S$ is a subset of the auxiliary nodes from the cut preserver $G(\mathcal{H})$, designed so that the cut function in $G(\mathcal{H})$ matches the cut function in $\mathcal{H}$. If the minimum s-t cut value in $G(\mathcal{H}, R, \alpha)$ is less than $\pi(R)$, this means that for the set S:

$$\begin{aligned}&\frac{1}{\alpha}\textbf{cut}_{\mathcal{H}}(S) + \pi(R \cap \bar{S}) + \eta\pi(\bar{R} \cap S) < \pi(R) \\ \implies &\textbf{cut}_{\mathcal{H}}(S) + \alpha\pi(R \cap \bar{S}) + \alpha\eta\pi(\bar{R} \cap S) < \alpha\pi(R) \\ \implies &\textbf{cut}_{\mathcal{H}}(S) < \alpha\pi(R \cap S) - \alpha\eta\pi(\bar{R} \cap S) \\ \implies &\frac{\textbf{cut}_{\mathcal{H}}(S)}{\pi(R \cap S) - \eta\pi(\bar{R} \cap S)} < \alpha.\end{aligned}$$

The result will hold as long as we can show that $\min\{\pi(S), \pi(\bar{S})\} \geq \pi(R \cap S) - \eta\pi(\bar{R} \cap S)$. First note that $\pi(S) \geq \pi(R \cap S) \geq \pi(R \cap S) - \eta\pi(\bar{R} \cap S)$. Additionally, we have

$$\begin{aligned}\pi(R \cap S) - \eta\pi(\bar{R} \cap S) &= \pi(R \cap S) + \eta\pi(\bar{R} \cap \bar{S}) - \eta\pi(\bar{R}) \\ &= \eta\pi(\bar{R} \cap \bar{S}) - \pi(R \cap \bar{S}) \leq \pi(\bar{S}).\end{aligned}$$

Proof of Lemma 4.2. Let $f_R = \sum_{(u,v)\in R\times\bar{R}} f_R^{(uv)}$ be the directed multicommodity flow in $G(\mathcal{H})$ obtained from the maximum s-t flow f in $G(\mathcal{H}, R, \alpha)$. In other words, for each pair for $u \in R$ and $v \in \bar{R}$, $f_R^{(uv)}$ is a u-v flow function that sends $|f_R^{(uv)}| = \mathbf{M}_R(u,v)$ flow from u to v. Each $u \in R$ will be a *deficit* node, and each $v \in \bar{R}$ will be an *excess* node for this multicommodity flow function.

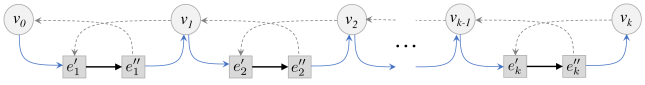


Figure 5: Every directed path in the reduced graph $G(\mathcal{H})$ between two nodes in V alternates between nodes in V and pairs of auxiliary nodes $\{e_i', e_i''\}$ from different CB-gadgets. If the original flow function sends flow from v_0 to v_k, replacing the flow along blue solid edges with flow along the dashed gray edges reverses the flow direction. Reversing flow paths in this way will not increase congestion.

We can assume without loss of generality that f is cycle-free, which implies that f_R is also cycle-free. Let $S \subseteq V$ be an arbitrary set; our goal is to edit f_R to turn it into a flow function $f_S = \sum_{(u,v)\in S\times\bar{S}} f_S^{(uv)}$ that routes $\mathbf{M}_R(u,v)$ flow from u to v whenever $u \in S$, $v \in \bar{S}$, and $\{u,v\}$ is an edge in M_R. In particular, this means we need to find a way to reverse the flow direction for any pair $(u,v) \in (S \cap \bar{R}) \times (\bar{S} \cap R)$, since for this type of pair f_R sends flow from v to u rather than from u to v.

We will first consider what it means to reverse flow on a single directed flow path. Consider an arbitrary edge $\{u,v\}$ in M_R such that $u \in S \cap \bar{R}$ and $v \in \bar{S} \cap R$. The original multicommodity flow function f_R includes a flow function $f_R^{(vu)}$ that sends $\mathbf{M}_R(u,v)$ units of flow from v to u. By Lemma 2.1, this can be decomposed as $f_R^{(vu)} = \sum_{j=1}^{r} f_j^{(vu)}$, where each f_j is a directed flow path from v to u. By the construction of $G(\mathcal{H})$, this flow path will travel through nodes and edges in a sequence of CB-gadgets corresponding to a sequence of hyperedges $\{e_1, e_2, \ldots, e_k\} \subseteq \mathcal{E}$ in the hypergraph $\mathcal{H} = (V, \mathcal{E})$. For the ith hyperedge e_i in this sequence and the corresponding ith CB-gadget, let e_i' be the first auxiliary node in the gadget for this hyperedge, and e_i'' be the second. Recall that there is an edge (e_i', e_i''), and for each node $x \in e_i$ there is a directed edge (x, e_i') and a directed edge (e_i'', x) (see Figure 1). A simple directed flow path $f_j^{(vu)}$ from $v = v_0$ to $u = v_{k+1}$ is therefore is a sequence of the form:

$$\begin{aligned}v = v_0 \to (e_1' \to e_1'') \to v_1 \to (e_2' \to e_2'') \to v_2 \to \cdots \\ \cdots \to v_{k-1} \to (e_k' \to e_k'') \to v_k = u,\end{aligned} \tag{15}$$

where $v_t \in V$ for each $t \in \{0, 1, \ldots, k\}$, and where the same amount of flow is sent along each edge. By the construction of every CB-gadget, there is a path in the opposite direction with the same exact set of edge weights:

$$\begin{aligned}u = v_k \to (e_k' \to e_k'') \to v_{k-1} \to \cdots \\ \cdots \to v_2 \to (e_2' \to e_2'') \to v_1 \to (e_1' \to e_1'') \to v_0 = v.\end{aligned} \tag{16}$$

Recall that f_R is cycle-free. This means that no flow is sent along edges of the form (v_i, e_i') for $i \in \{1, 2, \ldots, k\}$, because f_R contains a flow path $f_j^{(vu)}$ with positive flow on the edges (e_i', e_i'') and (e_i'', v_i). Therefore, we can replace the flow path $f_j^{(vu)}$ in (15) with the flow path $\hat{f}_j^{(vu)}$ defined on the edges in (16), with the same flow value as $f_j^{(vu)}$ but traveling in the opposite direction. Figure 5 illustrates this flow reversal process for a single flow path.

We construct a new multicommodity flow function f_S by simultaneously applying this flow reversal process to every flow path in f_R that goes in the wrong direction. Formally, if p_{vu} denotes the set of simple directed flow paths from v to u in f_R (obtained from a flow decomposition as in Lemma 2.1), then $f_R = \sum_{(v,u)\in R\times\bar{R}} \sum_{j\in p_{vu}} f_j^{(vu)}$, and we define:

$$f_S = f_R + \sum_{(v,u)\in(\bar{S}\cap R)\times(S\cap\bar{R})} \sum_{j\in p_{vu}} \hat{f}_j^{(vu)} - f_j^{(vu)}.$$

In other words, for every $(u,v) \in (S\cap\bar{R})\times(\bar{S}\cap R)$ and every directed flow path $f_j^{(vu)}$ from v to u in f_R, replace $f_j^{(vu)}$ with $\hat{f}_j^{(vu)}$.

It remains to prove that f_S has congestion at most $1/\alpha$. We prove this by considering different types of edges (v,e'), (e'',v), and (e',e''), where $v \in V$, and where $\{e',e''\}$ are the two auxiliary nodes from a CB-gadget for some hyperedge $e \in \mathcal{E}$. Observe first of all that this flow reversal procedure never changes the flow on edge (e',e''), so the congestion remains the same on edges that go between auxiliary nodes. The other two edges (v,e') or (e'',v) have the same weight, and because f_R is cycle free, at most one of these edges has a positive amount of flow in f_R. If (v,e') has a positive amount of flow in f_R, then (e'',v) has no flow in f_R. If the flow reversal process changes anything, it will take some of the flow through (v,e') and transfer it to (e'',v). In this case, the congestion on edge (v,e') cannot be worse because we are only removing flow. Meanwhile, the edge (e'',v) started with no flow and then received some of the flow from (v,e'). After the flow transfer process, the congestion on (e'',v) will not exceed $1/\alpha$ since it has the same weight as edge (v,e') and the congestion on (v,e') was at most $1/\alpha$. We can provide an analogous argument in the case where (e'',v) has a positive flow in f_R but (v,e') does not. Thus, this flow reversal process does not make the congestion worse.

Proof of Theorem 4.3. In each iteration the algorithm computes a maximum s-t flow in $G(\mathcal{H},R,\alpha)$. By Lemma 4.2, if the flow value is $\pi(R)$ then we can return a π-regular bipartite graph M_R that can be embedded with congestion $1/\alpha$. Otherwise, Lemma 4.1 guarantees that the minimum s-t cut set has expansion less than α. The algorithm is guaranteed to return a bipartite graph on the first iteration, because it is impossible to find a set with π-expansion less than the minimum value $2/\pi(V)$. If the algorithm finds a set S with $\phi_{\mathcal{H}}(S,\pi) < \alpha$ and then terminates, this means that in the previous iteration it found a bipartite graph M_R that can be embedded with congestion $\alpha/2$. For every $\alpha > U|\mathcal{E}|$, finding a maximum s-t flow in $G(\mathcal{H},R,\alpha)$ is guaranteed to return a cut set S with $\phi_{\mathcal{H}}(S,\pi) < \alpha$. This is because $\mathbf{cut}_{\mathcal{H}}(R) \le U|\mathcal{E}|$, so the node set $\{s\cup R\}$ is an s-t cut set in the auxiliary graph with cut value $\mathbf{cut}_{\mathcal{H}}(R)/\alpha \le U|\mathcal{E}|/\alpha < 1 \le \pi(R)$. Since HyperCutOrEmbed starts at $\alpha = 2/\pi(V)$ and doubles α at every iteration, it will take at most $O(\log U|\mathcal{E}|\pi(V))$ iterations before returning a cut set.

B RUNTIME ANALYSIS

To provide a runtime analysis for Algorithm 2, assume the hypergraph is connected and that all hyperedge weights are scaled to be integers. Let $n = |V|$, $m = |E|$, and $\mu = \sum_{e\in\mathcal{E}} |e|$. In order to focus on the main terms in the runtime, will use $\tilde{O}$ notation to hide logarithmic factors of m and n. For our analysis we also assume that the maximum edge penalty U and sum of node weights are small enough that $O(\log U)$ and $O(\log \pi(V))$ are both $\tilde{O}(1)$, and hence the maximum number of iterations of HyperCutOrEmbed is $\tilde{O}(1)$. When choosing a node weight function corresponding to conductance we have $\log\pi(V) = O(\log mn)$, and choosing the π corresponding to standard expansion we have $\log\pi(V) = \log n$.

In order to speed up the runtime for Algorithm 2, we can apply existing sparsification techniques for hypergraph-to-graph reduction [44]. This allows us to model the generalized cut function of $\mathcal{H}$ to within a factor of $(1+\varepsilon)$ for a small constant $\varepsilon > 0$ with an augmented graph $G(\mathcal{H})$ with $N = O(n + \sum_{e\in\mathcal{E}} \log|e|) = \tilde{O}(n+m)$ nodes and $M = O(\sum_{e\in\mathcal{E}} |e|\log|e|) = \tilde{O}(\mu)$ edges. Constructing this graph takes $O(M)$ time. For $\alpha > 0$, the graph $G(\mathcal{H},R,\alpha)$ also has $O(N)$ nodes and $O(M)$ edges.

The flow decomposition step in HyperCutOrEmbed can be accomplished in $O(M\log N) = \tilde{O}(\mu)$ time using dynamic trees [39]; note that for this step we do not need to explicitly return the entire flow decomposition but simply must identify the endpoints in each directed path for the bipartite graph we are embedding. Lemma 5.1 indicates that the total time spent on the cut player strategy will be $\tilde{O}(|E_{H_t}|)$, which is bounded above by $\tilde{O}(\mu)$. To see why, observe that the number of edges added to the bipartite graph constructed by FlowEmbed will be bounded above by the number of different directed flow paths, which by Lemma 2.1 is bounded above by $O(M) = O(\mu)$. Combining the edges from all $O(\log^2 n)$ bipartite graphs shows $\tilde{O}(|E_{H_t}|) = \tilde{O}(\mu)$.

The overall runtime of our algorithm is dominated by the time it takes to solve a maximum s-t flow in $G(\mathcal{H},R,\alpha)$. This overall runtime is $\tilde{O}(\mu + (n+m)^{3/2})$ if using the algorithm of van den Brand et al. [41]. The recent algorithm of Chen et al. [8] brings the runtime down to $\tilde{O}(\mu^{1+o(1)})$, nearly linear in terms of the hypergraph size μ. For comparison, the existing LP relaxation [20] and SDP relaxation [31], which only apply to all-or-nothing hypergraph cuts, both involve $\Omega(n^2+m)$ variables and $\Omega(n^3+\sum_{e\in E}|e|^2)$ linear constraints. When written in the form $\min_{\mathbf{Ax}=\mathbf{b}} \mathbf{c}^T\mathbf{x}$, the LP has $\Omega(n^3+\sum_{e\in E}|e|^2)$ constraints and variables. Even recent breakthrough theoretical results in LP solvers [12, 19] lead to runtimes significantly worse than $\Omega(n^6 + n^3\sum_e |e|^2 + \mu^2)$.

C PRACTICAL IMPROVEMENTS

In practice we use a slightly altered version of the HyperFlowEmbed procedure that returns a bipartite graph M_R that can be embedded with congestion $1/\alpha$ and a set S that has π-expansion equal to α, rather than just a set S with $\phi_{\mathcal{H}}(S,\pi) < 2\alpha$. To accomplish this, we set $\alpha = \phi_{\mathcal{H}}(R,\pi)$ in the first iteration and solve a maximum s-t flow problem on $G(\mathcal{H},R,\alpha)$ to search for a set S with π-expansion better than $\phi_{\mathcal{H}}(R,\pi)$. In each iteration, we update α to equal the π-expansion of the improved set found in the previous iteration, until no more improvement is found. This iterative refinement approach is standard and typically used in practice by related ratio cut improvement algorithms [35, 43, 47]. Although performing a bisection method over α leads to better theoretical runtimes, in practice it typically takes only a few iterations of cut improvement before the iterative refinement procedure converges. Thus, this is often faster in practice in addition to improving the approximation guarantee by a factor 2.