Densest Subhypergraph: Negative Supermodular Functions and Strongly Localized Methods

Yufan Huang
huan1754@purdue.edu
Purdue University
West Lafayette, IN, USA

David F. Gleich
dgleich@purdue.edu
Purdue University
West Lafayette, IN, USA

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

ABSTRACT

Dense subgraph discovery is a fundamental primitive in graph and hypergraph analysis which among other applications has been used for real-time story detection on social media and improving access to data stores of social networking systems. We present several contributions for localized densest subgraph discovery, which seeks dense subgraphs located nearby given seed sets of nodes. We first introduce a generalization of a recent *anchored densest subgraph* problem, extending this previous objective to hypergraphs and also adding a tunable locality parameter that controls the extent to which the output set overlaps with seed nodes. Our primary technical contribution is to prove when it is possible to obtain a strongly-local algorithm for solving this problem, meaning that the runtime depends only on the size of the input set. We provide a strongly-local algorithm that applies whenever the locality parameter is not too small, and show via counterexample why strongly-local algorithms are impossible below a certain threshold. Along the way to proving our results for localized densest subgraph discovery, we also provide several advances in solving global dense subgraph discovery objectives. This includes the first strongly polynomial time algorithm for the densest supermodular set problem and a flow-based exact algorithm for a heavy and dense subgraph discovery problem in graphs with arbitrary node weights. We demonstrate our algorithms on several web-based data analysis tasks.

CCS CONCEPTS

• **Theory of computation → Network flows**; **Network optimization**; *Linear programming*; • **Mathematics of computing → Hypergraphs**; • **Information systems** → *Web mining*.

KEYWORDS

Densest Subgraph, Strongly Localized Graph Algorithms, Hypergraph Algorithms, Maximum Flow, Supermodular Functions

ACM Reference Format:
Yufan Huang, David F. Gleich, and Nate Veldt. 2024. Densest Subhypergraph: Negative Supermodular Functions and Strongly Localized Methods. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589334.3645624

WWW '24, May 13–17, 2024, Singapore, Singapore
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0171-9/24/05.
https://doi.org/10.1145/3589334.3645624

1 INTRODUCTION

A common paradigm in unsupervised data analysis is to take as input a graph or hypergraph and to output extremal subsets. The types of extremal subsets range from sets of minimum cut [48] to minimum sparsest cut [41] to maximum clique [8]. The underlying hypothesis is that extremal sets reflect important and noteworthy structures that are informative for exploratory data analysis, or useful for downstream algorithms such as graph partitioners or machine learning pipelines that operate on subsets of the larger graph. This basic paradigm is fundamental in Web analysis and applied to problems such as detecting real-time stories on social media [6], improving access to data stores of social-networking systems [21], and a wide variety of clustering and community detection tasks over web-based datasets [4, 18, 31, 38, 55].

An issue with this paradigm is that there are many cases where extremal sets are trivial or simply unuseful. For instance, the minimum cut set in an unweighted graph with any degree 1 node is just that single node, which yields little information; a set of minimum conductance may be a simple small subgraph that just happens to have a small number of bridges to the rest of the graph [38]. A second and related challenge is that finding the extremal subset typically results in an NP-complete problem. Even if solved approximately, it may still consume substantial time.

Localized graph algorithms are a practical solution to this problem. The idea is that we rephrase the extremal search problem with respect to a reference set of nodes R. For instance, we may want the solution *within* R or *nearby* R with some measure of distance or fraction of R. This area has been most developed in the space of algorithms for finding small conductance cuts in a graph where techniques range between spectral methods [3, 4], flow methods [5, 19, 35, 44, 56, 57], and heat-kernel diffusions [33]. These techniques have also been extended to hypergraph analysis [28, 29, 40, 49, 53]. For methods that effectively *grow* small subsets, R may be as small as a single node; whereas for other techniques that *shrink or adapt* R, then R must be considerably larger. Often, a goal with these algorithms is to get a strongly localized runtime guarantee such that the total runtime scales with the size of the output instead of the size of the input graph. Using a localized algorithm enables one to analyze *many* interesting sets in the graph by varying R. Localized algorithms have already been widely used in web-based data analysis tasks such as detecting related retail products on Amazon [33, 52, 56], identifying groups of same-topic posts on Stackoverflow [52], and finding communities in various types of online social networks [33, 56, 57].

Although many extremal set problems in graph analysis focus on finding small *cut* values, another perspective on extremal sets seeks *high density* independently of cut values. The *densest subgraph* is one such example that seeks a subgraph S of maximum average degree. In a small surprise, this subset can be computed in polynomial time by a classic flow algorithm [22] or via linear programming [10]. A simple peeling algorithm that removes vertices from the graph one at a time has long been known to be a 2-approximation for the problem [10]. More recently, Boob et al. proposed an iterated peeling algorithm that has been shown to converge to the optimal solution [11]. Many variants and generalizations of the densest subgraph problem have been studied and considered (see [34] for a very recent survey). One of the most general of these is the densest supermodular subset problem (DSS) introduced by Chekuri et al. [11], where the goal is to maximize the ratio between a nonnegative monotone supermodular function f and the size of the returned set. Localized variants of the densest subgraph problem have also been considered recently [15]. However, localized algorithms for dense subgraph discovery remain underexplored and remain far less understood than localized algorithms for finding small cuts.

In this paper we greatly expand the scope of possible algorithms for dense subgraph computations, both in terms of global and local variants of the problem. We first provide a simple reduction that leads to efficient exact algorithms for a more general version of the densest supermodular subset (DSS) problem where the supermodular function does not need to be nonnegative (Theorem 1). This captures several dense subgraph problems that are not special cases of the standard nonnegative DSS problem [15, 42]. We then provide the first strongly polynomial algorithm for DSS (Algorithm 1, Theorem 2); previous approaches came with weakly polynomial runtimes. Our final contribution to *global* dense subgraph discovery algorithms is to design a flow-based exact algorithm for finding the densest subset of a node-weighted graph or hypergraph. Prior research on this problem showed how to obtain efficient flow-based solutions in the case of graphs with strictly non-negative weights [17, 22]; our results show how this can be extended to arbitrary node weights (Section 5.1).

In addition to our results for global dense subgraph discovery, we greatly advance the state of the art in localized densest subgraph computations (Section 5.2). First, we establish a parametric formulation of the discrete objective function underlying localized densest subgraph discovery (Problems 5, 6). This allows us to vary the degree of localization and continuously tradeoff between the degree of localization and the amount of computation. We explicitly delineate the region of strong locality where algorithms can have a runtime that scales independently of graph size (Theorem 4). Moreover, we show hypergraph generalizations of all of these algorithms. Our methods use max-flow / min-cut computations as a primitive and we show (in the appendix) counter-examples where standard *peeling* methods cannot approximate these objectives at all.

We demonstrate the advantages of the techniques on a variety of web-relevant datasets. This includes a hypergraph of web domains where hypergraphs are induced by hosts (Section 6.1). We show how our localized algorithms can help identify a densely connected set of about 1300 academic domains around the world.

2 PRELIMINARIES AND RELATED WORK

Let $G = (V, E)$ denote a graph with vertex set V and edge set E. Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with vertex set V and hyperedge set $\mathcal{E}$.

Table 1: Two kinds of degrees we consider.

	Normal	Fractional
Degree	$\deg(v) = \sum_{e \ni v} 1$	$\overline{\deg}(v) = \sum_{e \ni v} \frac{1}{\lvert e \rvert}$
Volume	$\mathrm{Vol}(S) = \sum_{v \in S} \deg(v)$	$\overline{\mathrm{Vol}}(S) = \sum_{v \in S} \overline{\deg}(v)$
Maximum	$\Delta(S) = \max_{v \in S} \deg(v)$	$\bar{\Delta}(S) = \max_{v \in S} \overline{\deg}(v)$

Each hyperedge $e \in \mathcal{E}$ is a subset of V and a graph is the special case of a hypergraph with $|e| = 2$. Our results for hypergraphs focus on unweighted and undirected hyperedges without self-loops, though the techniques can be easily extended to weighted hyperedges. At the same time, our results in some cases rely on reductions to weighted and directed graphs. By default, we use uv to denote a directed edge from vertex u to v, and a set of nodes $\{v_1, \dots, v_k\}$ to denote a hyperedge. For a hypergraph, let $r = \max_{e \in \mathcal{E}} |e|$ denote its rank. For a (hyper)graph and a set S, let $e[S]$ denote the number of (hyper)edges fully contained in S.

Throughout this paper, we mainly consider two kinds of degrees, the definitions of them and their corresponding volumes and maximums are summarized in Table 1. We have the following connection between degree and fractional degree.

Lemma 1. *For a hypergraph, we have*

$$2\overline{deg}(v) \leqslant deg(v) \leqslant r\overline{deg}(v), \quad \begin{matrix}\textit{and as}\\ \textit{a result}\end{matrix} \quad 2\overline{Vol}(S) \leqslant Vol(S) \leqslant r\overline{Vol}(S).$$

Proof. Because we focus on hypergraphs without self-loops, $2\overline{\deg}(v) = \sum_{e \in \mathcal{E}: e \ni v} \frac{2}{|e|} \leqslant \sum_{e \in \mathcal{E}: e \ni v} 1$. Since $r = \max_{e \in \mathcal{E}} |e|$, $r\overline{\deg}(v) = \sum_{e \in \mathcal{E}: e \ni v} \frac{r}{|e|} \geqslant \sum_{e \in \mathcal{E}: e \ni v} 1$. □

To evaluate any set function $f : 2^V \to \mathbb{R}$ we assume it is available as a value oracle as is standard practice. A set function f is normalized if $f(\emptyset) = 0$ and nonnegative if $f(S) \geqslant 0, \forall S \subseteq V$. Further, let $EO(f)$ be the maximum amount of time to evaluate $f(S)$ for a subset $S \subseteq V$ and $M(f)$ be an upper bound for $|f(S)|$ for all $S \subseteq V$. A set function f is supermodular if and only if $f(S) + f(T) \leqslant f(S \cap T) + f(S \cup T)$ for any $S, T \subseteq V$, and accordingly f is submodular if $-f$ is supermodular. A function is modular if it is both supermodular and submodular. Note that a normalized, nonnegative and supermodular set function f is monotone. Moreover, we call a set function cardinality-based if $f(S) = f(T)$, for all $S, T \subset V$ with $|S| = |T|$, and asymmetric if there exist $S \subset V$ where $f(S) \neq f(V \setminus S)$.

2.1 Graph Cut and Hypergraph Cut

Densest subgraph discovery (DSG) has a close connection with graph cut problems as the decision version of DSG is solvable by reducing it to a graph min s-t cut problem [22]. Here we briefly introduce some graph cut concepts that will show up in the following discussion. For a weighted directed graph $G = (V, E, w : E \to \mathbb{R})$ and a set S, the value of its induced cut is $\mathbf{cut}_G(S) = \sum_{u \in S, v \in \bar{S}} w(\{u, v\})$. The graph min s-t cut problem is to find the minimal graph cut while enforcing $s \in S$ and $t \in \bar{S}$. In other words, $\mathbf{min\text{-}st\text{-}cut}_G = \min_{S \subset V: s \in S, t \in \bar{S}} \mathbf{cut}_G(S)$.

The introduction of hyperedges enables a variety of definitions of cut as one hyperedge can be cut in more than one way now.

Here we adopt a recent generalized notion of a hypergraph cut function [39, 52]. Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, associate each hyperedge e with a splitting function $w_e : 2^e \to \mathbb{R}_{\geqslant 0}$ that maps each subset $A \subseteq e$ to a nonnegative splitting penalty. The value $w_e(A)$ indicates the penalty when $S \cap e = A$. Then for one vertex set $S \subseteq V$, the cut penalty it incurs is $\mathbf{cut}_{\mathcal{H}}(S) = \sum_{e \in \mathcal{E}} w_e(S \cap e)$. The corresponding hypergraph min s-t cut problem is $\textbf{min-st-cut}_{\mathcal{H}} = \min_{S \subset V: s \in S, t \in \bar{S}} \mathbf{cut}_{\mathcal{H}}(S)$.

2.2 Related Work

The classic densest subgraph problem is defined as

Problem 1 (Densest Subgraph (DSG)). *Given a graph $G = (V, E)$, find a vertex set S maximizing the fraction $e[S]/|S|$.* [1]

DSG and its variants have received significant attention over the past a few decades. They mainly admit two categories of exact solutions, one is flow-based [22] and the other one is based on a linear program (LP) [10]. One popular approximation algorithm for DSG and some variants is greedy peeling [10], which runs in linear time and is much faster than exact solutions. One variant of DSG, called densest subhypergraph (DSHG), is same as Problem 1 except the graph G is replaced by a hypergraph $\mathcal{H}$ [25, 26]. For a detailed introduction, refer to the recent tutorial [51] and survey [34].

Recently [9] introduces an iterative peeling method for Problem 1 called Greedy++ which shows quick convergence to the optimum. Then [11] showed that Greedy++ achieves a $(1-\varepsilon)$-approximation in $O(1/\varepsilon^2)$ iterations and extends iterative peeling to a broader class of problems called densest supermodular subset (DSS).

Problem 2 (Densest Supermodular Subset (DSS) [11]). *Given a normalized, nonnegative monotone supermodular function $f : 2^V \to \mathbb{R}_{\geqslant 0}$, maximize $f(S)/|S|$.*

This is an important breakthrough because numerous DSG variants are special cases of Problem 2 [17, 22, 25, 50, 54]. Therefore iterative peeling offers a faster algorithm for them compared with flow and LP. Moreover, [24] proposes an even faster and more scalable iterative algorithm for Problem 2 based on solving the quadratic relaxation of the dual of Charikar's LP. Although iterative peeling converges fast in practice, it is hard to terminate as soon as some user-defined approximation ratio is achieved as the optimum is not known in advance. Recently, [17] tackles this issue for a subclass of Problem 2, heavy and dense subgraph with nonnegative vertex weights.

Besides the line of designing faster *global* algorithms for DSG and its variants, there is some recent interest in studying seeded variants of DSG where a seed set R is given and the objective is to find a densest subgraph around this seed set [15, 17, 47]. Of these, only [15] provides an objective that gives a strongly local algorithm, meaning that the optimal answer is found only by exploring a small portion of the whole graph, via the objective:

Problem 3 (Anchored Densest Subgraph (ADS)). *Given a graph $G = (V, E)$ and a seed set $R \subset V$, find a vertex set S maximizing $\left(2e[S] - Vol(S \cap \bar{R})\right)/|S|$.*

In Problem 3, the bias towards the seed set is encoded by adding penalties onto vertices outside the seed set. Here for simplicity we omit the strict anchor set A defined in the ADS problem [15], but all our results and analysis go through with this additional constraint.

[1]We always treat $0/0 = -\infty$ and $\infty \cdot 0 = 0$.

3 GENERAL DENSE SUPERMODULAR SUBSET

The function $2e[S] - \mathrm{Vol}(S \cap \bar{R})$ in Problem 3 is a normalized supermodular function as $e[S]$ is supermodular and $\mathrm{Vol}(S \cap \bar{R})$ is modular. But it is not a special case of Problem 2 as this function is not guaranteed to be nonnegative. This inspires our broader class:

Problem 4 (Densest Supermodular Subset with Possible Negative Values). *Given a normalized supermodular function $f : 2^V \to \mathbb{R}$, maximize $f(S)/|S|$.*

In addition to the anchored densest subgraph objective mentioned above, this new formulation also generalizes the objective $\max_{S \subset V} \left(e[S] - \alpha \mathbf{cut}_G(S)\right)/|S|$ considered in [42] . We prove the following connection between this extension and the class DSS.

Theorem 1. *For any normalized supermodular function $f : 2^V \to \mathbb{R}$, one can construct a normalized, nonnegative monotone supermodular function $g : 2^V \to \mathbb{R}_{\geqslant 0}$ such that*

$$\operatorname*{argmax}_{S \subset V} f(S)/|S| = \operatorname*{argmax}_{S \subset V} g(S)/|S|$$

and g can be constructed in $O(|V| EO(f))$ time.

Proof. Let $C = \max\{0, \max_{v \in V} -f(\{v\})\}$, in other words C is the smallest nonnegative quantity such that $C + f(\{v\}) \geqslant 0, \forall v \in V$. Then we construct g as $g(S) := f(S) + C|S|, \forall S \subset V$. Since $C|S|$ is modular, g is still supermodular. Observe that because f is supermodular, for any set $S = \{v_1, v_2, \ldots, v_{|S|}\} \subset V$, we have $g(S) = f(S) + C|S| \geqslant f(S \setminus \{v_1\}) + f(\{v_1\}) + C|S| \geqslant \ldots \geqslant \sum_{i=1}^{|S|} f(\{v_i\}) + C|S| \geqslant 0$ which means that $g(S)$ is nonnegative and implies that $g(S)$ is monotone. Thus,

$$\max_{S \subset V} f(S)/|S| + C = \max_{S \subset V} g(S)/|S|.$$

And C can be computed by querying $f(\{v\})$ for each $v \in V$, which can be done in $O(|V| EO(f))$ time. □

This theorem implies that any exact algorithm for DSS will remain as an exact algorithm for the extended Problem 4, such as using linear programming or combining binary search with repeated submodular minimization.

While extending the definition to non-negative valued functions may seem a minor change as it is easy to adapt exact algorithms, this change has large implications for approximation algorithms. For instance, the efficient greedy peeling fails to hold a constant approximation ratio. In the Appendix B, we show one example where greedy peeling may perform arbitrarily badly. Even for the recent iterative peeling approach [11], the picture is more complex and the bounds are not straightforward. That said, we hypothesize that iterative peeling remains an effective practical heuristic.

4 A STRONGLY POLYNOMIAL ALGORITHM

Strongly polynomial algorithms are those having a running time bounded by a polynomial of the number of input numbers instead of their size. In the context of Problem 4, a strongly polynomial algorithm is one whose runtime is dependent on $|V|, EO(f)$ but independent of $M(f)$.

Algorithm 1 Density Improvement Framework for Problem 4

```
Input: A normalized supermodular function $f : 2^V \to \mathbb{R}$ via oracle
Output: $S^*$ maximizing $f(S)/|S|$.
1: $S_0 \leftarrow V, t \leftarrow 0$
2: repeat
3:     $t \leftarrow t+1, \beta_t \leftarrow d(S_{t-1})$
4:     Find an $S_t$ maximizing $\beta_t|S| - f(S)$
5: until $\beta_t|S_t| - f(S_t) = 0$
6: return $S_{t-1}$
```

As mentioned before, two common exact solutions for Problem 4 are linear programming or combining binary search with submodular minimization. However, neither of those two algorithms are strongly polynomial. In particular, there is no strongly-polynomial time solution for linear programming. Meanwhile, given a problem instance of Problem 4, one can binary search the optimum and answer the decision problem that given a parameter β, decide whether there exists one set S with $f(S)/|S| > \beta$. However, the range to perform binary search and the termination condition both depend on $M(f)$. For example, for the simplest case that f is nonnegative and integral, we have $M(f) = f(V) = \max_{S\subset V} f(S)$. Thus the optimum falls into the range $[0, f(V)]$ and for any two $S, T \subset V$ with $f(S)/|S|$ and $f(T)/|T|$ different, the minimum gap between $f(S)/|S|$ and $f(T)/|T|$ is $1/|V|^2$. This means the binary search takes $O(\log(M(f)|V|^2))$ iterations. Hence it has a dependence on $M(f)$ and is not strongly polynomial.[2]

Inspired by Dinklebach's algorithm [16], we give a simple strongly polynomial algorithm framework for a general normalized supermodular function f in Algorithm 1. Each iteration minimizes a submodular function in strongly polynomial time [30, 36, 45, 46]. And thus, we call it a density improvement framework as in each iteration the answer gets improved. More importantly, we will also show in numerical experiments that this in fact can be much faster than alternatives based on binary search commonly used in the literature.

The following standard result shows optimality at termination, of which proof is included in Appendix A.1.

Lemma 2. *For a normalized, supermodular function $f : 2^V \to \mathbb{R}$, and a given parameter β, $\min_{S\subset V} \beta|S| - f(S) < 0$ if and only if there exists a set S such that $f(S)/|S| > \beta$. As a result, $\min_{S\subset V} \beta|S| - f(S) = 0$ if and only if $\max_{S\subset V} f(S)/|S| \leqslant \beta$.*

Suppose Algorithm 1 runs for T iterations. By Lemma 2, $\beta_T|S_T| - f(S_T) = 0$ suggests that

$$\max_{S\subset V} f(S)/|S| \leqslant \beta_T = f(S_{T-1})/|S_{T-1}|,$$

which means S_{T-1} is optimal. We make one important observation that the size of S_t is strictly decreasing. This is intuitive since we can view $\beta|S|$ as an ℓ_1-norm penalty on $|S|$, thus with penalty coefficient β increasing, the size of the solution to $\min_{S\subseteq V} \beta|S| - f(S)$ tends to decrease. By our algorithm design, $\forall t < T$,

$$\beta_t|S_t| < f(S_t), \tag{1}$$

[2]This statement holds for a general function f. For specific f, we may have that $M(f)$ is a simple function of $|V|$ or $|E|$ and binary search would be strongly polynomial.

because we terminate the algorithm once at a point we get $\beta_t|S_t| = f(S_t)$. As f is normalized, for $t < T$, S_t is non-empty. Hence Equation (1) implies $\forall t < T$,

$$\beta_t < f(S_t)/|S_t| = \beta_{t+1}, \tag{2}$$

where the equality follows from the definition of β_t. This means the sequence of β is strictly increasing. By our algorithm design, for $\forall t > 0, S_t$ is the minimizer of $\beta_t|S| - f(S)$, hence we have $\forall t > 0$

$$\beta_t|S_t| - f(S_t) = \min_{S\subset V} \beta_t|S| - f(S) \leqslant \beta_t|S_{t+1}| - f(S_{t+1}). \tag{3}$$

Observe that via Equation (2), we have $\forall t < T-1$,

$$f(S_t)/|S_t| = \beta_{t+1} < \beta_{t+2} = f(S_{t+1})/|S_{t+1}|.$$

which further shows that $\forall t < T-1$,

$$\beta_{t+1}|S_{t+1}| - f(S_{t+1}) < 0 = \beta_{t+1}|S_t| - f(S_t). \tag{4}$$

Combining Equations (3) and (4), we get $\forall 1 \leqslant t < T-1$

$$\beta_t(|S_t| - |S_{t+1}|) \leqslant f(S_t) - f(S_{t+1}) < \beta_{t+1}(|S_t| - |S_{t+1}|).$$

As $\beta_t < \beta_{t+1}$ for $\forall t < T$, we get for all $1 \leqslant t < T-1$,

$$|S_t| > |S_{t+1}|.$$

Notice that if $T > 1$, then $d(S_1) = \beta_2 > \beta_1 = d(S_0) = d(V)$ where the last equality is because of our choice of S_0. Thus $S_1 \neq V$ and $|S_1| < |V| = |S_0|$.

Based on the discussion above, we have the following result.

Theorem 2. *Assume the density improvement procedure terminates after T iterations, then we have*

- $\forall t < T, \beta_t < \beta_{t+1}$.
- $\forall t < T-1, |S_t| > |S_{t+1}|$.

As a result, this procedure will terminate after at most $|S_0|+1 = |V|+1$ iterations and the algorithm runs in strongly polynomial time.

This conclusion shows that Algorithm 1 will iteratively decrease the size of the solution. The supermodularity of f ensures that Line 4 of Algorithm 1 can be done in strongly polynomial time. As a result, the whole procedure is strongly polynomial. We can see that in each iteration, when minimizing $\beta_t|S| - f(S)$, the minimization algorithm does not matter much as long as it is strongly polynomial.

Also here for simplicity of analysis, we take $S_0 = V$, but we can always start from some better initial sets and there is some potential to reuse information from previous solutions, which is sometimes more useful than solving the whole problem from scratch. The bound on the number of iterations is also rather loose, in other words, we believe in practice the number of iterations may be $o(|V|)$. Moreover, the supermodularity of f is not necessary as long as f has some special properties which enable a strong polynomial algorithm for minimizing $\beta|S| - f(S)$.

5 ANCHORED DENSEST SUBHYPERGRAPH

We now turn to concrete special cases of Problem 4 that focus on returning dense subhypergraphs that are localized around a given seed set in a hypergraph.

Problem 5 (Anchored Densest Subhypergraph (ADSH)). *Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a locality parameter $\varepsilon \geqslant 0$ and a seed set $R \subset V$, find a vertex set S maximizing $d(S) = (e[S] - \varepsilon Vol(S \cap \bar{R})/2)/|S|$, where $\bar{R} = V \setminus R$ is the complement set of R.*

Problem 6 (Anchored Densest Subhypergraph with Fractional Volume (ADSH-F)). *Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a locality parameter $\varepsilon \geqslant 0$ and a seed set $R \subset V$, find a vertex set S maximizing $\bar{d}(S) = (e[S] - \varepsilon\overline{Vol}(S \cap \bar{R}))/|S|$, where $\bar{R} = V \setminus R$.*

These problems are inspired by ADS (Problem 3) and we generalize it by applying to hypergraphs and including a locality parameter. They use the two different types of hypergraph volume and ADSH-F avoids situations where large hyperedges incur extremely large penalties. We focus results on ADSH in the interest of space.

5.1 A Flow-Based Exact Algorithm

We first introduce a flow-based exact algorithm that applies to the following problem that generalizes Problems 5 and 6:

$$\max\left(e[S] - p(S)\right)/|S|, \quad p : V \to \mathbb{R}_{\geqslant 0}. \tag{5}$$

Throughout the paper, for any $p : V \to \mathbb{R}$ and vertex set $S \subset V$, we let $p(S) = \sum_{v \in S} p(v)$ denote the summation of entries of p indexed by S. We show how to solve this by reducing it to a sequence of generalized hypergraph s-t cut problems, which can be solved in turn via reduction to graph s-t cut problems using existing techniques [52].

Consider the decision version of Eq. (5). For a parameter β, there exists an S such that $\left(e[S] - p(S)\right)/|S| > \beta$ if and only if there exists an S such that $p(S) + \beta|S| - e[S] < 0$. We have that $e[S] = \overline{\text{Vol}}(S) - \sum_{e \in \mathcal{E}} g_e(S)$ where $g_e(S) = \min \frac{1}{|e|}\{|e \cap S|, \infty|e \setminus S|\}$, as

$$\begin{aligned} e[S] &= \sum_{e \in \mathcal{E}} \mathbb{1}_{\{e \subset S\}} = \sum_{e \in \mathcal{E}} \left(\frac{|e \cap S|}{|e|} - g_e(S)\right) \\ &= \sum_{v \in S}\sum_{e \ni v} \frac{1}{|e|} - \sum_{e \in \mathcal{E}} g_e(S) = \overline{\text{Vol}}(S) - \sum_{e \in \mathcal{E}} g_e(S). \end{aligned} \tag{6}$$

We can therefore verify whether there exists an S such that $p(S) + \beta|S| - e[S] < 0$ by solving a hypergraph min s-t cut problem on the extended hypergraph $\mathcal{H}_\beta$ constructed as follows:

- Keep all of $\mathcal{H} = (V, \mathcal{E})$ and for each hyperegdge e, assign one splitting function $g_e(S) = \min \frac{1}{|e|}\{|e \cap S|, \infty|e \setminus S|\}$.
- Introduce one super source s and create one edge $\{s, v\}$ with weight $\overline{\deg}(v)$ for each $v \in V$.
- Introduce one super sink t and create one edge $\{v, t\}$ with weight $\beta + p(v)$ for each $v \in V$.

We focus here on the case where $\beta \geqslant 0$ since the optimal solutions to Problems 5 and 6 are always nonnegative. Note however that we can also handle $\beta < 0$ using slight adjustments to the construction above. We refer to edges directed connected to s or t *terminal* edges, denoted by $\mathcal{E}^{st}$. Their splitting function is the same as the cut function for a standard graph: the penalty is 0 if the edge is not cut and otherwise is equal to the weight of the edge. Every $S \subset V$ induces a hypergraph s-t cut on $\mathcal{H}_\beta$ with value

$$\begin{aligned} \textbf{cut}(S \cup \{s\}) &= \sum_{v \in \bar{S}} \overline{\deg}(v) + \sum_{v \in S} \left(p(v) + \beta\right) + \sum_{e \in \mathcal{E}} g_e(S) \\ &= \overline{\text{Vol}}(\bar{S}) + \beta|S| + p(S) + \sum_{e \in \mathcal{E}} g_e(S) \\ &= \overline{\text{Vol}}(\mathcal{H}) - e[S] + \beta|S| + p(S). \end{aligned} \tag{7}$$

where the last equality is due to Eq. (6). We summarize as:

Observation 1. *The minimum s-t cut of $\mathcal{H}_\beta$ is strictly smaller than $\overline{\text{Vol}}(\mathcal{H})$ if and only if there exists S with $\left(e[S] - p(S)\right)/|S| > \beta$.*

For each $e \in \mathcal{E}$, the splitting function g_e is submodular, cardinality-based and *asymmetric*. Under these conditions, Veldt et al. have shown how to reduce a generalized hypergraph s-t cut problem to a graph s-t cut problem [52]. We include details here for completeness. For this reduction, no change needs to be made to terminal edges, since by construction they already involve only two nodes. As shown in [25, 52], each $e \in \mathcal{E}$ can be replaced by the following *gadget*

- Introduce one auxiliary node v_e.
- For each $v \in e$, introduce a directed edge from v to v_e with weight $\frac{1}{|e|}$, and a directed edge from v_e to v with weight ∞.

This leads to a new directed graph $G_{\mathcal{H}}$ on an augmented node set. For any $S \subset V$, if we include v_e on the same side of S, then we incur a directed cut penalty of $\infty|e \setminus S|/|e|$, otherwise the incurred directed cut penalty is $|e \cap S|/|e|$. The minimum s-t cut solution in $G_{\mathcal{H}}$ will naturally place the auxiliary node v_e in a way that incurs the minimum possible penalty subject to the placement of the original node set V. Therefore, for a node set $S \subseteq V$, the penalty incurred because of nodes in hyperedge e is exactly $g_e(S)$.

With this core algorithmic step, what is left is to determine what βs to test. The density improvement framework introduced in Section 4 applies here and provides a strongly polynomial algorithm. One could also use binary search, as done in many densest subgraph variants. Observe that the answer falls in the interval $[-p(V), |\mathcal{E}|]$. When p is integral or rational, there exists some predetermined smallest gap between any two possible non-equal values of $\left(e[S] - p(S)\right)/|S|$, and we can determine the termination condition accordingly. When p is irrational, although this strategy fails, we can still apply parametric flow to solve it, as in [22].

New results for DSG in vertex-weighted graphs. We note in passing that our approach for solving Problem 5 implies more general results for solving densest subgraph problems in vertex-weighted graphs. The following problem was introduced in [22] and later considered in [17].

Problem 7 (Heavy and Dense Subgraph Problem (HDSP)). *Given an undirected graph (G, V, E, w_V, w_E) without self-loops, where $w_V : V \to \mathbb{R}_{\geqslant 0}$ and $w_E : E \to \mathbb{R}_{\geqslant 0}$, find $S^* \subset V$ such that*

$$S^* = \underset{S \subset V}{\text{argmax}} \frac{e[S] + \sum_{v \in S} w_V(v)}{|S|},$$

where $e[S]$ is the sum of the weights of edges fully contained in S.

This problem explicitly considers weighted edges. Note that our approach for solving Objective 5 can easily be extended to weighted settings as well by scaling hyperedges (and the resulting edges in the reduced graph). While HDSP focuses only on standard graphs, our approach applies more generally to hypergraphs. Furthermore, while HDSP focuses on nonnegative vertex weights, our approach effectively deals with nonpositive vertex weights. Combining our techniques with Goldberg's flow network for HDSP [22] leads to the following stronger result.

Observation 2. *There is an efficient flow-based exact algorithm for any problem of the form $\max_{S \subset V}\left(e[S] + p(S)\right)/|S|$, where $p : V \to \mathbb{R}$ is a vertex function with no sign constraint.*

5.2 A Strongly-local Flow Algorithm

We now show how to design a *strongly-local* algorithm for Problem 5, meaning that the runtime depends only on the size of R. Showing how to obtain a runtime that is independent of global graph properties is the most technically challenging contribution of our paper. Our goal here is to strike a balance between obtaining strong theoretical guarantees of this form while ensuring the algorithm is practical. Thus, rather than pursuing the tightest possible analysis, we focus on providing the simplest exposition that leads to a runtime that is bounded exclusively in terms of $|R|$ and $\text{Vol}(R)$.

We first provide high-level intuition as to why strongly-local algorithms are possible. For Problem 5, we have $p(S) = \varepsilon\text{Vol}(S \cap \bar{R})/2$. This means that in the directed graph $G_{\mathcal{H}}$ presented in Section 5.1, every vertex from the original hypergraph will have one directed edge from the source node s with weight $\overline{\deg}(v)$ and one directed edge to the sink node t with weight $\beta + \varepsilon\text{Vol}(v \cap \bar{R})/2$. When solving a maximum s-t flow problem in this graph, if ε is large enough we can pre-route a significant amount of flow and saturate many of the edges leaving the source node s. In particular, for large enough ε, pre-routing flow in this way will saturate all edges (s, v) for each $v \in \bar{R}$. In the remaining residual graph, the s will only be adjacent to nodes in R, and the total weight of edges leaving s will be much smaller than the total weight of edges entering t. In this way, the maximum s-t flow value will be bounded in terms of the size of R (rather than the size of the whole graph), and by carefully solving a sequence of smaller flow problems "nearby" R we will be able to find the minimum s-t cut of the entire graph $G_{\mathcal{H}}$ without having to visit all of its nodes and edges. In what follows we provide complete details for formalizing this intuition. Formally, we prove for $\varepsilon \geqslant 1$, Problem 5 can be solved exactly by a strongly-local algorithm.

We assume throughout our analysis that $d(R) = \Omega(1)$. In other words, the subhypergraph induced by R has a density lower bounded by some universal constant. This will simplify the technical exposition without significantly changing the analysis. We could alternatively weaken this to a natural assumption that R contains at least one hyperedge, which would only change the analysis slightly.

As a warm-up we prove that when $\varepsilon \geqslant 2$, Problem 5 is equivalent to finding the densest subhypergraph within R. A strongly-local algorithm can then easily be obtained by considering only subsets of R. This provides additional intuition as to why strongly-local algorithms are possible for large enough ε.

Lemma 3. *When $\varepsilon \geqslant 2$, $\max_{S \subset V} d(S) \Leftrightarrow \max_{S \subset R} d(S)$.*

The proof is provided in Appendix A.2. We now present a strongly-local algorithm for $1 \leqslant \varepsilon < 2$. We first bound the range of values containing the optimal value d^*.

Lemma 4. *Let $d^* = \max_{S \subset V} d(S)$, then for $1 \leqslant \varepsilon < 2$,*

$$\bar{\Delta}(R) \geqslant d^* \geqslant \max_{S \subset R} d(S) \geqslant d(R) = \Omega(1).$$

The proof is provided in Appendix A.3. Hence, we only need to test those βs falling into this range of values that contains d^*. Recall that for a given parameter β, one can verify whether there exists one set S such that $\left(e[S] - p(S)\right)/|S| > \beta$ by minimizing $p(S) + \beta|S| - e[S]$ and comparing the minimum to 0. Since $p(S) = \varepsilon\text{Vol}(S \cap \bar{R})/2$, we specifically minimize

$$\varepsilon\text{Vol}(S \cap \bar{R})/2 + \beta|S| - e[S]. \tag{8}$$

The following lemma bounds the size of the optimal set $S^* = \text{argmax}_{S \subseteq V} d(S)$, and the degree of nodes in S^*, in terms of the quantities that depend only on R.

Lemma 5. *When $\varepsilon \geqslant 1$, let $S^* = \text{argmax}_{S \subseteq V} d(S)$, then we have*

(1) $|S^*| \leqslant \overline{Vol}(R)$.
(2) $\forall v \in S^*, \deg(v) = O(\overline{Vol}(R) + \Delta(R))$.

Appendix A.4 provides a proof. This Lemma implies that when searching for the optimal S^*, we can ignore vertices with very high degrees. This is done by adding a directed edge from those vertices to t with weight ∞. These edges will never be a part of the minimum s-t cut, meaning that these vertices will never be part of S.

The main challenge is to find a minimum s-t cut in $\mathcal{H}_\beta$ in a strongly-local manner. Recall from the construction of $\mathcal{H}_\beta$ in Section 5.1 that for every vertex $v \in V$, there is an edge (s, v) and another edge (v, t). This means that the minimum s-t cut solution will have to cut one of these two edges for each vertex. Note that we can equivalently alter $\mathcal{H}_\beta$ so that each vertex in $\bar{R}$ has *either* an edge to the source s or sink t but not both. Concretely, for each $v \in \bar{R}$, we can remove the edge connected to s with weight $\overline{\deg}(v)$ and decrease the weight of the edge to t by $\overline{\deg}(v)$, so that the new weight is $\beta + \varepsilon \deg(v)/2 - \overline{\deg}(v)$. This is guaranteed to be nonnegative, since by Lemma 1 and the assumption that $\varepsilon \geqslant 1$ we have $\varepsilon \deg(v)/2 \geqslant \overline{\deg}(v)$. This adjustment will change the value of the minimum s-t cut by $\text{Vol}(\bar{R})$, but will not change the minimizer. In what follows we assume we are working with this slightly altered hypergraph; we overload the notation and still call this $\mathcal{H}_\beta$.

Our strongly-local procedure works by starting with a subset of $\mathcal{H}_\beta$ and growing it as needed in search for a global minimum s-t cut solution. We assume for this process that the hypergraph is given by oracle accesses. For each $v \in V$, let $\mathcal{N}_{\mathcal{E}}(v) = \{e \in \mathcal{E} : e \ni v\}$ be the set of hyperedges that v belongs to and $\mathcal{N}_{\mathcal{E}}(S) = \bigcup_{v \in S} \mathcal{N}_{\mathcal{E}}(v)$. Let $V(e) = \{v : v \in e\}$ be the set of vertices that belongs to e and $V(E) = \bigcup_{e \in E} V(e)$. Let $\mathcal{N}^{st}(v)$ denote those terminal edges incident to v and $\mathcal{N}^{st}(S) = \bigcup_{v \in S} \mathcal{N}^{st}(v)$. Given a vertex $v \in V$ or a hyperedge $e \in \mathcal{E}$, we can efficiently query $\mathcal{N}_{\mathcal{E}}(v)$ or $V(e)$ respectively. Combining these two oracles, we can efficiently compute the vertex neighborhood of one vertex $\mathcal{N}_V(v) = \{u \in V : \exists e \text{ s.t. } u, v \in e\}$ and $\mathcal{N}_V(S) = \bigcup_{v \in S} \mathcal{N}_V(v)$. We also assume some simple metadata are pre-stored together with the hypergraph, for example we can query $\deg(v)$, $\overline{\deg}(v)$ for any v, and $|e|$ for any e in $O(1)$ time. Hence $\mathcal{N}^{st}(S)$ can be constructed efficiently.

Instead of building all of $\mathcal{H}_\beta$ explicitly and computing the minimum s-t cut, we alternate between the following two steps:

- On the local hypergraph $\mathcal{L} \subseteq \mathcal{H}_\beta$, compute a minimum s-t cut induced by a vertex set $S_{\mathcal{L}}$ where $s \in S_{\mathcal{L}}$.
- Expand the local hypergraph $\mathcal{L}$ based on the min s-t cut $S_{\mathcal{L}}$ obtained in the above step.

This procedure ends at a point where we can certify that the s-t cut on $\mathcal{L}$ is also a solution to the s-t cut on the entire hypergraph $\mathcal{H}_\beta$. Concretely, let $\mathcal{L} = (V_{\mathcal{L}} \cup \{s, t\}, \mathcal{E}_{\mathcal{L}} \cup \mathcal{E}^{st}_{\mathcal{L}}, g)$ where $V_{\mathcal{L}} \subset V$ is a subset of the vertices of the hypergraph $\mathcal{H}$, $\mathcal{E}_{\mathcal{L}}$ is a subset of the hyperedges in $\mathcal{H}_\beta$ and $\mathcal{E}^{st}_{\mathcal{L}}$ is the set of the terminal edges in $\mathcal{H}_\beta$ between $V_{\mathcal{L}}$ and $\{s, t\}$, and g is the set of splitting functions corresponding to $\mathcal{E}_{\mathcal{L}} \cup \mathcal{E}^{st}_{\mathcal{L}}$. We initialize $V_{\mathcal{L}}$ to be $R \cup \mathcal{N}_V(R)$, in other

Algorithm 2 Strongly local method for solving Prob. 5 when $\varepsilon \geqslant 1$.

Input: R, ε, β, oracle access to a hypergraph $\mathcal{H}$.
Output: S minimizing Objective (8) for $p(S) = \varepsilon\mathrm{Vol}(S \cap \bar{R})/2$.

```
1: $V_{\mathcal{L}} \leftarrow R \cup \mathcal{N}_V(R), \mathcal{E}_L \leftarrow \mathcal{N}_{\mathcal{E}}(R), \mathcal{E}^{st}_{\mathcal{L}} \leftarrow \mathcal{N}^{st}(V_{\mathcal{L}}), X \leftarrow R$
2: repeat
3:     $S_{\mathcal{L}}$ = Solve min-st-cut in $\mathcal{L}$
4:     $S_{\text{new}} = S_{\mathcal{L}} \setminus (X \cup \{s,t\})$        ▷ Grow local hypergraph $\mathcal{L}$
5:     $V_{\mathcal{L}} \leftarrow V_{\mathcal{L}} \cup \mathcal{N}_V(S_{\text{new}})$
6:     $\mathcal{E}_{\mathcal{L}} \leftarrow \mathcal{E}_{\mathcal{L}} \cup \mathcal{N}_{\mathcal{E}}(S_{\text{new}}), \mathcal{E}^{st}_{\mathcal{L}} \leftarrow \mathcal{E}^{st}_{\mathcal{L}} \cup \mathcal{N}^{st}(S_{\text{new}})$
7:     $X \leftarrow X \cup S_{\text{new}}$
8: until $S_{\text{new}} = \emptyset$
9: return $S_{\mathcal{L}}$
```

words, the seed set union its vertex neighborhood. We initialize $\mathcal{E}_{\mathcal{L}}$ to be $\mathcal{N}_{\mathcal{E}}(R)$, in other words, those hyperedges touching the seed set R. Finally we initialize $\mathcal{E}^{st}_{\mathcal{L}}$ to be the terminal edges connected to $V_{\mathcal{L}}$. When we grow $\mathcal{L}$, we always guarantee it remains a subhypergraph of $\mathcal{H}_\beta$, which means we always have

$$\textbf{min-st-cut}_{\mathcal{L}} \leqslant \textbf{min-st-cut}_{\mathcal{H}_\beta}. \tag{9}$$

By carefully choosing how the local hypergraph $\mathcal{L}$ grows, we can guarantee that the inequality will reach equality, without ever having to explore the entire hypergraph. This growing process expands $\mathcal{L}$ by considering nodes in $S_{\mathcal{L}}$ and adding all of its neighboring edges and nodes from $\mathcal{H}_\beta$ that are not already in the local hypergraph $\mathcal{L}$. We specifically have the following two update rules:

- Update the vertex set by setting $V_{\mathcal{L}} \leftarrow V_{\mathcal{L}} \cup \mathcal{N}_V(S_{\mathcal{L}})$.
- Update the edge set by setting $\mathcal{E}_{\mathcal{L}} \leftarrow \mathcal{E}_{\mathcal{L}} \cup \mathcal{N}_{\mathcal{E}}(S_{\mathcal{L}}), \mathcal{E}^{st}_{\mathcal{L}} \leftarrow \mathcal{E}^{st}_{\mathcal{L}} \cup \mathcal{N}^{st}(S_{\mathcal{L}})$.

To avoid adding the neighbor of one vertex multiple times, we keep a list X and mark those vertices as *explored*. The algorithm ends when $S_{\mathcal{L}}$ does not introduce new vertices and edges. The whole procedure is summarized in Algorithm 2. Theorem 3 guarantees this will find the minimum s-t cut set, and Theorem 4 guarantees it will have a strongly-local runtime. We defer the proof for these results to the Appendix A.4.

Theorem 3. *When $\varepsilon \geqslant 1$, the optimal set S returned by Algorithm 2 minimizes the objective* (8).

Theorem 4. *For $\varepsilon \geqslant 1$, the local hypergraph $\mathcal{L}$ will contain $O((\overline{\mathrm{Vol}}(R) + |R|)\delta)$ hyperedges and at most $O((\overline{\mathrm{Vol}}(R) + |R|)\delta r)$ vertices where $\delta = O(\overline{\mathrm{Vol}}(R) + \Delta(R))$.*

6 EXPERIMENTS

We implement our proposed algorithms in Julia. We preprocess all the hypergraphs we use to remove dangling nodes, self-loops and multihyperedges. We defer some of the low-level details of implementation and experiments to Appendix E. Our code is available at https://doi.org/10.5281/zenodo.10681969.

To demonstrate the advantages and differences of the anchored densest subhypergraphs found by Problems 5 and 6, we compare them against running the anchored densest subgraph algorithm on the clique expansions of hypergraphs [15]. The specific clique expansions we consider are unweighted clique expansion (UCE) and weighted clique expansion (WCE). For WCE, each hyperedge e will be replaced by one clique where each edge has weight $\frac{1}{|e|}$. For UCE, it is replaced by one clique where each edge has weight 1.

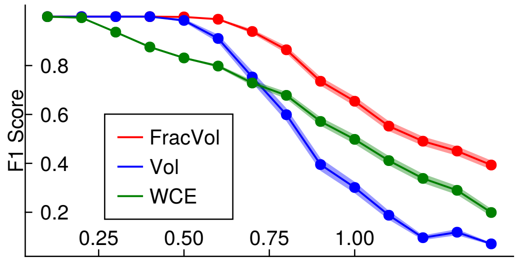


Figure 1: Solving Probs 5, 6 (Vol, FracVol) outperforms Anchored Densest Subgraph (WCE) in planted set models. The x-axis represents difficulty (m_1/m_2). Lines show mean F1 scores and bands show standard errors.

Table 2: Comparison between our Density Improvement (DI) Framework shown in Algorithm 1 and the standard binary search (BS). Time is the running time in seconds and iterations represent the number of subproblems solved. $\overline{|e|}$ indicates the average hyperedge size.

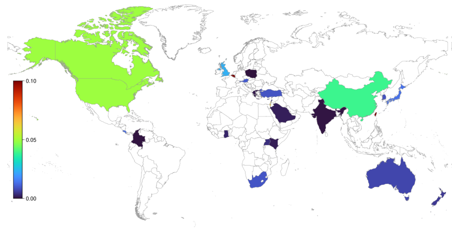
Datasets	n	m	$\overline{\lvert e\rvert}$	DI		BS	
				time	iters	time	iters
Walmart	87k	65k	6.9	6.4	9	18.5	43
Trivago	173k	220k	3.2	10.1	10	26.3	42
Math SX	153k	563k	2.6	19.5	8	89.5	47
Ask Ubuntu	82k	114k	2.3	2.6	10	8.7	43
Amazon	4.2M	2.3M	17.2	2239	10	9333	54

6.1 Density Improvement vs. Binary Search

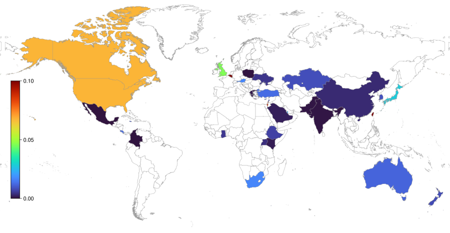
To demonstrate that our Density Improvement Framework shown in Algorithm 1 has good performance in practice, we perform comparison experiments against binary search on five different real-world hypergraph datasets, Walmart Trips [2], Amazon Reviews [43], Trivago Clicks [14], Threads Ask Ubuntu and Threads Math SX [7]. Since the search range and termination condition of binary search get complicated when the complexity of the objective function increases, here we simply study the densest subhypergraph problem , i.e. $f(S) = e[S]$. Each subproblem is solved by the same max. s-t flow solver. We compare two methods' running time and number of subproblems solved. The results are summarized in Table 2. We can see that on all five datasets, density improvement shows about 3.5x speed up, which demonstrates it is practical.

6.2 Experiments with Planted Dense Sets

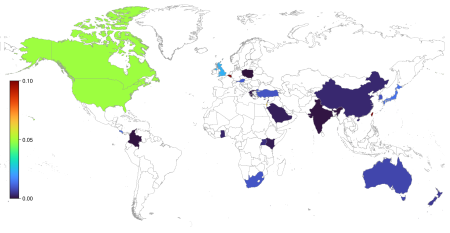
We first study the capacity of our objectives to find dense subhypergraphs on problems with planted dense subsets. Specifically, we build a graph with 1000 vertices and assign each vertex to one of the 30 clusters uniformly at random similar to a stochastic block model. Then we generate two kinds of hyperedges, m_1 hyperedges between clusters and m_2 hyperedges inside clusters. This allows us to plant 30 dense sets into this hypergraph. This is similar to



(a) A 1543 vertex, density 22.34 subhypergraph from Chinese universities

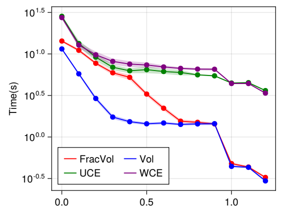


(b) A 1923 vertex, density 19.76 subhypergraph from on UK universities

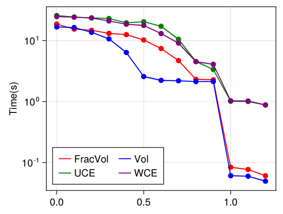


(c) A 1356 vertex, density 23.51 subhypergraph from the intersection

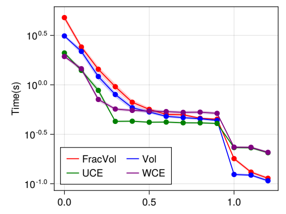
Figure 2: We show sets of domains as a colored map based on the number of domains associated with that region normalized by the total domains in the region. (Our attribution of domain to region is imperfect, but should capture general trends.)



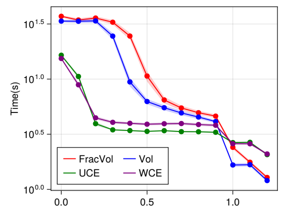
(a) Walmart Trips



(b) Trivago Clicks



(c) Threads Ask Ubuntu



(d) Threads Math SX

Figure 3: Running time comparison. The x-axis represents ε. We generate 100 different R for each dataset and run each method on them. Here we report the mean and stderr of the running time. Specifically, each R is generated by randomly sampling 10 seed nodes and then expanding them to a set with 200 nodes using random walks.

scenarios for planted partitions in *uniform* hypergraphs where each hyperedge has the same size [20] and is inspired by various ideas in random hypergraph and graph generation [1, 13, 23].

We let $m_2 = 50000$, and then compare our objectives with the baselines to see how well they can detect the underlying planted densest subhypergraphs when we vary m_1. The average hyperedge size in the hypergraphs we generate is 5.7. For each cluster, we generate 10 different seed sets R by sampling 5% vertices from that cluster and performing length-2 random walks to grow it to a set with size equal to 1.5 times the cluster size. In total, we generate 300 seed sets. For each objective, we compute F1 score between the detected subhypergraph and the ground truth planted cluster.

The results are summarized in Figure 1. Here we do not show the result for UCE as it exhibits similar behavior with WCE empirically. We can see that when the planted dense structures are relatively clear, i.e. the ratio $\frac{m_1}{m_2}$ is relatively small, both Vol and FracVol penalty are able to perfectly detect the planted dense cluster while WCE can not. As is expected, with m_1 increasing, it is harder for all methods to recover the planted dense cluster but FracVol has a clear advantage throughout.

6.3 Densely linked domains on the web

We perform a case study on the webgraph demonstrating our local dense subgraph tools' utility in network analysis. We build a domain-level hypergraph which has 147k vertices and 138k hyperedges, with average hyperedge size 11.3 from the host-level webgraph data. One common phenomenon of densest subgraph like objectives is that on real-world graphs, they usually do not have large densest subgraphs. On this hypergraph, the densest subhypergraph contains 105 nodes, 103 US domains and 2 UK domains (Oxford, Cambridge) with density 45.73.

Our tools allow us to go beyond this simple set. Here we take domains from the UK and mainland China as reference sets respectively and vary ε from 0.0 to 1.5 to find large, and reasonably dense subhypergraph. We identify one anchored densest subhypergraph with size 1923 and density 19.76 from the UK, and one anchored densest subhypergraph with size 1543 and density 22.34 from mainland China. By intersecting those two sets, we can get a denser subhypergraph with 1356 nodes and density 23.51 that spans universities throughout the world. This is illustrated in Figure 2.

6.4 Running Time Comparison

We compare runtime on real-world datasets and summarize results in Figure 3. All methods run faster when ε is large and their running time sharply decreases when ε enters the strongly local regime. Also, in general our anchored densest subhypergraph solvers are faster than clique expansion alternative when the hypergraph has a large mean hyperedge size, e.g. Walmart Trips and Trivago Clicks.

7 CONCLUSION

We propose two *localized* densest subhypergraph objectives and demonstrate their utility through experiments. Along the way, we also prove several interesting results for the general densest subgraph discovery problem. Future directions are making the algorithms scale to hypergraphs with billions of nodes and edges and further exploring the space of *localized* objectives for different scenarios.

ACKNOWLEDGEMENTS

DFG and YH are partially supported by NSF Award CCF-1909528, NSF IIS-2007481, and DOE DE-SC0023162.

REFERENCES

[1] Leman Akoglu and Christos Faloutsos. 2009. RTG: a recursive realistic graph generator using random typing. *Data Mining and Knowledge Discovery* 19, 2 (July 2009), 194–209. https://doi.org/10.1007/s10618-009-0140-7

[2] Ilya Amburg, Nate Veldt, and Austin R. Benson. 2020. Clustering in Graphs and Hypergraphs with Categorical Edge Labels. In *Proceedings of The Web Conference 2020*. 706–717. https://doi.org/10.1145/3366423.3380152 arXiv:1910.09943 [physics, stat]

[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. 2006. Communities from seed sets. In *Proceedings of the 15th international conference on World Wide Web*. 223–232.

[5] Reid Andersen and Kevin J Lang. 2008. An algorithm for improving graph partitions.. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA '08, Vol. 8)*. 651–660.

[6] Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanta Tirthapura. 2014. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB journal* 23 (2014), 175–199.

[7] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. 2018. Simplicial Closure and Higher-Order Link Prediction. https://arxiv.org/abs/1802.06916v2. https://doi.org/10.1073/pnas.1800683115

[8] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. 1999. The maximum clique problem. *Handbook of Combinatorial Optimization: Supplement Volume A* (1999), 1–74.

[9] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. 2020. Flowless: Extracting Densest Subgraphs Without Flow Computations. In *Proceedings of The Web Conference 2020 (WWW '20)*. Association for Computing Machinery, New York, NY, USA, 573–583. https://doi.org/10.1145/3366423.3380140

[10] Moses Charikar. 2000. Greedy Approximation Algorithms for Finding Dense Components in a Graph. In *Approximation Algorithms for Combinatorial Optimization (Lecture Notes in Computer Science)*, Klaus Jansen and Samir Khuller (Eds.). Springer, Berlin, Heidelberg, 84–95. https://doi.org/10.1007/3-540-44436-X_10

[11] Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. 2022. Densest Subgraph: Supermodularity, Iterative Peeling, and Flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 1531–1555. https://doi.org/10.1137/1.9781611977073.64

[12] Boris V. Cherkassky and Andrew V. Goldberg. 1995. On Implementing Push-Relabel Method for the Maximum Flow Problem. In *Integer Programming and Combinatorial Optimization*, Gerhard Goos, Juris Hartmanis, Jan Leeuwen, Egon Balas, and Jens Clausen (Eds.). Vol. 920. Springer Berlin Heidelberg, Berlin, Heidelberg, 157–171. https://doi.org/10.1007/3-540-59408-6_49

[13] Philip S Chodrow. 2020. Configuration models of random hypergraphs. *Journal of Complex Networks* 8, 3 (June 2020). https://doi.org/10.1093/comnet/cnaa018

[14] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. 2021. Generative Hypergraph Clustering: From Blockmodels to Modularity. *Science Advances* 7, 28 (July 2021), eabh1303. https://doi.org/10.1126/sciadv.abh1303

[15] Yizhou Dai, Miao Qiao, and Lijun Chang. 2022. Anchored Densest Subgraph. In *Proceedings of the 2022 International Conference on Management of Data*. ACM, Philadelphia PA USA, 1200–1213. https://doi.org/10.1145/3514221.3517890

[16] Werner Dinkelbach. 1967. On Nonlinear Fractional Programming. *Management Science* 13, 7 (1967), 492–498. jstor:2627691

[17] Adriano Fazzone, Tommaso Lanciano, Riccardo Denni, Charalampos E. Tsourakakis, and Francesco Bonchi. 2022. Discovering Polarization Niches via Dense Subgraphs with Attractors and Repulsers. *Proceedings of the VLDB Endowment* 15, 13 (Sept. 2022), 3883–3896. https://doi.org/10.14778/3565838.3565843

[18] Gary William Flake, Steve Lawrence, and C Lee Giles. 2000. Efficient identification of web communities. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 150–160.

[19] Kimon Fountoulakis, Meng Liu, David F Gleich, and Michael W Mahoney. 2023. Flow-based algorithms for improving clusters: A unifying framework, software, and performance. *SIAM Rev.* 65, 1 (2023), 59–143.

[20] Debarghya Ghoshdastidar and Ambedkar Dukkipati. 2014. Consistency of Spectral Partitioning of Uniform Hypergraphs under Planted Partition Model. In *Advances in Neural Information Processing Systems*, Vol. 27. Curran Associates, Inc.

[21] Aristides Gionis, Flavio PP Junqueira, Vincent Leroy, Marco Serafini, and Ingmar Weber. 2013. Piggybacking on social networks. In *VLDB 2013-39th International Conference on Very Large Databases*, Vol. 6. 409–420.

[22] A. V. Goldberg. 1984. *Finding a Maximum Density Subgraph*. Technical Report. University of California at Berkeley, USA.

[23] Lilith Orion Hafner, Chase Holdener, and Nicole Eikmeier. 2022. Functional Ball Dropping: A superfast hypergraph generation scheme. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE. https://doi.org/10.1109/bigdata55660.2022.10020506

[24] Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. 2022. Faster and Scalable Algorithms for Densest Subgraph and Decomposition. *Advances in Neural Information Processing Systems* 35 (Dec. 2022), 26966–26979.

[25] Shuguang Hu, Xiaowei Wu, and T-H. Hubert Chan. 2017. Maintaining Densest Subsets Efficiently in Evolving Hypergraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, Singapore Singapore, 929–938. https://doi.org/10.1145/3132847.3132907

[26] D.J.-H. Huang and A.B. Kahng. 1995. When Clusters Meet Partitions: New Density-Based Methods for Circuit Decomposition. In *Proceedings the European Design and Test Conference. ED&TC 1995*. 60–64. https://doi.org/10.1109/EDTC.1995.470419

[27] Yufan Huang, David F Gleich, and Nate Veldt. 2023. Densest Subhypergraph: Negative Supermodular Functions and Strongly Localized Methods. *arXiv preprint arXiv:2310.13792* (2023).

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

[29] Masahiro Ikeda, Atsushi Miyauchi, Yuuki Takai, and Yuichi Yoshida. 2022. Finding Cheeger Cuts in Hypergraphs via Heat Equation. *Theoretical Computer Science* 930 (Sept. 2022), 1–23. https://doi.org/10.1016/j.tcs.2022.07.006

[30] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. 2001. A Combinatorial Strongly Polynomial Algorithm for Minimizing Submodular Functions. *J. ACM* 48, 4 (July 2001), 761–777. https://doi.org/10.1145/502090.502096

[31] Yasushi Kawase, Yuko Kuroki, and Atsushi Miyauchi. 2019. Graph Mining Meets Crowdsourcing: Extracting Experts for Answer Aggregation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. 1272–1279. https://doi.org/10.24963/ijcai.2019/177 arXiv:1905.08088 [cs]

[32] Bryan Klimt and Yiming Yang. 2004. The Enron Corpus: A New Dataset for Email Classification Research. In *Machine Learning: ECML 2004 (Lecture Notes in Computer Science)*, Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi (Eds.). Springer, Berlin, Heidelberg, 217–226. https://doi.org/10.1007/978-3-540-30115-8_22

[33] 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.

[34] Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. 2023. A Survey on the Densest Subgraph Problem and Its Variants. https://doi.org/10.48550/arXiv.2303.14467 arXiv:2303.14467 [cs]

[35] Kevin Lang and Satish Rao. 2004. A Flow-Based Method for Improving the Expansion or Conductance of Graph Cuts. In *Conference on Integer Programming and Combinatorial Optimization (IPCO '04)*. 325–337. https://doi.org/10.1007/978-3-540-25960-2_25

[36] Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. 2015. A Faster Cutting Plane Method and Its Implications for Combinatorial and Convex Optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 1049–1065. https://doi.org/10.1109/FOCS.2015.68

[37] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data* 1, 1 (March 2007), 2–es. https://doi.org/10.1145/1217299.1217301

[38] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[39] Pan Li and Olgica Milenkovic. 2017. Inhomogeneous hypergraph clustering with applications. *Advances in neural information processing systems* 30 (2017).

[40] 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*. 2092–2103.

[41] David W Matula and Farhad Shahrokhi. 1990. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics* 27, 1-2 (1990), 113–123.

[42] Atsushi Miyauchi and Naonori Kakimura. 2018. Finding a Dense Subgraph with Sparse Cut. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*. Association for Computing Machinery, New York, NY, USA, 547–556. https://doi.org/10.1145/3269206.3271720

[43] 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)*. Association for Computational Linguistics, Hong Kong, China, 188–197. https://doi.org/10.18653/v1/D19-1018

[44] 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.

[45] James B. Orlin. 2009. A Faster Strongly Polynomial Time Algorithm for Submodular Function Minimization. *Mathematical Programming* 118, 2 (May 2009), 237–251. https://doi.org/10.1007/s10107-007-0189-2

[46] Alexander Schrijver. 2000. A Combinatorial Algorithm Minimizing Submodular Functions in Strongly Polynomial Time. *Journal of Combinatorial Theory, Series B* 80, 2 (Nov. 2000), 346–355. https://doi.org/10.1006/jctb.2000.1989

[47] Mauro Sozio and Aristides Gionis. 2010. The Community-Search Problem and How to Plan a Successful Cocktail Party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '10)*. Association for Computing Machinery, New York, NY, USA, 939–948. https://doi.org/10.1145/1835804.1835923
[48] Mechthild Stoer and Frank Wagner. 1997. A simple min-cut algorithm. *Journal of the ACM (JACM)* 44, 4 (1997), 585–591.
[49] 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 (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1970–1978. https://doi.org/10.1145/3394486.3403248
[50] Charalampos Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1122–1132. https://doi.org/10.1145/2736277.2741098
[51] Charalampos Tsourakakis and Tianyi Chen. 2021. *Dense Subgraph Discovery: Theory and Applications (Tutorial SDM 2021)*. https://tsourakakis.com/dense-subgraph-discovery-theory-and-applications-tutorial-sdm-2021/
[52] Nate Veldt, Austin R. Benson, and Jon Kleinberg. 2020. Hypergraph Cuts with General Splitting Functions. arXiv:2001.02817 [cs]
[53] 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 & Data Mining (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1708–1718. https://doi.org/10.1145/3394486.3403222
[54] Nate Veldt, Austin R. Benson, and Jon Kleinberg. 2021. The Generalized Mean Densest Subgraph Problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1604–1614. https://doi.org/10.1145/3447548.3467398
[55] Nate Veldt, David F Gleich, and Anthony Wirth. 2018. A correlation clustering framework for community detection. In *Proceedings of the 2018 World Wide Web Conference*. 439–448.
[56] 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)*.
[57] Di Wang, Kimon Fountoulakis, Monika Henzinger, Michael W Mahoney, and Satish Rao. 2017. Capacity releasing diffusion for speed and locality. In *International Conference on Machine Learning*. PMLR, 3598–3607.
[58] Jaewon Yang and Jure Leskovec. 2015. Defining and Evaluating Network Communities Based on Ground-Truth. *Knowledge and Information Systems* 42, 1 (Jan. 2015), 181–213. https://doi.org/10.1007/s10115-013-0693-z

A PROOFS

A.1 Proof of Lemma 2

Since f is normalized, if $\beta|S| - f(S) < 0$, then $S \neq \emptyset$. Therefore any S satisfying $\beta|S| - f(S) < 0$ has $f(S)/|S| > \beta$. Meanwhile any S with $f(S)/|S| > \beta$ naturally has $\beta|S| - f(S) < 0$.

Further, notice that $\beta|\emptyset| - f(\emptyset) = 0$, hence $\min_{S\subset V} \beta|S| - f(S) \leqslant 0$ always holds. As $\min_{S\subset V} \beta|S| - f(S) < 0$ and $\max_{S\subseteq V} f(S)/|S| > \beta$ are equivalent, the complement of them are also equivalent.

A.2 Proof of Lemma 3

By our assumption, $d(R)$ is positive. Hence the optimal S maximizing $d(S)$ has to intersect R, otherwise

$$\frac{e[S] - \varepsilon \mathrm{Vol}(S \cap \bar{R})/2}{|S|} = \frac{e[S] - \varepsilon \mathrm{Vol}(S)/2}{|S|} \leqslant 0.$$

For an S that intersects R, let $A = S \cap R$ and $B = S \cap \bar{R} = S \setminus A$. Then

$$\begin{aligned}\frac{e[S] - \varepsilon \mathrm{Vol}(S \cap \bar{R})/2}{|S|} &\leqslant \frac{e[S] - \mathrm{Vol}(S \cap \bar{R})}{|S|} \leqslant \frac{e[S \cap R]}{|S|} \\ &\leqslant \frac{e[S \cap R]}{|S \cap R|} = \frac{e[A] - \varepsilon \mathrm{Vol}(A \cap \bar{R})}{|A|}.\end{aligned}$$

For the second inequality we use the fact that any hyperedge fully contained in S and intersecting $S \cap \bar{R}$ is counted at least once in $\mathrm{Vol}(S \cap \bar{R})$. The last equality follows from the definition of A. This inequality shows that for any set S intersecting R, removing vertices outside R will not make the answer worse. Thus it is equivalent to maximizing $d(S)$ over $S \subset R$.

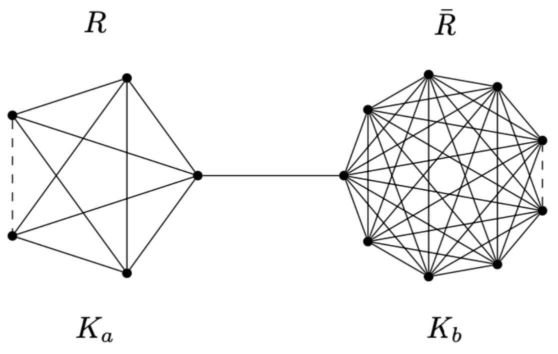


Figure 4: A counter example for greedy peeling.

A.3 Proof of Lemma 4

On the one hand, we know that $d^* \geqslant \max_{S\subset R} d(S) \geqslant d(R) = \Omega(1)$ by assumption. On the other hand, observe that

$$\begin{aligned} d(S) = \frac{e[S] - \varepsilon \mathrm{Vol}(S \cap \bar{R})/2}{|S|} &\leqslant \frac{\overline{\mathrm{Vol}}(S) - \frac{1}{2}\mathrm{Vol}(S \cap \bar{R})}{|S|} \\ &\leqslant \frac{\overline{\mathrm{Vol}}(S) - \overline{\mathrm{Vol}}(S \cap \bar{R})}{|S|} \leqslant \frac{\overline{\mathrm{Vol}}(S \cap R)}{|S \cap R|} \leqslant \bar{\Delta}(R).\end{aligned}$$

The first inequality relies on Eq. (6) and the second inequality follows from Lemma 1.

A.4 Proofs for Lemma 5, Theorems 3 and 4

The proofs of these results are included in a longer version [27].

B COUNTEREXAMPLE FOR GREEDY PEELING

We adopt the greedy peeling algorithm for Problem 2 mentioned in [11] (Theorem 3.1). For completeness, we restate it here. For a normalized nonnegative supermodular set function $f : 2^V \to \mathbb{R}_{\geqslant 0}$, we first initialize $S := V$ and then we recursively find $v = \underset{v\in S}{\mathrm{argmin}} f(v \mid S - v)$ and update $S := S \setminus \{v\}$ until S becomes empty. Here $f(v \mid S - v) := f(S) - f(S \setminus \{v\})$ is the marginal gain brought by element v to the set S. We see that when $f(S) = e[S]$ as in the classical DSG, $f(v \mid S - v)$ becomes the degree of vertex v in the subgraph $G[S]$.

Now we are ready to present one example which shows that greedy peeling may perform very poorly when f is not guaranteed to be nonnegative. Here we give a counterexample on a graph, which is a special case of a hypergraph. Consider the graph in Figure 4, which contains two cliques linked by one edge. The clique on the left-hand side contains a vertices and the clique on the right-hand side contains b vertices. We let $b = 9a$. We denote the left clique by R and the right clique by $\bar{R}$ accordingly. We consider the following objective

$$\max_{S\subset V} \frac{e[S] - p(S)}{|S|}$$

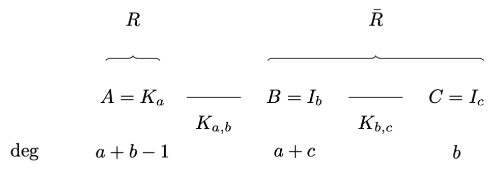


Figure 5: One counterexample that shows strong locality is not guaranteed when $\varepsilon < 1$. It consists of three set of vertices A, B, C, and they contain a, b, c vertices respectively. A form a clique and B, C are both independent sets. There is an edge between each vertex of A and B, in other words there is a complete bipartite graph between them, the same holds for B and C. The seed set R is A. The degree for vertices in A, B, C are $a-1+b, a+c$ and b respectively. We let $c \gg b \gg a$.

where $p(v) = \frac{2}{3}b$ for $v \in \bar{R}$ and $p(v) = 0$ for $v \in R$. For this example, greedy peeling will first remove vertices from R as the marginal gain brought by vertices from $\bar{R}$ is at least $\frac{1}{3}b = 3a$ and on the contrary the marginal gain brought by vertices from R is at most a. Hence greedy peeling will not start peeling off vertices from $\bar{R}$ until the whole R is peeled off. Then by symmetry, vertices from $\bar{R}$ will be peeled off in any random order.

We notice that when the intermediate subgraph only contains vertices inside $\bar{R}$, the objective is negative since $e[S] - p(S) = \binom{|S|}{2} - \frac{2}{3}b|S| < 0$.

When the intermediate subgraph still contains vertices from R, as pointed out before, at this time the whole clique inside $\bar{R}$ is also contained in the intermediate subgraph. Assume it contains x vertices from R, then the objective now is

$$\frac{\binom{x}{2} + 1 + \binom{b}{2} - \frac{2}{3}b^2}{x+b} \leqslant \frac{\binom{a}{2} + 1 + \binom{b}{2} - \frac{2}{3}b^2}{b} < 0$$

as we let $b = 9a$. This means the peeling algorithm will only output a negative answer on this example as we treat $0/0 = -\infty$. However the optimal solution is choosing $S = R$ and the optimum is $\frac{a-1}{2}$.

C EXAMPLE FOR $\varepsilon < 1$

We give one example illustrating why strong locality cannot be guaranteed when $\varepsilon < 1$. The example is summarized in Figure 5. The high-level intuition is that in this graph, the optimal answer consists of all vertices, whose size is impossible to be bounded by a polynomial of quantities only related to R. The detailed analysis is included in a longer version [27].

D ADDITIONAL EXPERIMENTS

To demonstrate the effectiveness of our proposed methods, we perform some additional experiments.

D.1 Density Improvement

We show that the density improvement framework described in Algorithm 1 also has good performance on the ordinary dense subgraph discovery problem. We perform comparison experiments against binary search on five real-world datasets, HepPh, AstroPh [37], Email-Enron [32, 38], com-amazon, and com-youtube [58]. The experimental results are summarized in Table 3. We can see that on all five datasets, our framework exhibits consistent improvement over binary search. Also, this Dinklebach-like framework demonstrated usefulness in other graph tasks, e.g. cut improvement, see a survey [19].

Table 3: Comparison between our Density Improvement (DI) Framework shown in Algorithm 1 and the standard binary search (BS) for the ordinary dense subgraph discovery problem. Time is the running time in seconds and iterations represent the number of subproblems solved.

Datasets	n	m	DI		BS	
			time	iters	time	iters
HepPh	11204	117619	0.29	4	1.97	35
AstroPh	17903	196972	1.17	7	5.21	37
Email-Enron	33696	180811	1.01	7	4.95	40
com-amazon	334863	925872	16.97	11	51.4	45
com-youtube	1134890	2987624	38.1	9	190.6	55

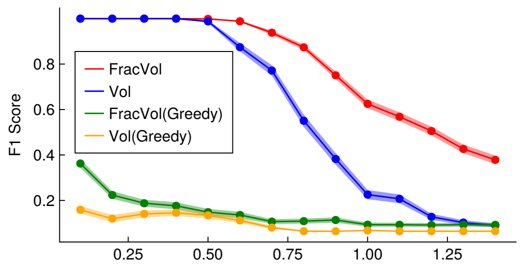


Figure 6: Solving Probs 5, 6 *exactly* outperforms *approximating* them using greedy peeling.

D.2 Comparison with Greedy Peeling

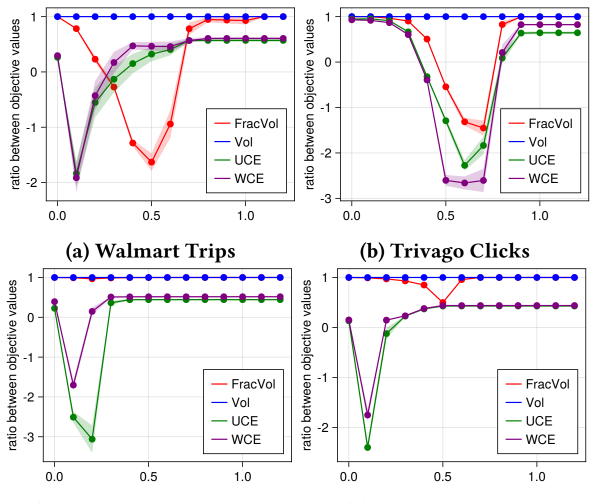
Greedy peeling is a prevalent approach for densest subgraph discovery because of its simplicity and efficiency. However, as shown in Section B, greedy peeling may fail when the supermodular function f is not guaranteed to be non-negative. Here we conduct further experiments to show some empirical evidence. On hypergraphs with planted dense sets, we apply greedy peeling on objectives 5 and 6, and compare the returned solutions with the exact solutions computed via flow. The results are shown in Figure 6. We can see that greedy peeling exhibits a much worse F1 score.

D.3 Comparison with Anchored Densest Subgraph

In Section 6.2, we compare our algorithms against running the anchored densest subgraph algorithm on the clique expansion of hypergraphs about their abilities of detecting planted dense sets. Here we provide some more empirical evidence with regard to their objective values. For a seed set R and a specific locality parameter ε, we solve objectives 5 and 6 on the hypergraph and objective 3 on the weighted clique expansion and unweighted clique expansion of the hypergraph, and then compute objective 5 of the dense sets they detect. In other words, we are comparing how well each of these methods does at approximating objective 5. So we normalize the

Table 4: Short descriptions of datasets used in the paper.

Datasets	Description
Walmart	sets of products bought on Walmart shopping trips, where labels are departments of products
Trivago	sets of hotels clicked on in a Web browsing session, where labels are the countries of the accommodation
Math SX	sets of users asking and answering questions on threads on math exchange
Ask Ubuntu	sets of tags applied to questions on askubuntu
Amazon	sets of products reviewed by users on Amazon, where labels are product categories
ca-AstroPh	Arxiv Astro Physics collaboration network
ca-HepPh	Arxiv High Energy Physics (Phenomenology) collaboration network
Email-Enron	email communication network related to Enron
com-amazon	product co-purchasing network on amazon
com-youtube	social network based on friendship on youtube



(a) Walmart Trips (b) Trivago Clicks

(c) Threads Ask Ubuntu (d) Threads Math SX

Figure 7: How well solving objective 6 or running the anchored densest subgraph objective 3 on the weighted or unweighted clique expansion of the hypergraph can approximately solve Problem 5. Objective values are normalized by the optimum.

objective of all other sets by the objective of the set computed by solving objective 5 and see how far they are from 1. From Figure 7, we can see that running the anchored densest subgraph algorithm usually cannot give a good result in terms of objective 5.

E MORE EXPERIMENT DETAILS

Throughout the paper, we solve minimum s-t cut problems using a highest-label push-relabel algorithm with optimizations from [12]. Concretely, as is standard, for binary search, we let the search range be $[|\mathcal{E}|/|V|, \bar{\Delta}(V)]$ and termination condition is when the search range becomes shorter than $1/(|V|(|V|-1))$ [34]. For our density improvement, we let $S_0 = V$.

E.1 Dataset Information

We provide some more information about the datasets we use. The 5 real-world hypergraph datasets in Section 6.1 are available at https://www.cs.cornell.edu/~arb/data/ and the 5 real-world graph datasets in Section D.1 are available at https://snap.stanford.edu/data/. We summarize them in Table 4.

E.2 Construction of the Domain-Level Hypergraph

We take the host-level data from Common Crawl (https://commoncrawl.org/blog/host-and-domain-level-web-graphs-oct-nov-jan-2020-2021). It contains 490 million nodes and 2.6 billion directed edges between hosts. We build a domain-level hypergraph by forming one hyperedge for each host within a domain. The contents of the hyperedge are all the domains linked from that host. We focus on the subgraph induced by the domain names of educational and academic institutions. Concretely, we take all domains ⋆.edu, ⋆.ac.⋆ or ⋆.edu.⋆.

E.3 Hyperedge Generation for Experiments with Planted Dense Sets

Each hyperedge is generated in a similar way. Given a vertex pool S, we first sample two different vertices from S, and then we iteratively grow the hyperedge. In each iteration, with probability p we stop the generating process and with probability $1-p$ we sample another unique vertex from S and continue to the next iteration. Once the hyperedge reaches some pre-determined max size threshold, we end the generating process. For those m_1 hyperedges between clusters, we let the vertex pool S be the whole vertex set V and for those m_2 hyperedges inside clusters, we pick a random cluster for each and set the vertex pool as vertices from that cluster. In this way, we plant 30 dense subhypergraphs in this 1000-vertex hypergraph. We let $m_2 = 50000, p = 0.2$ and set the max hyperedge size as 12. As m_1 increases, it will be much harder to detect the planted densest subhypergraphs as the background hypergraph gets denser and denser.

F ETHICS AND DATA

All of the data we use are publicly available and we do no mining of the data for specific human identifiable attributes. Some of the hypergraph data is based on public human activity, but we only use those experiments to calibrate performance on commonly used datasets. Our case study on the web graph is only based on linking patterns among web hosts and domains. Our dense subhypergraph tools have the potential to be used to identify extremal sets, which – like many general-purpose mining tools – could be used maliciously to infer attributes that people would prefer to stay secret if the information was represented by a dense graph. However, we believe that dense subgraph analysis and subhypergraph analysis is a common algorithmic framework that has substantial non-malicious uses including novel studies of graph data and characterizing dense interconnections in biological networks.