The Densest SWAMP problem: subhypergraphs with arbitrary monotonic partial edge rewards

Vedangi Bengali[1] (✉), Nikolaj Tatti[2], Iiro Kumpulainen[2], Florian Adriaens[2], and Nate Veldt[1]

[1] Texas A&M University, College Station, TX, USA
`{vedangibengali, nveldt}@tamu.edu`
[2] University of Helsinki, HIIT, Helsinki, Finland
`{nikolaj.tatti, iiro.kumpulainen, florian.adriaens}@helsinki.fi`

Abstract. We consider a generalization of the densest subhypergraph problem where nonnegative rewards are given for including partial hyperedges in a dense subhypergraph. Prior work addressed this problem only in cases where reward functions are convex, in which case the problem is poly-time solvable. We consider a broader setting where rewards are monotonic but otherwise arbitrary. We first prove hardness results for a wide class of non-convex rewards, then design a $1/k$-approximation by projecting to the nearest set of convex rewards, where k is the maximum hyperedge size. We also design another $1/k$-approximation using a faster peeling algorithm, which (somewhat surprisingly) differs from the standard greedy peeling algorithm used to approximate other variants of the densest subgraph problem. Our results include an empirical analysis of our algorithm across several real-world hypergraphs.

Keywords: Densest subhypergraphs · Approximation algorithms

1 Introduction

Dense subgraph discovery is a widely studied primitive in graph mining, with applications including team formation [12, 22], motif discovery [11], and fraud detection [9]. One of the most common problems in dense subgraph discovery is the densest subgraph problem (DSG). For a graph $G = (V, E)$, DSG seeks a node set $S \subseteq V$ that maximizes the ratio between (induced) edges and nodes, i.e.,

$$\underset{S \subseteq V}{\text{maximize}} \quad \frac{|E(S)|}{|S|}, \tag{1}$$

where $E(S) = \{(u, v) \in E \colon u, v \in S\}$. There are known polynomial-time algorithms for exactly solving DSG using flow-based methods [13] and linear programming [6]. There is also a simple greedy 1/2-approximation based on *peeling* (iteratively removing a minimum degree node) [4, 6]. Many results for variants of DSG focus on extending one or more of these basic techniques (flow, linear programming, and peeling) to more general settings [18, 8, 25, 19, 24, 15, 16, 26].

This paper focuses on generalizations of DSG to hypergraphs, which group nodes into (hyper)edges involving an arbitrary number of nodes (rather than just two). As a motivating application that we will consider throughout the manuscript, finding dense regions of a hypergraph provides a particularly natural approach for team formation. There are already a number of methods for team formation based on dense subgraph discovery [12, 22, 21]. These operate under the assumption that a dense region of a social network represents a good team for a future task, since it encodes a group of people who have already collaborated extensively in the past. Using a hypergraph rather than a graph for this application is arguably more natural, since it directly captures entire previous *team* interactions as hyperedges, rather than only previous pairwise relationships.

The simplest generalization of DSG to hypergraphs is to find a node set S that maximizes the ratio between the number of edges *completely contained in* S and $|S|$. This was first considered 30 years ago in the context of circuit decomposition [15] and has been considered by many others since [14, 5]. While this is a natural extension of the standard DSG problem in graphs, there are many applications in which *partially* including a hyperedge also intuitively contributes to notions of density. Consider again the example of team formation. When forming a new team, adding even a *subset* of people from a previous collaboration intuitively matches the belief that previous collaborations help contribute to good future team interactions. However, the standard densest subhypergraph objective only gives a reward for including an entire hyperedge (i.e., all members of some previous team) to the new team.

In this paper, we focus on new algorithms for a generalized densest subhypergraph objective introduced by Zhou et al. [26], which incorporates positive rewards for partially included edges. In more detail, each edge has a nonnegative monotonic reward function r, and the overall objective is to maximize a sum of partial rewards divided by the size of the output set (Section 2 includes a formal definition). Zhou et al. [26] proved that there exist reward functions for which this general problem becomes **NP**-hard. However, they then focused on convex reward functions, in which case the problem is a special case of the poly-time solvable densest supermodular subset problem [7]. They designed an exact algorithm for this convex case based on solving maximum s-t flow problems, and showed that a standard greedy peeling algorithm yields a $1/k$-approximation where k is the maximum hyperedge size.

Our contributions. We focus on the problem of finding a densest Subhypergraph With Arbitrary Partial edge rewards (SWAMP). By arbitrary, we mean that rewards are not required to be convex, as was the focus for Zhou et al. [26]. Our contributions include the following:

– We completely settle the complexity of SWAMP for all reward functions, significantly strengthening prior hardness results. Concretely, when applying the same reward function r to every edge, SWAMP is either poly-time solvable because r is convex, trivially optimized by a single node if r is non-convex but satisfies a certain extremal condition, or is otherwise **NP**-hard.

- We design peeling algorithms for the non-convex case that come with a $1/k$-approximation. Somewhat surprisingly, this approximation is not obtained by the standard greedy peeling method (which gives a $1/k$ approximation for the convex case), but rather with a peeling method that makes locally suboptimal choices about which node to remove in each step.
- We design approximation algorithms based on projecting non-convex rewards to convex rewards and then solving the latter problem. In the worst case, these come with a $1/k$-approximation.
- We introduce a new integer linear programming formulation for finding optimal solutions to SWAMP, that works even for (small-scale) **NP**-hard cases.
- We implement these algorithms and demonstrate their performance on real-world datasets, showing they exceed their theoretical guarantees in practice.
- We show that as a consequence of having approximation guarantees for SWAMP, we obtain approximations for constrained variants of our objective.

Other related work. We refer to Lanciano et al. [19] for an extensive recent survey on variants of the densest subgraph problem. We briefly cover several results that are particularly relevant to our paper. Several variants of DSG include constraints on the number of nodes in the output set S, including two variants introduced by Andersen and Chellapilla [2] called the densest at-most-k ($|S| \leq k$), and the densest at-least-k ($|S| \geq k$) problems. In other problem variants, the goal is to find a dense subgraph that ensures nodes with different node labels are included in the output. In some cases, node labels represent "skills" and the goal is to form a (dense) team of individuals to cover certain skill sets [12, 22]. In other cases, node labels represent disjoint protected classes and the goal is to form a dense subgraph that is "diverse" or "fair" [1, 21, 17]. Approximation algorithms for several of these variants build upon earlier approximation algorithms for the densest at-least-k variant [12, 21].

Recently, Chekuri et al. [8] introduced the densest supermodular subset problem (DSS) where the goal is to maximize $F(S)/|S|$ where $F: 2^V \to \mathbb{R}^+$ is a non-negative monotone supermodular function defined over a ground set V. Their results for this objective include a fast flow-based approximation algorithm, faster greedy peeling methods, and extensions to cardinality-constrained variants (e.g., the constraint $|S| \geq k$). The greedy peeling method starts with the entire node set V, finds a node $v = \operatorname{argmin} F(V) - F(V - v)$, and then removes it. At the next step, the same strategy is applied to the remaining node set. At the end, the method chooses the subset of nodes considered along the way with the best objective. This strategy is greedy in the sense that it removes the node that leads to the smallest decrease in the numerator of the objective at each step. This peeling method directly generalizes existing peeling methods for several special cases [6, 25, 26], including the $1/k$-approximation for the generalized densest subhypergraph objective of Zhou et al. [26].

2 The Densest SWAMP Problem

We consider a generalized maximum density problem with a hypergraph input $H = (V, E, w, \{r_e \colon e \in E\})$. Each $e \in E$ comes with a non-negative scalar weight $w(e) \geq 0$ and an *edge-reward function* $r_e \colon \{0, 1, \ldots, |e|\} \to \mathbb{R}_{\geq 0}$. The latter assigns a non-negative reward based on the *number* of nodes in the hyperedge included in a node set S, even if e is not entirely contained in S. We assume the reward function is monotonic and gives no reward for including no nodes from the edge:

$$0 = r_e(0) \leq r_e(1) \leq r_e(2) \leq \cdots \leq r_e(|e|).$$

This encodes the belief that including more of a hyperedge within a set should contribute more to the measure of density.

Problem 1. For input $H = (V, E, w, \{r_e \colon e \in E\}$, the Densest Subhypergraph With Arbitrary Monotonic Partial edge rewards problem (SWAMP) seeks to solve

$$\max_{S \subseteq V} \quad \Gamma(S) = \frac{f(S)}{|S|}, \quad \text{where } f(S) = \sum_{e \in \mathcal{E}} w(e) \cdot r_e(|e \cap S|). \tag{2}$$

If the rewards satisfy $r_e(i) = 0$ for $i < |e|$ and $r_e(|e|) = 1$, then this corresponds to the standard densest subhypergraph objective. Zhou et al. [26] were the first to study Problem 1, though focused almost exclusively on the convex case. A discrete monotonic edge-reward function $r_e : \{0, 1, \ldots, |e|\} \to \mathbb{R}_{\geq 0}$ is said to be convex if:

$$r_e(i+1) - r_e(i) \geq r_e(i) - r_e(i-1) \quad \text{for } i \in \{1, 2, \ldots, |e| - 1\}. \tag{3}$$

We instead focus on the case where rewards are monotonic but otherwise arbitrary, i.e., not restricted to be convex.

In our study of hardness results, we consider a special case of SWAMP where the input is a hypergraph with maximum edge size k and we use the same reward function r for every edge. We refer to this as SWAMP(r). In this case, r represents a parameter defining the *problem* rather than a reward function that is inherently part of the input. Our goal is to characterize the complexity of SWAMP(r) under different choices for r.

We also consider two-sized constrained variants. The first is to maximize $\Gamma(S)$ subject to a cardinality constraint $|S| \geq \ell$ (where ℓ is an input to the problem), which we refer to as CARD-SWAMP. We also consider a fair variant.

Problem 2. Assume nodes of $H = (V, E, w, \{r_e\})$ are partitioned into node classes $\{C_1, C_2, \ldots, C_m\}$, and let a parameter ℓ_i be given for C_i for $i = 1, 2, \ldots, m$. The Densest Fair SWAMP problem (FAIR-SWAMP) is defined by

$$\underset{S \subseteq V}{\text{maximize}} \quad \Gamma(S) \quad \text{subject to } |S \cap C_i| \geq \ell_i, \text{ for } i = 1, 2, \ldots, m. \tag{4}$$

If we think of nodes as individuals, the classes in Problem 2 can represent *skills* that must be present when forming a team of individuals based on their past collaborations. As another example, classes could represent categories that must be fairly represented in the output set (e.g., selecting faculty members from all faculty ranks to represent an academic department on some internal committee).

3 Computational complexity of SWAMP(r)

The complexity of SWAMP(r) is known in the convex case.

Theorem 1 ([26]). *If r is convex, SWAMP(r) is polynomial-time solvable.*

In addition, Zhou et al. [26] showed that for *some* choice of reward functions (where some rewards are convex but others are not), Problem 1 is **NP**-hard. We strengthen this result by showing that for *every* non-convex r (minus a corner case, described below), SWAMP(r) is **NP**-hard. This provides us with a full picture of the computational complexity of SWAMP(r).

Let us first describe a corner case when the optimal solution is a single node. This generalizes a result of Zhou et al. [26] which says that if r is concave, a single node defines the optimal solution.

Theorem 2. *Assume r such that for every $i \in \{1, 2, \ldots, k\}$, it holds that $r(1) \geq r(i)/i$. Then there is an optimal solution for SWAMP(r) consisting of a single node.*

Proof. For instance $H = (V, E, w)$, let O be the solution for SWAMP(r). Let $o \in O$ be the node maximizing $f(\{o\})$. Then

$$\begin{aligned}\mathit{\Gamma}(O) = \frac{f(O)}{|O|} &= \frac{1}{|O|} \sum_{e \in E} w(e) \cdot r(|e \cap O|) \\ &\leq \frac{1}{|O|} \sum_{e \in E} w(e) \cdot r(1) \cdot |e \cap O| = \frac{1}{|O|} \sum_{x \in O} f(\{x\}) \leq f(\{o\}) = \mathit{\Gamma}(\{o\}),\end{aligned}$$

proving the claim. □

The main result of this section now states that if the conditions in Theorems 1–2 are not satisfied, then SWAMP(r) is **NP**-hard, see Appendix for proof.

Theorem 3. *Assume r such that for some positive integers i and j we have $r(i)/i > r(1)$ and $2r(j) > r(j-1) + r(j+1)$. Then SWAMP(r) is **NP**-hard.*

4 Solving SWAMP with an integer linear program

We introduce a new ILP formulation for solving the decision version of the problem, which asks whether there exists some set S with density greater than α for some pre-specified α. More formally, finding S with $\mathit{\Gamma}(S) > \alpha$ is equivalent to finding S for which $f(S) - \alpha|S| > 0$. We will solve the latter using an ILP, and once we have the solution, we can solve SWAMP by performing a binary search on α. This construction is similar to the approach by Goldberg [13] for finding the densest subgraph.

We consider the following integer linear program.

$$\begin{aligned}
\text{maximize} \quad & \left(\sum_{e\in E} w(e) \sum_{i=1}^{|e|} \delta_{e,i} \cdot y_{e,i}\right) - \alpha \sum_{v\in V} x_v \\
\text{s.t.} \quad & y_{e,i} \leq \frac{1}{i} \sum_{v\in e} x_v && \text{for } e \in E,\ i = 1, 2, \ldots, |e|, \qquad (5)\\
& y_{e,i} \in \{0,1\} && \text{for } e \in E,\ i = 1, 2, \ldots, |e|,\\
& x_v \in \{0,1\} && \text{for } v \in V.
\end{aligned}$$

In the above, we have defined

$$\delta_{e,i} = r_e(i) - r_e(i-1) \geq 0$$

so that if we include k nodes from edge e in the set S, we know that this gives a reward of

$$r_e(k) = \sum_{i=1}^{k} \delta_{e,i}.$$

This ILP includes a variable $x_v \in \{0,1\}$ that indicates whether node v is contained in the optimal density set S ($x_v = 1$) or not ($x_v = 0$). The variable $y_{e,i}$ is designed to satisfy

$$y_{e,i} = \begin{cases} 1 & \text{if } |e \cap S| \geq i \\ 0 & \text{otherwise.} \end{cases} \qquad (6)$$

Observe that if this is the case, then the first part of the objective function of the ILP is exactly the edge reward

$$\sum_{e\in E} w(e) \sum_{i=1}^{|e|} \delta_{e,i} \cdot y_{e,i} = \sum_{e\in E} w(e) \sum_{i=1}^{|e\cap S|} \delta_{e,i} = \sum_{e\in E} w(e) \cdot r_e(|e\cap S|).$$

We just need to confirm that for the optimal solution, the $y_{e,i}$ variables will indeed satisfy Eq. (6). Observe first of all that maximizing the objective, plus the fact that the $\delta_{e,i}$ are all positive, will ensure that the $y_{e,i}$ variables will be set to 1 whenever possible. Now, note that the constraint

$$y_{e,i} \leq \frac{1}{i} \sum_{v\in e} x_v$$

is equivalent to a bound $y_{e,i} < 1$ if $|e \cap S| < i$, and otherwise it amounts to a bound $y_{e,i} \leq c$ for some $c \geq 1$ if $|e \cap S| \geq i$.

5 Approximating SWAMP

Our most significant algorithmic contributions are peeling-based approximation algorithms for the **NP**-hard regime of SWAMP. Our findings are surprising for

Algorithm 1 Approximates the densest subgraph problem, SWAMP

```
Require: Hypergraph $\mathcal{H} = (V, E, r_e)$, and a bound function $s_e(\cdot)$.
1: $X \leftarrow V$ and $Y \leftarrow V$
2: while $X \neq \emptyset$ do
3:     $v \leftarrow \arg\min_{u \in X} \sum_{e : u \in e} r_e(|e \cap X|) - s_e(|e \cap X| - 1)$.
4:     $X \leftarrow X \setminus \{v\}$.
5:     if $\Gamma(X) \geq \Gamma(Y)$, then $Y \leftarrow X$.
Ensure: $Y$.
```

two reasons. First of all, our peeling approach for the **NP**-hard non-convex case has a $1/k$-approximation, which is just as good as the peeling approximation guarantee for the (poly-time solvable) convex case. The second surprise is that our approximation guarantees *do not* come from using the standard existing greedy peeling algorithm, which peels away a node in a way that leads to the best objective in the subsequent step. Rather, our guarantees work only for certain peeling strategies that may make locally suboptimal choices for which node to remove at each step. In addition to our peeling algorithms, we design another $1/k$-approximation based on projecting non-convex reward functions to convex rewards and then exactly solving the resulting convex problem.

5.1 Peeling algorithm

Here we describe a peeling algorithm, which will result in a $1/k$ approximation for SWAMP. For notational simplicity, we will assume that $w(e) = 1$. Note that we can make this assumption without the loss of generality since we can incorporate the weights directly to the rewards r_e.

The greedy peeling algorithm by Zhou et al. [26] operates by iteratively deleting a vertex v with the smallest decrease in f, that is,

$$f(S) - f(S \setminus \{v\}) = \sum_{e : v \in e} r_e(|e \cap S|) - r_e(|e \cap S| - 1), \tag{7}$$

and then returning the best tested subgraph.

Unfortunately, this approach yields a guarantee only when r_e is convex and can fail for non-convex rewards. We extend the approach by replacing the second r_e in Eq. 7 with a different function s_e, which needs to be specified separately. The pseudo-code for the algorithm is given in Algorithm 1. We will show that certain choices for s_e lead to a guarantee.

The following result shows the conditions required for s_e so that Algorithm 1 yields an approximation guarantee.

Theorem 4. *Assume a hypergraph $H = (V, E, \{r_e\})$ and a function s_e for each $e \in E$ satisfying $0 \leq s_e(i) \leq r_e(i)$ and $r_e(i) - s_e(i-1) \leq r_e(i+1) - s_e(i)$ Then Algorithm 1 yields a $1/k$ approximation for SWAMP.*

To prove the claim, we need the following standard lemma.

Lemma 1. *Let $O \subseteq V$ be an optimal solution. Then for any $o \in O$,*

$$\Gamma(O) \leq \sum_{e:o\in e} r_e(|e \cap O|) - r_e(|e \cap O| - 1).$$

Proof. By optimality of O, it holds

$$\Gamma(O) \geq \frac{f(O \setminus \{o\})}{|O| - 1} = \frac{f(O) - (f(O) - f(O \setminus \{o\})}{|O| - 1}.$$

Rewriting this inequality and using $\Gamma(O) = \frac{f(O)}{|O|}$ shows that $\Gamma(O) \leq f(O) - f(O \setminus \{o\})$. See also [26, Theorem 4]. Thus,

$$\begin{aligned}\Gamma(O) &\leq f(O) - f(O \setminus \{o\}) \\ &= \sum_{e\in E}(r_e(|e \cap O|) - r_e(|e \cap (O \setminus \{o\})|) \\ &= \sum_{e:o\in e}(r_e(|e \cap O|) - r_e(|e \cap (O \setminus \{o\})|),\end{aligned}$$

proving the lemma. □

Proof (Proof of Theorem 4). Let $O \subseteq V$ be an optimal solution. Let X' be the subgraph X defined by the while loop in Algorithm 1 when the first vertex from O is deleted. Call that vertex x. Note that since each edge e contains at most k vertices, it holds that

$$\begin{aligned}k\Gamma(X') &\geq \frac{1}{|X'|}\sum_{u\in X'}\sum_{e:u\in e} r_e(|e \cap X'|) \\ &\geq \sum_{e:x\in e} r_e(|e \cap X'|) \\ &\geq \sum_{e:x\in e} r_e(|e \cap X'|) - s_e(|e \cap X'| - 1),\end{aligned}$$

where the second inequality follows from our choice of x in Algorithm 1. Furthermore, as $O \subseteq X'$ by the imposed conditions on s_e it holds that

$$\begin{aligned}\sum_{e:x\in e} r_e(|e \cap X'|) - s_e(|e \cap X'| - 1) &\geq \sum_{e:x\in e} r_e(|e \cap O|) - s_e(|e \cap O| - 1) \\ &\geq \sum_{e:x\in e} r_e(|e \cap O|) - r_e(|e \cap O| - 1).\end{aligned}$$

Since $x \in O$, we can use Lemma 1, and the theorem follows. □

Next, we show two options for s, both satisfying the conditions in Theorem 4.

Theorem 5. *Assume a hypergraph $H = (V, E, \{r_e\})$. Let $b_e(i) = 0$ and $u_e(i) = r_e(i + 1) - \max_{0\leq j\leq i}(r_e(j + 1) - r_e(j))$. Then b_e and u_e satisfy the conditions for s_e in Theorem 4. Moreover, any s_e that satisfies the conditions in Theorem 4 will have $b_e(i) \leq s_e(i) \leq u_e(i)$. If r_e is convex, then $u_e = r_e$.*

Proof. The function b_e satisfies the constraints since r_e is monotonic.

Let us write $M_e(i) = \max_{0 \leq j \leq i} \left(r_e(j+1) - r_e(j)\right)$. Then

$$u_e(i) = r_e(i+1) - M_e(i) \leq r_e(i+1) - (r_e(i+1) - r_e(i)) = r_e(i)$$

and

$$u_e(i) - u_e(i-1) = r_e(i+1) - r_e(i) - (M_e(i) - M_e(i-1)) \leq r_e(i+1) - r_e(i),$$

showing that u_e satisfies the constraints.

To prove the second claim, assume that s_e satisfies the constraints and assume inductively that $s_e(i-1) \leq u_e(i-1)$. If $M_e(i-1) < M_e(i)$, then $M_e(i) = r_e(i+1) - r_e(i)$, and immediately $s_e(i) \leq r_e(i) = u_e(i)$, proving the claim. Otherwise, assume $M_e(i-1) = M_e(i)$. Then

$$\begin{aligned} s_e(i) &\leq s_e(i-1) + r_e(i+1) - r_e(i) \\ &\leq u_e(i-1) + r_e(i+1) - r_e(i) \\ &= r_e(i+1) - M_e(i-1) = r_e(i+1) - M_e(i) = u_e(i), \end{aligned}$$

proving the claim.

If r_e is convex, then $M_e(i) = r_e(i+1) - r_e(i)$ and $u_e(i) = r_e(i)$, proving the last claim. □

Runtime analysis. Finding the node v in Algorithm 1 can be done with a priority queue. Maintaining such structure requires $\mathcal{O}(k \cdot \deg(v))$ updates when removing a single vertex v, each taking $\mathcal{O}(\log n)$ time for a total running time of $\mathcal{O}(pk \log n)$, where $p = \sum_{e \in E} |e| = \sum_{v \in V} \deg(v)$.

5.2 Approximations based on projecting to convexity

Given a set of reward functions $\{r_e \colon e \in E\}$ and a corresponding objective $\max_{S \subseteq V} \Gamma(S)$, another approach to approximating SWAMP is to replace each r_e with a nearby convex function $\hat{r}_e$, and maximize a related objective $\hat{\Gamma}(S) = \frac{1}{|S|} \sum_{e \in E} w(e) \hat{r}_e(|S \cap e|)$. As an initial observation, these objectives differ by at most the maximum ratio between original (r_e) and projected ($\hat{r}_e$) rewards.

Proposition 1. *If $r_e(i) \geq \hat{r}_e(i)$ for every $e \in E$ and every $i \in [|e|] = \{1, \ldots, |e|\}$, then for every $S \subseteq V$ we have $\hat{\Gamma}(S) \leq \Gamma(S) \leq \rho \cdot \hat{\Gamma}(S)$, where*

$$\rho = \max_{e \in E} \max_{i \in [|e|]} \frac{r_e(i)}{\hat{r}_e(i)}.$$

Proof. The first inequality follows from the assumption that $r_e(i) \geq \hat{r}_e(i)$. The definition of ρ implies that for every $e \in E$ and $i \in [|e|]$, we have $r_e(i) \leq \rho \cdot \hat{r}_e(i)$, which yields the second inequality $\Gamma(S) \leq \rho \cdot \hat{\Gamma}(S)$.

This approximation is tight in the sense that we can always construct a hypergraph H with a node set S for which $\Gamma(S) = \rho \cdot \hat{\Gamma}(S)$. In more detail, consider a pair of rewards functions r and $\hat{r}$ and let $t = \text{argmax}_i \frac{r(i)}{\hat{r}(i)}$. Then construct a k-uniform hypergraph $H = (V, E)$ with a node set S satisfying $|e \cap S| \in \{0, t\}$ for every $e \in E$. Use the same reward function r for every edge when defining Γ, and the reward function $\hat{r}$ for every edge when defining r. Then

$$\Gamma(S) = \frac{\sum_{e \in E} r(|e \cap S|)}{|S|} = \frac{\sum_{e \in E} \rho \cdot \hat{r}(t)}{|S|} = \rho \cdot \hat{\Gamma}(S).$$

Given a non-convex nonnegative monotonic reward function $r \colon [0, k] \to \mathbb{R}^+$, our goal is then to find a *convex* nonnegative monotonic function $\hat{r} \leq r$ such that $\max_i r(i)/\hat{r}(i)$ is small. This can be cast as a small linear program.

$$\begin{array}{lll} \underset{\hat{r}}{\text{maximize}} & \kappa & \\ \text{such that} & r(i)\kappa \leq \hat{r}(i) \leq r(i) & i = 0, 1, 2, \dots, k \\ & 2\hat{r}(i) \leq \hat{r}(i-1) + \hat{r}(i+1) & i = 2, 3, \dots, k-1 \\ & \hat{r}(i+1) \geq \hat{r}(i) & i = 1, 2, \dots, k-1 \end{array}$$

The resulting approximation factor is given by $\rho = 1/\kappa$. We have dropped the e from the subscript of r_e and $\hat{r}_e$ above since we must solve this generic optimization problem for each edge reward function individually. This problem is equivalent to finding the lower convex hull of the points $\{(0, 0), (1, r(1)), \dots, (k, r(k))\}$, which can be done in $O(k)$ time [3]. Using the monotonicity of r, we can bound the worst-case approximation factor. See Appendix for a proof.

Proposition 2. *Let $r \colon [0, k] \to \mathbb{R}^+$ be a monotonic reward function satisfying $r(0) = 0$. There exists a nonnegative monotonic convex function $\hat{r}$ satisfying $\hat{r}(i) \leq r(i) \leq k \cdot \hat{r}(i)$ for every $i \in \{1, 2, \dots, k\}$.*

Observe that this approximation is tight. If r is defined by $r(0) = 0$ and $r(x) = 1$ for $x \in (0, k]$, then $\hat{r}(x) = x/k$ and $r(1)/\hat{r}(1) = k$. Propositions 1 and 2 tell us that after performing optimal projections, Γ and $\hat{\Gamma}$ differ by at most a factor $1/k$, which leads to the following result.

Theorem 6. *A β-approximate solution to $\max_{S \subseteq V} \hat{\Gamma}(S)$ yields a $\frac{\beta}{k}$-approximate solution for $\max_{S \subseteq V} \Gamma(S)$.*

Since $\hat{\Gamma}$ includes only convex edge rewards, we can optimally solve it using flow-based methods, yielding a $1/k$-approximation for the original objective Γ. To provide a runtime analysis, we assume all original rewards $r_e(i)$ are integers. After projecting, the new rewards $\hat{r}_e$ are not necessarily integers. However, new rewards can be expressed as rational numbers with denominators that range between 1 and k. For a simple runtime analysis, we can scale all new rewards by $k!$ to obtain new convex integer reward functions $r'_e = k! \cdot \hat{r}_e \leq k^k \cdot r_e$. The flow-based approach of Zhou et al. [26] for this integer convex rewards case relies on performing a binary search over the interval $[0, W]$ where $W =$

Algorithm 2 Approximates the CARD-SWAMP problem

```
Require: Hypergraph $H$, the cardinality constraint $\ell$.
1: $S \leftarrow \emptyset$, $H_1 \leftarrow H$, $i \leftarrow 1$
2: while $|S| < \ell$ do
3:     $S_i \leftarrow$ (approximate) densest SWAMP in $H_i$
4:     $H_{i+1} \leftarrow$ CONTRACT$(H_i, S_i)$
5:     $S \leftarrow S \cup S_i$
6:     $i \leftarrow i + 1$
7: $S' \leftarrow S_1 \cup \cdots \cup S_{i-2}$ padded with arbitrary nodes so that $S'$ has $\ell$ nodes
Ensure: either $S$ or $S'$, whichever has the higher density.
```

$\sum_{e \in E} r'_e(|e|) \leq k^k \sum_{e \in E} r_e(|e|)$. This has a runtime of $\mathcal{O}(\text{mincut}(p, p \cdot k) \log W)$ time where $p = \sum_{e \in E} |e|$ is the size of the hypergraph and $\text{mincut}(N, M)$ is the complexity of solving a minimum s-t cut problem in a graph with N nodes and M edges. To put this expression into a form that only involves the original rewards $\{r_e : e \in E\}$, observe that $\log W \leq k \log k + \log(\sum_{e \in E} r_e(|e|))$.

Note finally that if we project the non-convex rewards and apply the existing $1/k$-approximation greedy peeling algorithm for $\hat{\Gamma}$ [26], this is only guaranteed to produce a $1/k^2$-approximate solution using this analysis. This again highlights utility of our peeling algorithms that work directly on the non-convex objective and yield a $1/k$-approximate solution.

6 Approximation algorithms for constrained variants

The approximability of SWAMP has immediate implications for the approximability of CARD-SWAMP and FAIR-SWAMP. We will now explore these results.

Let us first consider the CARD-SWAMP problem, where the solution must have at least ℓ nodes. Here we will adopt the algorithm by Khuller and Saha [18] which was used to solve the constrained variant in regular graphs, and further extended to work with supermodular rewards by Chekuri et al. [8].

The algorithm, given in Algorithm 2, iteratively finds an approximate densest SWAMP, say S_i from the current hypergraph, say H_i, removes S_i from H_i (while keeping the edges), and adds S_i to the solution S, until S is large enough. Then the returned value is either S, or S', a set corresponding to S during the previous round, plus padded arbitrary nodes to satisfy the constraint.

Algorithm 2 requires a subroutine for contracting the discovered set from the current hypergraph. Given a hypergraph $H = (V, E, w, \{r_e\})$ and a set of nodes U we define a contracted hypergraph $H' = (V', E', w', \{r'_e\}) = \text{CONTRACT}(H, U)$ as follows. The nodes are $V' = V \setminus U$, the hyperedges E' consist of the hyperedges in E with nodes in U removed, and the weights w' correspond to the weights w. To define the rewards, let $e \in E$ be a hyperedge and $a = e \setminus U$ be the contracted hyperedge. Let $j = |e \cap U|$. We define the contracted reward as $r'_a(i) = r_e(i + j) - r_e(j)$.

We have the following approximation result, which we prove in Appendix.

Theorem 7. *Assume that we can α-approximate* SWAMP*, then Algorithm 2 yields $\frac{\alpha}{\alpha+1}$ approximation for* CARD-SWAMP*. Consequently, using Algorithm 1 together with Algorithm 2 yields $\frac{1}{k+1}$ approximation.*

We can now use Theorem 7 and the algorithm proposed by Miyauchi et al. [21] to obtain an approximation result for FAIR-SWAMP.

Theorem 8. *Assume an instance* FAIR-SWAMP *with $\{\ell_i\}$ constraints. Let S be the α-approximation for* CARD-SWAMP *with $\ell = \sum_i \ell_i$. Let c_i be the number of nodes with color i in S. Let S' be S padded with any $\ell_i - c_i$ nodes of color i, for every color i. Then S' yields $\alpha/2$-approximation for* FAIR-SWAMP*. Consequently, using Algorithms 1 and 2 yields $1/(2k+2)$ approximation.*

Note that originally this algorithm was designed for standard graphs (i.e., hypergraphs with $k = 2$). However, the proof for Theorem 8 is identical to the proof by Miyauchi et al. [21], and therefore omitted. We conjecture that a better approximation is possible using an approach for normal graphs by Gajewar and Das Sarma [12]. We leave exploring this direction as a future work.

7 Experiments and Analysis

We now analyze the performance of algorithms over a variety of hypergraphs using several different convex and non-convex reward functions. We implement all the algorithms in Julia and use publicly available hypergraph datasets. All experiments were conducted on a research server with 1TB of RAM.[3]

Datasets. The *contact-high-school* (CHS) [10, 20] and *contact-primary-school* (CPS) [10, 23] datasets represent student interactions at a high school and primary school, respectively, with students as nodes and group interactions as hyperedges. *Senate-committees* (SC) and *House-committees* (HC) contain labeled nodes representing US Senate and House members with political party affiliations [10]. Here hyperedges denote the committee memberships. In the *Trivago* hypergraph (Triv), nodes are vacation rentals and hyperedges are rentals clicked during the same user browsing session on `Trivago.com`. We specifically use a subset of a larger hypergraph [10], defined by considering only vacation rentals in Fukuoka, Japan. We preprocess each hypergraph by eliminating self-loops and dangling nodes, and extracting their largest connected component while preserving multi-edges. Hypergraph statistics are shown in Table 2. We choose hypergraphs that are small enough so that we can find optimal solutions using the ILP, as a point of comparison for our approximation algorithms.

Reward functions. We use a range of edge-reward functions from non-convex to convex. For simplicity, we assume that in a given hypergraph, all edges use the same function to compute r_e and that each edge has weight 1. The function definitions are presented in Table 1. To avoid trivial solutions (as noted in Theorem 2), we set $r_e(1) = 0$ for functions 1, 2, 3, and 6. This is especially important for 2-node hyperedges while using reward functions 1, 2, and 3.

[3] The code and Appendix are available at repository: The Densest SWAMP Problem.

Table 1: Edge-reward functions and their definitions

	Function	r_e		Function	r_e
1.	atleast-two	$r_e(i) = \mathbf{1}[\, i \geq 2\,]$	4.	standard	$r_e(i) = \mathbf{1}[\, i = \lvert e\rvert\,]$
2.	atleast-half	$r_e(i) = \mathbf{1}\big[\, i \geq \lceil \lvert e\rvert/2\rceil\,\big]$	5.	quadratic	$r_e(i) = i^2/\lvert e\rvert$
3.	all-but-one	$r_e(i) = \mathbf{1}[\, i \geq \lvert e\rvert - 1\,]$	6.	square-root	$r_e(i) = \sqrt{i}$

Algorithms. For finding the optimal solution, we implement the Exact method that iteratively solves the ILP (using Gurobi optimization software) as described in Section 4. Instead of using a binary search, we begin with the entire hypergraph and iteratively search for a denser and denser subset until no more improvement is possible, as this tends to converge in 4-5 iterations. We compare Exact against several approximation algorithms. Algorithm 1 is referred to as as PeelMax when we set $s_e = u_e$ as specified in Theorem 5, as Greedy when $s_e(i) = r_e(i)$, and as PeelZero when $s_e(i) = 0$. DegPeel is the greedy peeling algorithm for the standard densest subhypergraph objective: peeling based solely on the degree (number of incident hyperedges) in the induced hypergraph. For the projection-based approximations, we first project each non-convex r_e onto its nearest convex $\hat{r}_e$. We then run a flow-based method and Greedy methods using projected rewards. The projection-plus-flow approach solves the projected problem exactly using a maxflow solver, as described in [26]. It comes with a $1/k$-approximation guarantee, while the latter is a $1/k^2$ approximation.

Performance analysis. Table 2 shows runtimes and objective values for our methods. As a first observation, we see that our projection-based flow method is exceptionally fast. Even the peeling methods, although not optimized for runtime, are still orders of magnitude faster than exactly solving the objective using the ILP solver. For instance, in the CPS hypergraph with the objective using *atleast-two* reward function for every edge, the ILP approach requires approximately 29 minutes, whereas Greedy and ProjFlow find the densest solution in under a second. Furthermore, our approximation algorithms all produce approximation ratios in practice that are close to 1, showing that these methods far exceed their theoretical guarantees and produce nearly optimal solutions. This illustrates that even for cases where SWAMP is **NP**-hard, peeling methods can provide a fast and accurate approach in practice, comparable to the success of peeling methods for poly-time solvable variants. We note also that for these hypergraphs and reward functions, applying direct peeling methods tends to give a slightly better approximation than projecting to nearby convex rewards. As another interesting observation, the standard greedy peeling method (setting $s_e = r_e$) usually produces the best approximation results among peeling methods, despite the fact that our approximation guarantees do not apply to this approach.

Qualitative comparison. As a final point of comparison, Table 3 reports qualitative aspects of the dense subsets produced by exactly solving SWAMP (using Exact) with different reward functions on two hypergraphs (CFS and CPS). The

Table 2: Objective and runtime values of ILP, peeling and projection based strategies. Dashes indicate that Exact did not complete within the allotted time.

	CHS		**CPS**		**SC**		**HC**		**Triv**	
$\lvert V\rvert, \lvert E\rvert, k$	327, 7818, 5		242, 12704, 5		282, 315, 31		1290, 340, 81		262, 910, 16	
	Obj.	Run	Obj.	Run	Obj.	Run	Obj.	Run	Obj.	Run
atleast-two										
Exact	27.078	416	60.549	1760	–	–	–	–	11.53	2.4
PeelMax	26.871	0.15	60.02	0.42	21.0	0.89	14.0	24.01	11.09	0.06
Greedy	27.065	0.2	60.549	0.59	15.90	1.14	11.88	32.77	11.41	0.02
PeelZero	26.871	0.16	60.022	0.45	21.0	1.11	14.0	23.98	11.09	0.02
DegPeel	26.725	0.06	58.714	0.06	11.5	0.01	7.625	3.20	10.5	0.008
ProjFlow	26.451	0.16	58.34	0.21	12.22	0.07	7.7	0.43	10.76	0.02
ProjGreedy	26.451	0.2	58.34	0.58	12.22	0.86	7.625	25.57	10.76	0.02
atleast-half										
Exact	27.078	515	60.549	1711	–	–	–	–	9.625	2.33
PeelMax	26.871	0.16	60.02	0.43	3.0	0.89	1.571	23.81	9.179	0.02
Greedy	27.065	0.2	60.549	0.6	2.775	1.02	2.0	29.07	9.56	0.02
PeelZero	26.871	0.16	60.022	0.43	3.0	1.10	2.38	23.76	9.51	0.02
DegPeel	26.725	0.05	58.714	0.07	2.66	0.01	1.321	3.2	9.02	0.008
ProjFlow	26.451	0.18	58.34	0.21	2.32	0.07	2.28	0.39	9.378	0.03
ProjGreedy	26.451	0.19	58.344	0.59	2.32	0.86	2.23	25.36	9.378	0.02
all-but-one										
Exact	26.926	249.9	60.16	1716	1.83	6794	1.22	100.97	7.769	1.39
PeelMax	26.784	0.16	59.66	0.44	1.66	0.91	1.07	24.32	7.725	0.02
Greedy	26.867	0.2	60.16	0.6	1.8	0.95	1.105	25.62	7.763	0.02
PeelZero	26.784	0.16	59.66	0.44	1.66	1.12	1.07	24.29	7.725	0.02
DegPeel	26.593	0.05	58.47	0.08	1.66	0.01	0.936	3.23	7.375	0.008
ProjFlow	26.424	0.28	58.26	0.32	1.233	0.05	0.976	0.32	7.34	0.02
ProjGreedy	26.424	0.19	55.266	0.6	1.233	0.88	1.0	24.31	7.36	0.02
standard										
ExactFlow	25.597	0.24	54.475	0.33	1.176	0.13	0.823	0.5	5.52	0.10
Greedy	25.575	0.15	54.475	0.4	1.176	0.88	0.77	24.87	5.52	0.02
PeelZero	25.575	0.15	54.475	0.43	1.176	1.13	0.77	24.82	5.52	0.02
DegPeel	25.581	0.15	54.475	0.14	1.176	0.04	0.794	3.22	5.52	0.04
quadratic										
ExactFlow	71.45	0.44	145.47	0.68	26.91	0.25	12.52	1.33	31.10	0.08
Greedy	71.34	0.15	145.377	0.42	26.91	0.84	12.52	23.50	31.10	0.02
PeelZero	70.874	0.16	145.06	0.41	26.78	1.09	12.25	23.14	31.10	0.02
DegPeel	69.85	0.06	143.70	0.07	25.24	0.01	11.59	3.20	29.50	0.008
square-root										
Exact	41.34	495	91.7	4469	–	–	–	–	17.15	17.22
PeelMax	41.06	0.16	91.21	0.42	29.69	0.89	20.48	24.02	16.86	0.02
Greedy	41.14	0.2	91.72	0.58	29.26	0.87	18.38	24.99	17.14	0.02
PeelZero	41.05	0.15	90.83	0.42	29.69	1.58	17.67	23.23	16.27	0.02
DegPeel	41.06	0.06	89.92	0.06	19.64	0.01	11.94	3.28	16.16	0.008
ProjFlow	40.86	0.29	89.76	0.35	21.88	0.2	13.6	0.97	16.35	0.11
ProjGreedy	40.61	0.2	89.76	0.58	21.88	0.86	12.80	24.76	16.35	0.02

Table 3: Evaluating edge composition of optimal densest SWAMP solutions using five reward functions on two hypergraphs: CHS and CPS.

	`atleast-two`		`atleast-half`		`all-but-one`		|E(S)|		|S|	
	CHS	CPS	CHS	CPS	CHS	CPS	CHS	CPS	CHS	CPS
atleast-two	3791	7932	3791	7932	3762	7956	3053	6052	140	131
atleast-half	3791	7932	3791	7932	3762	7956	3053	6052	140	131
all-but-one	4055	8349	4055	8349	4039	8303	3404	6574	150	138
standard	6182	11341	6182	11341	6175	11337	6041	10895	236	200
quadratic	757	4206	757	4206	755	4152	651	3040	29	75

columns `atleast-two`, `atleast-half`, and `all-but-one` list the number of hyperedges (fully or partially contained in S) that intersect S in at least two nodes, at least $\lceil |e|/2 \rceil$ nodes, or at least $|e| - 1$ nodes, respectively. We also report the number of edges completely contained ($|E(S)|$) and the subset size ($|S|$). Each reward function leads to a dense subset with distinct characteristics: for instance, the standard objective tends to produce larger subgraphs whereas the quadratic objective tends to produce smaller subgraphs. This provides a simple check that finding the densest SWAMP using different reward functions does indeed allow us to capture meaningfully different types of density patterns in practice.

8 Conclusions and Discussion

We have presented comprehensive hardness results and new approximation algorithms for a dense subhypergraph objective where rewards are given for partially included edges. Our most significant finding is that peeling algorithms can achieve the same approximation guarantee for **NP**-hard variants of the problem as they do for poly-time variants that are special cases of the densest supermodular subset problem. This is somewhat surprising given that previous approximations for peeling seem to inherently rely on the supermodularity property. As one interesting observation, our theory does not apply to the standard greedy peeling strategy ($s_e = r_e$), but this strategy still seems to work well in practice. One open direction is to explore whether there are cases where the standard greedy strategy performs poorly, or whether we can prove an approximation for this approach using a different technique. Another direction for future work is to explore hardness of approximation results for **NP**-hard variants of SWAMP.

Acknowledgments. This research is supported by the Academy of Finland project MALSOME (343045) and by the Helsinki Institute for Information Technology (HIIT). The research is also supported by the Army Research Office (ARO) under Award Number W911NF-24-1-0156. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Anagnostopoulos, A., Becchetti, L., Fazzone, A., Menghini, C., Schwiegelshohn, C.: Spectral relaxations and fair densest subgraphs. In: CIKM, pp. 35–44 (2020)

[2] Andersen, R., Chellapilla, K.: Finding dense subgraphs with size bounds. In: WAW, pp. 25–37 (2009)

[3] Andrew, A.M.: Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters **9**(5), 216–219 (1979)

[4] Asahiro, Y., Iwama, K., Tamaki, H., Tokuyama, T.: Greedily finding a dense subgraph. Journal of Algorithms **34**(2), 203–221 (2000)

[5] Bera, S.K., Bhattacharya, S., Choudhari, J., Ghosh, P.: A new dynamic algorithm for densest subhypergraphs. In: TheWebConf, pp. 1093–1103 (2022)

[6] Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: International Workshop on Approximation Algorithms for Combinatorial Optimization, pp. 84–95, Springer (2000)

[7] Chekuri, C., Quanrud, K.: $(1-\epsilon)$-approximate fully dynamic densest subgraph: linear space and faster update time. arXiv:2210.02611 (2022)

[8] Chekuri, C., Quanrud, K., Torres, M.R.: Densest subgraph: Supermodularity, iterative peeling, and flow. In: SODA, pp. 1531–1555 (2022)

[9] Chen, T., Tsourakakis, C.: Antibenford subgraphs: Unsupervised anomaly detection in financial networks. In: KDD, pp. 2762–2770 (2022)

[10] Chodrow, P.S., Veldt, N., Benson, A.R.: Hypergraph clustering: from blockmodels to modularity. Science Advances (2021)

[11] Fratkin, E., Naughton, B.T., Brutlag, D.L., Batzoglou, S.: Motifcut: regulatory motifs finding with maximum density subgraphs. Bioinformatics **22**(14), e150–e157 (2006)

[12] Gajewar, A., Das Sarma, A.: Multi-skill collaborative teams based on densest subgraphs. In: SDM, pp. 165–176 (2012)

[13] Goldberg, A.V.: Finding a maximum density subgraph. Tech Report, UC Berkeley (1984)

[14] Hu, S., Wu, X., Chan, T.H.: Maintaining densest subsets efficiently in evolving hypergraphs. In: CIKM, pp. 929–938 (2017)

[15] Huang, D.H., Kahng, A.B.: When clusters meet partitions: new density-based methods for circuit decomposition. In: ED&TC, pp. 60–64 (1995)

[16] Huang, Y., Gleich, D.F., Veldt, N.: Densest subhypergraph: Negative supermodular functions and strongly localized methods. In: TheWebConf, pp. 881–892 (2024)

[17] Kariotakis, E., Sidiropoulos, N.D., Konar, A.: Fairness-aware dense subgraph discovery. TMLR (2025), ISSN 2835-8856

[18] Khuller, S., Saha, B.: On finding dense subgraphs. In: ICALP, pp. 597–608 (2009)

[19] Lanciano, T., Miyauchi, A., Fazzone, A., Bonchi, F.: A survey on the densest subgraph problem and its variants. ACM Computing Surveys **56**(8), 1–40 (2024)

[20] Mastrandrea, R., Fournet, J., Barrat, A.: Contact patterns in a high school: A comparison between data collected using wearable sensors, contact diaries and friendship surveys. PLOS ONE **10**(9), e0136497 (2015)

[21] Miyauchi, A., Chen, T., Sotiropoulos, K., Tsourakakis, C.E.: Densest diverse subgraphs: How to plan a successful cocktail party with diversity. In: KDD, pp. 1710–1721 (2023)

[22] Rangapuram, S.S., Bühler, T., Hein, M.: Towards realistic team formation in social networks based on densest subgraphs. In: WWW, pp. 1077–1088 (2013)

[23] Stehlé, J., Voirin, N., Barrat, A., Cattuto, C., Isella, L., Pinton, J.F., Quaggiotto, M., den Broeck, W.V., Régis, C., Lina, B., Vanhems, P.: High-resolution measurements of face-to-face contact patterns in a primary school. PLoS ONE **6**(8), e23176 (2011)

[24] Tsourakakis, C.: The k-clique densest subgraph problem. In: Proceedings of the 24th international conference on world wide web, pp. 1122–1132 (2015)

[25] Veldt, N., Benson, A.R., Kleinberg, J.: The generalized mean densest subgraph problem. In: KDD, pp. 1604–1614 (2021)

[26] Zhou, Y., Hu, S., Sheng, Z.: Extracting densest sub-hypergraph with convex edge-weight functions. In: International Conference on Theory and Applications of Models of Computation, pp. 305–321, Springer (2022)