Finding Subgraphs with Maximum Total Density and Limited Overlap in Weighted Hypergraphs

Oana Balalau, Francesco Bonchi, T-H. Hubert Chan, Francesco Gullo, Mauro Sozio, Hao Xie

Finding Subgraphs with Maximum Total Density and Limited Overlap in Weighted Hypergraphs*

OANA BALALAU, Inria and Institut Polytechnique de Paris, France
FRANCESCO BONCHI, Centai, Turin, Italy
T-H. HUBERT CHAN, Dept. of Computer Science, The University of Hong Kong
FRANCESCO GULLO, UniCredit, Rome, Italy
MAURO SOZIO, Institut Polytechnique de Paris, Telecom Paris, France
HAO XIE, Dept. of Computer Science, The University of Hong Kong

Finding dense subgraphs in large (hyper)graphs is a key primitive in a variety of real-world application domains, encompassing social network analytics, event detection, biology, and finance. In most such applications, one typically aims at finding several (possibly overlapping) dense subgraphs which might correspond to communities in social networks or interesting events. While a large amount of work is devoted to finding a single densest subgraph, perhaps surprisingly, the problem of finding several dense subgraphs in weighted hypergraphs with limited overlap has not been studied in a principled way, to the best of our knowledge. In this work we define and study a natural generalization of the densest subgraph problem in weighted hypergraphs, where the main goal is to find at most k subgraphs with maximum total aggregate density, while satisfying an upper bound on the pairwise weighted Jaccard coefficient, i.e., the ratio of weights of intersection divided by weights of union on two nodes sets of the subgraphs. After showing that such a problem is NP-Hard, we devise an efficient algorithm that comes with provable guarantees in some cases of interest, as well as, an efficient practical heuristic. Our extensive evaluation on large real-world hypergraphs confirms the efficiency and effectiveness of our algorithms.

CCS Concepts: • **Human-centered computing → Social network analysis**; • **Mathematics of computing → Graph algorithms**;

Additional Key Words and Phrases: densest subgraphs; weighted hypergraphs

1 INTRODUCTION

Finding dense subgraphs in large graphs has emerged as a key primitive in a variety of real-world application domains [31, 33], ranging from biology [22, 32] to finance [20]. In the Web domain, Gibson *et al.* [24] have observed that dense subgraphs might correspond to thematic group of pages or spam link farms. In the context of social networks, finding dense subgraphs has been employed for organizing social events and community detection [38], as well as for expert team formation [11, 43]. Angel *et al.* [2] have shown how finding dense subgraphs in the entity co-occurrence graph constructed from micro-blogging streams can be used to automatically detect important events.

Many of the aforementioned applications ask for finding several (possibly overlapping) dense subgraphs, which might correspond to communities in social networks or important events. The work in [6] has been among the first works to study the problem of finding multiple densest subgraphs in a principled way. In our work, we extend theoretical results to weighted hypergraphs, which enable the representation of relationships involving more than two nodes. This is particularly

*This is an extended version of the WSDM 2015 conference paper [6]. While the original version focuses only on unweighted normal graphs, we have generalized the approaches to weighted hypergraphs, especially BasicLP, and Algorithms 1, 4 and 6. Moreover, we have performed additional experiments for hypergraphs.

Authors' addresses: Oana Balalau, Inria and Institut Polytechnique de Paris, France; Francesco Bonchi, Centai, Turin, Italy; T-H. Hubert Chan, Dept. of Computer Science, The University of Hong Kong; Francesco Gullo, UniCredit, Rome, Italy; Mauro Sozio, Institut Polytechnique de Paris, Telecom Paris, France; Hao Xie, Dept. of Computer Science, The University of Hong Kong.

useful in situations where dependencies cannot be adequately represented by binary relationships in standard graphs. For instance, hyperedges can model overlapping communities in social networks.

In a first attempt to give a formal definition for such a problem, one could aim at finding at most k subgraphs with maximum aggregate total density on hypergraphs. However, it turns out that such a formulation might lead to find several subgraphs being very similar between each other and in particular sharing a large fraction of nodes of a relatively dense subgraph. Such a solution is not really interesting as the dense subgraphs to be found should ideally exhibit some appreciable degree of diversity among each other. Therefore, we enforce an upper bound on the pairwise weighted Jaccard coefficient between the sets of nodes of the subgraphs, where $J_w(A, B) := \frac{w(A \cap B)}{w(A \cup B)}$.

Several definitions of density have been studied in the literature, among which the *average weighted degree* density stands outs as a natural and widely used definition. Subgraphs with maximum average weighted degree density are usually referred to as *densest subgraphs*. One appealing feature of such a definition is that densest subgraphs can be found in polynomial time using the linear programming (LP) algorithm presented in [26] or the maximum flow algorithm in [25] for normal graphs, while there are efficient algorithms which come with provable approximation guarantees [26]. In our work we focus on the average weighted degree density.

A natural heuristic for our main problem is the following one: Greedily find one densest subgraph in the current graph, remove all its vertices and edges, and iterate until k subgraphs are found or the current graph is empty (see e.g., [43]). This heuristic, although reasonable, might potentially deliver arbitrarily bad solutions in terms of our objective function, as we show in the remainder. Another observation is that such an approach would produce subgraphs which are pairwise disjoint. By allowing some limited amount of overlap, one could find more interesting (i.e., denser) solutions, while maintaining enough diversity among the subgraphs extracted. Another drawback of the previous heuristic is that algorithms for finding densest subgraphs (based on linear programming and maximum flow) cannot cope with large graphs containing millions of edges.

In our work we present an efficient algorithm for our problem which comes with provable guarantees in some cases of interest. We introduce the concept of minimality of a densest subgraph, (roughly speaking a densest subgraph is minimal if it does not contain any other densest subgraph) and develop efficient algorithms for finding minimal densest subgraphs, which will be pivotal in solving our main problem. Our algorithm for finding minimal densest subgraphs can handle large graphs containing up to 10 million edges, as shown by our experimental evaluation. We also devise an efficient heuristic, so as to find subgraphs with limited overlap on even larger input graphs. More in detail, the contributions of this paper are summarized as follows:

- We define the (k, α)-Dense Subhypergraph with Limited Overlap problem ((k, α)-DSLO): given an integer $k > 0$ as well as a real number $\alpha \in [0, 1]$, find at most k subhypergraphs that maximize the total aggregate density, i.e., the weighted degree of each subgraph, under the constraint that the the maximum pairwise weighted Jaccard coefficient between the set of nodes in the subgraphs be at most α. We prove that (k, α)–DSLO is NP-hard even when $\alpha = 0$ (disjoint subgraphs).
- We define and study the problem of finding minimal densest hypergraphs. We develop an LP-based algorithm for such a problem that requires a logarithmic number of calls to an LP solver, with high probability. This allows us to deal with large real-world graphs.
- We devise several algorithms for the (k, α)-DSLO problem. We prove that, in the case when the input graph contains k disjoint densest subgraphs, one of our algorithms is guaranteed to find an optimum solution for (k, α)-DSLO. In the general case, we show empirically that our algorithms can find solutions that are very close to an optimum solution of our problem.

- We conduct an extensive experimental evaluation on real-world graphs and hypergraphs evaluating our algorithms in terms of our main objective function. We also investigate the uniqueness of dense subgraphs and whether subgraphs with the same density are isomorphic in real-world graphs.

In Section 2 we discuss the related work, while in Section 3 we define our main problem formally and prove its NP-hardness. All our algorithms are presented in Section 4, while Section 5 contains an experimental evaluation on large real-world graphs. Finally, in Section 6 we draw our conclusions and discuss interesting directions for future work.

2 RELATED WORK

Finding a single densest subgraph. The problem of finding a dense subgraph from a large input graph has been widely studied [33]. Generally speaking, such a problem aims at finding a subgraph of a given input graph that maximizes some notion of density. A notion of density widely employed in the literature is the average degree. Due to its popularity, the corresponding problem of finding a subgraph that maximizes the average degree has been commonly referred to as the *densest-subgraph* problem. The densest subgraph can be identified in polynomial time by solving a parametric maximum-flow problem [25]. Charikar [13] introduces a linear-programming formulation of the problem, while also showing that the greedy algorithm proposed by Asashiro *et al.* [4] produces a $\frac{1}{2}$-approximation in linear time. Hu *et al.* [26] propose a similar algorithm that achieves $\frac{1}{r}$-approximation for hypergraph, where $r = \max_{e \in E} |e|$. Boob *et al.* [12] propose a iterative peeling algorithm which generalises the Charikar's algorithm, and Chekuri *et al.* [14] prove that a generalisation of their algorithm converges to a $(1 - \varepsilon)$-approximation. The densest-subgraph problem has also been studied in streaming [5] and in a dynamic environment [8, 10, 21].

A more difficult variant of the densest-subgraph problem is the so-called *DkS* problem, which consists of finding a densest subgraph of k vertices. Such a problem is known to be NP-hard [3], while an algorithm with approximation guarantee of $O(n^{\frac{1}{4}})$ has been presented in [9]. Chlamtác *et al.* [16] propose a $O(n^{4(4-\sqrt{3})/13+\epsilon})$-approximation algorithm for 3-uniform hypergraphs for every constant $\epsilon > 0$. It is also well known that there cannot be any PTAS for the *DkS* problem under reasonable complexity assumptions [28]. Some variants of the *DkS* problem are introduced by Andersen and Chellapilla [1] and further investigated in [29].

A number of works depart from the classic average-degree maximization problem and focus on extracting a subgraph maximizing other notions of density. Tsourakakis *et al.* [43] focus on the study of quasi-cliques, while Wang *et al.* [47] focus on a density based on triangle counting. Sozio *et al.* [38] focus on minimum degree density while enforcing so-called *monotone* constraints. In the k-clique densest subgraph problem [39, 42] the density of a subgraph is defined to be as the ratio between its number of k-cliques and its number of nodes.

Finding multiple densest subgraphs. Unlike its single-subgraph counterpart, the problem of finding a set of k dense subgraphs has received considerably less attention. Few authors [43, 46] have discussed it, without providing any rigorous formulation of the problem. Instead they consider the most obvious heuristic that iteratively finds and removes the densest subgraph until k subgraphs have been found. In our work, we precisely formulate and characterize the problem of finding at most k disjoint subgraphs that maximize the sum of densities, while also showing, both theoretically and empirically, that the aforementioned simple heuristic is not well-suited for such a problem. To the best of our knowledge, this was not known before the work in [6].

A variant on finding k overlapping densest subgraphs focus on maximizing the sum of density and the distance between subgraphs, i.e. maximizing $\sum \rho(G(S_i)) + \lambda \sum_{i<j} d(G(S_i), G(S_j))$, where ρ refers to density function, λ is a non-negative parameter, and $d : 2^{G(V)} \times 2^{G(V)} \rightarrow \mathbb{R}_{\geq 0}$ is the

distance function defined by $d(G(U), G(Z)) = 2 - \frac{|U \cap Z|^2}{|U||Z|}$ when $U \neq Z$ and 0 otherwise [19, 23]. This is different to our work, not only due to the measure of overlapness but the way we treat overlapness. In this variant, overlapness is measured by the proposed distance function and our work use Jaccard Similarity. Further, it treats overlapness as a cost function in objective to be maximized, but in our problem, overlapness is seen as a constraint. Some variants focus on finding k disjoint densest subgraphs, see [36].

Apart from that, existing research has considered tangentially related problems, such as finding *nested* subgraphs containing a set of query nodes and exhibiting non-increasing densities [40], decomposing the graphs according to their densities [18, 41], discovering overlapping dense subgraphs containing a query node [17], extracting all *large-enough* dense bipartite subgraphs in massive graphs [24], or partitioning a hypergraph subject to maximizing the conditional core subgraphs [34]. Furthermore, Angel *et al.* [2] focus on maintaining the set of all (possibly overlapping) subgraphs exceeding a density threshold under streaming edge weight updates. Chen and Saad [15] instead propose a matrix-blocking model to identify dense subgraphs that best cover the input graph. Particularly, the latter problem differs from ours as it aims at finding a set of dense subgraphs that cover most of the input graph, while discarding outlier (i.e., non-dense) graph zones, but it does not attempt at maximizing the sum of the densities of k subgraphs.

A survey on the densest subgraph subgraph problem and its variants can be found in [31].

3 DEFINITION AND COMPLEXITY

In this section, we define our problem formally and we study its computational complexity.

Given a weighted hypergraph $G = (V, E, w_V, w_E)$, where $E \subseteq 2^V$, $w_V : V \to \mathbb{R}_{>0}$ is the weight function of vertice and $w_E : E \to \mathbb{R}_{>0}$ is the weight function of edge. We define $w_V(S) = \sum_{u \in S} w_V(u)$ for any $S \subseteq V$ and $w_E(T) = \sum_{e \in T} w_E(e)$ for any $T \subseteq E$. When there is no ambiguity, we drop down the subscripts and use w for simplicity. For the same reason, we also use $G = (V, E)$ to denote $G = (V, E, w_V, w_E)$. Given such a graph G we define its density $\rho(G)$ to be $\frac{w(E)}{w(V)}$. For a set of vertices $S \subseteq V$, we denote the subgraph of G induced by S as $G[S] = (S, E(S))$, where $E(S) = E \cap 2^S$.

In a first attempt to give a formal definition for our problem, one could aim at finding at most k subgraphs with maximum aggregate total density. However, it turns out that such a formulation might lead to find several subgraphs being very similar between each other and in particular sharing a large fraction of nodes of a relatively dense subgraph. Such a solution is not really interesting as the dense subgraphs to be found should ideally exhibit some appreciable degree of diversity among each other. Therefore, we enforce an upper bound $\alpha \in [0, 1]$ on the pairwise weighted Jaccard coefficient between the sets of nodes of the subgraphs, with one indicating that the two subgraphs contain exactly the same nodes and zero indicating that they are disjoint. Our problem can then be formalized as follows.

Definition 3.1 ((k, α)-DSLO). Given a weighted hypergraph $G = (V, E, w_V, w_E)$, an integer $k > 0$, as well as a rational number $\alpha \in [0, 1]$, find a set of sets of vertices $\mathcal{S} = \{S_1, \ldots, S_{\bar{k}}\}$ with $\bar{k} \leq k$, and $S_i \subseteq V, \forall S_i \in \mathcal{S}$, such that

$$\sum_{i=1}^{\bar{k}} \rho(G(S_i))) \quad \text{is maximum, and}$$

$$J_w(S_i, S_j) := \frac{w(S_i \cap S_j)}{w(S_i \cup S_j)} \leq \alpha \quad \forall S_i, S_j \in \mathcal{S}. \tag{1}$$

3.1 Hardness of the Problem

Proof Intuition. We show that (k, α)-DSLO is NP-hard even when $\alpha = 0$ (subgraphs are disjoint), by a reduction from degree bounded maximum independent set problem. The high level idea is that given an input graph, in which one wishes to find an independent set, one creates a gadget graph for each input node such that for neighboring nodes, the corresponding gadget graphs intersect. Loosely speaking, we show that a subgraph with high density must contain an intact gadget graph. Hence, if there are k disjoint subsets having high densities, one can find k independent nodes in the input graph.

Theorem 3.2. *(k, α)-DSLO is NP-hard.*

More Detailed Explanation. We prove NP-hardness by reducing the well-known maximum independent set problem to a special case of (k, α)-DSLO i.e., to case where $\alpha = 0$, G is a constant-weight undirected normal graph which means $E \subseteq \binom{V}{2}$ and $w_u(u) = w_e(e) = 1$ for all $u \in V, e \in E$. Specifically, $\rho(G) = \frac{|E|}{|V|}$. Until the end of this section, the graphs we refer to are constant-weight undirected normal graph by default. We will show that given such a graph G and a positive integer k, it is NP-hard to find k disjoint subsets $S_1, S_2, \ldots, S_k$ of nodes such that the sum of densities $\sum_{i=1}^{k} \rho(S_i)$ is maximized. In particular, we consider a decisional version of the problem, in which we are given a target τ, and the problem is to decide if there are k disjoint subsets whose sum of densities is at least τ.

Our hardness proof reduces from the NP-hardness of maximum independent set on degree bounded graphs [37]. In particular, for any fixed $\Delta \geq 3$, given a graph G with maximum degree at most Δ and an integer k, it is NP-hard to decide if G contains an independent set with size k.

Reduction Construction. Given an instance $G = (V, E)$ of the maximum independent set problem with maximum degree at most Δ, we construct a graph $\widehat{G}$ and select a threshold τ such that G has an independent set of size k *iff* $\widehat{G}$ contains k disjoint subsets whose sum of densities is at least τ.

Node Gadget. We choose some $N = n^4$, where $n = |V|$. Given a node u, we create a gadget graph G_u as follows. Let C_u be a set of N independent nodes, which forms the *core* of G_u. Let A_u be a set Δ independent nodes, which are the *arms* of G_u. The graph G_u is a complete bipartite graph between C_u and A_u, and so contains $N\Delta$ edges.

Interaction between Node Gadgets. For distinct nodes u and v in V, the cores C_u and C_v are disjoint. If $\{u, v\} \notin E$, then A_u and A_v are disjoint; but if $\{u, v\} \in E$, then $|A_u \cap A_v| = 1$, i.e., the two gadgets G_u and G_v share exactly one arm. Moreover, each arm node can be shared by at most 2 gadget graphs. Since G has degree at most Δ, each arm of a gadget graph can be potentially used to connect with another gadget according to G. This completes the description of graph $\widehat{G}$.

Before we state our threshold τ, we consider the densities of subgraphs in $\widehat{G}$.

Lemma 3.3. *The density of any subset of nodes in $\widehat{G}$ is at most Δ. Moreover, if the density of a subset S is at least $\Delta - \frac{1}{2}$, then S must contain all the arms A_u of some u.*

Proof. The result follows from the following statement. Suppose S is a subset of nodes in $\widehat{G}$ that intersects the cores of some r gadgets. Suppose for some $D > 0$, for each $i \in [r]$, S contains d_i arm nodes of gadget graph G_i, where $d_i \leq D$. (Observe that each arm node can belong to more than one gadget graph G_i.) Then, we show that the density of S is at most D.

For $i \in [r]$, suppose S contains n_i nodes from the core C_i of gadget graph G_i. The total number of edges induced by S is $\sum_{i \in [r]} n_i d_i$. If none of the G_i's share an arm, the number of nodes in S is $\sum_{i \in [r]} (n_i + d_i)$. Observe that for each pair of G_i's that share an arm, the number of nodes decreases by 1. Since each G_i can share arms with at most d_i other gadgets, the maximum number of pairs that can share an arm is $\frac{\sum_{i \in [r]} d_i}{2}$.

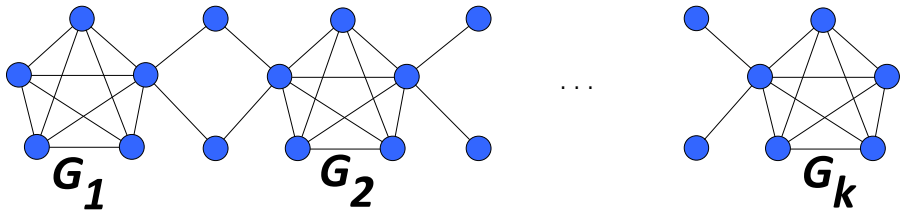

Fig. 1. A constant-weight undirected normal graph G where Naive gives poor results. Notice that G is a non-minimal densest subgraph.

Hence, $|S| \geq \sum_{i\in[r]}(n_i + d_i) - \frac{\sum_{i\in[r]} d_i}{2} = \sum_{i\in[r]}(n_i + \frac{d_i}{2})$.
Therefore, the density of S is at most
$\frac{\sum_{i\in[r]} n_i d_i}{\sum_{i\in[r]}(n_i+\frac{d_i}{2})} \leq \max_{i\in[r]} \frac{n_i d_i}{n_i+\frac{d_i}{2}} \leq \frac{ND}{N+\frac{D}{2}}$, where the last inequality follows because the expression $\frac{n_i d_i}{n_i+\frac{d_i}{2}}$ is increasing in both n_i and d_i.
Finally, observing that $\frac{ND}{N+\frac{D}{2}} \leq D$, we finish the proof of the statement. □

The following lemma completes the reduction proof.

Lemma 3.4. *The graph G (with maximum degree at most Δ) contains an independent set of size k* iff *the graph $\widehat{G}$ contains k disjoint subsets whose sum of densities is at least $\tau = k\Delta - \frac{1}{n}$.*

Proof. The forward direction is easy because each point in the independent set corresponds to a disjoint gadget graph, which has density $\frac{\Delta N}{N+\Delta} \geq \Delta - \frac{1}{n^2}$, because $\Delta < n$ and $N = n^4$. Hence, the sum of the densities of the k disjoint gadget graphs is at least $k\Delta - \frac{1}{n}$.

For the backward direction, observe that each of the k disjoint subsets S_i's in $\widehat{G}$ must have density at least $\Delta - \frac{1}{2}$. Otherwise, there exists $k-1$ disjoint subsets whose sum of densities is at least $(k-1)\Delta + \frac{1}{2} - \frac{1}{n} > (k-1)\Delta$. This implies that there exists a subset in $\widehat{G}$ with density strictly larger than Δ, which is impossible by Lemma 3.3.

Hence, we conclude that each of k disjoint subsets S_i's must have density at least $\Delta - \frac{1}{2}$, which implies by Lemma 3.3 that each S_i must contain all the arms of some core A_u for some $u \in V$. This means the k subsets S_i's correspond to k independent nodes in G. □

4 ALGORITHMS

As (k, α)-DSLO is NP-Hard, we devise heuristics that work well in practice while also exhibiting provable guarantees in some cases of interest. We start by considering one natural heuristic for the disjoint case ($\alpha = 0$): at each step, we compute the densest subgraph in the current hypergraph (using for example the approach in [26]), we remove all its nodes and edges from the current hypergraph, iterating until we find exactly k subgraphs or until the current hypergraph contains no edges. We hereinafter refer to this heuristic as Naive.

This simple heuristic gives unfortunately very poor results in the worst case even when the graph is constant-weight undirected normal graph, as illustrated in Figure 1.

In that example, each subgraph G_i is a complete graph with 5 nodes, and each G_i is joined with G_{i+1} by 2 independent nodes as shown in Figure 1. The density $\rho(G)$ of the graph is 2 (there are $10k + 4(k-1)$ edges and $5k + 2(k-1)$ nodes), like every subgraph G_i of G. Naive would find the whole graph G and then would stop giving a solution with total density equal to 2, while an optimum solution to our problem is composed of the disjoint subgraphs $G_1, \ldots, G_k$ with total

density equal to $2k$. This highlights one of the limitations of Naive and paves the way to the definition of minimal dense hypergraphs.

Definition 4.1. (Minimal dense hypergraphs) A weighted hypergraph G with density $\rho(G)$ is a *minimal dense hypergraph* if for any proper subgraph H of G, $\rho(H) < \rho(G)$. Moreover, we say that G is a minimal densest subgraph if it is minimal and has maximum density.

Notice that the constant-weight normal graph G in Figure 1 is a non-minimal densest subgraph.

Finding minimal densest hypergraphs plays an important role in solving our problem, as illustrated by the following variant of Naive. At each step we compute a *minimal* densest subgraph, we remove its nodes and edges from the current hypergraph, and we iterate until k subgraphs are found or there are no edges left in the current hypergraph. Including at each step a minimal densest subgraph H, rather than a supergraph G of H, would not decrease the total density of our solution (as H is as dense as G) and might actually increase it, as fewer edges are removed from the current hypergraph after including H in the solution. It turns out that this simple variant of Naive, finds k disjoint densest subgraphs, if they exist. This is proved in Section 4.2. In Section 4.1, we show how to efficiently compute minimal densest subgraphs.

Drawing inspiration from the techniques developed for computing minimal densest subgraphs, we then address the (k, α)-DSLO problem. The main challenge here is to take full advantage of the overlap between subgraphs so to maximize our objective function. Computing minimal subgraphs is desirable also in this case, in order to minimize the number of heavy-weighted edges that are removed at each step. This is discussed in Section 4.2. In Section 5, we perform an extensive evaluation of our algorithms showing the effectiveness of our heuristics for (k, α)-DSLO on large real-world hypergraphs and that minimal densest subgraphs can be computed efficiently on large hypergraphs containing millions of edges.

4.1 Finding Minimal Densest Subgraphs

Our algorithm for computing minimal densest subgraphs is inspired by the linear programming (LP)-based algorithm for the densest subgraph developed by Hu et al. [26]. In order to have some provable guarantees for our problem, we need to dive deeper into the structure of the solutions of the LP. We start by recalling the algorithm in [26].

Given a weighted hypergraph $G = (V, E, w_V, w_E)$, Hu et al. [26] proposed the following LP formulation for the densest subgraph problem. For each edge $e \in E$ we introduce a variable x_e, while for each node v we introduce variable y_v. We have the following linear program:

$$\max \quad \sum_{e \in E} w(e) x_e \tag{BasicLP}$$

$$\text{s.t.} \quad x_e \leq y_u \qquad \forall u \in e \in E \tag{2}$$

$$\sum_{u \in V} w(u) y_u \leq 1 \tag{3}$$

$$x_e, y_u \geq 0 \qquad \forall e \in E, u \in V. \tag{4}$$

Intuition of BasicLP. For $e \in E$, the variable x_e is positive means that edge e is included in the densest subgraph and for $u \in V$, the variable y_u is positive means that u is included in the densest subgraph. However, since the objective function should be related to the density of the chosen subgraph, these are not intended to be 0-1 variables. Instead, the positive values taken by these variables are related to the weight of the chosen subset of nodes.

We further illustrate the roles of these variables by considering some subgraph $H = (S, E(S))$, $S \subseteq V$. We can then define a feasible solution $z^S = (x^S, y^S)$ for BasicLP as follows:

$$x_e^S = \begin{cases} \frac{1}{w(S)} & \text{if } e \in E(S) \\ 0 & \text{otherwise.} \end{cases}$$

$$y_u^S = \begin{cases} \frac{1}{w(S)} & \text{if } u \in S \\ 0 & \text{otherwise.} \end{cases}$$

Recall that the density of the subgraph $H = (S, E(S))$ is defined as $\rho(H) = \frac{w(E(S))}{w(S)}$. Observe that z^S is feasible for the basic LP, and has objective value $\rho(H)$. Vice versa, given an optimum solution $z^* = (x^*, y^*)$ for BasicLP, Lemma 4.2 below shows how one can construct a densest subgraph. Specifically, one can solve BasicLP using efficient LP solvers such as Gurobi or CPLEX and then obtain the densest subgraph by applying the following result.

Lemma 4.2. *Each optimal solution of BasicLP is a convex combination of points in* $\{z^S : S \subseteq V,\ S \text{ is a densest subgraph}\}$.

Proof. Suppose $z^* = (x^*, y^*)$ is an optimal solution. Then, since the objective value to be maximized is the sum of all $w(e)x_e$'s, it follows that if z^* is optimal, it must be the case that $\sum_{u \in V} w(u) y_u^* = 1$, and for all $e \in E$, $x_e^* = \min_{u \in e} y_u^*$.

We prove the result by induction on the number k of non-zero coordinates of y^*. If E contains at least one edge, it follows that $k \geq \min_{e \in E} |e|$. For the base case $k = \min_{e \in E} |e|$, since $x_e^* = \min_{u \in e} y_u^*$ is maximized when $y_u^* = \frac{1}{\sum_{u \in E} w(u)}$ for all $u \in e$, the result follows.

For the inductive step, suppose y^* has $k > \min_{e \in E} |e|$ non-zero coordinates corresponding to some subset $S \subseteq V$, where $k = |S|$. If all the non-zero y_u^* are the same for $u \in S$, then it follows that $z^* = z^S$, and the result follows; otherwise, let $\alpha := \min_{u \in S} y_u^*$. Observe that $w(S)\alpha < 1$.

Define $\widehat{z} = (\widehat{x}, \widehat{y})$ as follows.

$$\widehat{x}_e = \begin{cases} \frac{x_e^* - \alpha}{1 - w(S)\alpha} & \text{if } e \in E(S) \\ 0 & \text{otherwise.} \end{cases}$$

$$\widehat{y}_u = \begin{cases} \frac{y_u^* - \alpha}{1 - w(S)\alpha} & \text{if } u \in S \\ 0 & \text{otherwise.} \end{cases}$$

Hence, $z^* = w(S)\alpha \cdot z^S + (1 - w(S)\alpha) \cdot \widehat{z}$, and the number of non-zero coordinates of $\widehat{y}$ is strictly less than k. Hence, to complete the inductive step, it suffices to show that $\widehat{z}$ is an optimal solution to the basic LP.

Observe that since the objective function is linear, $p(z^*) = w(S)\alpha \cdot p(z^S) + (1 - w(S)\alpha) \cdot p(\widehat{z})$, it is enough to show that $\widehat{z}$ is feasible, because if $p(\widehat{z}) < p(z^*)$, it must be the case that $p(z^S) > p(z^*)$, which violates the optimality of z^*.

To check the feasibility of $\widehat{z}$, we have for $e \in E(S)$, $\widehat{x}_e = \frac{x_e^* - \alpha}{1 - w(S)\alpha} = \frac{\min_{u \in e} y_u^* - \alpha}{1 - w(S)\alpha} = \min_{u \in e} \widehat{y}_u$.

Moreover, $\sum_{u \in S} w(u)\widehat{y}_u = \sum_{u \in S} \frac{w(u) y_u^* - w(u)\alpha}{1 - w(S)\alpha} = \frac{\sum_{u \in S} w(u) y_u^* - w(S)\alpha}{1 - w(S)\alpha} = 1$, because $\sum_{u \in S} w(u) y_u^* = 1$. Therefore, $\widehat{z}$ is feasible and this completes the inductive step. □

The following corollary follows from Lemma 4.2.

Corollary 4.3. *Suppose* S_1, S_2 *both induce densest subgraphs in* V. *Then, both* $S_1 \cap S_2$ *and* $S_1 \cup S_2$ *induce densest subgraphs in* V.

Constructing Densest Subset from LP solution. An important consequence of Lemma 4.2 is that we can find a densest subgraph by first solving BasicLP to obtain (x^*, y^*), and then returning

the subgraph consisting of all nodes u whose corresponding variables y_u^* have values strictly larger than zero in our solution.

In fact, for any value $0 < t \leq \max_{v \in V} y_v^*$, the nodes u having $y_u^* \geq t$ will form a densest subset. Since we would like to find a minimal densest subgraph, it is desirable to choose the largest possible value of t, i.e., we pick the nodes u attaining the maximum y_u^* values.

This can speed up the LP-based algorithm presented in [13]. Starting from a solution for BasicLP, our method returns a densest subgraph in $O(|V|)$ time, in contrast with the rounding algorithm described in [13], which requires $\Omega(|V| \log |V| + |E|)$ operations. To the best of our knowledge, this fact was not known before [6]. We shall refer to this more efficient algorithm for the densest subgraph as FastLP.

4.1.1 Finding Minimal Densest Subgraph. **Overall Strategy.** Our intuition is that it suffices to locate a node u that is in some minimal densest subgraph. Afterwards, we consider a subroutine TryEnhance which receives in input a hypergraph G, a node u, as well as density $\rho_{\max}$ of a densest subgraph in G and returns a densest subgraph H in G which contains u while having smallest size of nodes (or null in case there is no densest subgraph containing u).

Intuition of TryEnhance. This is achieved by solving a carefully defined LP whose objective is to maximize y_u subject to the constraint that the density is equal to $\rho_{\max}$. The main intuition is that for each densest subgraph in G with S nodes there is a solution to the LP where variables have all values $\frac{1}{w(S)}$. Therefore, by maximizing y_u we can find a densest subgraph containing u with smallest weight of nodes. This is proved in Lemma 4.4 and follows partially from Lemma 4.2. See Algorithm 1 for a pseudocode of TryEnhance.

Algorithm 1: TryEnhance$(u, G, \rho_{\max})$

Input: A weighted hypergraph $G = (V, E, w_V, w_E)$, a vertex $u \in V$, the maximum density $\rho_{\max}$ over all subgraphs in G.

Output: Returns the minimal densest subgraph in G containing u or *null* if there is no densest subgraph containing u.

1 Modify the BasicLP by adding the constraint $\sum_{e \in E} w(e) x_e = \rho_{\max}$, maximizing the objective function y_u; solve the modified LP.
2 **if** $y_u = 0$ *in the optimal solution of the modified LP* **then**
3 **return** *null*
4 Run FastLP to find a densest subgraph $H = (\bar{V}, \bar{E})$ starting from the LP solution.
5 **return** H

Lemma 4.4. *Given a weighted hypergraph $G = (V, E)$ and a node $u \in V$, the subroutine TryEnhance returns null or the densest subgraph in G containing u with the smallest $w(S)$.*

Proof. When $y_u = 0$, the subroutine TryEnhance returns *null*. Following we consider only the case $y_u > 0$.

The minimal densest subgraph containing u is unique, because by Corollary 4.3, the intersection of all densest subgraphs containing u is also a densest subgraph.

Consider the optimal solution $z = (x, y)$ computed from the modified LP in the subroutine TryEnhance$(u, G, \rho_{\max})$. By Lemma 4.2, z is a convex combination of z^{S_i}, where each S_i induces a densest subgraph in G. Since the objective is to maximize y_u, and z^S is a feasible solution, it follows that y_u is positive, which means that at least one of the S_i must contain u. Since for each S_i containing u, $y_u^{S_i} = \frac{1}{w(S_i)}$, it follows that if z is a convex combination of more than one S_i, the value y_u could be strictly improved by $z^{\overline{S}}$, where $\overline{S}$ is the intersection of all S_i's containing u.

Hence, it follows that $z = z^{\overline{S}}$ for some densest subgraph induced by $\overline{S}$, which has to be the smallest densest subgraph containing u. □

Deterministic Algorithm for Finding Minimal Densest Subgraphs. Given a densest subgraph $H = (\bar{V}, \bar{E})$ with density $\rho_{\max}$, we are ready for finding a minimal one within it, or claim the given one is minimal. A trivial solution is to iterate all the nodes u in $\bar{V}$, and in each iteration, we run TryEnhance(u). Among all the output subgraphs of TryEnhance's, the one with minimum size of nodes must be a minimal densest subgraph as we claimed in Lemma 4.4. This requires $O(|\bar{V}|)$ iterations, where each iteration needs to solve BasicLP.

Binary Search. A simple way to reduce the number of iterations is binary search. As aforementioned, we will locate a node u such that when u is the input of TryEnhance, the latter will return densest subgraph that is guaranteed to be minimal. The idea to locate u is as follows: first label the nodes with $[|\bar{V}|]$ in arbitrary order. For $k \in [|\bar{V}|]$, we solved BasicLP on the graph induced by nodes with first k labels. When there exists a k such that nodes with first k labels result in a solution with density $\rho_{\max}$, but nodes with first $k-1$ labels fail to achieve so, we could conclude that node with label k is the one with desired property. Note that as k increases, the solution given by BasicLP is non-decreasing and must reach $\rho_{\max}$ when $k = |\bar{V}|$. Hence we could apply binary search on k to locate the desired node u.

See Algorithm 2 for a detailed description of finding a minimal densest subgraph in a given graph G.

Algorithm 2: DetFindMinimal(G)

Input: A weighted hypergraph $G = (V, E, w_V, w_E)$ with n nodes.
Output: A minimal densest subgraph in G.

Find a densest subgraph $H = (\bar{V}, \bar{E})$ in $\bar{G}$ by running FastLP. Let $\rho_{\max}$ be its density.
Denote $n' = |\bar{V}|$. Label $\bar{V}$ with $[n']$ in arbitrary order and hence $\bar{V} = \{u_1, u_2, \cdots, u_{n'}\}$.
/* Binary Search on k */
let $l := 0, k := n$
while $l < k$ **do**
    let $k' := \lfloor \frac{l+k}{2} \rfloor$
    let $H' := G[\{u_1, \cdots, u_{k'}\}]$
    run a densest subgraph in H' by running FastLP, and let ρ be its density.
    if $\rho < \rho_{\max}$ **then**
        let $l := k' + 1$
    else
        let $k := k'$
return TryEnhance($u_k, H, \rho_{\max}$)

Lemma 4.5. *DetFindMinimal(G) returns a minimal densest subgraph of G.*

Proof. Algorithm 2 always terminates. The binary search ensures that the output k satisfying the property that running FastLP on $G[\{u_1, \cdots, u_{k-1}\}]$ result in a subgraph that is not densest in G, and it is so when on $G[\{u_1, \cdots, u_k\}]$.

By Lemma 4.4, TryEnhance($u_k, H, \rho_{\max}$) returns a denest subgraph H_k containing u_k with minimum number of nodes. For contradiction's sake, we suppose there exists $\bar{H}_k$ that $\bar{H}_k$ is a subgraph of H_k without u_k but has the same density. We denote $G[\{u_1, \cdots, u_k\}]$ as G_k and $G[\{u_1, \cdots, u_{k-1}\}]$ as $\bar{G}_k$. Note that H_k is a subgraph of G_k (otherwise H_k is not minimal by Corollary 4.3, which

contradicts Lemma 4.4), we conclude that $\bar{H}_k$ is a subgraph of $\bar{G}_k$. Hence running FastLP on $\bar{G}_k$ could give a subgraph with density $\rho_{\max}$, which leads to a contradiction. □

Randomized Algorithm for Finding Minimal Densest Subgraphs. We devise another randomized algorithm of finding minimal densest subgraph named FindMinimal(G). Theoretically, it requires only $O(\log|\bar{V}|)$ iterations with high probability. However, when we perform experiments in Section 5, we observe that on real-world data, it terminates in 1 iteration for almost 99.9% of the time. As comparison, we construct a synthetic example later for which the randomized algorithm takes an expected number $O(\log|\bar{V}|)$ of iterations.

We employ a subroutine named TryRemove that takes as input a hypergraph G, a node u and ρ_{max}, the density of the densest subgraph in G and checks whether u can be removed from G without decreasing the density of its densest subgraph. This is done by computing a densest subgraph in the input hypergraph $(V \setminus \{u\}, E)$ and checking whether the density drops.

Algorithm 3: TryRemove(u, G, ρ_{max})

Input: A weighted hypergraph $G = (V, E)$, a node u to be removed and ρ_{max}, the maximum density over subgraphs in G.

Output: Returns a densest subgraph in G not containing u, or *null* if every densest subgraph in G must contain u.

```
Solve the BasicLP on the induced subgraph G[V \ {u}] and run FastLP to find a densest
  subgraph H in G[V \ {u}].
if ρ(H) ≥ ρ_max then
    return H
else
    return null
```

The main steps for finding efficiently a minimal densest subgraph are the following ones. At each iteration: 1) we pick one node u from our current hypergraph, uniformly at random; 2) we execute the subroutines TryRemove and TryEnhance with input u and our current hypergraph; 3) our current hypergraph is then set to be the subgraph with smallest weight of nodes among the ones returned by the previous subroutines. It turns out that $2\log_{\frac{4}{3}} n$ iterations suffice to find a minimal densest subgraph, with high probability. This is proved in Corollary 4.9, while Lemma 4.6 proves that our algorithm finds a minimal densest subgraph.

In order to speed up even further our algorithm, we include a preprocessing phase where we remove nodes not belonging to any densest subgraph. This is done as follows. We first run the linear-time greedy algorithm presented by Hu et al. in [26] so to find a r-approximation solution to the densest subgraph, where $r = \max_{e \in E} |e|$. Let ρ_{apx} be the density of the hypergraphs so found. As noted in [29], no nodes u with value $\frac{w(e)}{w(u)}$ smaller than the density of the densest subgraph belongs to any densest subgraph, where $u \in e \in E$. Therefore, we can safely remove all nodes u

with $\frac{w(e)}{w(u)}$ smaller than ρ_{apx} from the input hypergraph. A pseudocode of our algorithm is shown in Algorithm 4.

Algorithm 4: FindMinimal(G)

Input: A weighted hypergraph $G = (V, E, w_V, w_E)$ with n nodes.
Output: A minimal densest subgraph in G.

Run the greedy algorithm to find a r-approximation solution for the densest subgraph. Let ρ_{apx} be the density of such a subgraph.
Remove iteratively node u whose $\frac{w(e)}{w(u)}$ less than ρ_{apx}, and edges containing u, and let $\bar{G}$ be the remaining hypergraph.
Find a densest subgraph $H = (\bar{V}, \bar{E})$ in $\bar{G}$ by running FastLP. Let $\rho_{\max}$ be its density.
while *true* **do**
    let u be a node picked uniformly at random from $\bar{V}$
    let $H_1 :=$ TryRemove(u, H, ρ_{max})
    let $H_2 :=$ TryEnhance$(u, H, \rho_{\max})$
    if H_1 *is* null **then**
        return H_2
    let $\widehat{H}$ be the subgraph with minimum size of nodes between H_1 and H_2, breaking ties arbitrarily
    $H := \widehat{H}$
return H

Lemma 4.6. *FindMinimal(G) returns a minimal densest subgraph of G.*

Proof. Note that Algorithm 4 always terminates. After each iteration of the *while* loop, either TryRemove(u, H, ρ_{max}) returns *null* and therefore the algorithm terminates or the node u is removed from the current hypergraph. Hence, at each iteration the number of nodes of the current hypergraph decreases by at least one. Observe that the hypergraph H is always a densest subgraph, throughout the execution of the algorithm. When the algorithm terminates, it must be the case that TryRemove(u, H, ρ_{max}) returns *null*, for some node u in H. This means that every densest subgraph of H must contain u. By Lemma 4.4, TryEnhance$(u, H, \rho_{\max})$ returns the densest subgraph containing u with smallest weight of nodes, and so it must be minimal. □

We prove that $O(\log n)$ iterations of the *while* loop of FindMinimal(G) suffice to find a minimal densest subgraph in G. To this end, we show that at each iteration the number of nodes in the current subgraph decreases by a constant fraction with constant probability. A standard measure concentration argument concludes the proof. Let $H = (\bar{V}, \bar{E})$ be a densest subgraph of G. For a node $u \in \bar{V}$, $\epsilon > 0$, we say that u is ϵ-bad in $H = (\bar{V}, \bar{E})$ if both $H_1 :=$ TryRemove(u, H, ρ_{max}) and $H_2 :=$ TryEnhance$(u, H, \rho_{\max})$ contain more than $(1 - \epsilon)|\bar{V}|$ nodes, and neither of them is *null*.

Lemma 4.7. *Given a hypergraph $H = (\bar{V}, \bar{E})$ with maximum density $\rho_{\max}$, the fraction of ϵ-bad nodes in $\bar{V}$ is at most 2ϵ, for any $\epsilon > 0$.*

Proof. For contradiction's sake, suppose the set B of ϵ-bad nodes has size more than $2\epsilon|\bar{V}|$. For each $u \in B$, define $A_u := \bar{V} \setminus$ TryRemove(u, H, ρ_{max}). Observe that $u \in A_u$, and by the definition of ϵ-bad, $|A_u| < \epsilon|\bar{V}|$.

Consider an arbitrary order of $B := \{u_1, u_2, \ldots, u_{|B|}\}$. Since B contains more than $2\epsilon|\bar{V}|$ points, and each $|A_u| < \epsilon|\bar{V}|$, there exists a smallest i such that $\epsilon|\bar{V}| \leq |\bigcup_{j=1}^{i} A_{u_j}| < 2\epsilon|\bar{V}|$.

Algorithm 5: FindAllMinimal(V)

Input: Weighted hypergraph $G = (V, E, w_V, w_E)$.
Output: Returns a list L of all minimal densest subgraphs.

```
L := ∅
while true do
    Ĥ = FindMinimal(G)
    if Ĥ is not a densest subgraph then
        break
    L := L ∪ {Ĥ}
    remove all nodes and edges incident to Ĥ from G
return L
```

Observe that there exists a $\widehat{u} \in B \setminus (\bigcup_{j=1}^{i} A_{u_j})$. Since for each j, TryRemove(u_j, H, ρ_{max}) is a densest subgraph, then by Lemma 4.3, $\bigcap_{j=1}^{i}$ TryRemove$(u_j, H, \rho_{max}) = \bar{V} \setminus (\bigcup_{j=1}^{i} A_{u_j})$ induces a densest subgraph that contains $\widehat{u}$ and has size at most $(1-\epsilon)|\bar{V}|$. Therefore, it follows that the algorithm TryEnhance$(\widehat{u}, H, \rho_{max})$ will return a densest subgraph that has size at most $(1-\epsilon)|\bar{V}|$, thereby contradicting that $\widehat{u}$ is ϵ-bad. □

By taking $\epsilon = \frac{1}{4}$ in Lemma 4.7, we have the following corollary.

Corollary 4.8. *At each iteration of* FindMinimal(G)*, the algorithm either terminates with a minimal densest subgraph, or with probability at least* $\frac{1}{2}$*, the number of nodes in* $\widehat{H}$ *is at most* $\frac{3}{4} \cdot |\bar{V}|$.

Corollary 4.9. *By standard Chernoff bound, with probability at least* $1 - \frac{1}{\text{poly}(|V|)}$*, the number of iterations in* FindMinimal(G) *is at most a constant times its mean, which is at most* $2\log_{\frac{4}{3}} |V|$.

Our algorithm FindMinimal(G) allows us to compute minimal densest subgraphs in hypergraphs containing millions of edges. In particular, the pruning step and the fact that we can bound the number of iterations by a logarithmic function allows us to save several order of magnitudes in terms of running time.

Synthetic Example. For comparison, we give an example such that our FindMinimal(G) will require an expected number $\Theta(\log n)$ of iterations. Consider a graph G with n nodes $\{u_1, u_2, \cdots, u_n\}$ and n edges, such that (1) u_1, u_2 and u_3 form a triangle, (2) $u_4, u_5, \cdots, u_n$ form a path, and (3) u_3 and u_4 are connected. Each time, the algorithm will randomly select a node u_k, and run TryRemove as well as TryEnhance. When $k \leq 3$, TryRemove will give null, TryEnhance will give the loop that is minimal, and the while loop terminates. When $k \geq 4$, TryRemove will give $G[\{u_1, \cdots, u_{k-1}\}]$, TryEnhance will give $G[\{u_1, \cdots, u_k\}]$, and while loop continues with $G[\{u_1, \cdots, u_{k-1}\}]$. Hence, when we denote $T(n)$ as the expected number of iterations on n nodes, we have $T(n) = 1 + \sum_{i=3}^{n-1} T(i)/n$, which gives $T(n) = \Theta(\log n)$.

4.1.2 Finding All Minimal Densest Subgraphs. Armed with an efficient algorithm for finding minimal densest subgraphs, we now present an algorithm that computes all such hypergraphs. Our algorithm computes at each step a minimal densest subgraph by running Algorithm 4, it removes all its nodes and edges from the current hypergraph and iterates until no densest subgraph can be found. A pseudocode is shown in Algorithm 5.

Minimal densest subgraphs must be disjoint, for otherwise, their intersection would be a densest subgraph (from Corollary 4.3), which would contradict the fact that they are minimal. Lemma 4.10 then follows.

Lemma 4.10. *Given a weighted hypergraph $G = (V, E, w_V, w_E)$, our algorithm* FindAllMinimal(V) *finds all minimal densest subgraphs in G.*

4.2 Main Algorithms

Drawing inspiration from the theory and the techniques developed in the previous section for finding minimal densest subgraphs, we devise one algorithm for our main problem (k, α)-DSLO. In this case, we face the additional challenge of taking full advantage of the overlap between the subgraphs.

Our algorithm is inspired by FindMinimal and proceeds as follows. At each step i, we compute a minimal densest subgraph $G_i = (V_i, E_i)$ of our current hypergraph $G = (V, E, w_V, w_E)$ and we remove some nodes from V such that at least $(1-\alpha)w(V_i)$ weight of nodes is decreased. We remove those nodes that are not well connected with nodes outside G_i, as they will contribute less to the total density in the next steps of the algorithm. Formally, for node u in V_i, we remove some nodes (and their edges) according to the value $\frac{w(e)}{w(u)}$ where e is the incident edge of u in current hypergraph, such that at least $(1-\alpha)w(V_i)$ weight of nodes is decreased. We iterate until k subgraphs are found or the current hypergraph becomes empty. Observe that the constraint on the weighted Jaccard coefficient is not violated. A pseudocode of our algorithm is shown in Algorithm 6.

We can prove an interesting property of MinAndRemove. Observe that if there are k disjoint densest subgraphs in G then MinAndRemove would return the same solution of FindAllMinimal. Then, our main theorem follows from Lemma 4.10.

Theorem 4.11. *If there are k disjoint densest subgraphs in the input hypergraph G, then* MinAndRemove *computes an optimal solution for (k, α)-DSLO, for any $\alpha \geq 0$.*

Although, we cannot give any guarantee for the general case of (k, α)-DSLO (i.e. for any $\alpha \geq 0$) we show the effectiveness of our main algorithm in our experimental evaluation. In particular, we are able to derive an upper bound on any optimum solution for (k, α)-DSLO and show that MinAndRemove is very close to such an upper bound in our experiments (up to a factor of 0.9).

Finally, we present a fast heuristic ApproxMinAndRemove for finding dense subgraphs with limited overlap. Our heuristic computes at each step i a r-approximation solution G_i to the densest subgraph problem using the greedy algorithm presented in [26]. Then, similarly to MinAndRemove $(1-\alpha)w(V_i)$ weight of nodes are descreased from the current hypergraph so to satisfy the requirement on the pairwise weighted Jaccard coefficient. A pseudocode is shown in Algorithm 6. Our experimental evaluation shows that although our heuristic is less accurate than MinAndRemove,

it is still not too far from an optimum solution while it can handle hypergraphs with more than 10 million edges within a few minutes.

Algorithm 6: (Approx)MinAndRemove(G, k, α)

Input: A weighted hypergraph $G = (V, E, w_V, w_E)$, an integer $k > 0$, $\alpha \in [0, 1]$
Output: A list $L = \{G_i = (V_i, E_i)\}_i$ of at most k subgraphs of G such that the constraint on the pairwise weighted Jaccard coefficient on the V_i's is not violated (Equation (1)).

$L := \emptyset; H := G$
while $< k$ *subgraphs are found and* H *is not empty* **do**
  Find a subgraph $G_i = (V_i, E_i)$ of H by either running (i) the minimal densest subgraph Algorithm 4, or (ii) the faster r-approximation greedy algorithm in [26].
  $L := L \cup \{G_i\}$
  $S := V_i$
  while $w(S) > \alpha \cdot w(V_i)$ **do**
    Let $u = \arg\min_{v \in S} \frac{\sum_{e \in E: v \in e} w(e)}{w(v)}$ /*alternatively we can use $u = \arg\min_{v \in S} \frac{\sum_{e \in E: v \in e \wedge e \setminus V_i \neq \emptyset} w(e)}{w(v)}$}*/
    Remove v from S and H, and remove all its incident edges from $E(H)$.
return L

5 EXPERIMENTS

We perform our experimental evaluation on a Linux server with Intel Xeon E5-2660 at 2.00GHz, while limiting the total amount of main memory avaiable to 128GB. We solve linear programs with the Gurobi Optimizer version 9.1.1. All our algorithms are implemented in C++[1].

DATASET. We consider 8 datasets[2] in total. The datasets contains up to 11 million edges, as illustrated in Table 1.

Name	Nodes	Edges	r	Sum of degree	Description
web-Google	875K	4.3M	2	8.6×10^6	Hyperlink network
web-Stanford	281K	1.9M	2	4.0×10^6	Hyperlink network
com-Youtube	1.1M	2.9M	2	6.0×10^6	Social network
as-Skitter	1.6M	11M	2	2.2×10^7	Internet topology
Reddit-Democrats	3.6K	4K	20	1.0×10^4	Social network
Reddit-Republican	2.7K	3.6K	27	7.7×10^3	Social network
Reddit-UKPolitics	34K	161K	220	6.2×10^5	Social network
Reddit-Politics	498K	1.4M	835	6.8×10^7	Social network

Table 1. Real-world datasets.

The Reddit datasets [7] are created from discussions on Reddit posted in 2019 on four forums: Democrats, Republican, UKPolitics and Politcs. On Reddit, there are two types of posts, submissions and comments. The submission will start a discussion on a certain topic, and comments will be reactions to that submission. A submission and its comments can be organized in a graph as a forest, where the submission represents the root of all the trees in the forest. We will refer to these trees as discussion trees. In order to create a hypergraph from this data, we consider that nodes in

[1]The source codes are available at: https://github.com/scidylanpno/dslo
[2]Datasets web-Google, web-Stanford, com-Youtube, as-Skitter are from snap.stanford.edu

our hypergraph represent users. A hyperedge represents all the users that posted comments in the same discussion tree. We only keep hyperedges with more than 1 node. A hyperedge is weighted by the number of comments we have in the discussion tree. A node is weighted by the number of distinct days in which the user posted comments in the discussion trees considered above.

Complexity of LP. In Section 4.1, the LP becomes more complicated for hypergraphs. Specifically, the number of constraints in (2) will increase when a hyperedge contains a large number of nodes. As a result, the hypergraphs we include in our experiments have a smaller number of nodes and edges compared to the normal graphs.

BENCHMARK ALGORITHMS. We evaluate our algorithm (Approx)MinAndRemove against two variants of the Naive algorithm. In the first variant we compute at each step a densest subgraph with the LP-based approach proposed in [26], we remove all its nodes and edges from the graph, and we iterate until k subgraphs have been found or the graph has become empty. We refer to this algorithm as NaiveDensest. In the second variant we compute at each step the $\frac{1}{r}$-approximation algorithm proposed in [26], where $r = \max_{e \in E} |e|$, instead of a densest subgraph. We refer to this algorithm as NaiveApprox. The reason to consider also this second variant is that the $\frac{1}{r}$-approximation algorithm is much faster than the LP-based optimal approach.

MinAndRemove AND OUR UPPER BOUND. We start by measuring the total density when $k = 10$, while varying α between 0.1 and 0.5. Let $\rho_{\max}$ be the density of a densest subgraph in the input graph. Observe that $k \cdot \rho_{\max}$ gives us an upper bound on the value of any optimum solution for (k, α)-DSLO, for any value of α. Therefore, we can use such an upper bound to give an idea of how close our results are to an optimum solution. Given that we use only an upper bound, clearly, our results might be even closer to the actual optimum solution.

In Table 2, we measure the ratio between the total density computed by our main algorithm and our upper bound, when $k = 10$. We can see that in all cases our algorithm yields a solution that is at least within a factor of 0.21 of the value of an optimum solution, while in many cases it yields an approximation factor of 0.6 (meaning that it reaches the 60% of the upper bound on the optimum objective-function value). We can also see that in most cases the quality of the solution increases as a function of α showing that our algorithm takes full advantage of the overlap between the subgraphs. We also recall that our upper bound might be loose and that we might have computed an optimum solution for such a dataset.

In Table 3, we set $\alpha = 0.3$ and we measure the ratio between the density of MinAndRemove and an upper bound to any optimum solution as a function of k. We can see that when k is at most 4 our solution is very close to an optimum solution for (k, α)-DSLO. For larger values of k, in most cases our solution is still within a factor of 0.6 of an optimum solution or better. This might depend on the fact that our upper bound becomes loose when k is large.

MinAndRemove AND NaiveDensest. Next, we evaluate our algorithm MinAndRemove against NaiveDensest, when $k = 10$. Table 4 shows the ratio between the total density of the subgraphs found by our algorithm and those of NaiveDensest. We can see that in most cases MinAndRemove yields a solution larger than that of NaiveDensest. On the other hand, for the graph "web-Google", the total density does not change much as α increases. This implies that subgraphs with high-density on do not overlap much with one another. We speculate the difference is that in a social network, people can be active in different communities with different context, which means that high-density subgraphs are more likely to have overlapped individuals. In contrast, web pages are often dedicated to specific topics and this may explain why the dense subgraphs do not have a large number of common web pages.

We expect that the advantage of MinAndRemove against NaiveDensest would be even more remarkable for larger values of k. Table 5 shows the total running time of our algorithm, which is always at most 4.5 hours, while in many cases is as less as 30 mins.

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	.719	.736	.780	.818	.840
com-Youtube	.479	.490	.501	.593	.634
web-Google	.808	.808	.808	.808	.808
as-Skitter	.586	.599	.602	.631	.652
Reddit-Democrats	.614	.664	.660	.681	.739
Reddit-Republican	.830	.817	.819	.826	.818
Reddit-UKPolitics	.219	.251	.292	.340	.403
Reddit-Politics	.263	.306	.358	.418	.478

Table 2. Ratio between the density of MinAndRemove and our upper bound

$\alpha = 0.3$	$k = 2$	$k = 4$	$k = 6$	$k = 8$	$k = 10$
web-Stanford	.991	.917	.863	.817	.780
com-Youtube	.843	.721	.613	.548	.501
web-Google	.991	.916	.864	.836	.808
as-Skitter	.915	.784	.694	.643	.602
Reddit-Democrats	.882	.792	.731	.691	.660
Reddit-Republican	.983	.918	.863	.836	.819
Reddit-UKPolitics	.868	.566	.427	.345	.292
Reddit-Politics	.850	.625	.505	.420	.358

Table 3. Ratio between the density of MinAndRemove and our upper bound

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	1.111	1.137	1.205	1.265	1.298
com-Youtube	1.066	1.091	1.116	1.319	1.411
web-Google	1.002	1.002	1.002	1.002	1.003
as-Skitter	1.017	1.039	1.045	1.096	1.131
Reddit-Democrats	1.043	1.127	1.121	1.155	1.255
Reddit-Republican	1.016	0.999	1.002	1.011	1.001
Reddit-UKPolitics	1.272	1.461	1.699	1.978	2.345
Reddit-Politics	1.322	1.539	1.800	2.104	2.404

Table 4. Ratio between the density of MinAndRemove and NaiveDensest

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	4.01×10^2s	4.25×10^2s	4.42×10^2s	4.65×10^2s	4.09×10^2s
com-Youtube	1.44×10^3s	1.30×10^3s	1.93×10^3s	1.31×10^3s	1.29×10^3s
web-Google	5.05×10^2s	5.04×10^2s	5.48×10^2s	5.54×10^2s	5.06×10^2s
as-Skitter	4.31×10^3s	8.93×10^3s	8.86×10^3s	6.32×10^3s	7.09×10^3s
Reddit-Democrats	3.10×10^0s	3.45×10^0s	3.53×10^0s	3.47×10^0s	3.50×10^0s
Reddit-Republican	4.25×10^0s	4.21×10^0s	4.61×10^0s	4.62×10^0s	5.01×10^0s
Reddit-UKPolitics	3.60×10^2s	3.64×10^2s	3.72×10^2s	3.74×10^2s	3.77×10^2s
Reddit-Politics	1.60×10^4s	7.10×10^3s	6.90×10^3s	6.91×10^3s	8.39×10^3s

Table 5. Running time when varying α for MinAndRemove

RUNNING TIME OF MINANDREMOVE. Our experimental evaluation shows that MINANDREMOVE can handle large graphs containing up to more than 10 million edges within 5 hours or less, while delivering near-optimal solutions for our problem. For even larger graphs we need to resort to our fast heuristic. Similarly to the previous case, we can still use $k \cdot \rho_{\max}$ be the upper bound on the value of any optimum solution for (k, α)-DSLO.

APPROXMINANDREMOVE. In Table 6, we set $k = 10$ while we measure the ratio between the density of APPROXMINANDREMOVE and an upper bound to any optimum solution, as a function of α. Although our heuristic turns out to be less accurate than MINANDREMOVE it delivers solutions with an approximation factor of around 0.2 or more, while it can handle large graphs within a few minutes. We also evaluate our algorithm APPROXMINANDREMOVE against NAIVEAPPROX, when $k = 10$. Table 7 shows that even though the approximation algorithm are equipped as a subroutine, our algorithm can still make good use of the overlapness. At most cases, our algorithm is at least a factor of 1.2 of the NAIVEAPPROX. Table 8 shows the running time of APPROXMINANDREMOVE when $k = 10$ and α varies between 0.1 and 0.5. We can observe that the running time is around few seconds for the smaller datasets and does not exceed 15 minutes on all the datasets.

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	.558	.617	.648	.681	.722
com-Youtube	.472	.491	.479	.572	.630
web-Google	.509	.540	.565	.612	.630
as-Skitter	.561	.571	.566	.595	.615
Reddit-Democrats	.431	.481	.541	.571	.622
Reddit-Republican	.507	.529	.602	.615	.666
Reddit-UKPolitics	.199	.244	.290	.337	.393
Reddit-Politics	.246	.301	.357	.419	.477

Table 6. Ratio between the density of APPROXMINANDREMOVE and our upper bound

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	1.143	1.263	1.326	1.394	1.478
com-Youtube	1.170	1.219	1.189	1.420	1.563
web-Google	1.005	1.067	1.115	1.208	1.243
as-Skitter	1.078	1.097	1.087	1.142	1.181
Reddit-Democrats	1.203	1.343	1.512	1.594	1.735
Reddit-Republican	0.992	1.034	1.177	1.204	1.302
Reddit-UKPolitics	1.313	1.607	1.911	2.223	2.589
Reddit-Politics	1.465	1.790	2.124	2.488	2.834

Table 7. Ratio between the density of APPROXMINANDREMOVE and NAIVEAPPROX

ISOMORPHISM AMONG NON-UNIQUE DENSEST SUBGRAPHS. We investigate the uniqueness of dense subgraphs in our datasets and whether subgraphs with the same density are isomorphic. Our first finding is that in all the datasets we considered, the densest subgraph is unique. Next, we focus on subgraphs with arbitrary density. Since we are not aware of an efficient algorithm to enumerate all subgraphs with a given density, we proceed as follows. We compute a minimal densest H in the input graph G and we remove all nodes and edges of H from G. If the density of H is larger than 1.25, then we add H to the set S (initially empty), which contains the graphs to

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	9.74×10^1s	9.78×10^1s	9.85×10^1s	9.98×10^1s	9.89×10^1s
com-Youtube	1.68×10^2s	1.73×10^2s	1.78×10^2s	1.84×10^2s	1.88×10^2s
web-Google	2.61×10^2s	2.60×10^2s	2.57×10^2s	2.50×10^2s	2.53×10^2s
as-Skitter	6.04×10^2s	6.21×10^2s	6.39×10^2s	6.32×10^2s	6.44×10^2s
Reddit-Democrats	1.42×10^{-1}s	1.57×10^{-1}s	1.53×10^{-1}s	1.92×10^{-1}s	1.92×10^{-1}s
Reddit-Republican	8.87×10^{-2}s	9.80×10^{-2}s	1.11×10^{-1}s	1.10×10^{-1}s	1.30×10^{-1}s
Reddit-UKPolitics	4.94×10^0s	5.34×10^0s	5.80×10^0s	6.49×10^0s	7.19×10^0s
Reddit-Politics	9.71×10^1s	1.08×10^2s	1.19×10^2s	1.30×10^2s	1.46×10^2s

Table 8. Running time of ApproxMinAndRemove on very large datasets as a function of α

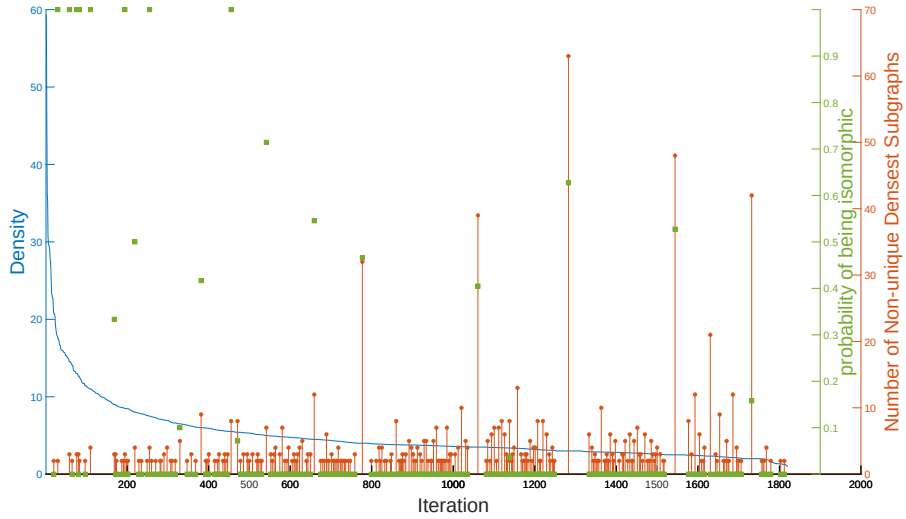

Fig. 2. Density (blue), the number of multiple densest subgraphs (orange) and the probability of being isomorphic given two subgraphs have the same density (green) in web-Stanford.

be studied further. Such a procedure is iterated until the graph becomes empty. When there are multiple subgraphs with the same density, we employ the approach in [27] to verify whether they are isomorphic. The results of this experiment for web-Stanford are depicted in Figure 2, which reports the density of the subgraphs in S found at each step of our procedure (in blue), the number of multiple densest subgraphs (in orange) and the probability of being isomorphic given that two subgraphs have the same density (in green). After 1820 iterations of our procedure, we obtain 260 sets, each one of which containing several subgraphs with the same density. We found that the probability of two graphs in S being isomorphic given that they have the same density > 9 is > 0.6, while such a probability is close to zero for smaller density values. Our experiments offer some preliminary evidence that subgraphs sharing the same density are relatively rare in real-world graphs, while they are often isomorphic when they are "sufficiently" dense.

6 CONCLUSIONS

This paper studies the problem of finding at most k subgraphs from a large hypergraph given in input such that the total weighted density be maximized, while satisfying a constraint on the pairwise weighted Jaccard coefficient between the subgraphs. Although very natural, this variant of the densest subgraph problem has been surprisingly neglected so far.

After showing the NP-hardness of this problem (even when the subgraphs are disjoint), we develop an algorithm that computes an optimum solution for our problem in the case when there are k disjoint densest subgraphs in the input graph. Moreover, our experimental evaluation on large real-world hypergraphs shows that our algorithm delivers near-optimal solutions, in that, they are very close to an upper bound on any optimum solution. We introduce the concept of minimality of densest subgraphs in hypergraphs, while we develop efficient algorithms for such a problem.

We will devote our future investigation to devise even more scalable algorithms and to adapt our algorithm into a dynamic environment, where edges might be added to or removed from the current graph. Another interesting direction is to determine whether finding minimal densest subgraphs can be a valuable tool in finding interesting patterns in social networks and other real-world graphs.

ACKNOWLEDGEMENTS

T-H. Hubert Chan was partially supported by the Hong Kong RGC grant 17203122. Mauro Sozio was partially supported by the French National Agency (ANR) under project APY (ANR-20-CE38-0011). We would like to thank Pierre Lafourcade and Artur Léonard for pointing out a simpler deterministic algorithm for efficiently finding minimal densest subgraphs.

REFERENCES

[1] Reid Andersen and Kumar Chellapilla. 2009. Finding Dense Subgraphs with Size Bounds. In *WAW*.
[2] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. 2012. Dense Subgraph Maintenance Under Streaming Edge Weight Updates for Real-time Story Identification. *PVLDB* 5, 6 (2012).
[3] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. 2002. Complexity of finding dense subgraphs. *Discr. Ap. Math.* 121, 1-3 (2002).
[4] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily finding a dense subgraph. *J. Algorithms* 34, 2 (2000).
[5] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest Subgraph in Streaming and MapReduce. *PVLDB* 5, 5 (2012).
[6] Oana Denisa Balalau, Francesco Bonchi, T.-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. 2015. Finding Subgraphs with Maximum Total Density and Limited Overlap. In *WSDM*. ACM, 379–388.
[7] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. 2020. The pushshift reddit dataset. In *Proceedings of the international AAAI conference on web and social media*, Vol. 14. 830–839.
[8] Suman K. Bera, Sayan Bhattacharya, Jayesh Choudhari, and Prantar Ghosh. 2022. A New Dynamic Algorithm for Densest Subhypergraphs. In *WWW*. ACM, 1093–1103.
[9] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. 2010. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*. 201–210.
[10] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. 2015. Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM, 173–182.
[11] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core Decomposition of Uncertain Graphs. In *KDD*.
[12] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos E. Tsourakakis, Di Wang, and Junxing Wang. 2020. Flowless: Extracting Densest Subgraphs Without Flow Computations. In *WWW*. ACM / IW3C2, 573–583.
[13] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, Klaus Jansen and Samir Khuller (Eds.). Springer.
[14] Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. 2022. Densest Subgraph: Supermodularity, Iterative Peeling, and Flow. In *SODA*. SIAM, 1531–1555.

[15] Jie Chen and Yousef Saad. 2012. Dense Subgraph Extraction with Application to Community Detection. *TKDE* 24, 7 (2012).

[16] Eden Chlamtác, Michael Dinitz, Christian Konrad, Guy Kortsarz, and George Rabanca. 2018. The Densest k-Subhypergraph Problem. *SIAM J. Discret. Math.* 32, 2 (2018), 1458–1477.

[17] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *SIGMOD*.

[18] Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. In *WWW*. ACM, 233–242.

[19] Riccardo Dondi, Mohammad Mehdi Hosseinzadeh, Giancarlo Mauri, and Italo Zoppis. 2021. Top-k overlapping densest subgraphs: approximation algorithms and computational complexity. *J. Comb. Optim.* 41, 1 (2021), 80–104.

[20] Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E. Lee, and John H. Thornton, Jr. 2009. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *KDD*.

[21] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient Densest Subgraph Computation in Evolving Graphs. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 300–310.

[22] Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. 2006. MotifCut: regulatory motifs finding with maximum density subgraphs. In *ISMB*.

[23] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2016. Top-k overlapping densest subgraphs. *Data Min. Knowl. Discov.* 30, 5 (2016), 1134–1165.

[24] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering Large Dense Subgraphs in Massive Graphs. In *VLDB*.

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

[26] Shuguang Hu, Xiaowei Wu, and T.-H. Hubert Chan. 2017. Maintaining Densest Subsets Efficiently in Evolving Hypergraphs. In *CIKM*. ACM, 929–938.

[27] Tommi A. Junttila and Petteri Kaski. 2007. Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs. In *ALENEX*. SIAM.

[28] Subhash Khot. 2006. Ruling Out PTAS for Graph Min-Bisection, Dense k-Subgraph, and Bipartite Clique. *J. Computing* 36, 4 (2006).

[29] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *ICALP*.

[30] Aritra Konar and Nicholas D. Sidiropoulos. 2021. Exploring the Subgraph Density-Size Trade-off via the Lovaśz Extension. In *WSDM*. ACM, 743–751.

[31] Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. 2023. A Survey on the Densest Subgraph Problem and its Variants. *CoRR* abs/2303.14467 (2023). arXiv:2303.14467 https://doi.org/10.48550/arXiv.2303.14467

[32] Michael A. Langston and et al. 2005. A Combinatorial Approach to the Analysis of Differential Gene Expression Data: The Use of Graph Algorithms for Disease Prediction and Screening. In *Methods of Microarray Data Analysis IV*.

[33] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu C. Aggarwal. 2010. A Survey of Algorithms for Dense Subgraph Discovery. In *Managing and Mining Graph Data*.

[34] Hairong Liu, Longin Jan Latecki, and Shuicheng Yan. 2015. Dense Subgraph Partition of Positive Hypergraphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 3 (2015), 541–554.

[35] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V. S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. Efficient Directed Densest Subgraph Discovery. *SIGMOD Rec.* 50, 1 (2021), 33–40.

[36] Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. 2017. Fully Dynamic Algorithm for Top-k Densest Subgraphs. In *CIKM*. ACM, 1817–1826.

[37] Christos H. Papadimitriou and Mihalis Yannakakis. 1991. Optimization, Approximation, and Complexity Classes. *J. Comput. Syst. Sci.* 43, 3 (1991).

[38] Mauro Sozio and Aristides Gionis. 2010. The Community-search Problem and How to Plan a Successful Cocktail Party. In *KDD*. 939–948.

[39] Bintao Sun, Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. *Proc. VLDB Endow.* 13, 10 (2020), 1628–1640.

[40] Nikolaj Tatti and Aristides Gionis. 2013. Discovering Nested Communities. In *ECML/PKDD (2)*.

[41] Nikolaj Tatti and Aristides Gionis. 2015. Density-friendly Graph Decomposition. In *WWW*. ACM, 1089–1099.

[42] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*. 1122–1132.

[43] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser Than the Densest Subgraph: Extracting Optimal Quasi-cliques with Quality Guarantees. In *KDD*.

[44] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *WWW*. ACM, 1122–1132.

[45] Charalampos E. Tsourakakis, Tianyi Chen, Naonori Kakimura, and Jakub Pachocki. 2019. Novel Dense Subgraph Discovery Primitives: Risk Aversion and Exclusion Queries. In *ECML/PKDD (1) (Lecture Notes in Computer Science, Vol. 11906)*. Springer, 378–394.

[46] Elena Valari, Maria Kontaki, and Apostolos N. Papadopoulos. 2012. Discovery of Top-k Dense Subgraphs in Dynamic Graph Collections. In *SSDBM*.

[47] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony K. H. Tung. 2010. On Triangulation-based Dense Neighborhood Graph Discovery. *PVLDB* 4, 2 (2010).