Finding Subgraphs with Maximum Total Density and Limited Overlap

Oana Denisa Balalau[*]
Institut Mines Telecom,
Telecom Paristech, CNRS
oana.balalau@telecom-paristech.fr

Francesco Bonchi
Yahoo Labs
Barcelona, Spain
bonchi@yahoo-inc.com

T-H. Hubert Chan[†]
Dept. of Computer Science
The University of Hong Kong
hubert@cs.hku.hk

Francesco Gullo
Yahoo Labs
Barcelona, Spain
gullo@yahoo-inc.com

Mauro Sozio[‡]
Institut Mines Telecom,
Telecom Paristech, CNRS
sozio@telecom-paristech.fr

ABSTRACT

Finding dense subgraphs in large 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 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, where the main goal is to find at most k subgraphs with maximum total aggregate density, while satisfying an upper bound on the pairwise Jaccard coefficient between the sets of nodes 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 graphs confirms the efficiency and effectiveness of our algorithms.

1. INTRODUCTION

Finding dense subgraphs in large graphs has emerged as a key primitive in a variety of real-world application domains [19], ranging from biology [13, 18] to finance [12]. In the Web domain, Gibson *et al.* [14] 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 [21], as well as for expert team formation [23, 8]. 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. Perhaps surprisingly, such a problem has not been studied in a principled way to the best of our knowledge. In this work we aim at filling this gap.

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. 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 Jaccard coefficient between the sets of nodes of the subgraphs.

Several definitions of density have been studied in the literature, among which the *average degree* density stands outs as a natural and widely used definition. Subgraphs with maximum average 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 [9] or the maximum flow algorithm in [15], while there are efficient algorithms which come with provable approximation guarantees [9]. In our work we focus on the average 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., [23]). 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,

[*]Partially funded by a Google Faculty Research Award.
[†]Partially funded by France/HK Grant: F-HK31/11T and Hong Kong RGC HKU17200214E.
[‡]Partially funded by a Google Faculty Research Award and the France/HK PHC Procore 2012 grant 26893QA.

WSDM'15, February 2–6, 2015, Shanghai, China.
Copyright 2015 ACM 978-1-4503-3317-7/15/02 ...$15.00.
http://dx.doi.org/10.1145/2684822.2685298.

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 turns out to be the fastest known algorithm for the exact computation of a densest subgraph as it can handle large graphs containing up to 10 million edges, as shown by our experimental evaluation. We finally devise an efficient heuristic so 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 Subgraph 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 subgraphs that maximize the total aggregate density, i.e., the sum of the average degree of each subgraph, under the constraint that the the maximum pairwise 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 improve the LP-based approach by Charikar [9], thus achieving the fastest known exact algorithm for the densest subgraph problem. Our algorithm has the desirable property of producing *minimal* densest subgraphs and use our fast LP solver as a subroutine. In particular, we prove that the number of calls to an LP-solver is logarithmic in the number of nodes with high probability. This allows us to deal with large real-world graphs.

- We devise an algorithm (MinAndRemove) for the (k, α)-DSLO problem. We prove that, in the case the input graph contains k disjoint densest subgraphs, our algorithm is guaranteed to find an optimum solution for (k, α)-DSLO. In the general case, we show empirically that our algorithm can find solutions that are very close to an optimum solution of our problem.

- We present a fast heuristic (FastDSLO) for (k, α)-DSLO which, albeit less accurate than MinAndRemove, is able to find dense subgraphs with limited overlap on even larger graphs (containing up to 100 million edges).

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 [19]. Generally speaking, such a problem aims at finding a subgraph of a given input graph that maximizes some notion of density. A density notion 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 [15]. Charikar [9] introduces a linear-programming formulation of the problem, while also showing that the greedy algorithm proposed by Asashiro *et al.* [5] produces a $\frac{1}{2}$-approximation in linear time. The densest-subgraph problem has also been studied in a streaming context [6].

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 [4], while an algorithm with approximation guarantee of $\mathcal{O}(n^{\frac{1}{4}})$ has been presented in [7]. It is also well known that there cannot be any PTAS for the *DkS* problem under reasonable complexity assumptions [16]. Some variants of the *DkS* problem are introduced by Andersen and Chellapilla [1] and further investigated in [17].

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.* [23] resort to the notion of quasi-clique to define an alternative measure of density, while Wang *et al.* [25] focus on a density based on triangle counting. Sozio *et al.* [21] focus on minimum degree density while enforcing so-called *monotone* constraints.

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 [24, 23] 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.

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 [22], discovering overlapping dense subgraphs containing a query node [11], or extracting all *large-enough* dense bipartite subgraphs in massive graphs [14]. Furthermore, Angel *et al.* [3] focus on maintaining the set of all (possibly overlapping) subgraphs exceeding a density threshold under streaming edge weight updates. Chen and Saad [10] 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.

3. DEFINITION AND COMPLEXITY

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

Given an undirected graph $G = (V, E)$, we define its density $\rho(G)$ to be $\frac{|E|}{|V|}$, which corresponds to half the average degree of the nodes in G. For a set of vertices $S \subseteq V$, we denote the subgraph of G induced by S as $G(S) = (S, E(S))$, where $E(S) = \{\{u, v\} \in E | u, v \in S\})$.

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 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 an undirected graph $G = (V, E)$, 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*

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

$$\frac{|S_i \cap S_j|}{|S_i \cup S_j|} \leq \alpha \quad \forall S_i, S_j \in \mathcal{S}. \tag{1}$$

Theorem 3.1 *(k, α)-DSLO is NP-hard.*

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 $\alpha = 0$. Particularly, we show that given 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 [20]. 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 3.1 *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 $\frac{\sum_{i\in[r]} d_i}{2}$.

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. $\square$

The following lemma completes the reduction proof.

Lemma 3.2 *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 3.1.

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 3.1 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. $\square$

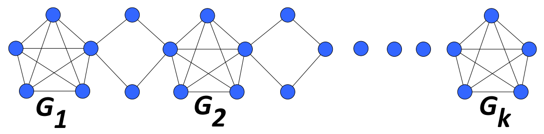

Figure 1: A graph G where Naive gives poor results. Notice that G is a non-minimal densest subgraph.

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 graph (using for example the approach in [9]), we remove all its nodes and edges from the current graph, iterating until we find exactly k subgraphs or until the current graph contains no edges. We hereinafter refer to this heuristic as Naive.

This simple heuristic gives unfortunately very poor results in the worst case, as illustrated in Figure 1.

In that example 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 graphs.

Definition 4.1 *(Minimal dense graphs) An undirected graph G with density $\rho(G)$ is a* minimal dense graph *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 graph G in Figure 1 is a non-minimal densest subgraph.

Finding minimal dense graphs 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 graph, and we iterate until k subgraphs are found or there are no edges left in the current graph. Including at each step a minimal dense 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 graph 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 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 graphs and that minimal densest subgraphs can be computed efficiently on large graphs 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 Charikar [9]. 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 [9].

Given a graph $G = (V,E)$, Charikar [9] proposed the following LP formulation for the densest subgraph problem. For each edge $ij \in E$ we introduce a variable x_{ij} taking values in $[0,1]$, while for each node v we introduce variable x_v (taking values in $[0,1]$). We have the following linear program:

$$\max \quad \sum_{ij \in E} x_{ij} \qquad \text{(BasicLP)}$$

$$\text{s.t.} \quad x_{ij} \le y_i \qquad \forall ij \in E \qquad (2)$$

$$x_{ij} \le y_j \qquad \forall ij \in E \qquad (3)$$

$$\sum_{i \in V} y_i \le 1 \qquad (4)$$

$$x_{ij}, y_i \ge 0 \qquad \forall i,j. \qquad (5)$$

To gain an intuition about the above LP, consider a densest 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:

$$x^S_{ij} = \begin{cases} \frac{1}{|S|} & \text{if both } i,j \in S \\ 0 & \text{otherwise.} \end{cases}$$

$$y^S_i = \begin{cases} \frac{1}{|S|} & \text{if } i \in S \\ 0 & \text{otherwise.} \end{cases}$$

Recall that the density of the subgraph $H = (S, E(S))$ is defined as $\rho(H) = \rho(S) = \frac{|E(S)|}{|S|}$. Observe that z^S is feasible for the basic LP, and has objective value $\rho(S)$. Vice versa, given an optimum solution $z^S = (x^S, y^S)$ for BasicLP, one can construct an optimum solution for the densest subgraph problem using the *rounding* algorithm described in [9]: We first order the y^S_i's by non-increasing order. Let $y_1, \ldots, y_n$ be the variables so ordered. We then find the prefix $y_1, \ldots, y_k$ in such ordering whose corresponding induced subgraph H achieves maximum density, for any value of k in $[2,n]$. It can be proved that H is a densest subgraph.

One can then solve BasicLP using efficient LP solvers such as Gurobi or CPLEX and then compute the densest subgraph by running the algorithm described above. In the rest of this section, we present a more efficient algorithm for computing the densest subgraph, which allows us to deal with large real-world graphs containing millions of edges. The latter algorithm uses the following fact.

Lemma 4.1 *Each optimal solution of BasicLP is a convex combination of points in $\{z^S : S \subseteq V,\ S \text{ is densest subgraph}\}$.*

Proof. Suppose $z^* = (x^*, y^*)$ is an optimal solution. Then, since the objective value to be maximized is the sum of all x_{ij}'s, it follows that if z^* is optimal, it must be the case that $\sum_{i \in V} y^*_i = 1$, and for all $ij \in E$, $x^*_{ij} = \min\{y^*_i, y^*_j\}$.

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 $k \geq 2$. For the base case $k = 2$, suppose $\{i, j\}$ is the support of y^*. Since $x^*_{ij} = \min\{y^*_i, y^*_j\}$ is maximized when $y^*_i = y^*_j = \frac{1}{2}$, the result follows.

For the inductive step, suppose y^* has $k > 2$ non-zero coordinates corresponding to some subset $S \subseteq V$, where $k = |S|$. If all the non-zero y^*_i's are the same for $i \in S$, then it follows that $z^* = z^S$, and the result follows; otherwise, let $\alpha := \min\{y^*_i : i \in S\}$. Observe that $k\alpha \leq 1$.

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

$$\widehat{x}_{ij} = \begin{cases} \frac{x^*_{ij} - \alpha}{1-k\alpha} & \text{if both } i, j \in S \\ 0 & \text{otherwise.} \end{cases}$$

$$\widehat{y}_i = \begin{cases} \frac{y^*_i - \alpha}{1-k\alpha} & \text{if } i \in S \\ 0 & \text{otherwise.} \end{cases}$$

Hence, $z^* = k\alpha \cdot z^S + (1-k\alpha)\widehat{z}$, 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, $p(z^*) = k\alpha \cdot p(z^S) + (1 - k\alpha)p(\widehat{z})$, 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 $ij \in E(S)$, $\widehat{x}_{ij} = \frac{x^*_{ij} - \alpha}{1-k\alpha} = \frac{\min\{y^*_i, y^*_j\} - \alpha}{1-k\alpha} = \min\{\widehat{y}_i, \widehat{y}_j\}$.

Moreover, $\sum_{i \in V} \widehat{y}_i = \sum_{i \in S} \frac{y^*_i - \alpha}{1-k\alpha} = \frac{\sum_{i \in S} y^*_i - k\alpha}{1-k\alpha} = 1$, because $\sum_{i \in S} y^*_i = 1$. Therefore, $\widehat{z}$ is feasible and this completes the inductive step. □

The following corollary follows from Lemma 4.1.

Corollary 4.1 *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.*

Another interesting consequence of Lemma 4.1 is that we can find a densest subgraph by first solving BasicLP, and then returning the subgraph consisting of all nodes whose corresponding variables have values strictly larger than zero in our solution.

We now focus on computing a minimal densest subgraph H of a graph G given in input. We recall that any (proper) subgraph of H must have density strictly smaller than H.

As we shall use an LP-based approach to find a densest subgraphs, we start by showing how to speed up the LP-based algorithm presented in [9]. As proved in Lemma 4.1, the subgraph induced by variables with maximum value in an optimum solution for BasicLP is a densest subgraph. Starting from a solution for BasicLP we can then derive a densest subgraph in $O(n)$, in contrast with the rounding algorithm described in [9], which requires $\Omega(n \log n + m)$ operations, where n, m are the number of nodes and edges in G, respectively. To the best of our knowledge, this fact was not known before. We shall refer to this more efficient algorithm for the densest subgraph as FastLP.

Our algorithm for finding minimal dense subgraphs employs two subroutines: TryRemove and TryEnhance. The former one takes as input a graph G and a node u and checks whether u can be removed from G without decreasing its density. This is done by computing a densest subgraph in the input graph $(V \setminus \{u\}, E)$ and checking whether the density drops. If this is not the case, u can be removed from G. A pseudocode for TryRemove is shown in Algorithm 1.

Algorithm 1 TryRemove(u, G)

```
1: Input: A graph $G = (V, E)$ and a node $u$ to be removed.
2: Output: Returns a densest subgraph in $G$ not containing $u$, or null if every densest subgraph in $G$ must contain $u$.
3: Solve the BasicLP with input $(V \setminus \{u\}, E)$ and run FastLP to find a densest subgraph $H$ in $(V \setminus \{u\}, E)$.
4: if $\rho(H) \geq \rho(G)$ then
5:     return $H$
6: else
7:     return null
8: end if
```

We could then compute H by iterating through all nodes and checking for each such a node whether it can be removed or not. This algorithm would not be efficient, in that, it requires to solve $\Theta(n)$ LPs, one for each node of G. Therefore, we devise a much more efficient algorithm which requires to solve $O(\log n)$ LPs, with high probability. Such an algorithm employs the subroutine TryEnhance, which receives in input a graph G, a node u, as well as density $\rho_{\max}$ of a densest subgraph in G. It returns a densest subgraph in G which contains u while having smallest number of nodes (or null in case there is no densest subgraph containing u).

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 $\frac{1}{|S|}$. Therefore, by maximizing y_u we can find a densest subgraph containing u with smallest number of nodes. This is proved in Lemma 4.2 and follows partially from Lemma 4.1. See Algorithm 2 for a pseudocode of TryEnhance.

Algorithm 2 TryEnhance$(u, G, \rho_{\max})$

```
1: Input: a graph $G = (V, E)$, a node $u \in V$, the density $\rho_{\max}$ of the densest subgraph in $G$.
2: Output: Returns a densest subgraph in $G$ containing $u$ with minimum cardinality or null if there is no densest subgraph containing $u$.
3: Modify the basic LP by adding the constraint $\sum_{ij \in E} x_{ij} = \rho_{\max}$, maximizing the objective function $y_u$; solve the modified LP by running FastLP.
4: If there is no feasible solution to the modified LP then return null.
5: Run FastLP to find a densest subgraph $H = (\bar{V}, \bar{E})$ starting from the LP solution.
6: return $H$
```

Lemma 4.2 *Given a graph $G = (V, E)$ and a node $u \in V$, the subroutine* TryEnhance *returns a densest subgraph in G containing u with the smallest number of nodes.*

Proof. The smallest densest subgraph containing u is unique, because by Corollary 4.1, 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 4.1, z is a convex combination of z^{S_i}'s, 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's must contain u. Since for each S_i containing u, $y_u^{S_i} = \frac{1}{|S_i|}$, it follows that if z is a convex combination of more than one S_i's, 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 subset containing u. □

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 graph, uniformly at random; 2) we execute the subroutines TryRemove and TryEnhance with input u and our current graph; 3) our current graph is then set to be the smallest subgraph 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 4.3, while Lemma 4.3 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 Charikar in [9] so to find a 2-approximation solution to the densest subgraph. Let ρ_{apx} be the density of the graphs so found. As noted in [17], no nodes with degree smaller than the density of the densest subgraph belongs to any densest subgraph. Therefore, we can safely remove all nodes with degree smaller than ρ_{apx} from the input graph. A pseudocode of our algorithm is shown in Algorithm 3.

Algorithm 3 FindMinimal(G)

```
1: Input: A graph $G = (V, E)$ with $n$ nodes.
2: Output: A minimal densest subgraph in $G$.
3: Run the greedy algorithm to find a 2-approximation solution for the densest subgraph. Let $\rho_{apx}$ be the density of such a subgraph.
4: Remove iteratively nodes with degree smaller than $\rho_{apx}$ and let $\bar{G}$ be the graphs so obtained.
5: Find a densest subgraph $H = (\bar{V}, \bar{E})$ in $\bar{G}$ by running FastLP. Let $\rho_{\max}$ be its density.
6: while (true) do
7:     let $u$ be a node picked uniformly at random from $\bar{V}$
8:     let $H_1 :=$ TryRemove$(u, H)$
9:     let $H_2 :=$ TryEnhance$(u, H, \rho_{\max})$
10:    IF $H_1$ is null return $H_2$
11:    let $\widehat{H}$ be the subgraph with minimum number of nodes between $H_1$ and $H_2$, breaking ties arbitrarily
12:    $H := \widehat{H}$
13: end while
14: return $H$
```

Lemma 4.3 FindMinimal(G) *returns a minimal densest subgraph of* G.

Proof. Algorithm 3 always terminates. After each iteration of the *while* loop, either TryRemove(u, H) returns *null* and therefore the algorithm terminates or the node u is removed from the current graph. Hence, at each iteration the number of nodes of the current graph decreases by at least one. Observe that the graph 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) returns *null*, for some node u in H. This means that every densest subgraph of H must contain u. By Lemma 4.2, TryEnhance($u, H, \rho_{\max}$) returns the smallest densest subgraph containing u, 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) and $H_2 :=$ TryEnhance($u, H, \rho_{\max}$) contain more than $(1-\epsilon)|\bar{V}|$ nodes, and neither of them is *null*.

Lemma 4.4 *Given a graph* $H = (\bar{V}, \bar{E})$ *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). Observe that $u \in A_u$, and by the definition of ϵ-bad, $|A_u| < \epsilon|\bar{V}|$.

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

Observe that there exists $\widehat{u} \in B \setminus (\cup_{j=1}^{i} A_{u_j})$. Since for each j, TryRemove(u_j, H) is a densest subgraph, then by Lemma 4.1, $\cap_{j=1}^{i}$TryRemove$(u_j, H) = \bar{V} \setminus (\cup_{j=1}^{i} A_{u_j})$ 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 4.4, we have the following corollary.

Corollary 4.2 *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 4.3 *By standard Chernoff bound, the number of iterations in* FindMinimal(G) *is at most a constant times its mean, which is at most* $2\log_{\frac{4}{3}} n$*, where* n *is the number of nodes in* G.

Our algorithm FindMinimal(G) allows us to compute minimal densest subgraphs in graphs 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.

4.1.1 Finding All Minimal Densest Subgraphs

Armed with an efficient algorithm for finding minimal densest subgraphs, we now present an algorithm that computes all such graphs. Our algorithm computes at each step a minimal densest subgraph by running Algorithm 3, it removes all its nodes and edges from the current graph and

Algorithm 4 FindAllMinimal(V)

```
1: Input: Graph $G = (V, E)$.
2: Output: Returns a list $L$ of all minimal densest subgraphs.
3: $L := \emptyset$
4: while (true) do
5:     $\widehat{H} = $ FindMinimal($G$)
       IF $\widehat{H}$ is not a densest subgraph break;
6:     $L := L \cup \{\widehat{H}\}$
7:     remove all nodes and edges incident to $\widehat{H}$ from $G$
8: end while
9: return $L$
```

iterates until no densest subgraph can be found. A pseudocode is shown in Algorithm 4.

Minimal densest subgraphs must be disjoint, for otherwise, their intersection would be a densest subgraph (from Corollary 4.1), which would contradict the fact that they are minimal. Lemma 4.5 then follows.

Lemma 4.5 *Given an undirected graph $G = (V, E)$, 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 graph $G = (V, E)$ and we remove $\lceil(1-\alpha)|V_i|\rceil$ nodes (and their edges) from V. 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 any node v in V_i let $\Delta_G(v)$ be the set of neighbors of v in G. We remove the $\lceil(1-\alpha)\rceil|V_i|$ nodes (and their edges) with minimum value $|\Delta_G(v) \setminus V_i|$. We iterate until k subgraphs are found or the current graph becomes empty. Observe that the constraint on the Jaccard coefficient is not violated. A pseudocode of our algorithm is shown in Algorithm 5.

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 4.5.

Theorem 4.1 *If there are k disjoint densest subgraphs in the input graph G, then* MinAndRemove *computes an optimum 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 FastDSLO for finding dense subgraphs with limited overlap. Our heuristic computes at each step i a 2-approximation solution G_i to the densest subgraph problem using the greedy algorithm presented in [9]. Then, similarly to MinAndRemove $\lceil(1-\alpha)\rceil|V_i|$ nodes are removed from the current graph so to satisfy the requirement on the pairwise 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 graphs with more than 100 million edges within a few hours.

Algorithm 5 MinAndRemove(G, k, α)

```
1: Input: A graph $G = (V, E)$, an integer $k > 0$, $\alpha \in [0, 1]$
2: Output: A list $L$ of at most $k$ subgraphs of $G$, $G_i = (V_i, E_i)$, s.t. the constraint on the pairwise Jaccard coefficient on the $V_i$'s is not violated (Equation (1)).
3: $L := \emptyset$
4: while $< k$ subgraphs are found and $G$ is not empty do
5:     Find a minimal densest subgraph $G_i = (V_i, E_i)$ of $G$ by running Algorithm 3
6:     $L := L \cup \{G_i\}$
7:     For each node $v$ in $V_i$, let $\Delta_G(v)$ be the set of neighbors of $v$ in $G$.
8:     Remove the $\lceil(1-\alpha)|V_i|\rceil$ nodes with minimum value $|\Delta_H(v) \setminus V_i|$ and all their edges from $G$.
9: end while
10: return L
```

Algorithm 6 FastDSLO(G, k, α)

```
1: Input: A graph $G = (V, E)$, an integer $k > 0$, $\alpha \in [0, 1]$
2: Output: A list $L$ of at most $k$ subgraphs of $G$, $G_i = (V_i, E_i)$, s.t. the constraint on the pairwise Jaccard coefficient on the $V_i$'s is not violated (Equation (1)).
3: $L := \emptyset$
4: while $< k$ subgraphs are found and $G$ is not empty do
5:     Find a 2-approximation solution $G_i = (V_i, E_i)$ to the densest subgraph problem by running the greedy algorithm in [9].
6:     $L := L \cup \{G_i\}$
7:     For each node $v$ in $V_i$, let $\Delta_G(v)$ be the set of neighbors of $v$ in $G$.
8:     Remove the $\lceil(1-\alpha)|V_i|\rceil$ nodes with minimum value $|\Delta_H(v) \setminus V_i|$ and all their edges from $G$.
9: end while
10: return L
```

5. EXPERIMENTS

We perform our experimental evaluation on a Linux server with Intel Xeon E7-4870 at 2.40GHz, while limiting the total amount of main memory available to 64 GB. We solve linear programs with the Gurobi Optimizer version 5.6.3. All our algorithms are implemented in Java.

We consider 8 datasets in total grouped according to their size[1]. We ignore the direction of the edges in the directed graphs, as it is irrelevant to our purposes. Most of our experiments are conducted on the datasets that contain up to 11 million edges, as illustrated in Table 1. We refer to this set of datasets as *large* datasets.

Name	Nodes	Edges	Description
web-Stanford	281K	1.9M	Hyperlink network
com-Youtube	1.1M	3M	Social network
web-Google	875K	4.3M	Hyperlink network
Youtube-growth	3.2M	9.3M	Social network
As-Skitter	1.69M	11M	Internet topology

Table 1: Large real-world datasets.

The second group of datasets consists of *very large* datasets containing up to more than 100 million edges where we evaluate our fast heuristic. In Table 2 we specify all the details for this group of datasets. We refer to this set of datasets as *very large* datasets.

Name	Nodes	Edges	Description
Live Journal	4.8M	43M	Social network
Hollywood-2009	1.1M	57M	Social network
Orkut	3M	117M	Social network

Table 2: Very large real-world datasets.

We evaluate our algorithm 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 [9], 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 $\frac{1}{2}$-approximation algorithm proposed in [9] instead of a densest subgraph. We refer to this algorithm as NaiveGreedy. The reason to consider also this second variant is that the $\frac{1}{2}$-approximation algorithm is much faster than the LP-based optimal approach.

We tested NaiveDensest on the large datasets and it did not terminate after 16 hours of computation on any dataset. We also evaluated the maximum flow algorithm proposed in [15], which turned out to be even slower. Therefore, in the rest of the experimental evaluation we focus on evaluating MinAndRemove against NaiveGreedy.

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 3, 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 0.44 of the value of an optimum solution, while in many cases it yields an approximation factor of 0.8 (meaning that it reaches the 80% of the upper bound on the optimum objective-function value). We can also see that the quality of the solution increases as a function of α showing that our algorithm takes full advantage of the overlap between the subgraphs. This is not the case for one dataset only ($web - Google$), however, we observe that the results are already very good for this dataset when $\alpha = 0.1$, making it harder to improve upon such a solution. We also recall that our upper bound might be loose and that we might have computed an optimum solution for such a dataset.

Next, we evaluate our algorithm MinAndRemove against NaiveGreedy, when $k = 10$. Table 4 shows the ratio between the total density of the subgraphs found by our algorithm and those of NaiveGreedy. We can see that in most cases MinAndRemove yields a solution that is a factor of 1.5 larger than that of MinAndRemove, and always at least 10% denser. We expect that the advantage of MinAndRemove against NaiveGreedy 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 3.2 hours, while in many cases is as less as 30 mins. We recall that instead the basic LP-based approach by Charikar [9] could not terminate after 16 hours of computation on any of the selected datasets. On the other hand, as expected, NaiveGreedy is faster than our method. However, NaiveGreedy does not ensure optimality in finding the densest subgraph at each step, and this results in consistently less dense solutions.

In Table 6, 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 (up to a factor of 0.9). For larger values of k, our solution is still within a factor of 1/2 of an optimum solution or better. This might depend on the fact that our upper bound becomes loose when k is large.

We then evaluate the impact of computing at each step minimal densest subgraphs in MinAndRemove, as opposed to computing densest subgraphs. We perform the following experiment. Starting from the input graph, we remove at each step minimal densest subgraphs until the current graph is left with no edges. We then perform a similar experiment where at each step (possibly non-minimal) densest subgraphs are removed from the current graph. Our experiments show (which are omitted for lack of space) that in the former case we gain a factor of 1% (on average) in the total density, which is significant given that MinAndRemove might deliver near-optimal solutions. We also observe that all our datasets contain at most one densest subgraph. This fact could not be verified prior to our work.

Our experimental evaluation shows that MinAndRemove can handle large graphs containing up to more than 10 million edges within 3 hours or less, while delivering near-optimal solutions for our problem. For even larger graphs we resort to our fast heuristic which is evaluated on graphs

[1]Sources: snap.stanford.edu , konect.uni-koblenz.de, law.di.unimi.it/datasets.php

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	.717	.738	.767	.790	.816
com-Youtube	.480	.521	.518	.613	.623
web-Google	.808	.808	.808	.808	.808
Youtube-growth	.440	.467	.538	.593	.579
As-Skitter	.585	.597	.599	.625	.647

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	1.469	1.512	1.570	1.617	1.671
com-Youtube	1.192	1.293	1.287	1.522	1.547
web-Google	1.595	1.595	1.595	1.595	1.595
Youtube-growth	1.2	1.271	1.467	1.617	1.578
As-Skitter	1.125	1.147	1.151	1.202	1.244

Table 4: Ratio between the density of MinAndRemove and NaiveGreedy

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
web-Stanford	0.3h	0.21h	0.21h	0.23h	0.21h
com-Youtube	0.54h	0.54h	0.63h	0.55h	0.47h
web-Google	2.15h	2.31h	2.31h	2.12h	2.73h
Youtube-growth	1.5h	1.74h	2.19h	1.8h	3.13h
As-Skitter	1.29h	1.47h	2.85h	2.53h	1.78h

Table 5: Running time when varying α for MinAndRemove

$\alpha = 0.3$	$k = 2$	$k = 4$	$k = 6$	$k = 8$	$k = 10$
web-Stanford	.991	.914	.858	.808	.767
com-Youtube	.840	.744	.638	.570	.518
web-Google	.991	.915	.863	.836	.808
Youtube-growth	.845	.738	.664	.593	.538
As-Skitter	.914	.783	.693	.641	.599

Table 6: Ratio between the density of MinAndRemove and our upper bound

containing up to more than 100 million edges. Similarly to the previous case, we can derive an upper bound on the optimum solution as a function of the densest subgraph found by our heuristic. Namely, let ρ_A be the density of the densest subgraph found by FastDSLO (which we recall is a 2-approximation for the densest subgraph problem). Then, $2k \cdot \rho_A$ gives an upper bound to any optimum solution for (k, α)-DSLO.

In Table 7, we set $k = 10$ while we measure the ratio between the density of FastDSLO 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 very large graphs with more than 100 million edges within a few hours. We also observe that our upper bound might be even looser in this case, as it is based on an approximation of the density of a densest subgraph. Table 8 show the running time of FastDSLO when $k = 10$ and α varies between 0.1 and 0.5. We can observe that the running time is around half an hour for the smaller datasets and does not exceed 2.3 hours on the largest dataset (Orkut).

6. CONCLUSIONS

This paper studies the problem of finding at most k subgraphs from a large graph given in input such that the total density be maximized, while satisfying a constraint on the pairwise 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 graphs 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 and develop efficient algorithms for finding minimal densest subgraphs, which play an important role in solving our main problem. It turns out that our algorithm for finding minimal densest subgraphs is the fastest known algorithm for the exact computation of a densest subgraph, as shown by our experimental evaluation. We presented an efficient heuristic for our problem which, albeit less accurate, can handle graphs containing up to 100 million edges.

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 the current graph or removed. Another interesting direction is to determine whether finding minimal densest subgraphs can

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
LiveJournal	.244	.245	.251	.285	.276
Hollywood-2009	.187	.190	.199	.210	.231
Orkut	.187	.200	.218	.249	.271

Table 7: Ratio between the density of FastDSLO and our upper bound

$k = 10$	$\alpha = 0.1$	$\alpha = 0.2$	$\alpha = 0.3$	$\alpha = 0.4$	$\alpha = 0.5$
LiveJournal	0.56h	0.52h	0.63h	0.54h	0.54h
Hollywood-2009	0.34h	0.3h	0.33h	0.37h	0.34h
Orkut	1.97h	2.15h	1.2h	2.16h	2.26h

Table 8: Running time of FastDSLO on very large datasets as a function of α

be a valuable tool in finding interesting patterns in social networks and other real-world graphs.

7. REFERENCES

[1] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, 2009.

[2] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6), 2012.

[3] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6), 2012.

[4] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discr. Ap. Math.*, 121(1-3), 2002.

[5] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *J. Algorithms*, 34(2), 2000.

[6] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5), 2012.

[7] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

[8] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, 2014.

[9] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In K. Jansen and S. Khuller, editors, *APPROX*. Springer, 2000.

[10] J. Chen and Y. Saad. Dense subgraph extraction with application to community detection. *TKDE*, 24(7), 2012.

[11] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, 2013.

[12] X. Du, R. Jin, L. Ding, V. E. Lee, and J. H. Thornton, Jr. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *KDD*, 2009.

[13] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. MotifCut: regulatory motifs finding with maximum density subgraphs. In *ISMB*, 2006.

[14] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *VLDB*, 2005.

[15] A. V. Goldberg. Finding a maximum density subgraph. Technical report, University of California at Berkeley, 1984.

[16] S. Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *J. Computing*, 36(4), 2006.

[17] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, 2009.

[18] M. A. Langston and et al. 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*. 2005.

[19] V. E. Lee, N. Ruan, R. Jin, and C. C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*. 2010.

[20] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3), 1991.

[21] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[22] N. Tatti and A. Gionis. Discovering nested communities. In *ECML/PKDD (2)*, 2013.

[23] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *KDD*, 2013.

[24] E. Valari, M. Kontaki, and A. N. Papadopoulos. Discovery of top-k dense subgraphs in dynamic graph collections. In *SSDBM*, 2012.

[25] N. Wang, J. Zhang, K.-L. Tan, and A. K. H. Tung. On triangulation-based dense neighborhood graph discovery. *PVLDB*, 4(2), 2010.