Machine Learning (2024) 113:1967–1987
https://doi.org/10.1007/s10994-023-06377-y

Dense subgraphs induced by edge labels

Iiro Kumpulainen[1] · **Nikolaj Tatti**[1]

Received: 6 March 2023 / Revised: 29 May 2023 / Accepted: 17 July 2023 /
Published online: 6 September 2023

Abstract
Finding densely connected groups of nodes in networks is a widely-used tool for analysis in graph mining. A popular choice for finding such groups is to find subgraphs with a high average degree. While useful, interpreting such subgraphs may be difficult. On the other hand, many real-world networks have additional information, and we are specifically interested in networks with labels on edges. In this paper, we study finding sets of labels that induce dense subgraphs. We consider two notions of density: average degree and the number of edges minus the number of nodes weighted by a parameter α. There are many ways to induce a subgraph from a set of labels, and we study two cases: First, we study conjunctive-induced dense subgraphs, where the subgraph edges need to have all labels. Secondly, we study disjunctive-induced dense subgraphs, where the subgraph edges need to have at least one label. We show that both problems are **NP**-hard. Because of the hardness, we resort to greedy heuristics. We show that we can implement the greedy search efficiently: the respective running times for finding conjunctive-induced and disjunctive-induced dense subgraphs are in $\mathcal{O}(p \log k)$ and $\mathcal{O}\left(p \log^2 k\right)$, where p is the number of edge-label pairs and k is the number of labels. Our experimental evaluation demonstrates that we can find the ground truth in synthetic graphs and that we can find interpretable subgraphs from real-world networks.

Keywords Dense subgraphs · Convex hull · Label-induced subgraphs

1 Introduction

Finding dense subgraphs in networks is a common tool for analyzing networks with potential applications in diverse domains, such as bioinformatics (Fratkin et al., 2006; Langston et al., 2005), finance (Du et al., 2009), social media (Angel et al., 2014), or web graph analysis (Fratkin et al., 2006).

Editor: Dino Ienco, Robert Interdonato, Pascal Poncelet.

✉ Iiro Kumpulainen
iiro.kumpulainen@helsinki.fi

✉ Nikolaj Tatti
nikolaj.tatti@helsinki.fi

[1] HIIT, University of Helsinki, Helsinki, Finland

While useful on their own, analyzing dense subgraphs without any additional explanation may be difficult for domain experts and consequently may limit its usability.

Fortunately, it is often the case that the network has additional information such as labels associated with nodes and/or edges. For example, in social networks, users may have tags describing themselves. In networks arising from communication, for example, by email or Twitter, the tags associated with the edge can be the tags associated or extracted with the message.

Using the available label information to provide explainable dense subgraphs may ease the burden of domain experts when, for example, studying social networks. In this paper, we consider finding dense subgraphs in networks with labeled edges. More formally, we are looking for a label set that *induces* a dense subgraph. As a measure of density, a subgraph (W, F) with nodes W and edges F will use $|F|/|W|$, the ratio of edges over the nodes, a popular choice for measuring the density of a subgraph.

We consider two cases: conjunctive-induced and disjunctive-induced dense subgraphs. In the former, the induced subgraph consists of all the edges that have the given label set. In the latter, the induced subgraph consists of all the edges that have at least one label common with the label set. We give an example of both cases in Fig. 1.

Finding the densest subgraph—with no label constraints—can be done in polynomial time (Goldberg et al., 1984) and can be 2-approximated in linear time (Charikar, 2000). Unfortunately, additional requirements on the labels will make solving the optimization problem exactly computationally intractable: we show that both problems are **NP**-hard, which forces us to resort to heuristics. We propose a greedy algorithm for both problems: we start with an empty label set and keep adding the best possible label until no additions are possible. We then return the best observed induced subgraph.

The computational bottleneck of the greedy method is selecting a new label. If done naively, evaluating a single label candidate requires enumerating over all the edges. Since this needs to be done for every candidate during every addition, the running time is $\mathcal{O}(p|L|)$, where $|L|$ is the number of labels and p is the number of edge-label pairs in the network. By keeping certain counters we can speed up the running time. We show that conjunctive-induced graphs can be discovered in $\mathcal{O}(p \log |L|)$ time using a balanced search tree, and that disjunctive-induced graphs can be discovered in $\mathcal{O}\left(p \log^2 |L|\right)$ time with the aid of an algorithm originally used to maintain convex hulls.

This is an extended version of our previously published conference paper (Kumpulainen & Tatti, 2022). We extend our earlier work by considering an alternative definition of density: namely, we search for label-induced subgraphs (W, F) with high α-density $|F| - \alpha|W|$. This density is closely related to the problem of finding a subgraph with maximum density (Goldberg et al., 1984) but also has been used to decompose graphs (Tatti, 2019;

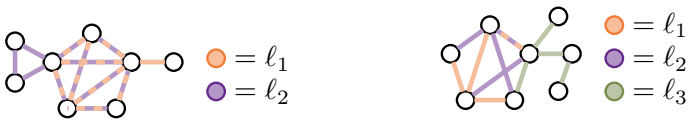

Fig. 1 Example graphs with labels on the edges. Edge labels are indicated by colors; dashed edges indicate edges with 2 labels. Left figure: Label ℓ_1 induces a subgraph with 6 nodes and 9 edges, and ℓ_2 induces a subgraph with 7 nodes and 11 edges, while the conjunction of ℓ_1 and ℓ_2 induces a subgraph with 5 nodes and 8 edges resulting in the highest density of $8/5 = 1.6$. Right figure: Labels ℓ_1, ℓ_2, and ℓ_3 each induce subgraphs with 5 nodes and 4 edges, while the disjunction of ℓ_1 and ℓ_2 induces a subgraph with 5 nodes and 7 edges, resulting in a density of $7/5 = 1.4$. The colours are used to indicate edge labels. For example, orange edges have one label while purple edges have another label, and dashed edges with both orange and purple colours have both labels

Danisch et al., 2017). We show that there are α such that the α-densest label-induced graph also has the highest density. We then modify the greedy algorithms to find subgraphs with high α-density in $\mathcal{O}(p \log |L|)$ for both the conjunctive and disjunctive cases.

The remainder of the paper is organized as follows. In Sect. 2 we introduce the notation and formalize the optimization problem. In Sects. 3–4 we present our algorithms. In Sect. 5, we analyze the case of using an alternative density metric and adapt the previous algorithms to this problem. Section 6 is devoted to the related work. Finally, we present the experimental evaluation in Sect. 7 and conclude with a discussion in Sect. 8. The computational complexity proofs are given in Appendix 1.

2 Preliminary notation and problem definition

In this section, we first describe the common notation and then introduce the formal definition of our problem.

Assume that we are given an *edge-labeled graph*, that is, a tuple $G = (V, E, lab)$, where V is the set of vertices, $E \subseteq \{(x, y) \mid (x, y) \in V^2, x \neq y\}$ is the set of undirected edges, and $lab : E \to 2^L$ is a function that maps each edge $e \in E$ to the set of labels $lab(e)$. Here L is the set of all possible labels.

Given a label $\ell \in L$, let us write $E(\ell)$ to be the edges having the label ℓ. In addition, let us write $V(\ell)$ to be the nodes adjacent to $E(\ell)$.

Our goal is to search for dense regions of graphs that can be explained using the labels. In other words, we are looking for a set of labels that induce a dense graph. More formally, we define an *inducing* function to be a function f that maps two sets of labels to a binary number. An example of such a function could be $f(A;B) = [B \subseteq A]$ which returns 1 if and only if B is a subset of A.

Given a set of labels $B \subseteq L$, an inducing function f, and a graph G, we define the *label-induced subgraph* $H = G(f, B)$ as $(V(B), E(B), lab)$, where

$$E(B) = \{e \in E \mid f(lab(e);B) = 1\}$$

is the subset of edges that satisfy f, and $V(B)$ is the set of vertices that are adjacent to $E(B)$.

Given a graph G with vertices V and edges E, we measure the *density* of the graph $d(G)$ as the number of edges divided by the number of vertices: $d(G) = \frac{|E|}{|V|}$.

We should point out that there are alternative choices for a notion of density. For example, one option is to consider a fraction of edges $|E|/\binom{|V|}{2}$. However, this measure is however problematic since a single edge will yield a maximum value. Consequently, either a size needs to be incorporated into the objective, which leads to discovering maximum cliques—an **NP**-hard problem with bad approximation guarantees (Håstad, 1996), or enumerating all pseudo-cliques with an exponential-time algorithm (Uno, 2010; Abello et al., 2002). On the other hand, finding graph with maximum $d(G)$ can be done in polynomial time (Goldberg et al., 1984), and 2-approximated in linear time (Charikar, 2000). See related work for additional discussion.

We are now ready to state our generic problem.

Problem 1 (LD) Let $G = (V, E, lab)$ be an edge-labeled graph over a set of labels L with multiple labels being possible for each edge. Assume an inducing function f. Find a set of labels L^* such that the density $d(H)$ of the label-induced subgraph $H = G(f, L^*)$ is maximized.

We consider two special cases of LD. Firstly, let us define $f_{AND}(A;B) = [B \subseteq A]$, that is, the induced edges need to contain every label in B. We will denote the problem LD paired with f_{AND} as LDAND. Secondly, we define $f_{OR}(A;B) = [B \cap A \neq \emptyset]$, that is, the induced edges need to have one common label with B. Then, we denote the corresponding problem as LDOR. In other words, LDAND is the problem of finding dense conjunctive-induced subgraphs, and LDOR is the problem of finding disjunctive-induced subgraphs.

In addition, we consider an alternative measure to the density $d(G)$ of a graph by instead measuring the *α-density* of the graph $g(G;\alpha)$ as the number of edges minus α times the number of vertices: $g(G;\alpha) = |E| - \alpha|V|$. This measure is closely related to finding the densest subgraph: Goldberg et al. (1984) finds a series of α-densest subgraphs when searching for the densest subgraph. However, this measure has been also studied on its own as it can be used to decompose graphs (Tatti, 2019; Danisch et al., 2017). Our optimization problem is as follows.

Problem 2 (LD-α) Let $G = (V, E, lab)$ be an edge-labeled graph over a set of labels L with multiple labels being possible for each edge. Assume an inducing function f and a constant $\alpha \in \mathbb{R}$. Find a set of labels L^* such that the α-density $g(H;\alpha)$ of the label-induced subgraph $H = G(f, L^*)$ is maximized.

3 Finding dense conjunctive-induced graphs

In this section, we focus on LDAND, that is, finding conjunctive-induced graphs that are dense. We will first prove that LDAND is **NP**-hard.

Theorem 1 *LDAND is* **NP***-hard.*

Proof The proof is given in Appendix 1. □

The **NP**-hardness forces us to resort to heuristics. Here, we use the algorithm for 2-approximating dense subgraphs (Charikar, 2000) as a starting point. The algorithm iteratively removes a node with the smallest degree, and returns the best solution among the observed subgraphs. We propose a similar greedy algorithm, where we greedily add the best possible label, and repeat until the induced subgraph is empty. We then select the best observed labels as the output.

To avoid enumerating over the edges every time we look for a new label, we maintain several counters. Let A be the current set of labels. For each label, we maintain the number of nodes n_k and edges m_k of the candidate graph, that is, $n_k = |V(A \cup \{k\})|$ and $m_k = |E(A \cup \{k\})|$. We store the densities m_k/n_k in a balanced search tree (for example, a red-black tree), which allows us to obtain the largest element quickly. Once we update set A, we also update the counters and update the search tree. Maintaining the node counts n_k requires us to maintain the counters $r_{v,k}$, number of edges labeled as k adjacent to v: once the counter reduces to 0, we reduce n_k by 1. The pseudo-code of the algorithm is given in Algorithm 1.

Algorithm 1: GreedyAnd, greedy search for the conjunctive-induced dense subgraphs

```
1  n_ℓ ← |V(ℓ)|, for each label ℓ ∈ L;
2  m_ℓ ← |E(ℓ)|, for each label ℓ ∈ L;
3  r_{v,ℓ} ← |{e ∈ E(ℓ) | e is adjacent to v}|, for each vertex v and label ℓ;
4  T ← labels sorted by the density values m_ℓ/n_ℓ (e.g., in a red-black tree);
5  A_0 ← ∅ and i ← 0;
6  while there are labels do
7      pick and remove label k that has the maximum density in T;
8      A_{i+1} ← A_i ∪ {k};
9      for each edge e without label k do
10         for each label ℓ of edge e = (u, v) do
11             m_ℓ ← m_ℓ − 1;
12             r_{v,ℓ} ← r_{v,ℓ} − 1; r_{u,ℓ} ← r_{u,ℓ} − 1;
13             if r_{v,ℓ} = 0 then n_ℓ ← n_ℓ − 1;
14             if r_{u,ℓ} = 0 then n_ℓ ← n_ℓ − 1;
15         remove edge e;
16     update T for all labels ℓ with changed values of m_ℓ or n_ℓ;
17     i ← i + 1;
18 return the set of labels A_i that yields the highest density;
```

We conclude with an analysis of the computational complexity of GreedyAnd.

Theorem 2 *GreedyAnd runs in $\mathcal{O}(p \log |L| + |V| + |E|)$ time, where p is the number of edge-label pairs $p = |\{(e,k) \mid e \in E, k \in lab(e)\}|$.*

Proof Initializing counters in GreedyAnd can be done in $\mathcal{O}(|V| + |E| + |L|)$ time while initializing the tree can be done in $\mathcal{O}(|L| \log |L|)$ time.

Let us consider the inner for-loop. Since an edge is deleted once it is processed, the inner for-loop is executed at most p times during the search. Since this is the only way the counters get updated, the tree T is updated p times, each update requiring $\mathcal{O}(\log |L|)$ time.

The outer loop is executed at most $|L|$ times. During each round, selecting and removing the label requires $\mathcal{O}(\log |L|)$ time.

In summary, the algorithm requires

$$\mathcal{O}(|V| + |E| + |L| + |L| \log |L| + p \log |L|) \subseteq \mathcal{O}(|V| + |E| + p \log |L|)$$

time, completing the proof. □

4 Finding dense disjunctive-induced graphs

In this section, we focus on LDor, that is, finding disjunctive-induced graphs that are dense. We will first prove that LDor is **NP**-hard.

Theorem 3 *LDOR is* **NP***-hard.*

Proof The proof is given in Appendix 1. □

Similar to LDAND, we resort to a greedy search to find good subgraphs: We start with an empty label set, and iteratively add the best possible label. Once done, we return the best observed label set.

However, we maintain a different set of counters as compared to GREEDYAND. The reason for having different counters is to avoid a significantly higher number of updates: the inner loop would need to go over the edge-label pairs that are *not* present in the graph. More formally, we maintain values n and m representing the number of nodes and edges in the subgraph induced by the current set of labels, say A. We also maintain n_k and m_k, the number of *additional* nodes and edges if k is added to A. At each iteration, we select the label optimizing $\frac{m+m_k}{n+n_k}$. We will discuss the selection process later. Once the label is selected, we update the counters m_k and n_k. To maintain n_k properly, we keep track of what nodes are already in $V(A)$, using an indicator r_v with $r_v = 1$ if $v \in V(A)$. The pseudo-code for the algorithm is given in Algorithm 2.

Algorithm 2: GREEDYOR, greedy search for the disjunctive-induced dense subgraphs

```
n ← 0; m ← 0;
n_ℓ ← |V(ℓ)|, for each label ℓ ∈ L;
m_ℓ ← |E(ℓ)|, for each label ℓ ∈ L;
S_v ← {ℓ ∈ L | there is an edge with label ℓ adjacent to v};
r_v ← 0, for each vertex v;
A_0 ← ∅ and i ← 0;
while there are labels do
    pick and remove label k that yields the maximum density (m+m_k)/(n+n_k);
    A_{i+1} ← A_i ∪ {k};
    for each edge e = (u, v) with label k do
        for each label ℓ of edge e = (u, v) do m_ℓ ← m_ℓ − 1 ;
        m ← m + 1;
        if r_v = 0 then
            for each label ℓ in S_v do n_ℓ ← n_ℓ − 1 ;
            n ← n + 1;
        if r_u = 0 then
            for each label ℓ in S_u do n_ℓ ← n_ℓ − 1 ;
            n ← n + 1;
        r_v ← 1; r_u ← 1;
        remove edge e;
    i ← i + 1;
return the set of labels A_i that yields the highest density;
```

During each iteration, we need to select the label maximizing $\frac{m+m_k}{n+n_k}$. We cannot use priority queues any longer since n and m change every iteration. However, we can speed up

the selection using a convex hull, a classic concept from computational geometry, see for example, (Li & Klette, 2011). First, let us formally define a lower-right convex hull.

Definition 1 Given a set of points $X = \{(x_i, y_i)\}$ in a plane, we define a *lower-right convex hull* $H = hull(H)$ to be a subset of X such that $q = (x_q, y_q) \in X$ is *not* in X if and only if there is a point $r = (x_r, y_r) \in H$ such that $x_q \leq x_r$ and $y_q \geq y_r$, or if there are two points $p, r \in H$ such that q is above or at the segment joining q and r.

If we were to plot X on a plane, then $hull(X)$ is the lower-right portion of the complete convex hull, that is, a set of points in X that form a convex polygon containing X. For notational simplicity, we will refer to $hull(X)$ as the convex hull. Note that if we order the points in $hull(X)$ by their x-coordinates, then the y-coordinates and the slopes of the intermediate segments are also increasing.

We will first argue that we only need to search the convex hull when looking for the optimal label.

Theorem 4 *Let X be a set of positive points (m_i, n_i), and let $H = hull(X)$ be the convex hull. Select $m, n \geq 0$. Then* $\max_{p \in X} \frac{m+m_i}{n+n_i} = \max_{p \in H} \frac{m+m_i}{n+n_i}$.

Proof Let $k = (m_k, n_k)$ be the optimal point in X. Assume that $k \notin H$. Assume that there is a point $q = (m_q, n_q)$ in H such that $m_q \geq m_k$ and $n_q \leq n_k$. Then $\frac{m+m_k}{n+n_k} \leq \frac{m+m_q}{n+n_q}$, so the point q is also optimal.

Assume there is no such point q. Then, the x-coordinate of point k falls between two consecutive points p and q in H, that is, $m_p < m_k < m_q$. Then k must be above the segment between p and q as otherwise, k would also be a part H. Therefore, the slope for the segment between p and k must be greater than the slope of the segment between p and q, and the slope for the segment between k and q must be smaller,

$$\frac{n_q - n_k}{m_q - m_k} \leq \frac{n_q - n_p}{m_q - m_p} \leq \frac{n_k - n_p}{m_k - m_p}. \tag{1}$$

Furthermore, since $k \notin H$, we must have $n_k > n_p$. By assumption, we also have $n_k < n_q$. In summary, we have $n_p < n_k < n_q$ and $m_p < m_k < m_q$, which means that the slopes in Eq. 1 are all positive. By taking the reciprocals this then gives,

$$\frac{m_q - m_k}{n_q - n_k} \geq \frac{m_q - m_p}{n_q - n_p} \geq \frac{m_k - m_p}{n_k - n_p}. \tag{2}$$

Denote then the objective value at point k by $c = \frac{m+m_k}{n+n_k}$. Let $x_1 = c(n + n_p) - m$. Then, the optimality of k implies $\frac{m+x_1}{n+n_p} = c \geq \frac{m+m_p}{n+n_p}$, which means $x_1 \geq m_p$. The definition of c leads to $m = c(n + n_k) - m_k$, which in turns leads to $x_1 = c(n_p - n_k) + m_k$. Solving for c we get $c = \frac{m_k - x_1}{n_k - n_p}$. Substituting $x_1 \geq m_p$ yields $c \leq \frac{m_k - m_p}{n_k - n_p}$, using Eq. 2 then yields $c \leq \frac{m_q - m_k}{n_q - n_k}$.

Next, let $x_2 = c(n_q - n_k) + m_k$ which means that $c = \frac{x_2 - m_k}{n_q - n_k}$. Now since $c \leq \frac{m_q - m_k}{n_q - n_k}$ we must have $x_2 \leq m_q$. Since $m_k = c(n + n_k) - m$, we also have $x_2 = c(n_q + n) - m$, yielding $c = \frac{m + x_2}{n + n_q} \leq \frac{m + m_q}{n + n_q}$, thus q is also optimal. □

Theorem 4 states that we need to only consider the convex hull H of the set $\{(m_i, n_i)\}$ when searching for the optimal new label. Note that H does not depend on n or m. Moreover, we can use the algorithm by Overmars and Van Leeuwen (1981) to maintain H as n_k and m_k are updated in $\mathcal{O}(\log^2 |L|)$ time per update. We will see that the number of needed updates is bounded by the number of edge-label pairs.

However, the convex hull can be as large as the original set, so our goal is to avoid enumerating over the whole set. To this end, we design a binary search strategy over the hull. We will first introduce two quantities used in our search.

Definition 2 Given two points $p, q \in hull(X)$, we define the inverse slope as $s(p, q) = \frac{m_q - m_p}{n_q - n_p}$ and the bias term as $b(p, q) = \frac{m_q n_p - m_p n_q}{n_q - n_p}$.

First, let us prove that both s and b are monotonically decreasing.

Lemma 1 *Let p, q, and r be three consecutive points in hull(X). Then we have $n \times s(q, r) + b(q, r) \leq n \times s(p, q) + b(p, q)$, for any $n \geq 0$.*

Proof The slope for the segment between p and q is less than or equal to the slope for the segment between q and r. Inversing the slopes leads to

$$s(q, r) = \frac{m_r - m_q}{n_r - n_q} \leq \frac{m_q - m_p}{n_q - n_p} = s(p, q).$$

By cross-multiplying, adding $m_q n_q - m_q n_p - m_q n_r + \frac{m_q n_p n_r}{n_q}$ to both sides, multiplying by $\frac{n_q}{(n_r - n_q)(n_q - n_p)}$, and simplifying, we get

$$b(q, r) = \frac{m_r n_q - m_q n_r}{n_r - n_q} \leq \frac{m_q n_p - m_p n_q}{n_q - n_p} = b(p, q).$$

Combining the two equations proves the claim. □

Next, we show the key necessary condition for the optimal point.

Lemma 2 *Let p, q, and r be 3 consecutive points in hull(X). Select $n, m \geq 0$. If q optimizes $\frac{m_q + m}{n_q + n}$, then $n \times s(q, r) + b(q, r) \leq m \leq n \times s(p, q) + b(p, q)$.*

Proof Since q is optimal, we have $\frac{m + m_p}{n + n_p} \leq \frac{m + m_q}{n + n_q}$. Solving for m gives us $m \leq n \frac{m_q - m_p}{n_q - n_p} + \frac{m_q n_p - m_p n_q}{n_q - n_p} = n \times s(p, q) + b(p, q)$. Similarly, due to optimality, $\frac{m + m_r}{n + n_r} \leq \frac{m + m_q}{n + n_q}$, and solving for m leads to $m \geq n \times s(p, q) + b(p, q)$, proving the claim. □

The two lemmas allow us to use binary search as follows. Given two consecutive points p and q we test whether $m \leq n \times s(p, q) + b(p, q)$. If true, then the optimal label

is q or to the right of q, if false, the optimal point is to the left of q. To perform the binary search, we can use directly the structure maintained by the algorithm by Overmars and Van Leeuwen (1981) since it stores the current convex hull in a balanced search tree. Moreover, the algorithm allows evaluating any function based on the neighboring points. Specifically, we can maintain s and b. In summary, we can find the optimal label in $\mathcal{O}(\log |L|)$ time.

Our next result formalizes the above discussion.

Theorem 5 *GreedyOr runs in* $\mathcal{O}\big(p \log^2 |L| + |V| + |E|\big)$ *time, where* p *is the number of edge-label pairs* $p = |\{(e,k) \mid e \in E, k \in lab(e)\}|$.

Proof The proof is similar to the proof of Theorem 2, except we have replaced a search tree with the convex hull structure by Overmars and Van Leeuwen (1981). The inner for-loops are evaluated at most $\mathcal{O}(p)$ times since an edge or a node is visited only once, and $\sum_v |S_v| \in \mathcal{O}(p)$. Maintaining the hull requires $\mathcal{O}\big(\log^2 |L|\big)$ time, and there are at most $\mathcal{O}(p)$ such updates. Searching for an optimal label requires $\mathcal{O}(\log |L|)$ time, and there are at most $|L|$ such searches. □

We should point out that a faster algorithm by Brodal and Jacob (2002) maintains the convex hull in $\mathcal{O}(\log |L|)$ time. However, this algorithm does not provide a search tree structure that we can use to search for the optimal addition.

5 Finding subgraphs with high α-density

In this section, we focus on the problem LD-α of finding subgraphs with high α-density.

The following classic result in fractional programming (Dinkelbach, 1967) shows how the problem of finding the maximum density subgraph reduces to maximizing the α-density of a subgraph for a large enough value of α. An immediate consequence of this result is that solving LD-α is **NP**-hard.

Theorem 6 *write* H_α *to be the solution to LD-*α*. There is* τ *such that* H_τ *also solves LD. Moreover, for any* $\alpha > \tau$*, the graph* H_α *either solves LD or is empty.*

Proof Let H^* be a solution to LD with $\sigma = d(H^*)$. Since there are a finite number of subgraphs, there is $\tau < \sigma$ such that any graph H with $d(H) \geq \tau$ has $d(H) = \sigma$.

Since $g(H^*;\tau) > 0$, we have $g(H_\tau;\tau) > 0$, or $\big|E(H_\tau)\big| - \tau\big|V(H_\tau)\big| > 0$ which implies $d(H_\tau) > \tau$. By definition of τ, the subgraph H_τ solves LD.

Similarly, for any $\alpha > \tau$, we have $g(H_\alpha;\alpha) \geq 0$. Consequently, either H_α is empty or $d(H_\alpha) \geq \alpha > \tau$, that is, H_α solves LD. □

Corollary 1 *LD-*α *is* **NP***-hard for both* f_{OR} *and* f_{AND}*. Moreover, both problems are inapproximable unless* $\mathbf{P} = \mathbf{NP}$.

Proof The proof is given in Appendix 1. □

To find solutions to LD-α in practice, we adapt the previous greedy algorithms to find subgraphs with high α-density. In the conjunctive case, we get the GreedyAnd-α algorithm

by simply changing the density on line 4 of Algorithm 1 from $\frac{m_\ell}{n_\ell}$ to $m_\ell - \alpha n_\ell$. This leads to the same computational complexity as for GreedyAnd.

In the disjunctive case, we again keep track of the counters to find the number of additional nodes and edges when a label is added to the current set of labels. However, the α -density to maximize now becomes $(m + m_k) - \alpha(n + n_k)$. As $m - \alpha n$ does not depend on the label, we only need to find the label k that maximizes $m_k - \alpha n_k$. We may thus use a balanced search tree as in the conjunctive case. The pseudo-code for this algorithm is given in Algorithm 3.

Algorithm 3: GreedyOr-α, greedy search for the disjunctive-induced subgraphs with high α-density

```
1  n ← 0; m ← 0;
2  n_ℓ ← |V(ℓ)|, for each label ℓ ∈ L;
3  m_ℓ ← |E(ℓ)|, for each label ℓ ∈ L;
4  S_v ← {ℓ ∈ L | there is an edge with label ℓ adjacent to v};
5  r_v ← 0, for each vertex v;
6  T ← labels sorted by the density values m_ℓ − αn_ℓ (e.g., in a red-black
     tree);
7  A_0 ← ∅ and i ← 0;
8  while there are labels do
9      pick and remove label k that has the maximum density in T;
10     A_{i+1} ← A_i ∪ {k};
11     for each edge e = (u, v) with label k do
12         for each label ℓ of edge e = (u, v) do m_ℓ ← m_ℓ − 1 ;
13         m ← m + 1;
14         if r_v = 0 then
15             for each label ℓ in S_v do n_ℓ ← n_ℓ − 1 ;
16             n ← n + 1;
17         if r_u = 0 then
18             for each label ℓ in S_u do n_ℓ ← n_ℓ − 1 ;
19             n ← n + 1;
20         r_v ← 1; r_u ← 1;
21         remove edge e;
22     update T for all labels ℓ with changed values of m_ℓ or n_ℓ;
23     i ← i + 1;
24 return the set of labels A_i that yields the highest density;
```

As GreedyOr-α does not need to use a convex hull but uses a balanced search tree instead, the running time becomes the same as for the conjunctive case.

Theorem 7 *GreedyAnd-α and GreedyOr-α run in $\mathcal{O}(p \log |L| + |V| + |E|)$ time, where p is the number of edge-label pairs $p = |\{(e, k) \mid e \in E, k \in lab(e)\}|$.*

Proof The proofs for both cases are virtually the same as the proof of Theorem 2. □

We conclude this section by considering the (lack of the) hierarchy property of α-density. Tatti (2019) showed that the subgraphs (without label constraints) optimizing $g(\cdot, \alpha)$ form a nested structure, that is, if we write H_α to be the optimal solution, then $H_\beta \subseteq H_\alpha$ for any $\beta > \alpha$. Such a decomposition may be useful as it partitions the nodes into increasingly dense regions. Unfortunately, this is not the case for us as shown in Fig. 2.

Interestingly enough, if we allow more flexible queries, we can show that we too obtain a nested structure. More formally, given a Boolean formula B we define $G(B)$ to be the subgraph consisting of edges whose labels satisfy B, and the incident vertices. Then the optimization problem would be to find the Boolean formula B maximizing $g(G(B);\alpha)$. We then have the following proposition.

Proposition 1 *Let H_α be a subgraph induced by a Boolean formula B_α that optimizes $g(\cdot;\alpha)$. Then $H_\alpha \subseteq H_\beta$ for any $\alpha > \beta$.*

Proof Assume otherwise. Write $X = H_\alpha \cup H_\beta$ and $Y = H_\alpha \cap H_\beta$. Note that X is induced by $B_\alpha \vee B_\beta$ and Y is induced by $B_\alpha \wedge B_\beta$. Then

$$g(X;\beta) - g(H_\beta;\beta) > g(X;\alpha) - g(H_\beta;\alpha) = g(H_\alpha;\alpha) - g(Y;\alpha) \geq 0,$$

where the last inequality is due to the optimality of H_α. Thus, $g(X;\beta) > g(H_\beta;\beta)$ violating the optimality of H_β. □

6 Related work

A closely related work to our method is an approach proposed by Galbrun et al. (2014). Here the authors search for multiple dense subgraphs that can be explained by conjunction on (or the majority of) the *node* labels. The authors propose a greedy algorithm for finding such subgraphs. Interestingly enough, the authors do not show that the underlying problem is **NP**-hard—although we conjecture that this is indeed the case—instead, they show that the subproblem arising from the greedy approach is an **NP**-hard problem.

Another closely related work is an approach proposed by Pool et al. (2014), where the authors search for dense subgraphs that can be explained by queries on the *nodes*. The quality of the subgraphs is a ratio S/C, where S measures the goodness of a subgraph using the edges within the subgraph as well as the cross-edges, and C measures the complexity of the query.

The major difference between our work and the aforementioned work is that our method uses labels on the edges. While conceptually a small difference, this distinction leads to different algorithms and different analyses of those algorithms. Moreover, we cannot apply directly the previously discussed methods to networks that only have labels on edges.

An appealing property of finding subgraphs that maximize $|E(W)|/|W|$, or equivalently an average degree, is that we can find the optimal solution in polynomial time (Goldberg et al., 1984). Furthermore, we can 2-approximate the graph with a simple linear algorithm (Charikar, 2000). The algorithm iteratively removes the node with the smallest degree and then selects the best available graph. This algorithm is essentially the same as the algorithm used to discover k-cores, subgraphs that have the *minimum* degree of at least k. The connection between the k-cores and dense subgraphs is further explored by Tatti

Fig. 2 Subgraphs with optimal α-density are not nested. Left figure: ℓ_1 is optimal for $\alpha = 3/4$ and ℓ_2 is optimal for $\alpha = 1/4$ when using f_{AND}. Right figure: ℓ_1 is optimal for $\alpha = 2.25$ and ℓ_2 is optimal for $\alpha = 1.75$ when using f_{OR}

(2019), where the dense subgraphs are extended to create an increasingly dense structure. A variant of a quality measure was proposed by Tsourakakis (2015), where the quality of the subgraph is the ratio of triangles over the vertices. In another variant by Bonchi et al. (2019), the edges were replaced with paths of at most length k. Finding such structures in labeled graphs poses an interesting line of future work.

While finding dense subgraphs is polynomial, finding cliques is an **NP**-hard problem with a very strong inapproximability bound (Håstad, 1996). Finding cliques may be impractical as they do not allow any absent edges. To relax the requirement, Abello et al. (2002) and Uno (2010) proposed searching for quasi-cliques, that is subgraphs with a high proportion of edges, $|E(W)|/\binom{|W|}{2}$. Another relaxation of cliques is k-plex where k absent edges are allowed for a vertex (Seidman, 1983). Finding k-plexes remain an **NP**-hard problem (Balasundaram et al., 2011). Alternatively, we can relax the definition by considering n-cliques, where vertices must be connected with an n-path (Bron & Kerbosch, 1973), or n-clans where we also require that the diameter of the graph is n (Mokken, 1979). Since 1-clique (and 1-clan) is a clique, these problems remain computationally intractable.

7 Experimental evaluation

In this section, we describe our experimental evaluation of the GreedyAnd and GreedyOr algorithms. First, we observe how the algorithms behave on synthetic data with increasing randomness. Then we apply the algorithms to real-world datasets and analyze the results.

We implement our algorithms in Python and the source code is available online.[1] Since the number of labels in our experiments was not exceedingly large, we did not use the speed up using convex hulls when implementing disjunctive-induced graphs. Instead, we search for the optimal label from scratch leading to a running time of $\mathcal{O}(p|L|)$.

Experiments with synthetic data: We evaluate the greedy algorithms on synthetic graphs of 200 vertices and 50 labels. We select 5 of the labels as target labels and construct graphs for the conjunctive and disjunctive cases such that selecting the subgraph induced by these 5 labels gives the best density. We then add random noise to the network by introducing a noise parameter ϵ, which controls the probability of randomly adding and removing edges as well as adding new labels to the edges.

For the conjunctive case, we create five disjoint cliques of 10 vertices such that all edges on the kth clique have all except the kth of the target labels. Finally, we add one more 20 vertex clique that has all of the target labels. Since each of the smaller cliques is missing one of the target labels, selecting the conjunction of all of them yields the densest subgraph as the clique of 20 vertices.

[1] https://version.helsinki.fi/dacs.

Given the noise parameter ϵ, we then add noise by having each of the edges in the cliques removed with probability ϵ, as well as having any other edges added between any pair of vertices with probability ϵ. Finally, for each of the edges in the cliques, we add any of the other labels with probability ϵ each, except for adding the remaining target labels to edges in the cliques.

For the disjunctive case, we have created one clique with 40 vertices. The edges in the clique are split into five sets, such that each set of edges gets one of the target labels. Now, selecting the disjunction of the five target labels induces the clique as the subgraph and results in the highest density.

We then add noise by removing edges from the clique and adding new edges between any other pair of vertices with probability ϵ. In addition, each edge gains any of the other labels also with probability ϵ.

We repeat the experiments with increasing values of ϵ and compare the density of the subgraph induced by the target labels to the density of the subgraph induced by the labels of the greedy algorithms. For each ϵ, we run the experiment 10 times and compute the mean and standard deviation of the runs. The results are shown in Fig. 3.

In both cases, the greedy algorithms correctly find the target labels for small values of ϵ. After $\epsilon > 0.25$ for GreedyAnd and after $\epsilon > 0.35$ for GreedyOr, the algorithms start to find other sets of labels, which yield higher densities than the target labels as many of the edges in the target clique have been removed and other edges have been added. However, at $\epsilon = 0.30$, the GreedyOr returns a suboptimal solution that yields a slightly lower density than the target labels.

We confirm the theoretical running times of the algorithms by setting $\epsilon = 0.2$ and performing experiments with increasingly large graphs, where the number of total vertices goes from 10000 up to 100000 while other aspects of the experiments remain constant. Similarly, we test how the running times of the algorithms scale as the number of total labels in our synthetic graph increases from 1000 to 10000. The results for GreedyAnd are shown in Fig. 4 and results for GreedyOr in Fig. 5.

As expected, the running times of both algorithms scale linearly with the number of vertices in the graph. Furthermore, the running time of our naive implementation of GreedyOr appears to scale quadratically with the number of labels, while the scaling for GreedyAnd is close to linear. These results confirm our theoretical analysis and show that our algorithms can be applied to large graphs in practice.

Experiments with real-world datasets: We test the greedy algorithms by running experiments on four real-world datasets. The first dataset is the Enron Email Dataset,[2] which consists of publicly available emails from employees of a former company called Enron Corporation. We collect the emails in sent mail folders and construct a graph where new edges are added between the sender and the recipients of each email. Each edge has labels consisting of the stemmed words in the email's title, with stop words and words including numbers removed.

The second dataset consists of high energy physics theory publications (HEP-TH) from the years 1992 to 2003. The data was originally released in KDD Cup[3] but we use a preprocessed version of the data available in GitHub[4] We create the network by adding authors as vertices, and edges between any two authors are added if they share at least

[2] https://www.cs.cmu.edu/~./enron/.

[3] https://www.cs.cornell.edu/projects/kddcup/datasets.html.

[4] https://github.com/chriskal96/physics-theory-citation-network.

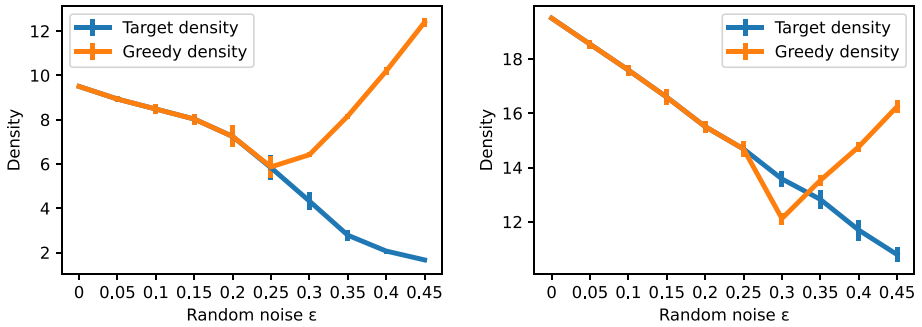

Fig. 3 Density of the subgraph induced by the target labels and the subgraph induced by the labels chosen by the greedy algorithms as a function of noise ϵ in the network. The line shows the mean density of 10 runs for each ϵ, and the vertical error bars show their standard deviation. The results for GreedyAnd algorithm are on the left and for GreedyOr on the right

two publications. The edges between authors are then given labels which consist of the stemmed words in the titles of the shared articles between the two authors. We exclude stop words and words including numbers from the titles the same way as for the Enron dataset.

The third dataset consists of publications from the DBLP[5] dataset (Tang et al., 2008). From this dataset, we chose publications from ECMLPKDD, ICDM, KDD, NIPS, SDM, and WWW conferences. The network is constructed in the same way as for the HEP-TH data, with authors as vertices, two or more shared publications as edges, and stemmed and filtered words from the titles as labels.

The fourth and final dataset consists of the latest 10000 tweets collected from Twitter API[6] with the hashtag #metoo by the 27th of May, 23:59 UTC. We create the network by having users as vertices with an edge between a pair of users if one of them has retweeted or responded to one of the other's tweets. The labels on the edge are then any hashtags in the retweets or response tweets between the two users.

We construct the networks by filtering out labels that appear in less than 0.1% of the edges in the Enron and Twitter datasets, or labels that occur in less than 0.5% of the papers in the case of the HEP-TH and DBLP datasets. The sizes, label counts, and densities of the resulting graphs are shown in Table 1.

We run the greedy algorithms on each of these graphs, and compare the results against the densest subgraph ignoring the labels (Dense). We report the statistics for the label-induced subgraphs and the densest subgraphs in Table 2.

For each of the datasets, both algorithms find label-induced subgraphs with higher densities than in the original graphs. In most cases, the restriction of constructing label-induced subgraphs results in clearly lower densities compared to the densest label-ignorant subgraphs. Interestingly, for the DBLP dataset GreedyAnd finds a label-induced subgraph with a very high density that is close to the density of the densest subgraph ignoring the labels. The running times are practical: the algorithm processes networks with 100 000 edge-label pairs in seconds.

For Enron and HEP-TH datasets, the GreedyOr returns large sets of labels resulting in large subgraphs, whereas the GreedyAnd algorithm selects only a few labels with smaller induced subgraphs in each case. For the Twitter dataset, both greedy algorithms

[5] https://www.aminer.org/citation.

[6] https://developer.twitter.com/en/docs/twitter-api.

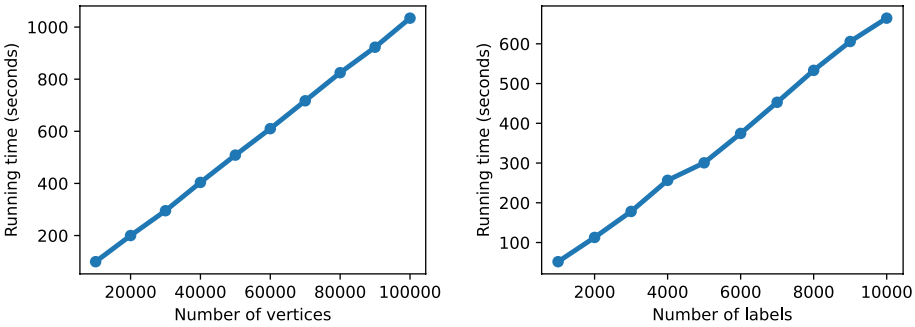

Fig. 4 Running time of the GreedyAnd algorithm as a function of the number of vertices (left) and the number of labels (right) in our synthetic graphs

select only one label, which induces a small subgraph with a notably higher density than the original graph.

Experiments with α-density: Next we consider finding α-dense subgraphs by running the GreedyAnd-α and GreedyOr-α algorithms on the same datasets. The results are shown in Tables 3 and 4, respectively.

As pointed out by Theorem 6, the optimal α-dense subgraph is also the densest for sufficiently large α. We use a binary search to find the maximum α for which the greedy algorithm yields a non-empty graph. The values of α in these tables are chosen by the binary search process while searching for the maximum. Additionally, we experiment with using a smaller α value of 0.25 times the maximum. For clarity, we exclude duplicated results where different values of α yield the same subgraph.

We see that the greedy algorithms for the two problems often find the same solution, as suggested by Theorem 6. However, this is not always the case due to the heuristic nature of these algorithms. Interestingly, with $\alpha = 2.5$ for the HEP-TH dataset, the GreedyAnd-α finds a denser subgraph than the one found by GreedyAnd, while an additional manual experiment with $\alpha = 1.4$ results in the greedy algorithm suboptimally returning an empty graph. For the DBLP dataset using $\alpha = 3.6$ leads to the same solution as GreedyAnd, but larger values of α lead the greedy algorithm to choose a suboptimal first label resulting in less dense subgraphs. For Enron and HEP-TH datasets,

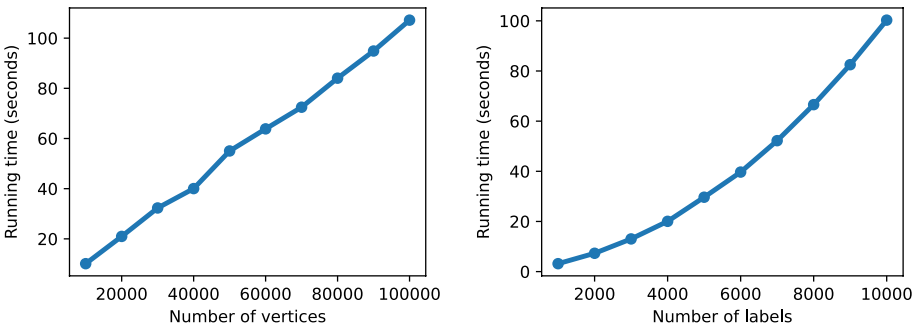

Fig. 5 Running time of the GreedyOr algorithm as a function of the number of vertices (left) and the number of labels (right) in our synthetic graphs.

Table 1 Basic characteristics of the networks: number or vertices $|V|$, number or edges $|E|$, number of labels $|L|$, number of edge-label pairs p, and the density $d(G) = |E|/|V|$

Dataset	$\|V\|$	$\|E\|$	$\|L\|$	p	$d(G)$
Enron	11 024	18 072	2 604	361 000	1.64
HEP − TH	4 738	7 767	240	78 078	1.64
DBLP	10 550	16 811	268	160 850	1.59
Twitter	7 973	9 314	248	19 849	1.17

Table 2 Statistics for the resulting subgraphs for the greedy algorithms and the label-ignorant densest subgraph algorithm. For the label-induced subgraphs, we have the number of vertices n, the number of edges m, the size of the best set of labels $|A|$, density d, and running time t in seconds. For the densest subgraph, we show the number of vertices n and density $d = m/n$

	GreedyAnd					GreedyOr					Dense	
Dataset	n	m	$\|A\|$	d	t	n	m	$\|A\|$	d	t	n	d
Enron	18	31	2	1.72	10.43	1 233	2 711	193	2.2	25.02	85	11.35
HEP − TH	7	14	4	2	1.96	3 284	5 588	40	1.7	5.74	58	3.81
DBLP	25	300	3	12	4.06	243	538	1	2.21	1.74	44	12.52
Twitter	12	31	1	2.58	0.77	12	31	1	2.58	1.89	19	3.37

GreedyOr-α only finds subgraphs with a slightly lower density than the ones found by GreedyOr.

In general, we observe that using smaller values of α results in subgraphs with more vertices and edges in both the disjunctive and conjunctive cases. Thus having α as a parameter gives us more control over the size of the resulting subgraph and allows us to look for both smaller and larger groups of densely connected nodes.

Case study: We analyze the label-induced dense subgraphs for the Twitter and DBLP datasets by repeatedly running the GreedyAnd algorithm for these graphs. After running the algorithm, we exclude the edges from the output edge-induced subgraph and run the algorithm again on the remaining graph. The first 8 resulting sets of labels, as well as densities and sizes for the induced subgraphs, are shown in Table 5.

For the DBLP graph, the algorithm finds a group of 25 authors that have each written at least two papers together with a shared topic, as well as other relatively large groups of authors whose edges form almost perfect cliques. The labels representing stemmed words can be used to interpret the topics of publications for these groups of authors having tight collaboration.

For the Twitter data of #metoo tweets, the densest label-induced subgraphs are formed by mostly looking at individual hashtags. This detects groups of people tweeting about #MeTooASE referring to the French Me Too movement for foster children, as well as groups closely discussing other topics in the context of the Me Too movement such as live streaming or the recent trial between Johnny Depp and Amber Heard.

We see that the same labels also appear when searching for α-dense subgraphs. For example, by looking at the labels for $\alpha = 1.548$ for the Twitter dataset in Table 4 and

Table 3 Results for running GreedyAnd-α on the four datasets with the different values of α. For each resulting subgraph, we have the density $d = m/n$, the chosen labels, the number of nodes n, and the number of edges m.

For each resulting subgraph, we have the density $d = m/n$, the chosen labels or their amount, the number of nodes n, and the number of edges m. Results matching the densest subgraph found by GreedyOr are shown in bold

Dataset	α	d	Labels	n	m
Enron	1.032	1.5088	Meet	1875	2829
	1.548	1.5971	Legal	479	765
	1.7218	**1.7222**	**Mopa, action**	**18**	**31**
HEP-TH	0.6249	1.3656	Theori	2410	3291
	2.4998	2.5	Casimir, light	6	15
DBLP	1.0927	1.3005	Learn	3780	4916
	3.6	**12**	**Novel, rate, techniqu**	**25**	**300**
	4.3708	6.2	Forecast, experi, use	15	93
Twitter	0.6457	1.1381	Metoo	8297	7290
	2.5828	**2.5833**	**Metooase**	**12**	**31**

Bold rows show when the value of α leads the GreedyAnd-α to find the same result as the one found by the GreedyAnd algorithm

Results matching the densest subgraph found by GreedyAnd are shown in bold

Table 4 Results for running GreedyOr-α on the four datasets with the different values of α. For each resulting subgraph, we have the density $d = m/n$, the chosen labels or their amount, the number of nodes n, and the number of edges m. Results matching the densest subgraph found by GreedyOr are shown in bold

Dataset	α	d	Labels	n	m
Enron	1.32	1.8497	(553 labels)	7982	14765
	1.98	2.1715	(964 labels)	3633	7889
	2.145	2.1810	(577 labels)	2497	5446
	2.1656	2.1798	(547 labels)	2364	5153
	2.1798	2.1799	(582 labels)	2329	5077
HEP-TH	0.4255	1.6393	(75 labels)	4738	7767
	1.02	1.6557	(68 labels)	4650	7699
	1.53	1.6938	(76 labels)	4063	6882
	1.6575	1.7011	(52 labels)	3567	6068
	1.7013	1.7023	(50 labels)	3426	5832
DBLP	0.5534	1.5953	(93 labels)	10532	16802
	1.326	1.6098	(83 labels)	10295	16573
	2.2137	**2.2140**	**Novel**	**243**	**538**
Twitter	0.6457	1.1685	(49 labels)	7969	9312
	1.548	1.8857	Metooase, streamer, anubhavmohanty, victimservices, causette, istandwithjohnny	70	132
	2.5828	**2.5833**	**Metooase**	**12**	**31**

Bold rows show when the value of α leads the GreedyAnd-α to find the same result as the one found by the GreedyAnd algorithm

comparing them with the labels in Table 5, we can see that this subgraph found by the GreedyOr-α algorithm in fact consists of multiple smaller groups of people discussing a variety of topics that we previously discovered.

Table 5 Label sets with corresponding subgraph densities and sizes selected by running the GreedyAnd algorithm repeatedly on the graphs for DBLP and Twitter datasets.

d	DBLP		
	Labels	*n*	*m*
12.0	Novel, rate, techniqu	25	300
10.74	Identif, combin, process	23	247
6.2	Forecast, experi, use	15	93
6.0	Heterogen, manag, stream, use	13	78
2.0	Heterogen, segment	5	10
3.13	Heterogen, manag, use, dynam	8	25
2.5	Heterogen, sourc, toward	6	15
2.5	Heterogen, construct, dimension, network	6	15
	Twitter		
d	Labels	*n*	*m*
2.58	Metooase	12	31
1.88	Streamer	16	30
1.75	Anubhavmohanty	16	28
1.71	Victimservices	7	12
1.83	Causette, lfi	6	11
1.63	Istandwithjohnny	8	13
1.43	Rupertmurdock	7	10
1.25	Marilynmanson	8	10

The labels are stemmed words from publication titles for DBLP, and tweet hashtags for Twitter data. The densities are not monotonically decreasing as the greedy algorithm does not always find the optimal solution

8 Concluding remarks

In this paper, we considered the problem of finding dense subgraphs that are induced by labels on the edges. More specifically, we considered two cases: conjunctive-induced dense subgraphs, where the edges need to contain the given label set, and disjunctive-induced dense subgraphs, where the edges need to have only one label in common. As a measure of quality, we used the average degree of a subgraph. We showed that both problems are **NP**-hard, and we proposed a greedy heuristic to find dense induced subgraphs. By maintaining suitable counters we were able to find subgraphs in quasi-linear time: $\mathcal{O}(p \log |L|)$ for conjunctive-induced graphs and $\mathcal{O}\big(p \log^2 |L|\big)$ for disjunctive-induced graphs. In addition, we analyzed the related problem of maximizing the number of edges minus α times the number of vertices and showed how the optimal solutions to these problems are connected. We proved that the problem of maximizing this α-density is **NP**-hard and inapproximable unless $\mathbf{P} = \mathbf{NP}$. We adopted the greedy algorithms for the conjunctive and disjunctive cases of this problem resulting in a running time of $\mathcal{O}(p \log |L|)$ for the disjunctive case as well. We then demonstrated that the algorithms are practical, they can find ground truth in synthetic datasets, and find interpretable results from real-world networks.

While this paper focused on the conjunctive and disjunctive cases, future work could explore other ways to induce graphs from a label set and design efficient algorithms for such tasks. Another direction for future work is to relax the requirement that every edge/node must be induced from labels. Instead, we can allow some deviation from this requirement but then penalize the deviations appropriately when assessing the quality of the subgraph.

A Computational complexity proofs

Proof of Theorem 1 We will prove the claim by reducing 3ExactCover to the densest subgraph problem. In 3ExactCover we are given a set X and a family $\mathcal{C}$ of subsets of size 3 over X and asked if there is a disjoint subset of $\mathcal{C}$ whose union is X.

Assume that we are given a set X and a family $\mathcal{C} = \{C_1, \ldots, C_N\}$ of N subsets. We set labels to be $L = \{1, \ldots, N\}$. The vertices V contain N vertices $y_1, \ldots, y_N$, and an additional vertex z. We connect each y_i to z, labeled with $L \setminus \{i\}$. For each overlapping C_i and C_j, we introduce $4N$ additional vertices and $2N$ edges, each edge connecting two *unique nodes*, and labeled as $L\setminus\{i,j\}$.

We claim that for $|X| \geq 5$, 3ExactCover has a solution if and only if there is an induced graph H with $d(H) \geq |X|/(|X| + 3)$.

Assume that we are given a set of labels $A \subset L$. Let $B = L\setminus A$. Let k be the number of set pairs in B that are overlapping, that is,

$$k = \left|\left\{\{i,j\} \mid i,j \in B, C_i \cap C_j \neq \emptyset\right\}\right|.$$

Then the density of the corresponding graph $H = G(f_{AND}, A)$ is equal to

$$d(H) = \frac{|B| + 2Nk}{|B| + 1 + 4Nk} \quad .$$

Assume that $k > 0$. Since $|B| \leq N$, we can bound the density with

$$d(H) = \frac{|B| + 2Nk}{|B| + 1 + 4Nk} \leq \frac{N + 2Nk}{N + 1 + 4Nk} < \frac{N + 2Nk}{N + 4Nk} \leq \frac{3}{5} \quad .$$

Assume that $k = 0$. Then the density is equal to $|B|/(|B| + 1)$. Let $\mathcal{U} = \{C_i \mid i \in B\}$. Since $\mathcal{U}$ is disjoint, $3|B| \leq |X|$ and the equality holds if and only if $\mathcal{U}$ covers X.

Assume that there is a subgraph $H = G(f_{AND}, A)$ with $d(H) \geq |X|/(|X| + 3)$. Since we assume that $|X| \geq 5$, we have $d(H) \geq 5/8 > 3/5$, and the preceding discussion shows that the sets corresponding to A form an exact cover of X.

On the other hand, if there is an exact cover in $\mathcal{C}$, then $d(G(f_{AND}, A)) = |X|/(|X| + 3)$, where A is the set of labels corresponding to the cover. This shows that maximizing the density of the label-induced subgraph is an **NP**-hard problem. □

Proof of Theorem 3 We will prove the claim by reducing 3ExactCover to the densest subgraph problem. In 3ExactCover we are given a set X and a family $\mathcal{C}$ of subsets of size 3 over X and asked if there is a disjoint subset of $\mathcal{C}$ whose union is X.

Assume that we are given a set X and a family $\mathcal{C} = \{C_1, \ldots, C_N\}$ of N subsets. The vertices V consists of the set X, N additional vertices $y_1, \ldots, y_N$, and 2 more vertices $Z = z_1, z_2$. We have N labels, $L = \{1, \ldots, N\}$.

Next, we define the edges E. Connect each $x \in X$ to Z, and label the edges with labels $\{i \mid x \in C_i\}$. Then for each C_i, we connect z_1 to y_i, labeled with i.

We claim that 3ExactCover has a solution if and only if the optimal label-induced graph has the density of $7|X|/(6+4|X|)$.

Given a non-empty set of labels $A \subseteq L$, the density of the corresponding graph H is equal to $g(k, |A|)$, where $g(s,t) = \frac{2s+t}{2+s+t}$, and k is the size of the union of sets in $\mathcal{C}$ corresponding to A.

Note that since $k \geq 3$, we have $2k > 2 + k$. Thus, $\partial \log g/\partial t = 1/(2k+t) - 1/(2+k+t) < 0$, and consequently $g(k,t) > g(k,t')$ when $t < t'$.

Since each set in $\mathcal{C}$ is of size 3, we have $|A| \geq k/3$. Thus,

$$g(k, |A|) \leq g(k, k/3) = \frac{7k}{6+4k} \leq \frac{7|X|}{6+4|X|},$$

where the equalities hold if and only if $k = |X|$ and $3|A| = k$, that is, A corresponds to an exact cover of X. □

Proof of Corollary 1 Let us adopt the notation of the proof of Theorem 3. The proof shows that 3ExactCover has a solution if and only if there is an induced graph H with $7d(H) \geq |X|/(4|X|+6)$. Moreover, there are $N+2$ nodes in the graph, so a difference between two densities is at least $(N+2)^{-2}$. Consequently, if we set $\tau = 7d(H) \geq |X|/(4|X|+6) - 0.5(N+2)^{-2}$, then Theorem 6 implies that 3ExactCover has a solution if and only if there is H with $g(H,\tau) > 0$. This proves the hardness and the inapproximability since any algorithm with a multiplicative guarantee will find the optimal solution. The proof for f_{AND} is similar. □

Author contributions NT formulated the problems. IK implemented the algorithms and conducted the experiments. Both authors wrote the manuscript.

Funding Open Access funding provided by University of Helsinki including Helsinki University Central Hospital. This research is supported by the Academy of Finland projects MALSOME (343045).

Data availability Publicly available datasets were used. See Sect. 7 for the links.

Code availability https://version.helsinki.fi/dacs/

Declarations

Conflict of interest Not applicable

Ethical approval Not applicable

Consent to participate Not applicable

Consent for publication Not applicable

References

Abello, J., Resende, GC., & Sudarsky S. (2002). Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics*, pp 598–612.

Angel, A., Koudas, N., Sarkas, N., Srivastava, D., Svendsen, M., & Tirthapura, S. (2014). Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB Journal, 23*(2), 175–199.

Balasundaram, B., Butenko, S., & Hicks, Illya V. (2011). Clique relaxations in social network analysis: The maximum *k*-plex problem. *Operations Research, 59*(1), 133–142.

Bonchi, F., Khan, A., & Severini, L. (2019). Distance-generalized core decomposition. In *SIGMOD*, pp 1006–1023.

Brodal, GS., Jacob, R. (2002). Dynamic planar convex hull. In *FOCS*, pp 617–626.

Bron, C., & Kerbosch, J. (1973). Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM, 16*(9), 575–577.

Charikar, M. (2000). Greedy approximation algorithms for finding dense components in a graph. *APPROX*.

Danisch, M., Chan, T-HH., & Sozio, M. 2017). Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web*, pp 233–242. International World Wide Web Conferences Steering Committee.

Dinkelbach, W. (1967). On nonlinear fractional programming. *Management Science, 13*(7), 492–498.

Du, X., Jin, R., Ding, L., Lee, VE., & Thornton Jr, John H. (2009). Migration motif: a spatial-temporal pattern mining approach for financial markets. In *KDD*, pp 1135–1144.

Fratkin, E., Naughton, BT., Brutlag, DL., & Batzoglou, S. (2006). Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics, 22*(14), e150–e157.

Galbrun, E., Gionis, A., & Tatti, N. (2014). Overlapping community detection in labeled graphs. *DMKD, 28*(5), 1586–1610.

Goldberg, AV. (1984). Finding a maximum density subgraph. *University of California Berkeley Technical report.*

Håstad, J. (1996). Clique is hard to approximate within $n^{1-\epsilon}$. In *FOCS*, pp 627–636.

Kumpulainen, I., & Tatti, N. (2022). Community detection in edge-labeled graphs. In *Discovery Science: 25th International Conference, DS 2022, Montpellier, France, October 10–12, 2022, Proceedings*, pp 460–475.

Langston, MA., Lin, L., Peng, X., Baldwin, NE., Symons, CT., Zhang, B., & Snoddy, JR. (2005). A combinatorial approach to the analysis of differential gene expression data. In *Methods of Microarray Data Analysis*, pp 223–238. Springer.

Li, F., & Klette, R. (2011). *Euclidean Shortest Paths: Exact or Approximate Algorithms*, chapter Convex Hulls in the Plane, pp 93–125. Springer .

Mokken, RJ. (1979). Cliques clubs and clans. *Quality & Quantity, 13*(2), 161–173.

Overmars, MH., & Van Leeuwen, J. (1981). Maintenance of configurations in the plane. *Journal of computer and System Sciences, 23*(2), 166–204.

Pool, S., Bonchi, F., & van Leeuwen, M. (2014). *Description-driven community detection. TIST, 5*(2), 1–28.

Seidman, SB. (1983). Network structure and minimum degree. *Social Networks, 5*(3), 269–287.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., & Su, Z. (2008). Arnetminer: Extraction and mining of academic social networks. In *KDD*, pp 990–998.

Tatti, N. (2019). *Density-friendly graph decomposition. TKDD, 13*(5), 1–29.

Tsourakakis, CE. (2015). The k-clique densest subgraph problem. In *WWW*, pp 1122–1132.

Uno, T. (2010). An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica, 56*(1), 3–16.