Jaccard-constrained dense subgraph discovery

Chamalee Wickrama Arachchi[1*] and Nikolaj Tatti[1]

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

*Corresponding author(s). E-mail(s):
chamalee.wickramaarachch@helsinki.fi;
Contributing authors: nikolaj.tatti@helsinki.fi;

Abstract

Finding dense subgraphs is a core problem in graph mining with many applications in diverse domains. At the same time many real-world networks vary over time, that is, the dataset can be represented as a sequence of graph snapshots. Hence, it is natural to consider the question of finding dense subgraphs in a temporal network that are allowed to vary over time to a certain degree. In this paper, we search for dense subgraphs that have large pairwise Jaccard similarity coefficients.

More formally, given a set of graph snapshots and input parameter $\boldsymbol{\alpha}$, we find a collection of dense subgraphs, with pairwise Jaccard index at least $\boldsymbol{\alpha}$, such that the sum of densities of the induced subgraphs is maximized. We prove that this problem is **NP**-hard and we present a greedy, iterative algorithm which runs in $\mathcal{O}(nk^2 + m)$ time per single iteration, where k is the length of the graph sequence and n and m denote number of vertices and total number of edges respectively. We also consider an alternative problem where subgraphs with large pairwise Jaccard indices are rewarded. We do this by incorporating the indices directly into the objective function. More formally, given a set of graph snapshots and a weight $\boldsymbol{\lambda}$, we find a collection of dense subgraphs such that the sum of densities of the induced subgraphs plus the sum of Jaccard indices, weighted by $\boldsymbol{\lambda}$, is maximized. We prove that this problem is **NP**-hard. To discover dense subgraphs with good objective value, we present an iterative algorithm which runs in $\mathcal{O}(n^2k^2 + m \log n + k^3 n)$ time per single iteration, and a greedy algorithm which runs in $\mathcal{O}(n^2k^2 + m \log n + k^3 n)$ time. We show experimentally that our algorithms are efficient, they can find ground truth in synthetic datasets and provide good results from real-world datasets. Finally, we present two case studies that show the usefulness of our problem.

1 Introduction

Finding dense subgraphs is a core problem in graph mining with many applications in diverse domains such as social network analysis [1], temporal pattern mining in financial markets [2], and biological system analysis [3]. Often, many real-world networks vary over time, in which case a sequence of graph snapshots naturally exists. Consequently, mining dense subgraphs over time has gained attention in data mining literature [1, 4–6].

Our goal is to find dense subgraphs in a temporal network. To measure the density, we will use the definition of the ratio between the number of induced edges and vertices. We should point out that there are other definitions of density such as the proportion of edges. However, our choice is popular since the densest subgraph can be found in polynomial time [7] and approximated efficiently [8].

Given a graph sequence, there are natural extremes to find the densest subgraphs: the first approach is to find a common subgraph that maximizes the sum of densities for individual snapshots, as proposed by Semertzidis et al. [1] among other techniques. The other approach is to find the densest subgraphs for each snapshot individually.

In this paper, we study the problem that bridges the gap between these two extremes, namely, we seek dense subgraphs in a temporal network that are allowed to vary over time to a certain degree. More formally, given a graph sequence $\mathcal{G}$ and parameter α, we seek a sequence of subgraphs, with pairwise Jaccard index at least α, such that the sum of densities is maximized.

We demonstrate the differences in the following toy example.

Example 1. *Consider a graph sequence (G_1, G_2, G_3) shown in Figure 1, each graph consisting of 6 vertices and varying edges. We denote the density induced by the vertex set S_i by $d(S_i)$, defined as the ratio between number of induced edges and vertices, $d(S_i) = \frac{|E(S_i)|}{|S_i|}$. We define the sum of densities as $\sum_{i=1}^{3} d(S_i)$.*

The densest common subgraph, that is, a single common subgraph over each snapshot maximizing the sum of densities, is $S = (a, b, d, e, f)$. The density sum induced by S is $\frac{4}{5} + \frac{5}{5} + \frac{8}{5} = 3.40$.

On the other hand, the densest subgraphs for individual snapshots are $S'_1 = (a, b, d)$, $S'_2 = (a, b, c, f)$, and $S'_3 = (a, b, d, e, f)$. The sum of densities of individually densest subgraphs is given by $\frac{3}{3} + \frac{6}{4} + \frac{8}{5} = 4.1$.

Note that the Jaccard index between set S and T is defined as $J(S, T) = \frac{|S \cap T|}{|S \cup T|}$. Therefore, $J(S'_1, S'_2) = \frac{3}{5} = 0.6$, $J(S'_1, S'_3) = \frac{4}{5} = 0.8$, and $J(S'_2, S'_3) = \frac{3}{6} = 0.5$. Therefore, $\min_{i<j} J(S'_i, S'_j)$ is 0.5.

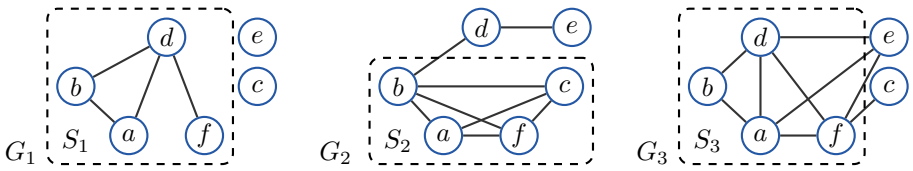

Figure 1: Toy graphs used in Example 1 and 2

Let $\alpha = 0.6$ *and let* $S_1 = (a, b, d, f)$, $S_2 = (a, b, c, f)$, *and* $S_3 = (a, b, d, f)$. *Then,* $J(S_1, S_2) = \frac{3}{5} = 0.6$, $J(S_1, S_3) = \frac{4}{4} = 1$, *and* $J(S_2, S_3) = \frac{3}{5} = 0.6$. *The total density induced by the sets with pairwise Jaccard index at least* 0.6 *is given by* $\frac{4}{4} + \frac{6}{4} + \frac{5}{4} = 3.75$. *Hence, three different density results are ordered as* $3.4 \leq 3.75 \leq 4.1$.

We also consider an alternative problem, where we incorporate the Jaccard index into an objective function instead of having it as a constraint. Here, we reward similar graphs over the snapshots. More formally, given a graph sequence $\mathcal{G}$ and a parameter λ, we seek a sequence of subgraphs, such that the sum of densities plus the sum of Jaccard indices, weighted by λ, is maximized.

We demonstrate the objective in the following example.

Example 2. *Consider a graph sequence* (G_1, G_2, G_3) *shown in Figure 1. Given a weight parameter* λ *and a sequence of subgraphs* (S_1, S_2, S_3), *we define our objective function as* $\sum_{i=1}^{3} d(S_i) + \lambda \sum_{i<j} J(S_i, S_j)$.

Assume that we set $\lambda = 0.3$ *and select* $S_1 = (a, b, d, f)$, $S_2 = (a, b, c, f)$, *and* $S_3 = (a, b, d, f)$. *The sum of densities is* $\frac{4}{4} + \frac{6}{4} + \frac{5}{4} = 3.75$. *Here,* $J(S_1, S_2) = \frac{3}{5} = 0.6$, $J(S_1, S_3) = \frac{4}{4} = 1$, *and* $J(S_2, S_3) = \frac{3}{5} = 0.6$. *Therefore, our objective is equal to* $3.75 + 0.3 \times (0.6 + 1 + 0.6) = 4.41$.

We show that both of our problems are **NP**-hard and consequently propose greedy algorithms. For the constrained version of the problem, we propose an iterative algorithm that starts from the common densest subgraph solution and greedily improves the solution by adding or removing vertices.

For the second problem, we propose two algorithms. The first approach is an iterative algorithm where we start either with the common densest subgraph or a set of the densest subgraphs for each individual snapshot and then iteratively improve each individual snapshot. The improvement step is done with a classic technique used to approximate the densest subgraph (in a single graph) [8, 9]. We start with the complete snapshot, and iteratively seek out the vertex so that the remaining graph yields the largest score. We remove the vertex, and the procedure is repeated until no vertices remain; the best subgraph for that snapshot is selected. This is repeated for each snapshot until no improvement is possible. The second algorithm uses a similar approach with the following differences. The algorithm does not iterate over snapshots. Instead, we search for the best snapshot and the best vertex in that snapshot and delete the vertex. We continue until there are no vertices left, and select the best set of subgraphs seen during the deletion.

The appeal of this approach is that, when dealing with a single graph, finding the next vertex can be done efficiently using a priority queue [8, 9]. We cannot use this approach directly due to the updates in Jaccard indices. Instead, we maintain a *set* of priority queues that allow us to find vertices quickly in practice.

This paper extends the earlier paper [10] by considering maximizing the density while constraining Jaccard indices.

The remainder of the paper is organized as follows. In Section 2, we provide preliminary notation along with the formal definitions of our optimization problems. All our algorithms and their running times are presented in Section 3. Related work is discussed in Section 4. In Section 5 we present an extensive experimental study both with

synthetic and real-world datasets followed by two case studies. Finally, Section 6 summarizes the paper and provides directions for future work. This paper is an extension of a conference paper [10].

2 Preliminary notation and problem definition

We begin by providing preliminary notation and formally defining our problem.

Our input is a sequence of graphs $\mathcal{G} = (G_1, \ldots, G_k)$, where each snapshot $G_i = (V, E_i)$ is defined over the same set of vertices. We denote the number of vertices and edges by $n = |V|$ and $m_i = |E_i|$, or $m = |E|$, if i is omitted.

Given a graph $G = (V, E)$, and a set of vertices $S \subseteq V$, we define $E(S) \subseteq E$ to be the subset of edges having both endpoints in S.

As mentioned before, our goal is to find dense subgraphs in a temporal network, and for that, we need to quantify the density of a subgraph. More formally, assume an unweighted graph $G = (V, E)$, and let $S \subseteq V$. We define the *density* $d(S)$ of a single vertex set S and extend this definition for a sequence of subgraphs $\mathcal{S} = (S_1, \ldots, S_k)$ by writing

$$d(S_i) = \frac{|E(S_i)|}{|S_i|} \qquad \text{and} \qquad d(\mathcal{S}) = \sum_{i=1}^{k} d(S_i) \quad .$$

We will use the Jaccard index to measure the similarity between two subgraphs. More formally, given two sets of vertices S and T, we write

$$J(S, T) = \frac{|S \cap T|}{|S \cup T|} \quad .$$

Given a sequence of graph snapshots and an input parameter α, our goal is to find a collection of subsets of vertices, one per each snapshot, such that the sum of the densities of subgraphs is maximized while maintaining pairwise Jaccard coefficient between the sets to be at least α.

Problem 2.1 (Jaccard Constrained Densest Subgraphs (JCDS))**.** *Given a graph sequence $\mathcal{G} = (G_1, \ldots, G_k)$, with $G_i = (V, E_i)$, and a real number $\alpha \in [0, 1]$, find a collection of subset of vertices $\mathcal{S} = (S_1, \ldots, S_k)$, such that $d(\mathcal{S})$ is maximized and $J(S_i, S_j) \geq \alpha$ for all i and j.*

We will consider two extreme cases. The first case is $\alpha = 1$, which we refer to as *densest common subgraph* or DCS that maximizes total density. This problem can be solved by first flattening the graph sequence into one weighted graph, where an edge weight is the number of snapshots in which an edge occurs. The problem is then a standard (weighted) densest subgraph problem that can be solved using the method given by Goldberg [7] in $\mathcal{O}(n(n+m)(\log n + \log k))$ time. The other extreme case is $\alpha = 0$ which can be solved by solving the densest subgraph problem for each individual snapshot.

The main difference from prior studies [1, 4] is that we allow the subsets to be varied within a given margin (which is defined by Jaccard coefficient), without enforcing subsets to be fully identical. In other words, we extend the idea of the common densest subgraph in which $\alpha = 1$, for a more generalized case where α can be less than one.

In addition to the hard constraint version, we consider a variant where we incorporate the constraints directly into the optimization score. Ideally, we would like each subgraph to have high density, and share as many vertices as possible with each other. This leads to the following score and optimization problem. More specifically, given a weight parameter λ and a sequence of subgraphs $\mathcal{S} = (S_1, \ldots, S_k)$, we define a score

$$q(\mathcal{S}; \lambda) = d(\mathcal{S}) + \lambda \sum_{i=1}^{k} \sum_{j=i+1}^{k} J(S_i, S_j) \quad .$$

Problem 2.2 (Jaccard Weighted Densest Subgraphs (JWDS))**.** *Given a graph sequence $\mathcal{G} = (G_1, \ldots, G_k)$, with $G_i = (V, E_i)$, and a real number λ, find a collection of subset of vertices $\mathcal{S} = (S_1, \ldots, S_k)$, such that $q(\mathcal{S})$ is maximized.*

The purpose of user parameter λ is to maintain a predefined balance between the two terms of $q(\mathcal{S})$; namely the density term and Jaccard similarity term. Here we will consider changing λ according to our needs. If we want the collection of subgraphs to be more identical, we set λ to be very large. In this case, we favor the vertex sets who incur higher overlap with other snapshots. Nevertheless, if we are more focused on the density than the similarity between sets, we assign a small value to λ which is less than 1.

3 Algorithms

The problem of finding a common subgraph which maximizes the sum of densities can be solved optimally in polynomial time. Moreover, if we set $\lambda = 0$, then we can solve the problem by finding optimal dense subgraphs for each snapshot individually. However, JCDS and JWDS are both **NP**-hard. The proof of the propositions is in Appendix A.1–A.2.

Proposition 1. JCDS *is **NP**-hard.*

Proposition 2. JWDS *is **NP**-hard.*

Next, we will present our heuristic algorithms.

3.1 Finding Jaccard-constrained densest subgraphs

First, we consider a straightforward greedy algorithm for JCDS. The idea of the proposed iterative algorithm is as follows. We start by solving the densest common subgraph problem, that is, JCDS with $\alpha = 1$.

We initialize the sets to be the densest common subgraph. Note that the Jaccard constraint is automatically satisfied. We then try to improve the sum of the densities by either adding or removing vertices to each subset, while satisfying the pairwise Jaccard similarity coefficient constraints. We repeat the same process until the algorithm converges. The objective value only either improves or stays constant at the end of each iteration. Thus our algorithm always converges. The pseudo-code for the algorithm is given in Algorithm 1.

To quickly test each vertex, we maintain the sizes of intersections and the unions between S_i and S_j. This leads to the following running time for a single iteration.

Algorithm 1: HARD$(\mathcal{G}, \alpha)$, finds $\mathcal{S}$ with good $d(\mathcal{S})$ s.t. $\min J(S_i, S_j) \geq \alpha$.

```
S ← S_1, ..., S_k, where S_i is the densest common subgraph (α = 1);
while changes in the score do
    foreach i = 1, ..., k do
        foreach v ∈ V do
            if v ∈ S_i then S' ← S with S_i replaced with S_i \ {v};
            else S' ← S with S_i replaced with S_i ∪ {v};
            if S' satisfies the Jaccard constraint and d(S') > d(S) then S ← S';
return S;
```

Proposition 3. *Assume a graph sequence $G_1, \ldots, G_k$ with n vertices and total $m = \sum_i m_i$ edges. Then the running time of a single iteration of* HARD *is in $\mathcal{O}(nk^2 + m)$.*

Proof. Computing the density gain $d(\mathcal{S}') - d(\mathcal{S})$ requires iterating over the edges adjacent to v. Consequently, testing over all v and i results in visiting every edge twice, amounting to a total of $\mathcal{O}(m)$ time.

Testing the Jaccard constraint can be done in $\mathcal{O}(k)$ time since we maintain the sizes of intersections and the unions between S_i and S_j. Once S_i is updated, updating these sizes can be updated in $\mathcal{O}(k)$ time. Since there are n vertices in k snapshots, the claim follows. □

3.2 Finding Jaccard-weighted densest subgraphs

In this section, we present two algorithms that we will use to find a sequence of subgraphs with good score $q(\cdot; \lambda)$.

The first algorithm is as follows. We start with an initial candidate $\mathcal{S}$. This set is either the solution of the densest common subgraph or the densest subgraphs of each individual snapshot; we test both and select the best end result.

To improve the initial set we employ the strategy used in [8, 9] when approximating the densest subgraph: Here, the algorithm starts with the whole graph and removes a vertex with the minimum degree, or equivalently, removes a vertex such that the remaining subgraph has the highest density. This is continued until no vertices remained, and among the tested subgraphs the one with the highest density is selected.

We employ a similar strategy. For a snapshot G_i, we start with $S_i = V$, and then iteratively remove the vertices so that the score is maximal. After removing all vertices, we pick the subgraph for S_i which maximizes our objective $q(\mathcal{S}; \lambda)$. We iterate over all snapshots, we keep on modifying the sets until the algorithm converges. Since the objective value either only increases or stays constant at the end of each iteration, the algorithm always converges. The pseudo-code for this approach is given in Algorithm 2.

Next, we will show that, due to the choice of the initial sets, ITR is a 2-approximation algorithm.

Proposition 4. *Assume a graph sequence $\mathcal{G}$ and let $\mathcal{S}^*$ be the optimal solution. Let $\mathcal{S}$ be the subgraph sequence consisting of the densest common subgraph. Let $\mathcal{S}'$ be the*

Algorithm 2: ITR$(\mathcal{G}, \lambda, \mathcal{S})$, finds subgraphs with good $q(\cdot; \lambda)$

```
S ← any of the two solutions to JCDS(α = 1) and JCDS(α = 0);
while changes in the score do
    foreach i = 1, ..., k do
        C ← V;
        foreach j = 2, ..., |V| do
            u ← arg max_{v∈C} q(S_1, ..., S_{i−1}, C \ {v}, S_{i+1}, ..., S_k);
            C ← C \ {u};
        S_i ← best tested C, if the score improves;
return S;
```

subgraph sequence consisting of the densest subgraphs for each snapshot. Then

$$q(\mathcal{S}^*) \leq 2\max(q(\mathcal{S}), q(\mathcal{S}')) \quad .$$

Before proving this result we should point out that any sequence with a common subgraph in place of $\mathcal{S}$ will yield the result. However, choosing the densest common subgraph is a sensible choice.

Proof. Let us write $f(\mathcal{S}) = \lambda \sum_{i=1}^{k} \sum_{j=i+1}^{k} J(S_i, S_j)$. Note that $\mathcal{S}'$ maximizes the density term in $q(\cdot)$ while $\mathcal{S}$ maximises the Jaccard term in $q(\cdot)$. Thus,

$$\begin{aligned} q(\mathcal{S}^*) &= d(\mathcal{S}^*) + f(\mathcal{S}^*) \\ &\leq 2\max(d(\mathcal{S}^*), f(\mathcal{S}^*)) \leq 2\max(d(\mathcal{S}'), f(\mathcal{S})) \leq 2\max(q(\mathcal{S}), q(\mathcal{S}')), \end{aligned}$$

proving the claim. □

Since iterative phase cannot decrease the score, the following approximation result is imminent.

Corollary 3.1. *Let $\mathcal{S}^*$ be the solution to* JWDS *and let $\mathcal{S} =$* ITR$(\mathcal{G}, \lambda)$ *be the sequence produced by* ITR. *Then $q(\mathcal{S}^*, \lambda) \leq 2q(\mathcal{S}, \lambda)$.*

Our second algorithm is similar to ITR. In Algorithm 2 we consider each snapshot separately and peel off vertices. In our second algorithm, we initialize each S_i with V. In each iteration, we find a snapshot S_i and a vertex v so that the remaining subgraph sequence is maximized. We remove the vertex and continue until no vertices are left. In the process, we choose the one which maximizes our objective function. The pseudo-code for this method is given in Algorithm 3.

The bottleneck in both algorithms is finding the next vertex to delete. Let us now consider how we can speed up this selection. To this end, select S_i and let $v \in S_i$. Let us write $\mathcal{S}'$ to be $\mathcal{S}$ with S_i replaced with $S_i \setminus \{v\}$. We can write the score difference

Algorithm 3: GRD$(\mathcal{G}, \lambda)$, finds subgraphs with good $q(\cdot; \lambda)$

$\mathcal{S} \leftarrow S_1, \ldots, S_k$, where $S_i = V$;
while there are vertices **do**
  $u, j \leftarrow \arg\max_{v,i|v\in S_i} q(S_1, \ldots, S_{i-1}, S_i \setminus \{v\}, S_{i+1}, \ldots, S_k)$;
  $S_j \leftarrow S_j \setminus \{u\}$;
return best tested $\mathcal{S}$;

between $\mathcal{S}$ and $\mathcal{S}'$ as

$$q(\mathcal{S}') - q(\mathcal{S}) = \frac{|E(S_i)| - \deg v}{|S_i| - 1} - \frac{|E(S_i)|}{|S_i|} + \sum_{j\neq i} J(S_i \setminus \{v\}, S_j) - J(S_i, S_j) \quad . \quad (1)$$

Let us first consider GRD. To find the optimal v and i, we will group the vertices in S_i such that the the sum in Eq. 1 is equal for the vertices in the same group. In order to do that we group the vertices based on the following condition: if two vertices $u, v \in S_i$ belong to the exactly same S_j for each j, that is, $u \in S_j$ if and only if $v \in S_j$, then u and v belong to the same group. More formally, let us first define $act(v) = \{j \mid v \in S_j\}$ to be the set of indices of all S_j that have v as a member. The function $act(\cdot)$ induces a partition $\mathcal{P}_i$ of S_i: each group in $\mathcal{P}_i$ consists of vertices v having the same $act(v)$.

Select $P \in \mathcal{P}_i$. Since the sum in Eq. 1 is constant for all vertices in P, the vertex in P maximizing Eq. 1 must have the smallest degree. Thus, we maintain the vertices in P in a priority queue keyed by the degree. We also maintain the difference of the Jaccard indices, the sum in Eq. 1. In order to maintain the difference, we maintain the sizes of intersection $|S_i \cap S_j|$ and the union $|S_i \cup S_j|$ for all i and j. To find the optimal v and i, we find the vertex with the smallest degree in each group, and then compare these candidates among different groups.

This data structure leads to the following running time.

Proposition 5. *Assume a graph sequence $G_1, \ldots, G_k$ with n vertices and total $m = \sum_i m_i$ edges. Let $\mathcal{P}_{ir}$ be the groups of S_i (based on the vertex memberships in other snapshots) when deleting the rth vertex. Define $\Delta = \max |\mathcal{P}_{ir}|$. Then the running time of* GRD *is in*

$$\mathcal{O}\big(nk^2\Delta + m \log n + k^2 n(k + \log n)\big) \subseteq \mathcal{O}\big(n^2k^2 + m \log n + k^3 n\big) \quad .$$

See Appendix A.3 for proof.

We should point out that the running time depends on Δ, the number of queues in a single snapshot. This number may be as high as the number of vertices, n, but ideally $\Delta \ll n$.

The same data structure can be also used ITR. The only difference is that we do not select optimal i; instead, i is fixed when looking for the next vertex to delete. Trivial adjustments to the proof of Prop. 5 imply the following claim.

Proposition 6. *Assume a graph sequence $G_1, \ldots, G_k$ with n vertices and total $m = \sum_i m_i$ edges. Let $\mathcal{P}_{ir}$ be the groups of S_i (based on the vertex memberships in other snapshots) when deleting rth vertex. Define $\Delta = \max |\mathcal{P}_{ir}|$. Then the running time of a single iteration of* Itr *is in*

$$\mathcal{O}\big(nk^2\Delta + m \log n + k^2 n(k + \log n)\big) \subseteq \mathcal{O}\big(n^2k^2 + m \log n + k^3 n\big) \quad .$$

4 Related work

In this section we discuss previous studies on discovering the densest subgraph in a single graph, the densest common subgraph over multiple graphs, overlapping densest subgraphs, and other types of density measures.

The densest subgraph: Given an undirected graph, finding the subgraph which maximizes density has been first studied by Goldberg [7] where an exact, polynomial time algorithm which solves a sequence of min-cut instances is presented. Asahiro et al. [9] provided a linear time, greedy algorithm proved to be a 1/2-approximation algorithm by Charikar [8]. The idea of the algorithm is that at each iteration, a vertex with the minimum degree is removed, and then the densest subgraph among all the produced subgraphs is chosen.

Several variants of the densest subgraph problem constrained on the size of the subgraph $|S|$ have been studied: finding the densest k-subgraph ($|S| = k$) [9, 11, 12], *at most* k-subgraph ($|S| \leq k$) [13, 14], and *at least* k-subgraph ($|S| \geq k$) [13, 14]. Unlike the densest subgraph problem, when the size constraint is applied, the densest k-subgraph problem becomes **NP**-hard [11]. Furthermore, there is no polynomial time approximation scheme (PTAS) [12]. Approximating the problem of finding at most k-subgraph is shown as hard as the densest k-subgraph problem by Khuller and Saha [14]. To find exactly k-size densest subgraph, Bhaskara et al. [15] gave an $\mathcal{O}\big(n^{1/4+\epsilon}\big)$-approximation algorithm for every $\epsilon > 0$ that runs in $n^{\mathcal{O}(1/\epsilon)}$ time. Andersen and Chellapilla [13] provided a linear time 1/3-approximation algorithm for at least k densest subgraph problem.

The densest common subgraph over multiple graphs: Jethava and Beerenwinkel [4] extended the densest subgraph problem (DCS) for the case of multiple graph snapshots. As a measure, the authors' goal was to maximize the minimum density. Moreover, Semertzidis et al. [1] introduced several variants of this problem by varying the aggregate function of the optimization problem, one variant, BFF-AA, is closely related to the DCS problem discussed in Section 2. DCS which maximizes the total density can be solved exactly through a reduction to the densest subgraph problem, and is consequently polynomial. The hardness of DCS variants has been addressed [16]. For a survey on the densest subgraph problem and its variants, we refer the reader to a recent survey by Lanciano et al. [17].

Overlapping densest subgraphs of a single graph: Finding multiple dense subgraphs in a single graph which allows graphs to be overlapped is studied by adding a hard constraint to control the overlap of subgraphs [18]. Later, Galbrun et al. [19] formulated the same problem adding a penalty in the objective function for the overlap.

The difference between our problem and the works of Balalau et al. [18] and Galbrun et al. [19] is that our goal is to find a collection of dense subgraphs over multiple graph snapshots (one dense subgraph for each graph snapshot) while they discover a set of dense subgraphs within a single graph. Due to this difference, we want to reward similar subgraphs while the authors want to penalize similar subgraphs.

Other density measures: We use the ratio of edges over the vertices as our measure as it allows us to compute it efficiently. Alternative measures have been also considered. One option is to use the proportion of edges instead, that is, $|E|/\binom{|V|}{2}$. The issue with this measure is that a single edge yields the highest density of 1. Moreover, finding the largest graph with the edge proportion of 1 is equal to finding a clique, a classic problem that does not allow any good approximation [20]. As an alternative approach, Tsourakakis et al. [21] proposed finding subgraphs with large score $|E| - \alpha\binom{|V|}{2}$. Optimizing this measure is an **NP**-hard problem but an algorithm similar to the one given by Asahiro et al. [9] leads to an additive approximation guarantee. In a similar vein, Tatti [22] considered subgraphs maximizing $|E| - \alpha|V|$ and showed that they form a nested structure similar to k-core decomposition. An alternative measure called triangle-density has been proposed by Tsourakakis [23] as a ratio of triangles and vertices, possibly producing smaller graphs. Like the density, optimizing this measure can be done in polynomial time. We leave adopting these measures as a future work.

5 Experimental evaluation

The goal of this section is to experimentally evaluate our algorithms. We first generate several synthetic datasets and plant dense subgraph components, in each snapshot and test how well our algorithms discover the ground truth. Next, we study the performance of the algorithm on real-world temporal datasets in terms of running time. We compare our results with the solutions obtained with the densest common subgraph which maximizes the total density [1] and the sum of densities of individually densest subgraphs [7]. Finally, we present interpretative results from two case studies.

We implemented the algorithms in Python[1] and performed the experiments using a 2.4GHz Intel Core i5 processor and 16GB RAM.

Synthetic datasets: Next, we explain in detail how the synthetic datasets are generated, the statistics and the related parameters are given in Table 1.

Each dataset consists of k graphs given as $(G_1, \ldots, G_k)$. We split the vertex set into dense and sparse components V^d and V^s. To generate the ith snapshot we create two components V_i^d and V_i^s by starting from V^d and V^s and moving vertices from V^s to V^d with a probability of η_i. The probability η_i is selected randomly for each snapshot from a uniform distribution $[0.01, 0.09]$. Once the vertices are generated, we sample the edges using a stochastic block model, with the edge probabilities being p_d, p_s, and p_c for dense component, sparse component, and cross edges, respectively. We created 3 such synthetic datasets in total to test our algorithms. We consider V^d as the ground truth vertex set and we denote the density of V_d by d_{true}.

[1] The source code is available at https://version.helsinki.fi/dacs/jaccard-constrained-densest-subgraph.

Table 1: Characteristics of synthetic datasets. Here, $|V^d|$ and $|V^s|$ give initial number of dense and sparse vertices respectively, $\mathrm{E}[|E|]$ is the expected number of edges, k is the number of snapshots, p_d, p_s, and p_c give the dense, sparse, and cross edge probabilities, $J_{min} = \min_{i<j} J(V_i^d, V_j^d)$ is the minimum Jaccard index between ground truth sets, d_{true} is the ground truth density of dense components, d_{dcs} gives the density of a common subgraph which maximizes the total density, and d_{ind} gives the sum of densities of individual densest subgraph from each graph snapshot.

Dataset	$\lvert V^d\rvert$	$\lvert V^s\rvert$	$\mathrm{E}[\lvert E\rvert]$	k	p_d	p_s	p_c	d_{true}	d_{dcs}	d_{ind}	J_{min}
Syn-1	120	1 200	3 922	5	0.06	0.005	0.002	27.4	17.66	27.62	0.45
Syn-2	500	3 500	12 136.29	7	0.05	0.0003	0.0003	112.11	87.27	112.12	0.53
Syn-3	350	3 500	32 015	5	0.06	0.005	0.002	67.96	52.18	67.96	0.49

Table 2: Computational statistics from the experiments with the synthetic datasets using HARD algorithm. Here, α is the constraint, i gives the number of iterations, d_{dis} is the discovered sum of densities, J_{min} and J_{avg} provide the minimum and average pairwise Jaccard coefficients between discovered sets of vertices, respectively, $\rho = k^{-1} \sum J\left(S_i, V_i^d\right)$ gives the average Jaccard index between discovered and ground-truth sets of vertices, and time gives the computational time in seconds.

Data	α	d_{dis}	J_{min}	J_{avg}	ρ	time	i
Syn-1	0.3	27.62	0.43	0.48	0.97	6	4
	0.5	25.23	0.5	0.52	0.88	5	4
	0.7	19.98	0.7	0.82	0.73	4	3
Syn-2	0.3	112.12	0.53	0.66	1	48	3
	0.5	112.12	0.53	0.66	1	50	3
	0.7	102.12	0.7	0.76	0.92	69	4
Syn-3	0.3	67.96	0.49	0.65	1	35	3
	0.5	67.43	0.5	0.65	0.99	34	3
	0.7	60.68	0.7	0.76	0.91	31	3

Results of synthetic datasets: We report our results for the JCDS and JWDS problems in Tables 2 and 3 respectively.

First, we compare the discovered density, d_{dis} in Table 2, with the ground truth density, d_{true} in Table 1. For all synthetic datasets, HARD obtains approximately similar densities or better densities compared to the ground truth density.

We compare the discovered sets against the ground truth by computing the Jaccard index between the sets, shown in the ρ column in Table 2. We see that we achieve high overlap with the ground truth: ρ column shows at least 0.97 for all the datasets. High values of ρ are expected as our synthetic datasets have a prominent dense component.

Next let us examine d_{dis} column of Table 2 separately for each dataset. As expected, d_{dis} increases as we decrease α. Finally, we see that the minimum Jaccard index J_{min} between discovered vertices increases as α increases.

Next, we consider the results of the JWDS problem, shown in Table 3. First, we observe that the discovered density values d_{dis} approximately match each other, that is, both ITR and GRD perform equally well in terms of densities. A similar result holds

Table 3: Computational statistics from the experiments for synthetic datasets using ITR and GRD algorithms. Here, λ is the parameter in $q(\cdot;\lambda)$, i is the number of iterations using ITR, columns d_{dis} and q are the sum of densities and scores of the discovered sets, J_{min} and J_{avg} provide the minimum and average Jaccard index between discovered sets, columns ρ give the average Jaccard index between discovered and ground truth sets, and columns time give the computational time in seconds.

		ITR							GRD					
Data	λ	d_{dis}	q	J_{min}	J_{avg}	ρ	time	i	d_{dis}	q	J_{min}	J_{avg}	ρ	time
Syn-1	0.3	27.59	29.09	0.44	0.5	0.97	18	3	27.6	29.09	0.44	0.5	0.97	31
	0.5	27.59	30.1	0.44	0.5	0.97	19	3	27.6	30.08	0.44	0.5	0.97	29
	0.7	27.47	31.12	0.45	0.52	0.98	25	4	27.6	31.06	0.43	0.49	0.98	33
Syn-2	0.3	112.12	116.3	0.53	0.66	1	121	3	112.12	116.3	0.53	0.66	1	276
	0.5	112.12	119.09	0.53	0.66	1	118	3	112.12	119.09	0.53	0.66	1	266
	0.7	112.11	121.88	0.53	0.66	1	116	3	112.11	121.88	0.53	0.66	1	279
Syn-3	0.3	67.96	69.92	0.49	0.65	1	64	3	67.96	69.91	0.65	0.65	1	251
	0.5	67.96	71.22	0.49	0.65	1	71	3	67.96	71.22	0.49	0.65	1	288
	0.7	72.53	67.96	0.49	0.65	1	80	3	67.96	72.52	0.49	0.65	1	273

also for the scores $q(\cdot;\lambda)$ and minimum Jaccard coefficients J_{min}. However, ITR runs faster than GRD. This is probably due to the fact that GRD takes more time to select the next vertex to delete, which is the bottleneck in both algorithms despite having the same asymptotic time complexity per iteration in ITR and overall time complexity in GRD. Next, we compare the discovered sets to the ground truth, given in columns ρ. We see both algorithms give similar values which indicates equally good performance of ITR and GRD.

Our next step is to study the effect of the input parameters: α or λ. First, we observe Figure 2 which demonstrates ρ and discovered density d_{dis} as a function of α. In Figure 2a, we see that ρ remains relatively constant in 0.97 at the start and as we increase α beyond 0.45 it starts to decrease more prominently. That is because J_{min} which gives the minimum pairwise Jaccard coefficient between the planted dense components is 0.45 for *Syn-1* dataset. When we set $\alpha > 0.45$, HARD deviates from the ground truth as the ground truth no longer satisfies the constraints. Let us now consider Figure 2b which demonstrates how the discovered sum of densities changes with respect to α. In Figure 2b, we observe that d_{dis} remains constant until 0.45 and decreases further as we increase α.

Next, we observe Figure 3 which demonstrates ρ as a function of λ. In Figure 3a, we see that ρ gradually decreases as we increase λ. This is due to the fact that when we increase the weight of the constraint part of q, the algorithms try to find dense sets with higher Jaccard coefficients which eventually forces to deviate from their ground truth. Furthermore, if we set $\lambda = 2$, we can see a drastic change in ρ.

Let us now consider Figure 3b which demonstrates how the discovered sum of densities changes with respect to λ. We see the decreasing trend showing that the Jaccard term in the objective function starts to dominate with the increase of λ. As expected, in Figure 3c we see that both ITR and GRD yield increasing scores when we increase λ.

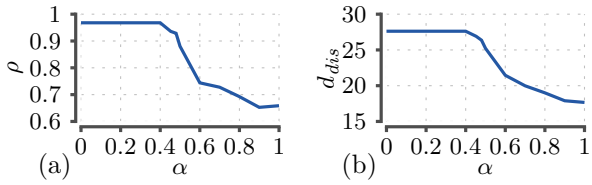

Figure 2: Jaccard index to the ground truth ρ as a function of α for HARD in Figure 2a and discovered density d_{dis} as a function of α for HARD in Figure 2b. This experiment was performed using *Syn-1* dataset.

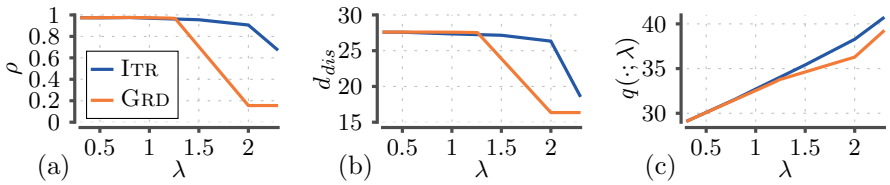

Figure 3: Average Jaccard index to the ground truth ρ as a function of λ in Figure 3a. Discovered density d_{dis} as a function of λ in Figure 3b. Scores $q(\cdot;\lambda)$ as a function of λ in Figure 3c. This experiment was performed using *Syn-1* dataset.

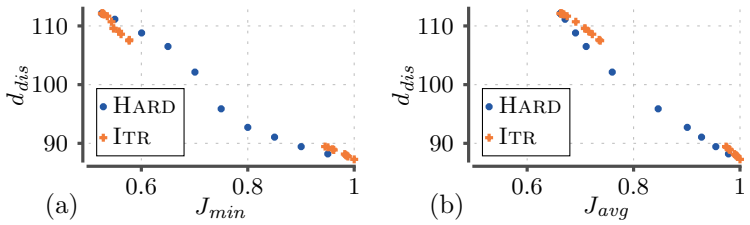

Figure 4: The discovered density d_{dis} as a function of J_{min} for HARD and ITR in Figure 4a and discovered density d_{dis} as a function of J_{avg} for HARD and ITR in Figure 4b. This experiment was performed using *Syn-2* dataset.

Note that the problems JCDS and JWDS optimize two different objective functions. The JCDS problem focuses solely on maximizing the sum of the induced densities while satisfying the similarity constraints. On the other hand, JWDS maximizes the sum of total density and weighted Jaccard indices without explicitly enforcing similarity constraints over the solution sets. Next, we study how the discovered density behaves as a function of J_{min} and J_{avg} for HARD and ITR algorithms, as shown in Figure 4. Let us first observe Figure 4a. We see that HARD achieves higher density for low J_{min} values. On the other hand, ITR performs better with high J_{min} values even though ITR optimizes a different problem. The discrepancy is due

Table 4: Characteristics of real-world datasets. Here, $|V|$ gives the number of vertices, $|E|$ is the expected number of edges, k is the number of snapshots, d_{dcs} gives the density of a common subgraph which maximizes the total density, d_{ind} gives the sum of densities of individual densest subgraph from each graph snapshot, and J_{min}, J_{max}, and J_{avg} give the minimum, maximum, and average pairwise Jaccard coefficients between the set of individual densest subgraphs from each graph snapshot, respectively.

Data	$\|V\|$	$\|E\|$	k	d_{dcs}	d_{ind}	J_{min}	J_{max}	J_{avg}
Twitter-#	806	101.2	15	9.8	38.8	0	0.38	0.019
Enron	1 079	23.2	183	52.7	185.94	0	1	0.068
Facebook	4 117	83.13	104	14	88.65	0	0.46	0.005
Students	889	43.68	122	26.32	118.01	0	0.67	0.045
Twitter-user	4 605	109.19	93	23	90.63	0	0.6	0.013
Tumblr	1 980	65.3	89	55.83	103.99	0	0.8	0.131

Table 5: Computational statistics from the experiments with real-world datasets using HARD algorithm. Here, α is the constraint, i gives the number of iterations, d_{dis} is the discovered sum of densities, J_{min} and J_{avg} provide the minimum and average pairwise Jaccard coefficients between discovered sets of vertices, respectively, and time gives the computational time in seconds.

Data	α	d_{dis}	time	i	J_{min}	J_{avg}
Twitter-#	0.3	16.08	0.12	4	0.31	0.53
	0.5	13.23	0.11	4	0.5	0.67
	0.7	11.9	0.05	2	0.71	0.75
Enron	0.3	102.32	1.6	4	0.3	0.51
	0.5	70.98	0.61	3	0.5	0.79
	0.7	54.65	0.32	3	0.71	0.97
Facebook	0.3	16.33	1.37	2	0.5	0.94
	0.5	16.33	1.37	2	0.5	0.94
	0.7	14	0.13	1	1	1
Students	0.3	43.61	1.69	4	0.3	0.69
	0.5	36.36	0.99	3	0.5	0.79
	0.7	29.36	0.98	2	0.7	0.92
Twitter-user	0.3	37.43	1.26	2	0.3	0.62
	0.5	33.32	0.93	2	0.5	0.72
	0.7	23.2	0.38	2	0.8	1
Tumblr	0.3	69.65	0.56	3	0.3	0.57
	0.5	65.38	0.29	2	0.5	0.71
	0.7	58.21	0.22	2	0.75	0.91

to HARD being stuck in a local minimum. Nevertheless, the densities are similar for both algorithms.

In Figure 4b, we observe that ITR achieves higher average Jaccard indices as HARD for the same density. This is expected since ITR uses J_{avg} as a part of its objective function whereas HARD uses *minimum* Jaccard index.

Table 6: Computational statistics from the experiments for real-world datasets. Here, $\lambda = \frac{d_{dcs}}{k}\lambda'$ where λ is the parameter in $q(\cdot;\lambda)$, i gives the number of iterations using ITR, columns d_{dis} are the discovered sum of densities, J_{avg} columns provide the average pairwise Jaccard coefficients between discovered sets of vertices, columns q are the discovered scores, and columns time give the computational time in seconds.

		ITR					GRD			
Data	λ'	d_{dis}	q	J_{avg}	time	i	d_{dis}	q	J_{avg}	time
Twitter-#	0.3	38.57	39.01	0.02	1	3	37.65	37.94	0.01	6
	0.5	37.34	39.53	0.06	2	4	37.65	38.55	0.03	7
	0.7	21.78	43.16	0.45	1	2	35.93	39.68	0.08	6
Enron	0.3	110.54	618.03	0.35	59	4	73.08	529.37	0.32	867
	0.5	104.6	958.13	0.36	52	4	83.95	797.72	0.3	783
	0.7	102.2	1 299.92	0.36	54	4	20	1 277.59	0.37	770
Facebook	0.3	62.22	84.7	0.1	560	13	58.17	67.81	0.04	9 849
	0.5	60.48	100.31	0.11	268	8	57.31	75.4	0.05	9 759
	0.7	56.08	122.24	0.13	205	6	26.75	86.46	0.12	10 290
Students	0.3	52.74	212.88	0.34	215	10	42.37	209.22	0.35	2 312
	0.5	48.65	310.01	0.33	124	6	38.84	321.45	0.36	2 410
	0.7	47.1	414.85	0.33	112	6	37.91	434.34	0.36	2 479
Twitter-user	0.3	17.05	183.64	0.52	318	8	15.86	164.67	0.47	7 577
	0.5	12.04	329.79	0.6	194	6	13.64	264.03	0.47	6 953
	0.7	11.49	474.1	0.62	160	5	11.24	363.56	0.48	6 550
Tumblr	0.3	59.25	630.05	0.77	51	4	63.25	524.77	0.63	1 323
	0.5	59.25	1 010.59	0.77	51	4	62.97	833.87	0.63	1 263
	0.7	59.25	1 391.12	0.77	41	3	62.97	1 142.24	0.63	1 358

Real-world datasets: We consider 6 publicly available, real-world datasets. The details of the datasets are shown in Table 4. *Twitter-#* [24][2] is a hashtag network where vertices correspond to hashtags and edges corresponds to the interactions where two hashtags appear in a tweet. This dataset contains 15 such daily graph snapshots in total. *Enron*[3] is a popular dataset which contains email communication data within senior management of Enron company. It contains 183 daily snapshots in which the daily email count is at least 5. *Facebook* [25][4] is a network of Facebook users in New Orleans regional community. It contains a set of Facebook wall posts among these users from the 9th of June to the 20th of August, 2006. *Students*[5] is an online message network at the University of California, Irvine. It spans over 122 days. *Twitter-user* [5][6] is a network of twitter users in Helsinki 2013. The edges correspond to the mentions of users. *Tumblr* [26][7] contains phrases or quote mentions that appeared in blogs and news media. It contains author and meme interactions of users over 3 months from February to April 2009.

Results of real-world datasets: We report the results obtained from the experiments with real-world datasets for the JCDS problem in Table 5 and for the JWDS

[2] https://github.com/ksemer/BestFriendsForever-BFF-
[3] http://www.cs.cmu.edu/~./enron/
[4] https://networkrepository.com/fb-wosn-friends.php
[5] https://toreopsahl.com/datasets/#online_social_network
[6] https://github.com/polinapolina/segmentation-meets-densest-subgraph/tree/master/data
[7] http://snap.stanford.edu/data/memetracker9.html

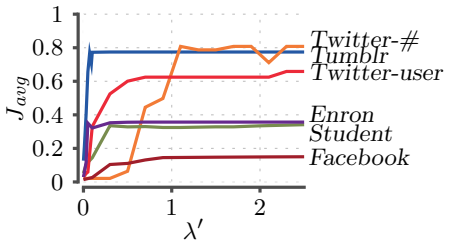

Figure 5: Average Jaccard index J_{avg} as a function of λ' for the ITR algorithm.

problem in Table 6. In Table 5, we can see that the discovered density values d_{dis} exceed the density of the common subgraph d_{dcs} (see column d_{dcs} in Table 4). As expected, we can observe that d_{dis} increases as α decreases. Moreover, we see that running time which is shown in the *time* column increases when we increase α. Furthermore, we see in the i column that HARD converges in less than 5 iterations which is reasonable in practice. Finally, we see that J_{min} and J_{avg} increase as α increases, as expected.

Next, let us focus on the performance of ITR and GRD algorithms with real-world datasets that are given in Table 6. Note that, as a normalization step, we set $\lambda = \frac{d_{dcs}}{k}\lambda'$ for the real-world experiments where λ is the parameter in $q(\cdot;\lambda)$.

First, we compare the scores q obtained using ITR and GRD. As we can see, apart from the cases in *Tumblr* dataset, ITR achieves a greater score than GRD. As expected, we see that q and J_{avg} increase and d_{dis} decreases as a function of λ' for both algorithms. Furthermore, we observe in *time* columns, that ITR runs faster than GRD. Next, let us observe column i which gives the number of iterations with ITR algorithm. We can see that we have at most 13 number of iterations which is reasonable to deal with real-world datasets.

Next, we study the effect of λ' on J_{avg} across the real-world datasets as shown in Figure 5. In general, we observe that J_{avg} has an increasing trend with λ' up to a certain value and stays almost constant afterwards.

Case studies: In this section, we present two case-studies and analyze their results which illustrate trending twitter hash tags over a span of 8 days and strongly connected DBLP co-authorships over a decade, under several Jaccard-constrained and weighted environments.

Twitter-8 dataset: *Twitter-8* contains a hashtag network from November 2013. This dataset was created by extracting the first 8 daily graph snapshots from the *Twitter-#* dataset. Here, each vertex of the graph represents a specific hashtag. As seen in the tags from Table 7, Formula-1 racing car event which occurred on Abu Dhabi has been trending during the period. By varying input parameters (α or λ), we discovered different sets of dense subgraphs. Note that often, due to simplicity, we give the set of hashtags with reference to the DCS solution which maximizes the sum of the densities. For example, if we set $\alpha = 0.8$, only the *indiangp* tag has been added to Day 1 to the DCS. On Day 5, tags *teamlh* and *japanesegp* have been added to the dense hashtag collection which indicates that additional racing car event related tags are trending. For $\alpha = 0.8$, the discovered density is 7.79 whereas letting $\alpha = 0.5$ increases

Table 7: Twitter hash tags discovered for *Twitter-8* dataset.

DCS: abudhabigp, fp1, abudhabi, guti, f1, pushpush, skyf1, hulk, allowing, bottas, kimi, fp3, fp2, density: 7.15, size: 13	
HARD algorithm: $\alpha = 0.8$, density: 7.79	
Day 1	**added**: indiangp (14)
Day 2	**added**: teamlh, indiangp, india (16)
Day 3–4	DCS (13)
Day 5	**added**: indiangp, teamlh, japanesegp (16)
Day 6–8	DCS (13)
HARD algorithm: $\alpha = 0.5$, density: 9.63	
Day 1	**added**: indiangp (14)
Day 2	**added**: teamlh, indiangp, india, dubai (17)
Day 3	DCS (13)
Day 4	**added**: sandracing, japanesegp, f1ad, williamstrackwalk, mclaren50, **removed**: bottas (17)
Day 5	**added**: hothot, indiangp, justsaying, lhf1, teamlh, **removed**: abudhabi, bottas (16)
Day 6	**added**: fridayschedule, toomanyts, pitlaneparking, socialfriday (17)
Day 7	**added**: quali, q1, q2, mclarenlive (17)
Day 8	**added**: wenevergiveup, lhf1, unleashthehulk, formula1, xs, **removed**: abudhabi, bottas (16)
ITR algorithm: $\lambda = 0.8$, density: 14.55, objective: 21.25	
Day 1	indiangp, skyf1, kimi, f1 (4)
Day 2	abudhabigp, abudhabi, f1, skyf1, mexico, us, germany, brazil, whys (9)
Day 3	kimi, skyf1, abudhabigp, f1 (4)
Day 4	abudhabigp, abudhabi, f1, skyf1, bottas, williamstrackwalk (6)
Day 5	abudhabigp, english, arabic, spanish, french, danish, swedish, f1, endimpunitybh, skyf1, bahrain (11)
Day 6	abudhabigp, fp1, abudhabi, guti, f1, skyf1, hulk, allowin, bottas, kimi, fp3, fp2 (12)
Day 7	abudhabigp, abudhabi, guti, f1, pushpush, skyf1, hulk, allowin, bottas, kimi, fp3, quali (12)
Day 8	atxfloods, mutualaid, abudhabi, abudhabigp, guti, f1, pushpush, skyf1, hulk, allowin, bottas, kimi, texas, austin, pets, horses (16)

the density up to 9.63. To gain higher density, some of the tags *sandracing*, *formula-1*, *wenevergiveup*, and *quali* have newly added to several daily dense collections and *abudhabi* is removed from some of the dense collections but *abudhabigp* still remains in dense subgraphs. For the results of JWDS, we can observe more new tags like *bahrain*, *english*, *arabic*, *french*, *danish*, and *swedish* have appeared which seem not directly related to racing car event. Moreover, the new dense collection gives a higher density of 14.55. The larger value for JWDS is evident since JWDS finds solution sets without enforcing similarity constraints hardly as JCDS.

We see that discovered hashtags vary during 8 days. For example, let us consider the results of HARD. The tag *indiangp* was included only during Day 1, 2, and 5. However, *abudhabigp* tag was included over the entire period. The tags like *teamlh* and *japanesegp* were included only during Day 5.

DBLP dataset: The *DBLP*[2] dataset contains annual co-authorship connections in top database and data mining conferences over a decade from 2006 to 2015. An edge corresponds to being a co-author pair in a publication. We pick authors who

Table 8: Authors discovered for *DBLP* dataset and the number in parentheses indicates the size of the set.

DCS: J.Pei, H.Wang, J.Han, J.XuYu, X.Yu, W.Wang, W.Fan, L.Qin, L.Chang, Y.Sun, X.Lin, W.Zhang, M.AamirCheema, J.Gao, C.Wang, X.Yan, Y.Zhang, P.S.Yu, H.Cheng, C.C.Aggarwal, B.Ding, density: 15.9, size: 21	
HARD algorithm: $\alpha = 0.4$, density: 21.86	
2006	**added**: H.Liu, D.W.Cheung, G.PuiCheongFung, P.Wang, W.Wang, B.Shi, C.Chen, J.Xu, S.Wang, F.Korn, F.Zhu, J.Yang, D.Xin, J.Zhang, A.Wai-CheeFu, J.Cheng, **removed**: J.Gao, W.Fan (35)
2007	**added**: H.Liu, L.Liu, B.Gedik, M.Kitsuregawa, X.Yin, X.Zhou, G.PuiCheongFung, C.Chen, K.Chen-ChuanChang, E.Bouillet, S.Wang, F.Zhu, K.Wu, **removed**: Y.Sun, J.Pei, W.Wang (31)
2008	**added**: K.Zhang, D.W.Cheung, Z.Yin, X.Yin, T.Wu, O.Verscheure, C.Chen, B.Zhao, C.XideLin, F.Zhu, B.Jiang, J.Cheng, **removed**: Y.Zhang, W.Zhang, W.Wang, L.Qin, C.C.Aggarwal (28)
2009	**added**: C.Zhang, D.Lo, T.Wu, **removed**: C.C.Aggarwal, X.Yu, C.Wang (21)
2010	**added**: L.Khan, B.M.Thuraisingham, **removed**: X.Yu, H.Wang (21)
2011	**added**: Y.Ke, D.S.Turaga, M.Winslett, B.M.Thuraisingham, D.Lo, Z.Li, L.Khan, T.Weninger, **removed**: W.Wang (28)
2012	**added**: W.Yu, H.Gonzalez, L.Chen, **removed**: C.C.Aggarwal, C.Wang, W.Wang (21)
2013	**added**: S.Ma, W.Liang, C.Liu, B.Zhao, C.XideLin, Q.Gu, Y.Li, H.Ji, W.Yu, T.Weninger (31)
2014	**added**: K.Zhang, J.Hu, J.Zhou, A.K.Singh, Y.Chang, J.Ye, H.Ji, W.Yu, **removed**: X.Yu (28)
2015	**added**: S.Xie, W.Wang, B.Zhao, J.Wang, Y.Chang, X.Wang, Q.Hu, H.Ji, **removed**: X.Yu (28)
ITR algorithm: $\lambda = 0.8$, density: 17.47, objective: 48.13	
2006	**removed**: Y.Sun, W.Zhang, M.AamirCheema, X.Yu, L.Chang, C.Wang (15)
2007	**removed**: W.Zhang, M.AamirCheema, X.Yu, L.Chang, C.Wang (16)
2008	**removed**: L.Chang, M.AamirCheema, X.Yu, C.Wang (17)
2009	**removed**: M.AamirCheema (20)
2010–12	DCS (21)
2013	**removed**: M.AamirCheema (20)
2014	**removed**: W.Wang (20)
2015	**removed**: H.Cheng, W.Wang (19)

have co-authored papers at least in 4 different years during a period of 10 years. As shown in Table 8, we have uncovered a set of prominent co-authors. We see that the algorithms provide denser solutions than the densest common subgraph. For example, JCDS achieves a density of 21.86 while JWDS induces density value of 17.47 with score of 48.13.

We see that there is a temporal dependency of the collaborations between authors. The set found by ITR matches to the densest common subgraph only during 2010–2012, and during other years some authors are removed.

6 Concluding remarks

We introduced a Jaccard constrained, dense subgraph discovery problem (JCDS) for graphs with multiple snapshots. Here, our goal was to find a dense subset of vertices

from each graph snapshot such that the sum of densities is maximized while pairwise Jaccard index constraint is satisfied.

We considered also an alternative problem (JWDS), where we incorporated Jaccard indices directly into the objective function, weighted by a user parameter.

We proved that both problems are **NP**-hard and hence we resorted to heuristics. First, we designed a greedy, iterative algorithm for JCDS which runs in $\mathcal{O}\big(nk^2 + m\big)$ time, where k is the length of the graph sequence and n and m denote number of vertices and total number of edges respectively. For JWDS, we designed an iterative algorithm which runs in $\mathcal{O}\big(n^2k^2 + m\log n + k^3n\big)$ time per single iteration, and a greedy algorithm which runs in $\mathcal{O}\big(n^2k^2 + m\log n + k^3n\big)$ time.

We experimentally showed that the number of iterations was low in iterative algorithms, and that the algorithms could find the ground truth using synthetic datasets and could discover dense collections in real-world datasets. We also studied the effect of our user parameters: the threshold for the Jaccard index α in JCDS, and weight λ for Jaccard indices in JWDS. Finally, we performed two case studies showing interpretable results.

The paper introduces several interesting directions for future work. In this paper, we enforced the pairwise Jaccard constraint between all available pairs of snapshots. However, we can relax this constraint further by letting only a portion of sets which lies within a specific window to assure the Jaccard similarity constraint which may lead to future work. Another possible direction is adopting different types of density for our problem setting.

References

[1] Semertzidis, K., Pitoura, E., Terzi, E., Tsaparas, P.: Finding lasting dense subgraphs. DMKD **33**(5), 1417–1445 (2019)

[2] Du, X., Jin, R., Ding, L., Lee, V.E., Thornton, J.H.: Migration motif: A spatial-temporal pattern mining approach for financial markets. In: KDD, pp. 1135–1144 (2009)

[3] Fratkin, E., Naughton, B.T., Brutlag, D.L., Batzoglou, S.: Motifcut: regulatory motifs finding with maximum density subgraphs. Bioinformatics **22**(14), 150–157 (2006)

[4] Jethava, V., Beerenwinkel, N.: Finding dense subgraphs in relational graphs. In: ECMLPKDD, pp. 641–654 (2015)

[5] Rozenshtein, P., Bonchi, F., Gionis, A., Sozio, M., Tatti, N.: Finding events in temporal networks: segmentation meets densest subgraph discovery. KAIS **62**(4), 1611–1639 (2020)

[6] Galimberti, E., Bonchi, F., Gullo, F., Lanciano, T.: Core decomposition in multilayer networks: Theory, algorithms, and applications. TKDD **14**(1), 1–40 (2020)

[7] Goldberg, A.V.: Finding a maximum density subgraph (1984)

[8] Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: APPROX, pp. 84–95 (2000)

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

[10] Wickrama Arachchi, C., Tatti, N.: Jaccard-constrained dense subgraph discovery. In: Discovery Science, pp. 508–522 (2023). Springer

[11] Feige, U., Peleg, D., Kortsarz, G.: The dense k-subgraph problem. Algorithmica **29**, 410–421 (2001)

[12] Khot, S.: Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. SIAM Journal on Computing **36**(4), 1025–1071 (2006)

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

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

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

[16] Charikar, M., Naamad, Y., Wu, J.: On Finding Dense Common Subgraphs. arXiv (2018). https://doi.org/10.48550/ARXIV.1802.06361

[17] Lanciano, T., Miyauchi, A., Fazzone, A., Bonchi, F.: A survey on the densest subgraph problem and its variants. arXiv preprint arXiv:2303.14467 (2023)

[18] Balalau, O.D., Bonchi, F., Chan, T.H., Gullo, F., Sozio, M.: Finding subgraphs with maximum total density and limited overlap. In: WSDM, pp. 379–388 (2015)

[19] Galbrun, E., Gionis, A., Tatti, N.: Top-k overlapping densest subgraphs. DMKD **30**(5), 1134–1165 (2016)

[20] Håstad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. In: STOC, pp. 627–636 (1996)

[21] Tsourakakis, C., Bonchi, F., Gionis, A., Gullo, F., Tsiarli, M.: Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: KDD, pp. 104–112 (2013)

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

[23] Tsourakakis, C.: The k-clique densest subgraph problem. In: WWW, pp. 1122–1132 (2015)

[24] Tsantarliotis, P., Pitoura, E.: Topic detection using a critical term graph on news-related tweets. In: EDBT/ICDT Workshops, pp. 177–182 (2015)

[25] Viswanath, B., Mislove, A., Cha, M., Gummadi, K.P.: On the evolution of user interaction in facebook. In: WOSN, pp. 37–42 (2009)

[26] Leskovec, J., Backstrom, L., Kleinberg, J.: Meme-tracking and the dynamics of the news cycle. In: KDD, pp. 497–506 (2009)

[27] Chlebík, M., Chlebíková, J.: Approximation hardness for small occurrence instances of np-hard problems. In: CIAC, pp. 152–164 (2003)

A Appendix

A.1 Computational complexity proof of JCDS problem

Proof of Proposition 1. We prove the hardness from k-Clique, a problem where, given a graph H, we are asked if there is a clique of size at least k.

Assume that we are given H with n vertices, and $k \geq 2$. Let $r = 3n$. Our graph sequence $\mathcal{G} = \{G_1, \ldots, G_r\}$ is as follows: The first graph snapshot G_1 consists of the graph H and additional k singleton vertices, say U. The remaining snapshots are identical, and consist of a k-clique over the vertices U. We set $\alpha = 1/2$.

We claim that there is a collection of subset of vertices $\mathcal{S} = \{S_1, \ldots, S_r\}$ yielding $d(\mathcal{S}) = (k-1)(1/4 + (r-1)/2)$ if and only if there is a k-clique in H.

Let $\mathcal{S}$ be the solution to JCDS. Assume $d(\mathcal{S}) \geq (k-1)(1/4+(r-1)/2)$. We show that there is a k-clique in H.

First, we can safely assume that $S_2, \ldots, S_r$ are all equal. We argue that the sets S_2 to S_r are equal to U. We prove our argument by contradiction.

If $S_2 \setminus U \neq \emptyset$, then we can show that $d(S_2) \leq \frac{k(k-1)}{2(k+1)}$. Due to optimality of $\mathcal{S}$,

$$(r-1)\frac{k(k-1)}{2k} = (r-1)d(U) \leq d(\mathcal{S}) \leq (r-1)\frac{k(k-1)}{2(k+1)} + d(S_1)$$

which we can rewrite as

$$d(S_1) \geq (r-1)\frac{k(k-1)}{2}\left(\frac{1}{k} - \frac{1}{k+1}\right) = (r-1)\frac{k-1}{2(k+1)} \geq \frac{n}{2},$$

which is a contradiction, since $d(S_1) \leq (n-1)/2$. Consequently, $S_2 \subseteq U$. Now, if $S_2 \neq U$, then $d(S_2) \leq \frac{(k-1)(k-2)}{2k} < \frac{k(k-1)}{2(k+1)}$, and we can repeat the previous argument. Thus, $S_2 = U$.

Let $x = |S_1 \cap S_2|$ and let $y = |S_1 \setminus U|$. Then, by definition, $\frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{x}{y+k} \geq \frac{1}{2}$, or, since also $x \leq k$,

$$y + k \leq 2x \leq x + k \quad .$$

Consequently, $k \geq x \geq y$. The density of S_1 is at most

$$d(S_1) \leq \frac{y(y-1)}{2(x+y)} \leq \frac{y(y-1)}{2(y+y)} = \frac{y-1}{4} \leq \frac{k-1}{4} \quad .$$

By assumption $d(S_1) \geq \frac{k-1}{4}$. Therefore equalities in the previous equation hold only if $x = y = k$ and $S_1 \setminus S_2$ is a k-clique, proving the claim.

On the other hand, assume there is a clique C of size k in H. Then set $S_1 = C \cup U$ and $S_i = U$ for $i = 2, \ldots, r$. Immediately, $d(\mathcal{S}) = (k-1)(1/4+(r-1)/2)$ and Jaccard index $J(S_i, S_j) \geq 1/2$ for all i and j, proving the claim. □

A.2 Computational complexity proof of JWDS problem

Proof of Proposition 2. We will prove the hardness by reducing from a 2-bounded 3-set packing problem 3DM-2, a problem where we are given a set of items U, a family

of sets $\mathcal{C}$ each of size 3 such that each item in U is included in exactly two sets, and are asked to find a disjoint cover [27].

Assume that we are given an instance with $r = |U|$ items and $\ell = |\mathcal{C}|$ sets. For each set C_i we introduce two vertices v_i and v'_i, and for each item u_i we introduce a vertex w_i. We also introduce four additional vertices z_1, z_2, z_3, and z_4. In total, we have $n = 2\ell + r + 4$ vertices.

Let u_i be an item and let C_a and C_b be two sets containing u_i. We add two snapshots: the first graph G_i contains (z_1, z_2), (z_1, z_3), (w_i, z_1), (z_1, v_a), and (z_1, v_b) edges and the second graph G'_i contains (z_1, z_2), (z_1, z_3), (w_i, z_1), (z_1, v'_a), and (z_1, v'_b) edges.

We also add $q = 2r(10n)^4$ graphs F_j, each with three edges (z_1, z_2), (z_1, z_3), and (z_1, z_4). We set $\lambda = 0.55/q$.

Let $\mathcal{S}$ be the optimal solution. We claim that

$$q(\mathcal{S}) \geq T_1 + T_2 + T_3, \text{ where } T_1 = q\frac{3}{4} + \lambda\binom{q}{2}, \; T_2 = 2r(\frac{3}{4} + 0.55\frac{3}{5}), \text{ and} \\ T_3 = 3\lambda\binom{2r}{2} + 8\lambda p + 4\lambda r/5, \tag{2}$$

if and only if there is a matching with p sets.

Assume that Eq. 2 holds. Let Q_j be the optimal subgraph in F_j. Write S_i to be the optimal subgraph in G_i and S'_i to be the optimal subgraph in G'_i. Due to symmetry, we can safely assume that the subgraphs Q_j are all equal. Next, we claim that $Q_j = \{z_1, z_2, z_3, z_4\}$. Assume otherwise. Case 1: let $Q_j = \{z_1, z_2, z_3\}$. The score $q(\mathcal{S})$ contains $2r$ density terms from S_i and S'_i with each density at most 1. In addition, Jaccard index is always less than or equal to 1. Therefore $q(\mathcal{S}) \leq \frac{2q}{3} + 2r + \lambda(2rq + \binom{2r}{2}) < \frac{2q}{3} + 6r$. The difference in the density value of Q_js between our case and $Q_j = \{z_1, z_2, z_3, z_4\}$ case is $\frac{q}{12} > 2 \times (40)^4 r$ which is a contradiction. Case 2: let $Q_j = \{z_1, z_2\}$ or $Q_j = \{z_1, z_3\}$. This follows a similar proof as Case 1 proving our claim. Therefore the values of the densities and Jaccard indices of Q_js now correspond to T_1.

Next, we claim that S_i (and S'_i) consists of 4 vertices and 3 edges, two of them being (z_1, z_2) and (z_1, z_3). Fix S_i with $y = |E(S_i)|$ edges and $x = |S_i|$ vertices. Let $z = |S_i \cap \{z_1, z_2, z_3, z_4\}|$ be the intersection with Q_j. We can show that the score is

$$q(\mathcal{S}) = \frac{y}{x} + 0.55\frac{z}{x + 4 - z} + R + C,$$

where R contains the Jaccard terms using S_i and not any Q_j, and C contains the remaining densities and Jaccard terms not depending on S_i. The first two terms form a fraction with a denominator of at most $(10n)^2$. Consequently, any changes to x, y, and z change the first two terms by at least $(10n)^{-4}$. Note that R contains only $2r - 1$ terms, and due to λ, we have $R < (10n)^{-4}$. In other words, $\mathcal{S}$ must optimize the first two terms. Since there are only 6 non-singleton vertices in G_i, $x \leq 6$. Moreover, $z \leq \min(3, x)$ and $y \leq \min(5, x - 1)$. Enumerating all the possible combinations show that $x = 4$ and $z = y = 3$ yield optimal score. This is only possible if S_i consists of 4

vertices and 3 edges, two of them being (z_1, z_2) and (z_1, z_3). Now, densities of S_i (and S'_i) and Jaccard indices between S_i (and S'_i) and Q_j correspond to T_2.

Finally, let us look now at the Jaccard terms between S_i and/or S'_j. These terms will constitute T_3. Let a be the number of vertices v_i or v'_i that are included in 3 subgraphs; any such vertex will yield 3 Jaccard terms of value 1. Let b be the number of vertices v_i, v'_i, or w_i that are included in 2 subgraphs; any such vertex will yield one Jaccard term of value 1. The remaining Jaccard terms between S_i and S'_i are all of value $3/5$. In summary, the terms are equal to

$$\lambda\binom{2r}{2}\frac{3}{5} + \lambda a 3\left(1-\frac{3}{5}\right) + \lambda b\left(1-\frac{3}{5}\right) = \frac{\lambda 3}{5}\binom{2r}{2} + \lambda a\frac{4}{5} + \frac{\lambda(a+b)2}{5} \quad .$$

Assume that $a < 2p$. Then since $a + b \leq 2r$ these terms are less than T_3, which is a contradiction. Therefore, $a \geq 2p$. Now, there are p vertices in $\{v_i\}$ or p vertices in $\{v'_i\}$ that are included in 3 subgraphs. These sets correspond to matchings, and at least one of them will have p sets.

To prove the other direction, assume there is a matching $\mathcal{M}$ with p sets. To form the subgraph sequence, we first select (z_1, z_2) and (z_1, z_3) in every set. For $u_i \in C_j \in \mathcal{M}$, we also select (v_j, z_1) in G_i and (v'_j, z_1) in G'_i. For an item u_i not covered by $\mathcal{M}$, we select (w_i, z_1) in G_i and (w_i, z_1) in G'_i. A straightforward calculation shows that this sequence yields the score given in Eq. 2. □

A.3 Computational complexity of Grd

Proof of Proposition 5. Finding the best vertex u and the snapshot S_i requires $\mathcal{O}(\sum_i |\mathcal{P}_{ir}|) \in \mathcal{O}(k\Delta)$ time. Consider deleting u from S_i, and assume that it is the rth vertex being deleted.

Deleting u from its queue requires $\mathcal{O}(\log n)$ time. Upon deletion, we update the degrees of the neighboring vertices in the corresponding queues, in *total* time of $\mathcal{O}(m \log n)$. Updating the intersection and the union sizes requires $\mathcal{O}(k)$ time.

We also need to update the gain coming from Jaccard indices for each group $P \in \mathcal{P}_{jr}$. Only one term changes if $P \notin \mathcal{P}_{ir}$; there are at most $k\Delta$ such groups. Otherwise, if $P \in \mathcal{P}_{ir}$, then $k-1$ terms change; there are at most Δ such groups. In summary, we need $\mathcal{O}(k\Delta)$ time.

Vertex u is included in $\mathcal{O}(k)$ queues. As we remove u from S_i, these queues need to be updated by moving u to the correct queue. A single such update requires deleting u from its current queue, finding (and possibly creating) the new queue, and adding u to it. This can be done in $\mathcal{O}(k + \log n)$ time.

Combining these times, and the fact that there are kn vertices, proves the claim. □