2018 IEEE 34th International Conference on Data Engineering

Mining Density Contrast Subgraphs

Yu Yang*, Lingyang Chu*, Yanyan Zhang*, Zhefeng Wang§, Jian Pei*‡ and Enhong Chen†
*Simon Fraser University, Burnaby, Canada
§Huawei Technologies, Hangzhou, China
‡JD.com, Beijing, China
†University of Science and Technology of China, Hefei, China
{yya119,lca117,yanyanz}@sfu.ca, wangzhefeng@huawei.com, jpei@cs.sfu.ca, cheneh@ustc.edu.cn

***Abstract*—Dense subgraph discovery is a key primitive in many graph mining applications, such as detecting communities in social networks and mining gene correlation from biological data. Most studies on dense subgraph mining only deal with one graph. However, in many applications, we have more than one graph describing relations among a same group of entities. In this paper, given two graphs sharing the same set of vertices, we investigate the problem of detecting subgraphs that contrast the most with respect to density. We call such subgraphs Density Contrast Subgraphs, or DCS in short. Two widely used graph density measures, average degree and graph affinity, are considered. For both density measures, mining DCS is equivalent to mining the densest subgraph from a "difference" graph, which may have both positive and negative edge weights. Due to the existence of negative edge weights, existing dense subgraph detection algorithms cannot identify the subgraph we need. We prove the computational hardness of mining DCS under the two graph density measures and develop efficient algorithms to find DCS. We also conduct extensive experiments on several real-world datasets to evaluate our algorithms. The experimental results show that our algorithms are both effective and efficient.**

I. Introduction

Dense subgraph extraction lies at the core of graph data mining. The problem and its variants have been intensively studied. Most of the existing studies focus on finding the densest subgraph in one network. For example, polynomial time algorithms and efficient approximation algorithms are devised to find the subgraph with maximum average degree [12]. There are also quadratic programming methods for extracting subgraphs with high graph affinity density [16], [19].

In many real-world applications, there are often more than one kind of relations among objects studied. Thus, it is common to have more than one graph describing a same set of objects, one kind of relation captured by one graph. As a result, an interesting contrast data mining problem arises. Given two graphs sharing the same set of vertices, what is the subgraph such that the gap between its density in the two graphs is the largest? We call such a subgraph the **Density Contrast Subgraph** (**DCS**).

To demonstrate the power of DCS, consider the task of surveying and summarizing the trends of an area, say data mining research. Such a task is practical and common for technical writers, academic researchers and graduate students among many others. Based on a database of published data mining papers, how can we detect trends from the database automatically? Angel *et al.* [1] proposed to build a keyword association graph from the input text data, and identify stories/topics via groups of densely-connected keywords from it. For example, applying the method of [1] on data mining papers we may find a topic "scalable tensor factorization", because the words "scalable", "tensor" and "factorization" often co-occur in papers. However, directly extracting dense subgraphs corresponding to densely-connected keywords from a keyword association graph like [1] may not help us detect trends effectively. For example, in our experiments, if we just extract dense subgraphs from the graph indicating pairwise keywords association strength in the titles of data mining papers published in the last 10 years, we find topics "time series" and "feature selection". But these two topics have been intensively investigated ever since and do not present a new trend. In the recent 10 years, according to the graph density measure on the data we have, the topic "time series" even cooled down a little bit.

To detect trends effectively, we take two keyword association graphs into consideration and apply DCS algorithms. Besides the keyword association graph based on papers published recently, we also need the other keyword association graph derived from the papers published in early years. Those groups of keywords whose connection strengths are much tighter in the recent keyword association graph than in the early keyword association graph are identified as trends in data mining. In our experiments we obtained results like "social networks", "matrix factorization" and "unsupervised feature selection". These topics all became popular only in recent years.

DCS can also be applied to detecting current anomalies against historical data. Specifically, we can build a weighted graph where the edge weights are our expectation of how tightly the vertices are connected to each other, which can be derived from, for example, historical data. Then, we observe the current pairwise connection strength of vertices, and build another weighted graph based on our observations. We apply DCS on these two weighted graphs. Some typical application scenarios include detecting emerging traffic hotspot clutters, emerging communities in social networks, and money launderer dark networks.

In this paper, we study the Density Contrast Subgraph problem under two widely adopted graph density measures, average degree and graph affinity. One may notice that for

This work was supported in part by the NSERC Discovery Grant program, the Canada Research Chair program, the NSERC Strategic Grant program, and the National Natural Science Foundation of China (Grants No. U1605251 and 61727809). All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00029

both density measures, we may form a "difference" graph, where the weight of each edge is obtained by the difference of the weights of this edge in the two graphs. However, this does not mean that traditional densest subgraph extraction methods can be applied to find density contrast subgraphs. In the traditional densest subgraph problems, edge weights are always positive. In the difference graph of the density contrast subgraph problem, we may have negative edge weights. The existence of negative edge weights changes the nature of densest subgraph finding substantially. For example, finding the densest subgraph with respect to average degree in a graph with only positive edge weights is polynomial time solvable [12], and has an efficient 2-approximation algorithm [7], while if the graph has negative edge weights, it becomes NP-hard and also hard to approximate as to be proved in Section IV.

To tackle the Density Contrast Subgraph problem, we make several technical contributions. We prove the computational hardness of finding DCS under the two density measures of average degree and graph affinity. For the average degree measure, we also prove it is hard to approximate within a factor of $O(n^{1-\varepsilon})$. An efficient $O(n)$-approximation algorithm is then developed to solve this problem. The DCS problem under the graph affinity measure is also NP-hard, and is a QP (Quadratic Programming) which is non-concave. For this problem, we first devise an efficient 2-Coordinate Descent algorithm that is guaranteed to converge to a KKT point. Based on the 2-coordinate descent algorithm, we give a constructive proof of that edges of negative weights cannot appear in a DCS with respect to graph affinity. Using our construction, we can further improve a KKT point solution to a positive clique solution. A smart initialization heuristic is proposed to reduce the number of initializations for our iterative algorithm, which in experiments brings us speedups of 1-3 orders of magnitude. Extensive empirical studies are conducted to demonstrate the effectiveness and efficiency of our algorithms.

The rest of the paper is organized as follows. We review the related work in Section II. In Section III, we briefly introduce the two density measures used in our work, average degree and graph affinity, and formulate the Density Contarst Subgraph problem. In Section IV, we give our solutions to the DCS problem under the measure of average degree. In Section V, we tackle the DCS problem under the graph affinity measure. We report the experimental results in Section VI and conclude the paper in Section VII.

For the interest of space, we skip some proofs and some experimental results which can be found in the full version of this paper[1].

II. Related Work

Dense subgraph extraction is a key problem in both algorithmic graph theory and graph mining applications [1], [10], [17], [22]. One of the most popular definitions of subgraph density is the average degree. Intensive studies have been conducted on finding a subgraph with the maximum average degree in one single graph [4], [7], [9], [12]. Goldberg [12] first proposed a polynomial time algorithm based on maximum flow. Charikar [7] described a simple greedy algorithm which has an approximation ratio of 2.

[1]https://arxiv.org/abs/1802.06775

Besides average degree, graph affinity, which is a quadratic function $\mathbf{x}^\top A\mathbf{x}$ of a subgraph embedding $\mathbf{x} \in \triangle^n$, is also widely adopted as a measure of subgrah density [8], [16], [19], [24]. Motzkin and Straus [18] proved that, for unweighted graphs, maximizing graph affinity is equivalent to finding the maximum clique in the graph. Pavan and Pelillo [19] first proposed an algorithm based on replicator dynamics to find local maximas of the quadratic function $\mathbf{x}^\top A\mathbf{x}$ on the simplex $\triangle^n$. Liu *et al.* [16] proposed a highly efficient algorithm called SEA (see Appendix) to solve the problem, where the core idea is to use a shrink-and-expand strategy to accelerate the process of finding Karush-Kuhn-Tucker (KKT) points. Wang *et al.* [24] discussed the trade-off between the graph affinity density and subgraph size in extracting dense subgraphs.

Please note that the existing work on maintaining dense subgraphs on temporal graphs [1], [2], [4], [9] cannot solve our problem, although two consecutive snapshots of a temporal graph can be regarded as a special case of the input to DCS. [1], [2], [4], [9] are all for extracting dense subgraphs from the latest snapshot, where for a valid input there are no edges with negative weights. The algorithms in [4], [9] can only deal with unweighted graphs. In our problem, mining DCS from two graphs is equivalent to mining dense subgraphs from a "difference graph", which may have negative edge weights. We show in Section IV that, when the density measure is average degree, the existence of negative edge weights makes our DCS problem NP-hard and hard to approximate. This is dramatically different from extracting densest subgraph with respect to average degree [2], [4], [9], which is polynomial time solvable and has an efficient 2-approximation algorithm. For the graph affinity density measure, [1] considers a general definition of subgraph density where edge density, the discrete version of graph affinity, is used as a special case. However, [1] is for maintaining all subgraphs whose density is greater than a threshold, and only subgraphs with size (#vertices) smaller than $N_{max} = 5$ is considered. Thus, although mining DCS with respect to graph affinity can be reduced to finding the densest subgraph in one single graph (the "difference" graph), techniques in [1] still cannot be used.

Mining dense subgraphs from multiple networks to find "coherent" dense subgraphs also attracts much research interest [13], [15], [25]. For example, Wu *et al.* [25] studied the problem of finding a subgraph that is dense in one conceptual network and also connected in a physical network.

Another line of related research is contrast graph mining, which aims at discovering subgraphs that manifest drastic differences between graphs. Wang *et al.* [23] and Gionis *et al.* [11] studied how to find the anomalous subgraphs that contrast others in one graph. Ting and Bailey [21] proposed algorithms to find the minimal contrast subgraph, which is a graph pattern appears in one graph but not in the other graph, and all of its proper subgraphs are either shared by or not contained in the two graphs. Yang *et al.* [27] studied the problem of detecting the most frequently changing subgraph in two consecutive snapshots of a time-evolving graph. The major difference between these studies and our work is that, none of these studies adopt subgraph density as the measure for mining contrast subgraphs.

[6] is the work closest to ours in literature. In [6], Cadena *et al.* investigated how to extract the subgraph whose total

Notation	Description
$G = \langle V, E, A \rangle$	An undirected and weighted graph, where each edge $(u,v) \in E$ is associated with a positive weight $A(u,v)$
$G(S) = \langle V, E(S), A(S) \rangle$	The induced subgraph of S in graph G
$W(S)$	The total degree of S in graph G. $W(S) = \sum_{(u,v)\in E(S)} A(u,v)$
$\triangle^n$	A simplex. $\triangle^n = \{\mathbf{x} \mid \sum_{i=1}^n x_i = 1, x_i \geq 0\}$
$\mathbf{x} \in \triangle^n$	An embedding of a subgraph, x_u denotes the participation of u in this subgraph
$S_\mathbf{x}$	Support set of $\mathbf{x}$. $S_\mathbf{x} = \{u \mid x_u > 0\}$
$G_1 = \langle V, E_1, A_1 \rangle$, $G_2 = \langle V, E_2, A_2 \rangle$	Inputs of our Density Contrast Subgraph problem
$G_D = \langle V, E_D, D \rangle$	The difference graph between G_2 and G_1, where $D = A_2 - A_1$ and $E_D = \{(u,v) \mid D(u,v) \neq 0\}$
$G_{D^+} = \langle V, E_{D^+}, D^+ \rangle$	The "positive" part of G_D, where $D^+(i,j) = \max\{D(i,j), 0\}$ and $E_{D^+} = \{(u,v) \mid D(u,v) > 0\}$
$N_D(i)$	The set of i's neighbors in G_D. $N_D(i) = \{j \mid D(i,j) \neq 0\}$
$W_D(i; G_D(S))$	The degree of vertex i in the induced subgraph $G_D(S)$. $W_D(i; G_D(S)) = \sum_{j \in N_D(i) \cap S} A(i,j)$

TABLE I. FREQUENTLY USED NOTATIONS.

edge weight deviates from its expected total edge weight the most. The total edge weight is related to the density measures adopted in our work, average degree and graph affinity, since the total edge weight of a subgraph is the numerator of this subgraph's average degree and edge density, which is often regarded as the discrete version of graph affinity [16], [19], [24]. However, these three measures are still quite different from each other. Thus, properties of the problem in [6] and our problems are very different.

III. PRELIMINARIES

In this section, we introduce several essential concepts in our discussion and formulate the Density Contrast Subgraph problem. For readers' convenience, Table I lists the frequently used notations.

A. Measures of Graph Density

An undirected and weighted graph is represented by $G = \langle V, E, A \rangle$, where V is a set of vertices, E is a set of edges and A is an affinity matrix. Since G is undirected, if $(u,v) \in E$ then $(v,u) \in E$. Denote by $n = |V|$ the number of vertices and $m = |E|$ the number of edges. A is an $n \times n$ symmetric matrix. The entry $A(u,v) > 0$ denotes the weight of the edge (u,v), and $A(u,v) = 0$ if $(u,v) \notin E$. Given an undirected graph $G = \langle V, E, A \rangle$ and a subset of vertices S, the induced subgraph of S is denoted by $G(S) = \langle S, E(S), A(S) \rangle$, where $E(S) = \{(u,v) \mid (u,v) \in E \wedge u \in S \wedge v \in S\}$, and $A(S)$ is a submatrix of A so that only the row and columns of vertices in S are present.

Average degree is a widely investigated graph density measure. Given an undirected graph $G = \langle V, E, A \rangle$ and a set of vertices S, the total degree of the induced subgraph $G(S)$ is $W(S) = \sum_{(u,v)\in E(S)} A(u,v)$. The **average degree** of the induced subgraph $G(S)$ is defined by

$$\rho(S) = \frac{W(S)}{|S|} = \frac{1}{|S|} \sum_{u \in S} \sum_{(u,v)\in E(S)} A(u,v) = \frac{1}{|S|} \sum_{u \in S} W(u; G(S)) \quad (1)$$

where $W(u; G(S)) = \sum_{(u,v)\in E(S)} w(u,v)$ is u's degree in $G(S)$.

Graph affinity is another popularly adopted graph density measure. In graph affinity, a subgraph is represented by a **subgraph embedding** in a standard simplex $\triangle^n = \{\mathbf{x} \mid \sum_{i=1}^n x_i = 1, x_i \geq 0\}$. For a subgraph embedding $\mathbf{x} = [x_1, x_2, ..., x_n]$, the entry x_u indicates the participation importance of vertex u in the subgraph. Denote by $S_\mathbf{x} = \{u \mid x_u > 0\}$ the **support set** of $\mathbf{x}$. The **graph affinity** density of a subgraph embedding $\mathbf{x} \in \triangle^n$ is defined by

$$f(\mathbf{x}) = \mathbf{x}^\top A \mathbf{x} = \sum_{i=1}^n \sum_{j=1}^n x_i x_j A(i,j) = \sum_{(u,v)\in E_S} x_u x_v A(u,v) \quad (2)$$

In traditional dense subgraph mining problems, when the density measure is average degree, the densest subgraph is often large in size [1], while if the density measure is graph affinity, the support set of the densest subgraph embedding is normally small [24].

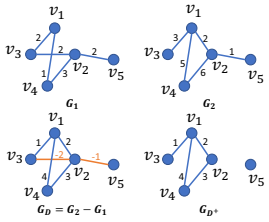


Fig. 1. An Example of the Difference Graph

B. Mining Density Contrast Subgraph

Given two undirected graphs $G_1 = \langle V, E_1, A_1 \rangle$ and $G_2 = \langle V, E_2, A_2 \rangle$, we want to find a subgraph such that its density in G_1 minus its density in G_1 is a large value. Similar to traditional dense subgraph mining, we are more interested in the subgraph whose density difference is the greatest among all subgraphs. Thus, the **Density Contrast Subgraph** (**DCS**) problem can be formulated as an optimization problem. Specifically, if the density measure is the average degree, the optimization problem is

$$\max_{S \subseteq V} \rho_2(S) - \rho_1(S) = \frac{W_2(S)}{|S|} - \frac{W_1(S)}{|S|} \quad (3)$$

where $W_1(S)$ and $W_2(S)$ are total degrees of S in G_1 and G_2, respectively. We call Eq. 3 the problem of **Density Contrast Subgraphs with respect to Average Degree** (**DCSAD**).

If we adopt graph affinity as the density measure, the optimization problem then becomes

$$\max_{\mathbf{x} \in \triangle^n} f_2(\mathbf{x}) - f_1(\mathbf{x}) = \mathbf{x}^\top A_2 \mathbf{x} - \mathbf{x}^\top A_1 \mathbf{x} \quad (4)$$

We call Eq. 4 the problem of **Density Contrast Subgraphs with respect to Graph Affinity** (**DCSGA**). Note that, similar to maximizing graph affinity in G, which is often used to maximize the edge density ($\frac{W(S)}{|S|^2}$) [24], we can also solve (**DCSGA**) for finding a subgraph whose edge density gap in G_1 and G_2 ($\frac{W_D(S)}{|S|^2}$) is maximized. To convert the solution $\mathbf{x}$ to a set of vertices S, we just set $S = S_\mathbf{x}$.

It is easy to find that to find the subgraph such that the absolute value of its density difference is maximized, besides solving Eq. 3 or Eq. 4, we also solve $\max_{S \subseteq V} \rho_1(S) - \rho_2(S)$ or $\max_{\mathbf{x} \in \triangle^n} f_1(\mathbf{x}) - f_2(\mathbf{x})$.

A nice property that both Eq. 3 and Eq. 4 have is that the objective equals the density of S's induced subgraph (or $\mathbf{x}$ if using graph affinity as density) in a "difference graph" between G_2 and G_1. Given $G_1 = \langle V, E_1, A_1 \rangle$ and $G_2 = \langle V, E_2, A_2 \rangle$, the **difference graph** $G_D = \langle V, E_D, D \rangle$ is the graph associated with the affinity matrix $D = A_2 - A_1$. Thus, $E_D = \{(u,v) \mid D(u,v) \neq 0\}$. We also define the graph that contains only edges with positive weights as $G_{D^+} = \langle V, E_{D^+}, D^+ \rangle$, where $E_{D^+} = \{(u,v) \mid D(u,v) > 0\}$. Fig. 1 gives an example of G_1, G_2, G_D and G_{D^+}. It is easy to verify that Eq. 3 is equivalent to

$$\max_{S \subseteq V} \rho_D(S) = \frac{W_D(S)}{|S|} \tag{5}$$

where $W_D(S)$ is the total degree of S in G_D. Also, Eq. 4 is equivalent to

$$\max_{\mathbf{x} \in \triangle^n} f_D(\mathbf{x}) = \mathbf{x}^\top D \mathbf{x} \tag{6}$$

The major difference between finding dense subgraphs in a difference graph G_D and the traditional dense subgraph detection problems is that there are negative edge weights in a difference graph. In Sections IV and V we will analyze how negative edge weights affect properties and algorithms of dense subgraph mining problems.

Also, from Eq. 5 and Eq. 6 we can see that the optimal value is positive if and only if the matrix D has at least one positive entry, that is, the difference graph has at least one edge with potisive weight. If D does not have positive entries, the optimal values to Eq. 5 and Eq. 6 are both 0, the optimal S to Eq. 5 contains one single vertex, and the optimal $\mathbf{x}$ to Eq. 6 has only one entry that equals 1 and all other entries 0.

C. Why not Ratio of Difference?

Instead of the absolute value of density difference, why don't we consider the ratio of density difference, i.e. $\frac{\rho_2(S)}{\rho_1(S)}$ or $\frac{f_2(\mathbf{x})}{f_1(\mathbf{x})}$, as the objective? The reason is that the ratio of density difference sometimes is not well-defined or has trivial solutions. Consider a single vertex u as a subgraph. Its densities in G_1 and G_2 are both 0 so the ratio of density difference is $\frac{0}{0}$. Also, in Fig. 1, the edge (v_1, v_2) has density ratio $+\infty$ since it only appears in G_2 but not G_1.

D. Generalization of the Difference Graph

In Sections IV and V we will introduce our DCS finding algorithms that can take any weighted graphs as input, where the weight of an edge can be positive or negative. Thus, the definition of the difference graph of $G_1 = \langle V, E_1, A_1 \rangle$ and $G_2 = \langle V, E_2, A_2 \rangle$ is not restricted to the graph whose affinity matrix is $A_2 - A_1$. For example, we can set the difference graph as $G_D = \langle V, E_D, D = A_2 - \alpha A_1 \rangle$ and maximizing $\rho_D(S)$ (or $f_D(\mathbf{x})$) is equivalent to finding S (or $\mathbf{x}$) such that $\rho_2(S) \geq \alpha \rho_1(S)$ (or $f_2(\mathbf{x}) \geq \alpha f_1(\mathbf{x})$), and $\rho_2(S) - \alpha \rho_2(S)$ (or $f_2(\mathbf{x}) - \alpha f_1(\mathbf{x})$) is maximized. This is similar to the optimal α-quasi-clique problem [22]. Also, when there is one edge in the difference graph whose weight is much heavier than all the other edges, such an edge itself is very possible to be the optimal subgraph. To avoid this, for edges with too heavy weights in G_D, we can adjust their weights such that they are not too much heavier than other edge weights in G_D. Then the DCS extracted usually will become larger in size.

IV. DCS with respect to Average Degree

In this section, we first explore some key properties of the **DCSAD** problem. Then, we devise an efficient greedy algorithm with a data-dependent ratio.

A. Complexity and Approximability

Like traditional dense subgraph discovery problem, the **DCSAD** prefers "connected" subgraphs, of course, in the difference graph G_D.

Property 1. *Let G_D be the difference graph of G_1 and G_2. For any $S \subseteq V$, if $G_D(S)$ is not a connected subgraph, then there exists a set $S' \subseteq S$ such that $G_D(S')$ is connected and the density difference $\rho_D(S') \geq \rho_D(S)$.*

Traditional dense subgraph discovery with respect to average degree can be solved in polynomial time [12], and has an efficient 2-approximation algorithm [7]. Unfortunately, our problem does not have the same computational properties.

Theorem 1. *The **DCSAD** (Eq. 5) problem is NP-hard.*

Proof: We prove this by a reduction from the maximum clique problem, which is known NP-hard. Given an instance of the maximum clique problem, which is an undirected and unweighted graph $G = \langle V, E \rangle$, we build two graphs G_1 and G_2 as the input of the **DCSAD** problem. Let $E_1 = \{(u,v) \mid (u,v) \in V \times V \wedge u \neq v \wedge (u,v) \notin E\}$. We set $G_1 = \langle V, E_1, A_1 \rangle$ and for every edge $(u,v) \in E_1$, we set the weight $A_1(u,v) = |E| + 1$. Clearly building G_1 and G_2 can be done in polynomial time w.r.t. the size of G. We set $G_2 = \langle V, E_2, A_2 \rangle$ where $E_2 = E$. For every edge $(u,v) \in E_2$, we set the weight $A_2(u,v) = 1$.

It is obvious that for any $S \subseteq V$, the density difference $\frac{W_2(S)}{|S|} - \frac{W_1(S)}{|S|} < 0$ if $G_1(S)$, the induced subgraph of S in G_1, contains at least one edge in E_1. Thus, the optimal S must satisfy that $G_1(S)$ does not contain any edges in E_1. Due to the definition of E_1, $G_2(S)$, the induced subgraph of S in G_2 is a clique. So the optimal density difference is $|S| - 1$ where S is the maximum clique in G_2. Because G_2 and G actually are the same, the optimal density difference of G_2 and G_1 is at least $k-1$ if and only if G contains a clique with at least k vertices. Due to the NP-hardness of the maximum clique problem, the **DCSAD** problem is also NP-hard. ■

The **DCSAD** problem is not only NP-hard but also hard to approximate under reasonable complexity assumptions.

Corollary 1. *Assuming **P≠NP**, the **DCSAD** problem (Eq. 5) cannot be approximated within $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$.*

B. Greedy Algorithms

Although **DCSAD** cannot be approximated within $O(n^{1-\varepsilon})$, an $O(n)$ approximation is easy to achieve. We have two cases,

1) If there are no edges with positive weights in G_D, apparently any S that only contains a single vertex is an optimal solution to the **DCSAD** problem, and the optimal density difference is 0.
2) If G_D has at least one edge with positive weight, $S = \{u,v\}$ is an $O(n)$ approximation solution, where $(u,v) =$

Algorithm: Greedy

Input: $G = \langle V, E, A \rangle$
Output: S

1: $S \leftarrow V$, $S_1 \leftarrow V$
2: **while** $|S_1| \geq 1$ **do**
3: **if** $\frac{W(S_1)}{|S_1|} > \frac{W(S)}{|S|}$ **then**
4: $S \leftarrow S_1$
5: $i \leftarrow \arg\min_{j \in S_1} W(j; G(S_1))$
6: $S_1 \leftarrow S_1 \setminus \{i\}$
7: **return** S

Algorithm 1: Greedy Algorithm.

$\arg\max_{(u,v)\in E_D} D(u,v)$. The reason is as follows. For any $S' \subseteq V$, $\rho_D(S')$ must be no greater than the density of an n-clique where every edge's weight is $D(u,v)$. Such an n-clique has density $(n-1)D(u,v)$. Note that $\rho_D(S) = D(u,v)$. Thus, $\frac{\rho_D(S)}{\max_{S' \subseteq V} \rho_D(S')} \geq n-1 = O(n)$.

Utilizing the above results, and inspired by the greedy approximation algorithm (shown in Algorithm 1) for the traditional dense subragph discovery problem [7], we devise an $O(n)$ approximation algorithm, the DCSGreedy algorithm (Algorithm 2), which also has a data-dependent ratio.

The idea of the Algorithm 2 is to generate multiple potentially good solutions and pick the best one. As discussed above, when G_D has positive weighted edges, the edge (u,v) with the maximum weight is a candidate solution since it is $\frac{1}{n-1}$-optimal. The Greedy algorithm may also generate a good solution, although for the **DCSAD** problem its approximation ratio is no better than $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$. Thus, we run Algorithm 1 on G_D to generate S_1. We also run Algorithm 1 on G_{D^+} to get S_2, because not only S_2 may be a better solution, but also $\rho_{D^+}(S_2)$, the average degree of S_2 in G_{D^+}, helps us derive a data-dependent ratio of Algorithm 2, which will be shown in Theorem 2. In line 9 of Algorithm 2, $CC_D(S)$ is the set of connected components of $G_D(S)$, where a connected component is represented by a set of vertices. Line 9 is for refining the solution S obtained at line 7 when $G_D(S)$ is not connected, since **DCSAD** prefers "connected" subgraphs.

Theorem 2. *The S returned by Algorithm 2 has a data-dependent ratio of $\frac{2\rho_{D^+}(S_2)}{\rho_D(S)}$, where S_2 is the set in line 6 of Algorithm 2.*

We analyze the time complexity of Algorithm 2. Suppose $|V| = n$, $|E_1| = m_1$ and $|E_2| = m_2$. The difference graph G_D can be built in $O((m_1+m_2)\log n + n)$ time, if we sort the adjacent lists of G_1 and G_2 first, and use then a merge sort to build u's adjacent list in G_D for each $u \in V$. Finding the maximum edge weight can be done in $O(m_1+m_2)$ time since G_D has at most m_1+m_2 edges. Running the Greedy algorithm on a graph $G = \langle V, E, A \rangle$ can be finished in $O((|E|+|V|)\log|V|)$ time, if we adopt a segment tree [3] to store the current degrees of vertices in S_1. Thus, **Greedy**(G_D) and **Greedy**(G_{D^+}) together can be done in $O((m_1+m_2+n)\log n)$ time. Lines 8 and 9 obviously can be done in $O(m_1+m_2+n)$ time. Thus, in total Algorithm 2 can be efficiently implemented in $O((m_1+m_2+n)\log n)$ time.

V. DCS WITH RESPECT TO GRAPH AFFINITY

In this section, we first explore several properties of the **DCSGA** problem. Then, we devise a Coordinate-Descent algorithm which is guaranteed to converge to a KKT point. We also propose a refinement step to further improve a KKT point solution. Since **DCSGA** is non-concave, normally we need multiple initializations to find a good solution. To reduce the number of initializations, we utilize a smart initialization heuristic. Combining the Coordinate-Descent algorithm, the refinement step and the smart initialization heuristic together, we have our NewSEA algorithm for the **DCSGA** problem.

Algorithm: DCSGreedy

Input: $G_1 = \langle V, E_1, A_1 \rangle$, $G_2 = \langle V, E_2, A_2 \rangle$
Output: S, and a data-dependent ratio β

1: Build the difference graph $G_D = \langle V, E_D \rangle$
2: **if** G_D does not have edges with positive weights **then**
3: Randomly pick a vertex v
4: **return** $S \leftarrow \{v\}$
5: $(u,v) \leftarrow \arg\max_{(u,v)\in E_D} D(u,v)$
6: $S \leftarrow \{u,v\}$, $S_1 \leftarrow$ **Greedy**(G_D) , $S_2 \leftarrow$ **Greedy**(G_{D^+})
7: $S \leftarrow \arg\max_{S' \in \{S, S_1, S_2\}} \frac{W_D(S')}{|S'|}$
8: **if** $G_D(S)$ is not connected **then**
9: $S \leftarrow \arg\max_{S' \in CC_D(S)} \frac{W_D(S')}{|S'|}$
10: **return** S and $\beta \leftarrow \frac{2\rho_{D^+}(S_2)}{\rho_D(S)}$

Algorithm 2: DCSGreedy algorithm for solving **DCSAD**.

A. Properties

We first show that, like the **DCSAD** problem, **DCSGA** also prefers connected subgraphs in the difference graph G_D.

Property 2. *Let $G_D = \langle V, E_D, D \rangle$ be the difference graph of G_1 and G_2. For any $\boldsymbol{x} \in \triangle^n$ such that $f_D(\boldsymbol{x}) = \boldsymbol{x}^\top D\boldsymbol{x} \geq 0$, if $G_D(S_{\boldsymbol{x}})$ is not connected, where $S_{\boldsymbol{x}}$ is the support set of $\boldsymbol{x}$, then there exists $\boldsymbol{x}'$ whose support set $S_{\boldsymbol{x}'} \subseteq S_{\boldsymbol{x}}$, and $G_D(S_{\boldsymbol{x}'})$ is connected, and $f_D(\boldsymbol{x}') \geq f_D(\boldsymbol{x})$.*

The **DCSGA** is a standard Quadratic Programming (QP) problem, which in general is NP-hard. We prove that **DCSGA** is NP-hard.

Theorem 3. *The **DCSGA** (Eq. 6) problem is NP-hard.*

Proof: Consider an undirected and unweighted graph G whose adjacency matrix is A, where the entries of A are either 0 or 1. It is known that maximizing $\mathbf{x}^\top A\mathbf{x}$ s.t. $\mathbf{x} \in \triangle^n$ is NP-hard, because the optimum is $1 - \frac{1}{k}$, where k is the size of the maximum clique of G [18]. Given an arbitrary undirected and unweighted graph G, we create a corresponding instance of the **DCSGA** problem by building G_1 as a graph without any edges and setting $G_2 = G$. Clearly for any $\mathbf{x} \in \triangle^n$, we have $\mathbf{x}^\top A\mathbf{x} = \mathbf{x}^\top D\mathbf{x}$, where D is the affinity matrix of the difference graph between G_2 and G_1. Thus, this simple reduction proves that the **DCSGA** problem is also NP-hard. ■

B. The SEACD Algorithm

Since **DCSGA** is NP-hard and is a QP, we employ local search algorithms to find good solutions. Because the density difference $\mathbf{x}^\top D\mathbf{x}$ is normally non-concave, we seek for $\mathbf{x}$ that satisfies the Karush-Kuhn-Tucker (KKT) conditions [5], which are necessary conditions of local maxima points. It is easy to derive that, if $\mathbf{x}$ is a KKT point of the **DCSGA** problem, it

should satisfy

$$\nabla_u f_D(\mathbf{x}) = 2(D\mathbf{x})_u \begin{cases} = \lambda & x_u > 0 \\ \leq \lambda & x_u = 0 \end{cases} \quad \forall u \in V \tag{7}$$

where $\nabla_u f_D(\mathbf{x}^*)$ is the partial derivative with respect to x_u, and $(D\mathbf{x}^*)_u$ is the u-th entry of the vector $D\mathbf{x}^*$. Since $x \in \triangle^n$, when Eq. 7 holds, we have $f_D(\mathbf{x}) = \sum_{u \in V} x_u * (D\mathbf{x})_u = \frac{\lambda}{2}$.

The condition in Eq. 7 is also equivalent to

$$\max_{k:x_k<1} \nabla_k f_D(\mathbf{x}) \leq \min_{k:x_k>0} \nabla_k f_D(\mathbf{x}) \tag{8}$$

The Shrink-and-Expansion (SEA[2]) algorithm in [16] utilizes a replicator dynamic to solve the problem that maximizes $\mathbf{x}^\top A\mathbf{x}$ s.t. $\mathbf{x} \in \triangle^n$, where A is an affinity matrix of an undirected graph. Although D in Eq. 6 can also be regarded as an affinity matrix, unfortunately the SEA algorithm cannot be directly applied to our problem. This is because the replicator dynamic can only deal with non-negative matrices, while in our problem the matrix D may have negative entries. Thus, we devise a 2-Coordinate Descent algorithm to solve Eq. 6.

In every iteration of the 2-Coordinate Descent algorithm, we only pick two variables x_i and x_j, and fix the rest $n-2$ variables. We adjust the values of x_i and x_j to increase the objective $f_D(\mathbf{x})$ without violating the simplex constraint. Suppose $x_i + x_j = C$, and let $b_i = \sum_{a \in N_D(i), a \neq j} D(a,i)x_a$, $b_j = \sum_{a \in N_D(j), a \neq i} D(a,j)x_a$, where $N_D(i)$ is the set of i's neighbors in G_D. We adjust x_i and x_j by solving a simple optimization problem involving only one variable, since x_j should always equal $C - x_i$ when the rest $n-2$ variables are fixed. Specifically, the optimization problem is

$$\begin{aligned} \max \quad & \frac{1}{2} f_D(\mathbf{x}) = g(x_i) = b_i x_i + b_j(C - x_i) + D(i,j)x_i(C - x_i) + Cnst \\ \text{s.t.} \quad & 0 \leq x_i \leq C \end{aligned} \tag{9}$$

where $Cnst$ is a constant independent from x_i and x_j.

Eq. 9 can be solved analytically. There are two cases,

1) $D(i,j) = 0$, which means i and j are not adjacent in the difference graph G_D. Then $g(x_i) = (b_i - b_j)x_i + b_j C + Cnst$. Obviously we should set $x_i = C$ if $b_i > b_j$, and set $x_i = 0$ if $b_i < b_j$. We do not adjust b_i or b_j if $b_i = b_j$.
2) $D(i,j) \neq 0$, which means i and j are adjacent in G_D. We have $g(x_i) = -D(i,j)x_i^2 + Bx_i + b_j C + Cnst$ where $B = D(i,j)C + b_i - b_j$. Let $r = \frac{B}{2D(i,j)}$. If $0 \leq r \leq C$, we set $x_i = \arg\max_{x \in \{0,r,c\}} g(x)$. If $r < 0$ or $r > C$, we set $x_i = \arg\max_{x \in \{0,C\}} g(x)$.

To pick x_i and x_j for an iteration, we exploit the partial derivatives. We pick $i = \arg\max_{k:x_k<1} \nabla_k f_D(\mathbf{x})$ and $j = \arg\min_{k:x_k>0} \nabla_k f_D(\mathbf{x})$. If $\nabla_i f_D(\mathbf{x}) \leq \nabla_j f_D(\mathbf{x})$, which means we reach a KKT point, the algorithm stops.

The 2-Coordinate Descent algorithm is guaranteed to converge to a stationary point, which is equivalent to a KKT point because the constraint $\mathbf{x} \in \triangle^n$ in Eq. 6 is linear [5].

Picking x_i and x_j at the beginning of every iteration clearly can be done in $O(n)$ time. But $O(n)$ may still be too costly for large graphs. Thus, to further improve the efficiency of our algorithm, we adopt the strategy of the Shrink-and-Expansion algorithm. We define a **local KKT point** on $S \subseteq V$ as a point $\mathbf{x} \in \triangle^n$ that satisfies the following conditions,

$$\begin{aligned} & x_u = 0 \quad if \quad u \notin S \\ & \nabla_u f_D(\mathbf{x}) = 2(D\mathbf{x})_u \begin{cases} = \lambda & x_u > 0 \\ \leq \lambda & x_u = 0 \end{cases} \quad \forall u \in S \\ & \lambda = 2 f_D(\mathbf{x}) \end{aligned} \tag{10}$$

where the major difference from Eq. 7 is that only the vertices in $S \subseteq V$ are considered. It is also equivalent to

$$\max_{k \in S: x_k<1} \nabla_k f_D(\mathbf{x}) \leq \min_{k \in S: x_k>0} \nabla_k f_D(\mathbf{x}) \tag{11}$$

The 2-Coordinate Descent algorithm is guaranteed to converge to a local KKT point on S, when we keep $x_u = 0$ for every $u \notin S$, and x_u is involved in iterations only when $u \in S$.

Algorithm 3 shows our method. We start with an initial embedding $\mathbf{x} \in \triangle^n$. Line 3 is the Shrink stage, since after calling the 2-coordinate descent algorithm, the support set of $\mathbf{x}$ may shrink due to some originally positive x_i is set to 0. Line 6 is the start of the expansion stage. We first enlarge S by adding to S the vertices whose partial derivatives are greater than $\lambda = 2f_D(\mathbf{x})$, and then do exactly the same expansion operation of the original SEA algorithm [16] (see Appendix). If Z in Line 6 is empty, the current $\mathbf{x}$ is already a KKT point satisfying conditions in Eq. 7 and the SEA iterations stop.

Algorithm: SEACD

Input: G_D, an initial embedding $\mathbf{x} \in \triangle^n$
Output: $\mathbf{x}$

1 $S \leftarrow S_{\mathbf{x}}$
2 **while true do**
3 Use the 2-Coordinate Descent algorithm and take $\mathbf{x}$ as the initial value to find a local KKT point $\mathbf{x}^{new}$ on S
4 $\mathbf{x} \leftarrow \mathbf{x}^{new}$
5 $S \leftarrow \{v \mid x_v > 0\}$, $\lambda \leftarrow 2f_D(\mathbf{x})$
6 $Z \leftarrow \{i \mid \nabla_i f_D(\mathbf{x}) > \lambda, i \in V\}$
7 **if** $Z = \emptyset$ **then**
8 **break**
9 Do the SEA Expansion operation on $S \cup Z$ to adjust $\mathbf{x}$
10 $S \leftarrow S_{\mathbf{x}}$
11 **return** $\mathbf{x}$

Algorithm 3: Coordinate Descent SEA Algorithm.

Like the original SEA algorithm [16], The SEACD algorithm converges to a KKT point.

Theorem 4. *The SEACD algorithm (Algorithm 3) is guaranteed to converge to a KKT point.*

We analyze the computational cost of Algorithm 3. It is worth noting that to efficiently run Algorithm 3, the initial embedding $\mathbf{x}$ should have a small support set such that during the execution of Algorithm 3, S and Z in the while loop are normally small sets. In the Shrink stage, for every iteration, we need $O(|S|)$ time to pick x_i and x_j, $O(1)$ time to adjust x_i and x_j, and $O(|N_D(i)| + |N_D(j)|)$ time to update the partial derivatives of the vertices affected by adjusting x_i or x_j. S is usually a small set and $|N_D(i)| + |N_D(j)|$ is often a small number since real-world graphs are normally sparse. Thus, the cost of each iteration of the shrink stage is low. In Line 6 of the Expansion stage, we only need to check the partial

[2]The details of SEA algorithm are illustrated in Appendix.

derivatives of the vertices that have at least one neighbor in S, since the partial derivatives of all other vertices are 0. Thus, the cost of Line 6 is $\sum_{v\in S}|N_D(v)|$. Line 9 is the same as the Expansion operation of the SEA algorithm, whose cost is $O(\sum_{v\in S\cup Z}|N_D(v)|)$ [16]. Since both S and Z are normally small sets, the cost of one SEA iteration (one Shrink stage + one Expansion stage) is low.

C. Refining a KKT Point Solution

After a KKT point solution is reached, we may further improve the solution. We call a clique in G_D as a **positive clique** if all its edge weights are positive, and $\mathbf{x}\in\triangle^n$ a **positive clique solution** if $G_D(S_\mathbf{x})$ is a positive clique. Utilizing the 2-Coordinate Decent algorithm, we give a construction that refines a KKT point $\mathbf{x}$ such that $G(S_\mathbf{x})$ is not a positive clique to a better solution.

Theorem 5. *For any KKT point $\boldsymbol{x}$, let $S_\boldsymbol{x}=\{v\mid x_v>0\}$. If $G_D(S_\boldsymbol{x})$ is not a positive clique, we can find a $\mathbf{y}$ such that $G_D(S_\mathbf{y})$ is a positive clique and $f_D(\mathbf{y})\geq f_D(\boldsymbol{x})$, where $S_\mathbf{y}=\{v\mid y_v>0\}$ and $S_\mathbf{y}\subseteq S_\boldsymbol{x}$.*

Proof: Suppose $\mathbf{x}$ is a KKT point and $G_D(S_\mathbf{x})$ is not a positive clique. We pick x_i and x_j from $S_\mathbf{x}$ such that $D(i,j)\leq 0$.

If $D(i,j)=0$, since $\nabla_i f_D(\mathbf{x})=\nabla_j f_D(\mathbf{x})$, we have $(D\mathbf{x})_i=(D\mathbf{x})_j$ which means $\sum_{a\in N_D(i)}D(a,i)x_a=\sum_{a\in N_D(j)}D(a,j)x_a$. Thus, $b_i=b_j$ in Eq. 9 and we have $g(x_i)=b_jC+Cnst$. Note that b_jC is independent of x_i and x_j, as long as $x_i+x_j=C$. We set $x_i=C$ and $x_j=0$ to remove vertex j from the current subgraph, and the objective $f_D(\mathbf{x})$ remains the same.

If $D(i,j)<0$, we solve the optimization problem in Eq. 9. Apparently $g(x_i)$ is a convex function with respect to x_i because $-D(i,j)>0$. To maximize the objective $g(x_i)$, we should set $x_i^{new}=\arg\max_{x\in\{0,C\}}g(x)$. Thus, after solving Eq. 9, either x_i or x_j becomes 0 and the objective $f_D(\mathbf{x})$ is improved.

Thus, if $G_D(S_\mathbf{x})$ is not a positive clique, we can always remove one vertex i (by setting $x_i=0$) that is incident to an edge with negative weight or is not adjacent to all other vertices in $S_\mathbf{x}$, and keep the objective non-decreasing. Suppose after removing this vertex we get $\mathbf{y}$. We use the 2-coordinate descent algorithm to adjust $\mathbf{y}$ to a local KKT point on $S_\mathbf{y}$, and obviously the objective $f_D(\mathbf{y})$ is not decreased. If $G_D(S_\mathbf{y})$ is still not a positive clique, we repeat the above procedure of removing one vertex and adjusting to a local KKT point. During this process, the support set shrinks if the current solution is not a positive clique solution. Since the support set cannot shrink forever (it should has at least 1 vertex), finally we will reach a positive clique solution $\mathbf{y}$ such that $S_\mathbf{y}\subseteq S_\mathbf{x}$. Moreover, during the process of reaching $\mathbf{y}$, the objective is non-decreasing. Thus, we have $f_D(\mathbf{y})\geq f_D(\mathbf{x})$. ■

Since an optimal $\mathbf{x}$ must be a KKT point, Theorem 5 implies that there exist a solution $\mathbf{y}\in\triangle^n$ such that $\mathbf{y}$ is an optimal solution to Eq. 6, and $G_D(S_\mathbf{y})$ is a positive clique in G_D. Note that a positive clique in G_D is a clique in G_{D^+}. Thus, we can run Algorithm 3 directly on G_{D^+} instead of G_D to get a solution $\mathbf{x}$. If $G_{D^+}(S_\mathbf{x})$ is not a clique in G_{D^+}, we use the construction in the proof of Theorem 5 to find a new solution $\mathbf{y}$ whose $G_{D^+}(S_\mathbf{y})$ is a clique. Algorithm 4 shows the construction, where we do not consider the case when $D^+(i,j)<0$ since D^+ only has non-negative entries.

Algorithm: Refinement

Input: G_{D^+}, a KKT point $\mathbf{x}$
Output: $\mathbf{y}$

1: $\mathbf{y}\leftarrow\mathbf{x}$
2: **while** $G_{D^+}(S_\mathbf{y})$ is not a clique **do**
3: Pick u and v such that (u,v) is not an edge in G_{D^+}
4: $y_u\leftarrow y_u+y_v$, $y_v\leftarrow 0$
5: Use the 2-Coordinate Descent algorithm and take $\mathbf{y}$ as the initial value to find a local KKT point $\mathbf{y}^{new}$ on $S_\mathbf{y}$
6: $\mathbf{y}\leftarrow\mathbf{y}^{new}$
7: **return** $\mathbf{y}$

Algorithm 4: Refining a KKT point.

Since the edges with negative weights can be ignored, it seems we can run the original SEA algorithm [16] on G_{D^+} directly to find DCS. However, SEA in [16] is not guaranteed to return a positive clique solution $\mathbf{x}$. If $G_{D^+}(S_\mathbf{x})$ is not a clique, $G_D(S_\mathbf{x})$ may have some edges with negative weights and $\mathbf{x}$ is definitely not an optimal solution. This is because according to the proof of Theorem 5, if $D(i,j)<0$ where $x_i>0$ and $x_j>0$, we can solve the optimization problem in Eq. 9 over x_i and x_j to further improve the objective. Therefore, we still need our refinement step (Algorithm 4).

Always returning a positive clique solution as the DCS is one advantage of adopting graph affinity as the density measure, since the returned DCS has very good interpretability. From G_1 to G_2, for every pair of vertices in the DCS, their connection is enhanced.

Please note that, although there exists an optimal solution $\mathbf{x}$ such that $G_D(S_\mathbf{x})$ is a positive clique, it does not mean a maximum clique finding algorithms like [20] can be applied to solve the **DCSGA** problem. The major reason is that G_D in the **DCSGA** problem is a weighted graph while maximum clique finding algorithms deal with unweighted graphs.

Advantages of the Coordinate-Descent SEA With the help of the refinement step (Algorithm 4), the original SEA algorithm [16] works for the DCSGA problem. However, our Coordinate-Descent SEA algorithm has some advantages over the original SEA algorithm. The correctness of the Expansion operation (see Appendix) depends on that a local KKT point is reached in the Shrink stage. Thus, when implementing the Shrink stage, the correct condition of convergence should be $\max_{k\in S:x_k<1}\nabla_k f_D(\mathbf{x})-\min_{k\in S:x_k>0}\nabla_k f_D(\mathbf{x})\leq\varepsilon$, where S is the set of vertices on which we try to find a local KKT point, and ε is the parameter of precision. However, the original SEA [16] adopts $f_D(\mathbf{x})-f_D(\mathbf{x}^{old})\leq\varepsilon$ as the convergence condition, where $\mathbf{x}$ and $\mathbf{x}^{old}$ are the solutions after and before a Shrink iteration by the replicator dynamics. In Section VI, we will show that such a convergence condition may fail to achieve a local KKT point and as a result, the objective $f_D(\mathbf{x})$ is even reduced in the following Expansion stage. Moreover, when the convergence condition of the Shrink stage is correctly set, the replicator dynamics of the original SEA [16] converges much slower than the coordinate-descent method, especially on dense graphs. Since our algorithm can also deal with graph with only positive edge weights, it is also a competitive solution to the traditional graph affinity maximization problem.

D. Smart Initializations of $\mathbf{x}$

One problem remaining unsettled is how to choose the initial embedding $\mathbf{x}$ for running Algorithm 3. Since the **DCSGA** problem is non-concave, we adopt the strategy of multiple initializations, that is, we run Algorithm 3 multiple times with different initial embeddings. The best solution generated in all runs is returned as the final solution. For the interest of efficiency of the SEACD algorithm as illustrated in Section V-B, an initial embedding $\mathbf{x}$ should have a small support set.

One simple way of initialization is to set $\mathbf{x} = \mathbf{e}_u$, where in $\mathbf{e}_u$, only the u-th entry is 1 and all other entries are 0. The original SEA algorithm employs this simple method and it uses every vertex $u \in V$ to set the initial embedding [16]. Thus, in [16], the SEA algorithm is called $n = |V|$ times.

For large graphs, $O(n)$ initializations are clearly very time-consuming. We adopt a smart heuristic to reduce the number of initializations. The major idea is to first find an upper bound μ_u for each $u \in V$, where μ_u is the upper bound of $\mathbf{x}^\top D\mathbf{x}$ for any $\mathbf{x} \in \triangle^n$ such that $x_u > 0$ and $G_{D^+}(S_\mathbf{x})$ is a clique. Then we only use the vertices with big upper bounds to do initializations.

Define the **ego net** of u in G_{D^+} as $G_{D^+}(T_u)$ where T_u is the set containing u and all u's neighbors in G_{D^+}. Let $w_u = \max_{i \in T_u \vee j \in T_u} D^+(i,j)$. Clearly, w_u is an upper bound of the maximum edge weight in u's ego net. Using $O(|E_{D^+}|)$ time, we compute w_u for every $u \in V$.

Theorem 6. *For any $u \in V$, $\boldsymbol{x}^\top D\boldsymbol{x} \leq \frac{(k-1)w_u}{k}$, where $\boldsymbol{x} \in \triangle^n$ and $G_{D^+}(S_\mathbf{x})$ is a k-clique containing u, and w_u is an upper bound of the maximum edge weight in $G_{D^+}(T_u)$, the ego net of u in G_{D^+}.*

Proof: Suppose for $\mathbf{x} \in \triangle^n$, $G_{D^+}(S_\mathbf{x})$ is a k-clique containing u. Thus, $\mathbf{x}^\top D^+\mathbf{x} = \sum_{(i,j)\in E_{D^+}(S_\mathbf{x})} x_i x_j D^+(i,j) = \mathbf{x}^\top D\mathbf{x}$. Since $G_{D^+}(S_\mathbf{x})$ is a k-clique containing u, for any $(i,j) \in E_{D^+}(S_\mathbf{x})$, $D(i,j) \leq w_u$. Thus, $\mathbf{x}^\top D\mathbf{x} \leq w_u \sum_{(i,j)\in E_{D^+}(S_\mathbf{x})} x_i x_j$. When $G_{D^+}(S_\mathbf{x})$ is a k-clique, we have $\sum_{(i,j)\in E_{D^+}(S_\mathbf{x})} x_i x_j \leq \sum_{(i,j)\in E_{D^+}(S_\mathbf{x})} \frac{1}{k}\frac{1}{k} = \frac{k-1}{k}$. Thus, $\mathbf{x}^\top D\mathbf{x} \leq \frac{(k-1)w_u}{k}$. ∎

Based on Theorem 6, assuming k_u is the size of the maximum clique in G_{D^+} that contains u, then $\mathbf{x}^T D^+\mathbf{x}$ is no more than $\frac{(k_u-1)w_u}{k_u}$, where $\mathbf{x} \in \triangle^n$, $x_u > 0$ and $G_{D^+}(S_\mathbf{x})$ is a clique. Although computing k_u for every $u \in V$ is NP-hard, it is easy to find an upper bound of k_u, which is $\tau_u + 1$ where τ_u is the core number of u in G_{D^+} [20]. Thus, we use $\mu_u = \frac{\tau_u w_u}{\tau_u + 1}$ as the upper bound of the affinity of a clique in G_{D^+} that contains u. Note that computing τ_u for every $u \in V$ can be done in $O(|E_{D^+}|)$ time [20].

We sort all vertices in V in the descending order of μ_u. Then we use the new order of vertices to initialize $\mathbf{x}$. Suppose we have tried some vertices and $\mathbf{y}$ is the current best solution. Then all vertices v such that $\mu_v \leq f_D(\mathbf{y})$ will not be used to initialize $\mathbf{x}$. In such a case, normally we only need to do a small number of initializations.

Note that when we use a vertex u to initialize $\mathbf{x}$, it is not guaranteed that after running the SEACD algorithm and the Refinement algorithm a solution $\mathbf{x}$ is returned where $x_u > 0$. It is possible that $x_u = 0$ in the returned solution $\mathbf{x}$. Thus, our method for reducing the number of initializations is not a pruning technique, but a heuristic. In Section VI we show that our smart initialization heuristic is very effective and it never impairs the quality of the final solution $\mathbf{x}$ compared to trying all vertices for initializations in experiments.

Algorithm: NewSEA

Input: G_{D^+}
Output: $\mathbf{y}$

$\mathbf{y} \leftarrow \mathbf{0}$
Compute w_u, τ_u for every $u \in V$
Compute $\mu_u = \frac{\tau_u w_u}{\tau_u + 1}$ for every $u \in V$
Sort V in descending order of μ_u
for $u \in V$ **do**
  if $\mu_u \leq f_D(\mathbf{y})$ **then**
    break
  Set $\mathbf{x}$ such that $\mathbf{x}_u = 1$ and $\mathbf{x}_v = 0$ for all $v \neq u$
  $\mathbf{x} \leftarrow$ **SEACD**$(G_{D^+}, \mathbf{x})$
  $\mathbf{x} \leftarrow$ **Refinement**$(G_{D^+}, \mathbf{x})$
  if $f_D(\mathbf{x}) > f_D(\mathbf{y})$ **then**
    $\mathbf{y} \leftarrow \mathbf{x}$
return $\mathbf{y}$

Algorithm 5: The NewSEA algorithm for solving **DCSGA**.

Combining all results in this section, we propose the NewSEA algorithm shown in Algorithm 5.

VI. Experiments

In this section, we report a series of experiments to verify the effectiveness and efficiency of our algorithms.

A. Algorithms and Datasets in Experiments

For the **DCSAD** problem, we tested our DCSGreedy algorithm, the Greedy algorithm on G_D (denoted by G_D **only**) and the Greedy algorithm on G_{D^+} (denoted by G_{D^+} **only**). For the **DCSGA** problem, we tested our NewSEA algorithm, our SEACD algorithm plus the Refinement step but without our smart initializations heuristic (denoted by **SEACD+Refine**), and the original SEA algorithm [16] plus the Refinement step (denoted by **SEA+Refine**). All **DCSGA** algorithms were run on G_{D^+} directly. The convergence condition of the Shrink stage in NewSEA and SEACD+Refine are all set to $\max_{k\in S: x_k<1} \nabla_k f_D(\mathbf{x}) - \min_{k\in S: x_k>0} \nabla_k f_D(\mathbf{x}) \leq 10^{-2} * \frac{1}{|S|}$, where S is the current set on which we want to reach a local KKT point. This convergence condition very often is too difficult to achieve for the replicator dynamic in the Shrink stage of SEA+Refine, because the replicator dynamic converges too slowly. Thus, for the Shrink stage in SEA+Refine, the convergence condition was set to that the improvement of the objective $f_D(\mathbf{x})$ is less than 10^{-6} after one iteration. As pointed out in Section V-C, this convergence condition actually is not enough for achieving a local KKT point. Thus, in our experiments, the Shrink stage of SEA+Refine sometimes could not converge to a local KKT point and as a result, in the following Expansion stage error occurred, the objective $f_D(\mathbf{x})$ was even reduced after expansion.

We list the statistics of all data sets used in our experiments in Table II. The setting of "Weighted" represents that we built G_D as $G_2 - G_1$ directly. In some graphs there are several edges with weights significantly greater than the weights of other edges, they make the DCS with respect to graph affinity a very small subgraph, sometimes even a single edge. Thus, to limit

the influence of these small number of edges with too heavy weights, we also tried the Discrete setting, where we set edge weights in G_D discrete values such that the maximum weight is not too much greater than the other edge weights. Details of how to set edge weight in the Discrete Setting and "G_D Type" are illustrated in each task in the rest of this section.

B. Finding Emerging and Disappearing Co-author Groups

We applied DCS to find emerging/disappearing co-author groups from co-author networks. We adopted the DBLP dataset (https://static.aminer.org/lab-datasets/citation/dblp.v8.tgz) and extracted all papers published in the top conferences according to the CS Ranking website (http://csrankings.org/). Based on these papers, we built two co-author graphs. The first graph $G_1 = \langle V, E_1, A_1 \rangle$ contains the co-authorships before the year of 2010, and the second one $G_2 = \langle V, E_2, A_2 \rangle$ contains the co-authorships from 2010 to 2016. For an edge linking two authors in a co-author graph, the weight is the number of papers written by these two authors together.

To build the difference graph $G_D = \langle V, E_D, D \rangle$, we tried two settings, the Weighted setting and the Discrete setting. In the Weighted setting, we set $D(u,v) = A_2(u,v) - A_1(u,v)$, which is the standard setting of the DCS problem. In the Discrete setting, the entries of D are set to discrete values. Specifically, if $A_2(u,v) - A_1(u,v) \geq 5$, which means u and v have at least 5 more co-authored papers in G_2 than in G_1, we set $D(u,v) = 2$. If $2 \leq A_2(u,v) - A_1(u,v) < 5$, we set $D(u,v) = 1$. If $-4 < A_2(u,v) - A_1(u,v) < 0$, we set $D(u,v) = -1$. If $A_2(u,v) - A_1(u,v) \leq -4$, we set $D(u,v) = -2$. The two different settings of G_D normally lead to different DCS.

Running our DCS algorithms on G_D described above, no matter in Weighted setting or Discrete Setting, what we find is the Emerging co-author group whose strength (density) of collaborations was enhanced after 2010. Thus, the type of G_D described above is called Emerging. We also wanted to mine the disappearing co-author group whose collaboration strength was weakened the most after 2010. Therefore, we tried another type of G_D, the Disappearing G_D, which was obtained by flipping the sign of weight of each edge in the Emerging G_D.

It turned out that, under the same G_D and the same density measure, all algorithms find the same group of authors. We list all co-author groups obtained in Table III. If a group is found under the graph affinity measure, the weight (in the simplex) of each author is also given. We give a short note on the affiliation/address and research interest of each group. Table IV reports the groups found under different settings and density measures. For the average degree measure, we also report the approximation ratio $\frac{2\rho_{D^+}(S_2)}{\rho_D(S)}$. For each group, we report its density differences under the two measures. Note that, for $\mathbf{x}$ under the graph affinity measure, its average degree is $\frac{W_D(S_\mathbf{x})}{|S_\mathbf{x}|}$. We also report the edge density difference, defined as $\frac{W_D(S)}{|S|^2}$ of each co-author group, since edge density can be regarded as a discrete version of graph affinity.

The results show that the research topics of the emerging groups are machine learning and security, which both are hot topics in recent years. As to the disappearing groups, Compiler & Software System are all relatively mature areas of computer science, and, for the 3 Japanese Robotics research groups, it is known that recently Japanese researchers do not publish as many papers in international conferences as they did before.

C. Mining Emerging and Disappearing Data Mining Topics

Using the same DBLP dataset, we extracted titles of papers published in some famous Data Mining venues including KDD, ICDM, SDM, PKDD, PAKDD, TKDE, TKDD and DMKD. Similar to [1], we built keyword association graphs from the paper titles. Unlike [1], we tried to identify emerging and disappearing data mining topics during 2008-2017, compared to the time period 1998-2007. Thus, we split all paper titles in two parts based on their publication years, and built two keyword association graphs G_1 (1998-2007) and G_2 (2008-2017). We removed all stop words and used the rest words as keywords. The edge weights of G_1 and G_2 were set based on the pairwise co-occurrences of keywords as suggested by [1]. Specifically, for an edge between two keywords, we set its weight as 100 times the percentage of paper titles containing both the keywords. Statistics of the difference graphs can be found in Table II (the DM dataset).

This time again all DCSGA algorithms found the same emerging topic {**social (0.5), networks (0.5)**} and the same disappearing topic {**mining (0.12), association (0.44), rules (0.44)**}. Our DCSGreedy algorithm for solving DCSAD also found the disappearing topic {mining, association, rules}. We skip the emerging topic w.r.t. the average degree measure, because DCSGreedy found a large set of 38 keywords which lacks interpretability. Since a research topic/story often only has a few keywords, the graph affinity which prefers small subgraphs is a more proper density measure in this task compared to the average degree. [1] also suggests to use small and dense subgraphs to identify stories in text data.

To further demonstrate the effectiveness of applying DCS in identifying emerging/disappearing research topics, we also display the top results returned by our SEACD+Refinement algorithm. Remember this algorithm does initializations using every vertex in G_D and returns multiple positive cliques in G_D. We removed the duplicate cliques and the cliques that are sub-graphs of other cliques found. We list the top-5 positive cliques with the highest graph affinity difference found by the SEACD+Refinement in Table V.

From the results we can find that our DCSGA algorithms are very effective. Social networks, matrix factorization, semi-supervised learning and unsupervised feature selection all became hot topics only in recent years, and they were not that popular in early years. Moreover, due to the need from industry and the development of computation power, large scale is turning into one of the most important concerns in data mining research. For the disappearing topics, association rule mining, support vector machines, inductive logic programming and intrusion detection are all relatively mature research topics which were majorly investigated in early years. "Knowledge discovery" used to be a popularly adopted term when data mining as a research area arouse.

What's more, we also report the top-5 topics in G_1 and G_2 in Table VI. Since average degree density measure prefers large subgraphs and is not very proper for identifying topics/stories, we do not report the top topics w.r.t. average degree. The aim of displaying such results is to show the necessity of

Data	Setting	G_D Type	n	m^+	m^-	Max w	Min w	Average w
DBLP	Weighted	Emerging	22,572	61,703	61,551	46	-100	-0.015
DBLP	Weighted	Disappearing	22,572	61,551	61,703	100	-46	0.015
DBLP	Discrete	Emerging	22,572	21,367	61,551	2	-2	-0.518
DBLP	Discrete	Disappearing	22,572	61,551	21,367	2	-2	0.518
DM	—	Emerging	9890	140,705	67,541	1.988	-5.997	0.0007
DM	—	Disappearing	9890	67,541	140,705	5.997	-1.988	-0.0007
Wiki	—	Consistent	116,836	762,999	1,264,872	9.619	-12.46	-0.474
Wiki	—	Conflicting	116,836	1,264,872	762,999	12.46	-9.619	0.474
Movie [26]	—	Interest−Social	55,710	338,524	914,292	1	-1	-0.46
Movie [26]	—	Socia−Interest	55,710	914,292	338,524	1	-1	0.46
Book [26]	—	Interest−Social	55,710	124,027	918,925	1	-1	-0.762
Book [26]	—	Social−Interest	55,710	918,925	124,027	1	-1	0.762
DBLP-C	Weighted	—	1,282,461	2,538,746	2,359,487	400	-186	0.188
DBLP-C	Discrete	—	1,282,461	2,538,746	2,359,487	2	-2	-0.013
Actor	Weighted	—	382,219	15,038,083	0	216	1	1.101
Actor	Discrete	—	382,219	15,038,083	0	10	1	1.098

TABLE II. STATISTICS DIFFERENCE GRAPHS IN EXPERIMENTS (n represents #vertices, m^+ is #edges with positive weights and m^- is #edges with negative weights. "Max w" is the maximum edge weight while "Min w" is the minimum one. We also report the average edge weight in the column of "Average w". "Setting" and "G_D Type" denote how the difference graph was built. "G_D Type" denotes which graph is used as G_1 and which is used as G_2.)

List of Authors	Note
Feiping Nie(0.4428), Heng Huang(0.462), Chris H. Q. Ding(0.0230), Hua Wang(0.0717)	UTA Machine Learning
Lorrie Faith Cranor(0.1428), Nicolas Christin(0.1428), Blase Ur(0.1428), Richard Shay(0.1428), Saranga Komanduri(0.1428), Michelle L. Mazurek(0.1428), Lujo Bauer(0.1428)	CMU Privacy & Security
Kensuke Harada, Kiyoshi Fujiwara, Fumio Kanehiro, Hirohisa Hirukawa, Shuuji Kajita, Kenji Kaneko	Japan Robotics 1
Toshio Fukuda(0.5), Fumihito Arai(0.5)	Japan Robotics 2
Fumio Kanehiro(0.1428), Shuuji Kajita(0.1428), Kenji Kaneko(0.1428), Kensuke Harada(0.1428), Kiyoshi Fujiwara(0.1428), Hirohisa Hirukawa(0.1428), Mitsuharu Morisawa(0.1428)	Japan Robotics 3
Monica S. Lam, Katherine A. Yelick, Alok N. Choudhary, Michael L. Scott, James C. Browne, Marina C. Chen, Rudolf Eigenmann, Dennis Gannon, Charles Koelbel, Wei Li 0015, Thomas J. LeBlanc, David A. Padua, Constantine D. Polychronopoulos, Sanjay Ranka, Ian T. Foster, Carl Kesselman, Geoffrey Fox, Tomasz Haupt, Allen D. Malony, Janice E. Cuny, Joel H. Saltz, Alan Sussman	Compiler & Software System

TABLE III. CO-AUTHOR GROUPS

Setting	G_D Type	Density	Co-author Group	#Authors	Positive Clique?	Ave. Degree Difference	Approx. Ratio	Graph Affinity Difference	Edge Density Difference ($\frac{W_D(S)}{\lvert S\rvert^2}$)
Weighted	Emerging	Average Degree	UTA Machine Learning	4	Yes	81.5	2	—	20.375
Weighted	Emerging	Graph Affinity	UTA Machine Learning	4	Yes	81.5	—	23.167	20.375
Weighted	Disappearing	Average Degree	Japan Robotics 1	6	Yes	143	2	—	23.833
Weighted	Disappearing	Graph Affinity	Japan Robotics 2	2	Yes	50	—	50	50
Discrete	Emerging	Average Degree	CMU Privacy & Security	7	Yes	12	2	—	1.714
Discrete	Emerging	Graph Affinity	CMU Privacy & Security	7	Yes	12	—	1.714	1.714
Discrete	Disappearing	Average Degree	Compiler & Software System	22	Yes	21.45	2	—	0.975
Discrete	Disappearing	Graph Affinity	Japan Robotics 3	8	Yes	14	—	1.714	1.714

TABLE IV. INFORMATION OF CO-AUTHOR GROUPS

Rank	Keyword Set/Topic	
	Emerging	Disappearing
1	{social (0.5), networks (0.5)}	{mining (0.12), association (0.45), rules (0.43)}
2	{large (0.5), scale (0.5)}	{knowledge (0.5), discovery (0.5) }
3	{matrix (0.5), factorization (0.5)}	{support (0.39), vector (0.38), machines (0.23) }
4	{semi (0.45), supervised (0.45), learning (0.1)}	{logic (0.36), inductive (0.26), programming (0.38)}
5	{unsupervised (0.34), feature (0.29), selection (0.27)}	{intrusion (0.5), detection (0.5)}

TABLE V. TOP 5 EMERGING/DISAPPEARING TOPICS W.R.T. GRAPH AFFINITY

Rank	Keyword Set/Topic	
	G_1 (1998-2007)	G_2 (2008-2017)
1	{time (0.5), series (0.5)}	{social (0.5), networks (0.5)}
2	{support (0.41), vector (0.41), machines (0.18)}	{time (0.5), series (0.5)}
3	{feature (0.5), selection (0.5)}	{large (0.5), scale (0.5)}
4	{decision (0.5), trees (0.5)}	{feature (0.5), selection (0.5)}
5	{nearest (0.5), neighbor (0.5)}	semi (0.46), supervised (0.47), learning (0.07)}

TABLE VI. TOP 5 TOPICS W.R.T. GRAPH AFFINITY

applying DCS to find emerging/disappearing topics. If we mine emerging topics only in one graph like [1] does, the results may be not effective. For example, if we only consider G_2 to mine emerging topics, we would find {time (0.5), series (0.5)} and {feature (0.5), selection (0.5)}. However, {time (0.5), series (0.5)} and {feature (0.5), selection (0.5)} were hot topics before 2008 so they were not emerging topics during 2008-2017. The topic {time (0.5), series (0.5)} even cooled down in the last ten years, since its graph affinity density dropped from 1.185 (in G_1) to 1.049 (in G_2) based on our calculation.

D. Efficiency Comparison

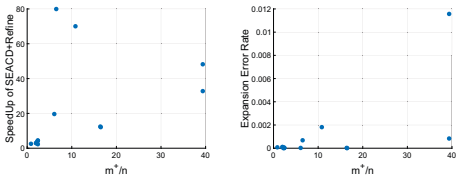
Limited by space, we focus on the running time of the **DCSGA** algorithms, since all **DCSAD** algorithms have quasi-linear time complexity $O((m_1+m_2+n)\log n)$, and are efficient and scalable in practice.

Besides the above DCS mining tasks, to compare the efficiency of the algorithms, we also employed several other data sets whose statistics can be found in Table II. How these datasets were generated and the description of experiments on these datasets please refer to the full version.

Table VII reports the running time of each **DCSGA** algorithm on each data set. Since we set different convergence

Data	Setting	G_D Type	NewSEA	SEACD+ Refine	SEA+ Refine	#Errors in SEA
DBLP	Weighted	Emerging	0.05	3.2	14.3	1
DBLP	Weighted	Disappearing	0.05	3.2	13.7	1
DBLP	Discrete	Emerging	0.06	2.9	7.3	2
DBLP	Discrete	Disappearing	0.06	2.9	6.8	0
DM	—	Emerging	0.35	14.1	185.3	0
DM	—	Disappearing	0.21	6.9	36.3	0
Wiki	—	Consistent	56.6	452	36121	80
Wiki	—	Conflicting	23.8	110	7703	211
Movie	—	Interest−Social	16.3	29.6	580.6	1
Movie	—	Social−Interest	23.1	32.7	404.8	1
Book	—	Interest−Social	2.02	14.5	53.2	0
Book	—	Social−Interest	20.9	32.7	397	0
DBLP-C	Weighted	—	2.01	8054	23090	118
DBLP-C	Discrete	—	12.3	7678	22837	131
Actor	Weighted	—	2.3	2249	73671	321
Actor	Discrete	—	155	2574	124132	4419

TABLE VII. RUNNING TIME IN SECONDS.



(a) Speed-Up (b) Errors of SEA

Fig. 2. SpeedUp of SEACD+Refine and Errors in Expansions of SEA+Refine

conditions for the Shrink stage of each algorithm, one may wonder whether the convergence condition for SEA+Refine is too strict and makes SEA+Refine not as efficient as the other two algorithms. Thus, we also report the number of errors made by SEA+Refine in the Expansion stages. Note that the errors in Expansion are caused by that the Shrink stage cannot reach a local KKT point. From Table VII we find that the SEA+Refine algorithm often made mistakes in the Expansion stage, which means the convergence condition for the Shrink stage of SEA+Refine is still too loose to achieve a local KKT point. It is worth noting that the two algorithms using our coordinate descent algorithm in the Shrink stage, NewSEA and SEACD+Refine, never made mistakes in the Expansion stage. We also find that our NewSEA algorithm often is much faster than the other two algorithms. Note that the only difference between NewSEA and SEACD+Refine is the smart initialization heuristic. Compared to SEACD+Refine, the smart initialization heuristic sometimes brings us a speed up of 3 orders of magnitude. Moreover, SEACD+Refine is always faster than SEA+Refine, sometimes 80 times faster. It seems when the input G_{D^+} is sparse, SEACD+Refine and SEA+Refine are close in efficiency. When G_{D^+} becomes denser, the gap in efficiency gets larger. The Expansion error rate (defined by $\frac{\#\text{Errors in SEA}}{n}$) seems correlated with how dense G_{D^+} is. The results are shown in Fig. 2, where m^+/n measures how dense G_{D^+} is, and m^+ is the number of edges in G_{D^+}.

E. Comparison with EgoScan [6]

Both DCSAD and DCSGA are new problems that were not discussed in literature before, and this paper focuses on algorithmic solutions to the two problems, so there are no very suitable baselines for our algorithms. However, in this section, we still compare our DCS mining algorithms with the EgoScan algorithm in [6], which is the work closest to ours in literature. The objective of EgoScan is to maximize $W_D(S)$ subject to $S \subseteq V$ on the difference graph G_D.

We ran the EgoScan algorithm[3] on the datasets used in our experiments. Unfortunately, since EgoScan needs a Semi-Definite Programming (SDP) solver as a frequently used subroutine, and the SDP solver is really slow and consumes too much memory when ego nets of vertices are large (having more than thousands of vertices), we only got results on the 4 DBLP co-author difference graphs that we used to draw emerging/disappearing co-author groups. For the 4 graphs, EgoScan always spent more than 100 seconds to finish. For other datasets, either EgoScan could not finish running in one day or the memory (16GB) of our machine was not enough for the SDP solver. The high computational cost is actually one drawback of applying EgoScan in practice.

We display the results of running EgoScan on the DBLP co-author data. Since all co-author groups found by EgoScan have at least 44 authors, we cannot list all the authors. We only show statistics of these co-author groups. From Table VIII and referring to Table IV which shows statistics of the author groups found by our DCS algorithms, we find that our DCS algorithms are much better than EgoScan in finding DCS w.r.t. average degree and edge density. Moreover, subgraphs found by EgoScan are all big, even bigger than the subgraphs found by our DCSGreedy algorithms.

We also compare our DCS algorithms with EgoScan in finding subgraphs w.r.t. the total edge weight difference $W_D(S)$, which is shown in Table IX. Note that the total edge weigh difference of a solution $\mathbf{x}$ returned by our NewSEA algorithm is defined as $W_D(S_{\mathbf{x}})$. Under the evaluation metric of total edge weight difference, EgoScan outperforms our DCS algorithms.

Table VIII, IX and IV show that DCS w.r.t. different measures could be very different. We have the following rough suggestions for deciding which measures to use in practice: (1) if users prefer small DCS and good interpretability, we should take graph affinity as the density measure and apply our NewSEA algorithm, since it always returns a positive clique where for every pair of vertices, their connection in G_2 is tighter than their connection in G_1; (2) if users prefer a medium sized subgraph, then average degree should be the measure and we apply our DCSGreedy algorithm; (3) If users want a even larger subgraph, total edge weight maybe the suitable measure because it seems that such a measure encourages even bigger subgraphs than average degree.

VII. CONCLUSION

In this paper, we studied the Density Contrast Subgraph problem that have interesting applications in practice. Two popularly adopted graph density measures, average degree and graph affinity, were considered. We proved the hardness of the DCS problem under the two measures, and devised algorithms that work well in practice for finding DCS under both density measures. We reported a series of experiments on both real

[3]We thank the authors of [6] for providing us the code of EgoScan.

Setting	G_D Type	#Authors	#Edges	Positive Clique?	Ave. Degree Difference	Edge Density Difference ($\frac{W_D(S)}{\|S\|^2}$)
Weighted	Emerging	82	473	No	26.95	0.3287
Weighted	Disappearing	59	311	No	45.39	0.7693
Discrete	Emerging	44	124	No	7.46	0.1694
Discrete	Disappearing	80	527	No	13.8	0.1725

TABLE VIII. STATISTICS OF CO-AUTHOR GROUPS (SUBGRAPHS) FOUND BY EGOSCAN

Setting	G_D Type	DCSGreedy	NewSEA ($W_D(S_\mathbf{x})$)	EgoScan
Weighted	Emerging	326	326	2210
Weighted	Disappearing	858	100	2678
Discrete	Emerging	84	84	328
Discrete	Disappearing	472	112	1104

TABLE IX. TOTAL EDGE WEIGHT DIFFERENCE ($W_D(S)$) OF CO-AUTHOR GROUPS FOUND BY DCS ALGORITHMS AND EGOSCAN

and synthetic datasets and demonstrated the effectiveness and efficiency of our algorithms.

There are some interesting future directions. For example, our methods are based on graph density, but density sometimes cannot reflect how "dissimilar" a subgraph looks in two graphs. Thus, how to extract subgraphs that are dissimilar in two graphs with respect to some graph similarity measures [14] is interesting. Also, our methods only mine one DCS with the greatest density difference, how to mine multiple subgraphs with big density difference is another interesting direction.

REFERENCES

[1] A. Angel et al. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012.

[2] B. Bahmani et al. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012.

[3] J. L. Bentley. Solutions to klee's rectangle problems. Technical report, Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, 1977.

[4] S. Bhattacharya et al. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC*, pages 173–182. ACM, 2015.

[5] S. Boyd et al. *Convex optimization*. Cambridge university press, 2004.

[6] J. Cadena et al. On dense subgraphs in signed network streams. In *ICDM*, pages 51–60. IEEE, 2016.

[7] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*, pages 84–95. Springer, 2000.

[8] L. Chu et al. Alid: scalable dominant cluster detection. *PVLDB*, 8(8):826–837, 2015.

[9] A. Epasto et al. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310. ACM, 2015.

[10] E. Fratkin et al. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

[11] A. Gionis et al. Bump hunting in the dark: Local discrepancy maximization on graphs. In *ICDE*, pages 1155–1166. IEEE, 2015.

[12] A. V. Goldberg. *Finding a maximum density subgraph*. University of California Berkeley, CA, 1984.

[13] H. Hu et al. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):i213–i221, 2005.

[14] D. Koutra et al. Deltacon: A principled massive-graph similarity function. In *SDM*, pages 162–170. SIAM, 2013.

[15] W. Li et al. Pattern mining across many massive biological networks. In *Functional Coherence of Molecular Networks in Bioinformatics*, pages 137–170. Springer, 2012.

[16] H. Liu et al. Fast detection of dense subgraphs with iterative shrinking and expansion. *IEEE TPAMI*, 35(9):2131–2142, 2013.

[17] M. Mitzenmacher et al. Scalable large near-clique detection in large-scale networks via sampling. In *KDD*, pages 815–824. ACM, 2015.

[18] T. S. Motzkin et al. Maxima for graphs and a new proof of a theorem of turán. *Canad. J. Math*, 17(4):533–540, 1965.

[19] M. Pavan et al. Dominant sets and pairwise clustering. *IEEE TPAMI*, 29(1), 2007.

[20] R. A. Rossi et al. Fast maximum clique algorithms for large graphs. In *WWW*, pages 365–366. ACM, 2014.

[21] R. M. H. Ting et al. Mining minimal contrast subgraph patterns. In *SDM*, pages 639–643. SIAM, 2006.

[22] C. Tsourakakis et al. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112. ACM, 2013.

[23] B. Wang et al. Spatial scan statistics for graph clustering. In *SDM*, pages 727–738. SIAM, 2008.

[24] Z. Wang et al. Tradeoffs between density and size in extracting dense subgraphs: A unified framework. In *ASONAM*, pages 41–48. IEEE, 2016.

[25] Y. Wu et al. Mining dual networks: Models, algorithms, and applications. *ACM TKDD*, 10(4):40, 2016.

[26] T. Xu et al. Towards annotating media contents through social diffusion analysis. In *ICDM*, pages 1158–1163. IEEE, 2012.

[27] Y. Yang et al. Mining most frequently changing component in evolving graphs. *World Wide Web*, 17(3):351–376, 2014.

APPENDIX: THE SEA ALGORITHM

The SEA algorithm [16] solves Eq. 6 when the symmetric matrix D only has non-negative entries. The strategy of SEA is to iteratively find a local KKT point (Shrink stage) and expand it to more vertices (Expansion stage) until convergence.

Shrink Stage. To find a local KKT point on a set of vertices S, a replicator dynamic is exploited. The replicator equation is

$$x_i(t+1) = x_i(t)\frac{(D\mathbf{x})_i}{\mathbf{x}(t)^\top D\mathbf{x}(t)},\ i \in S \tag{12}$$

where $x_i(t)$ is the value of x_i in the t-th iteration. To make this replicator dynamic converge, D should be non-negative.

Expansion Stage. In the Expansion stage, SEA firstly find the set Z as Algorithm 3 does in Line 6. According to Eq. 12, if x_i at the beginning of the replicator dynamic is 0, it will stay 0 forever. Thus, SEA needs to give a positive initial value x_v to each vertex $v \in Z$. To do that, we first define the γ vector,

$$\gamma_i = \begin{cases} 0 & i \notin Z \\ \nabla_i f_D(\mathbf{x}) - f_D(\mathbf{x}) & i \in Z \end{cases}$$

where $\mathbf{x}$ is the local KKT point to be expanded by Z. Let $s = \sum_{i\in Z}\gamma_i$, $\zeta = \sum_{i\in Z}\gamma_i^2$ and $\omega = \sum_{i,j\in Z}\gamma_i\gamma_j D(i,j)$. The Expansion stage updates $\mathbf{x}$ along the direction $\mathbf{b}$, where

$$b_i = \begin{cases} -x_i s & i \in S_\mathbf{x} \\ \gamma_i & i \in Z \end{cases}$$

Let $f_D(\mathbf{x}) = \bar{\lambda}$. We maximize $f_D(\mathbf{x}+\tau\mathbf{b}) - f_D(\mathbf{x}) = -(\bar{\lambda}s^2 + 2s\zeta - \omega)\tau^2 - 2\zeta\tau$ over τ. The best τ can be found analytically. Let $a = \bar{\lambda}s^2 + 2s\zeta - \omega$, when $a \le 0$ we set $\tau = \frac{1}{s}$, and we set $\tau = \min\{\frac{1}{s}, -\frac{1}{a}\}$ otherwise. Then $\mathbf{x}$ is updated to $\mathbf{x}+\tau\mathbf{b}$.