Efficient Directed Densest Subgraph Discovery

Chenhao Ma
Department of Computer Science, The University of Hong Kong
chma2@cs.hku.hk

Yixiang Fang
School of Data Science, The Chinese University of Hong Kong, Shenzhen
fangyixianghku@gmail.com

Reynold Cheng
Department of Computer Science, The University of Hong Kong
ckcheng@cs.hku.hk

Laks V.S. Lakshmanan
Department of Computer Science, The University of British Columbia
laks@cs.ubc.ca

Wenjie Zhang
School of Computer Science and Engineerinng, University of New South Wales
wenjie.zhang@unsw.edu.au

Xuemin Lin
School of Computer Science and Engineerinng, University of New South Wales
lxue@cse.unsw.edu.au

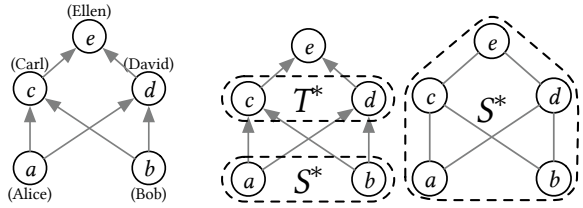
ABSTRACT

Given a directed graph G, the directed densest subgraph (DDS) problem refers to the finding of a subgraph from G, whose density is the highest among all the subgraphs of G. The DDS problem is fundamental to a wide range of applications, such as fraud detection, community mining, and graph compression. However, existing DDS solutions suffer from efficiency and scalability problems: on a three-thousand-edge graph, it takes three days for one of the best exact algorithms to complete. In this paper, we develop an efficient and scalable DDS solution. We introduce the notion of $[x, y]$-core, which is a dense subgraph for G, and show that the densest subgraph can be accurately located through the $[x, y]$-core with theoretical guarantees. Based on the $[x, y]$-core, we develop both exact and approximation algorithms. We have performed an extensive evaluation of our approaches on eight real large datasets. The results show that our proposed solutions are up to six orders of magnitude faster than the state-of-the-art.

1. INTRODUCTION

In emerging systems that manage complex relationships among objects, directed graphs are often used to model the relationships [12, 4, 1, 17]. For example, in online microblogging services (e.g., Twitter and Weibo), the "following" relationships between users can be captured as directed edges [12]. Figure 1a depicts a directed graph of the following relationship for five users in a microblogging network. Here, `Alice` has a link to `David` because she is a follower of `David`. As another example, in Wikipedia, each article can be considered as a vertex, and each link between two articles is represented by a directed edge from one vertex to another [4]. As yet another example, the Web can also be viewed as a huge directed graph [1].

In this paper, we study the problem of finding the densest subgraph from a directed graph G, which was first proposed in [13]. Conceptually, this *directed densest subgraph* (DDS) problem aims to find two sets of vertices, S^* and T^*, from



(a) A directed graph (b) Directed DS (c) Undirected DS

Figure 1: Illustrating the problem of densest subgraph discovery (or DDS problem) on the directed graph.

G, where (1) vertices in S^* have a large number of outgoing edges to those in T^*, and (2) vertices in T^* receive a large number of edges from those in S^*. To understand DDS, let us explain its usage in fake follower detection [20, 11] and community mining [15]:

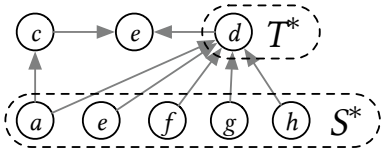


Figure 2: An example of fake follower detection.

• *Fake follower detection* [11] aims to identify fraudulent actions in social networks [11]. Figure 2 illustrates a microblogging network, with edges representing the "following" relationship. By issuing a DDS query, two sets of users S^* and T^*, are returned. Compared with other users, d (in T^*) has unusually a huge number of followers (a, e, f, g, h) in S^*. It may be worth to investigate whether d has bribed the users in S^* for following him/her.

• *Community mining* [15]. In [15], Kleinberg proposed the *hub-authority* concept for finding web communities, based on a hypothesis that a web community is often comprised of a set of *hub pages* and a set of *authority pages*. The hubs are characterized by the presence of a large number of edges to the authorities, while the authorities often receive a large number of links from the hubs. A DDS query can be issued on this network to find hubs and authorities. In Figure 3, for example, websites in S^* can be viewed as hubs providing

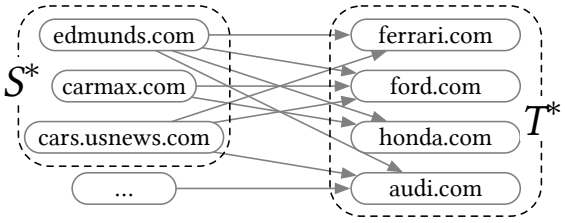


Figure 3: An example of web community.

car rankings and recommendations, while websites in T^* play the roles of authorities, as the official websites for well-known automakers.

Now let us give more details about the DDS query [13, 14, 5, 2]. Given a directed graph $G = (V, E)$ and sets of (not necesssarily disjoint) vertices $S, T \subseteq V$, the density of the directed subgraph induced by (S, T) is the number $|E(S, T)|$ of edges linking vertices in S to the vertices in T over the square root of the product of their sizes, i.e., $\rho(S, T) = \frac{|E(S,T)|}{\sqrt{|S||T|}}$. Based on this definition, the DDS problem aims to find a pair of sets of vertices, S^* and T^*, such that $\rho(S^*, T^*)$ is the maximum among all possible choices of $S, T \subseteq V$. For instance, for the directed graph in Figure 1a, the DDS is the subgraph induced by $S^*=\{a, b\}$ and $T^*=\{c, d\}$, (see Figure 1b). Its density is $\rho^* = \frac{4}{\sqrt{2\times 2}} = 2$. For each of the aforementioned applications, we can verify that the required solution corresponds to a DDS.

In undirected graphs, the density of a graph $G = (V, E)$ is defined to be $\rho(G) = \frac{|E|}{|V|}$ [9], which is different from that in directed graphs. Hence, finding the densest subgraph in undirected graphs (DS problem for short) amounts to finding the subgraph with the highest average degree [9, 7]. For example, for the undirected graph G in Figure 1c, the DS is G itself and its density is $\frac{6}{5}$, since there is no subgraph with higher density. We can observe that when $S = T$, the density of a directed graph reduces to the classical notion of the density of undirected graphs. Thus, it naturally generalizes the notion of the density of undirected graphs. On the other hand, the solution to the DDS problem returns two sets, S^* and T^*, which provide the advantage to distinguish different roles of vertices in the above applications.

Impact. The densest subgraph problem lies at the core of large scale data mining [2]. DDS is an important primitive for real-world applications, such as fake follower detection and community detection. Theoretically, the densest subgraph problem closely connects to fundamental graph problems such as network flow and bipartite matching [21]. Hence, the DDS problem receives much attention from the communities of the database, data mining, theory, and network analysis.

State-of-the-art. For a directed graph G=(V, E), we denote its number of vertices and edges by n and m respectively. In the literature, both exact [5, 14] and approximation algorithms [13, 5, 2] for DDS have been studied. The state-of-the-art exact algorithm is a flow-based algorithm [14], which mainly involves two nested loops: the outer loop enumerates all the n^2 possible values of $\frac{|S|}{|T|}$ ($1 \le |S|, |T| \le n$), while the inner loop computes the maximum density by using binary search on a flow network, regarding a specific value of $\frac{|S|}{|T|}$. The inner and outer loops take $O(nm \log n)$ and $O(n^2)$ time respectively, so its overall time complexity is $O(n^3 m \log n)$, which is prohibitively expensive for large graphs.

Table 1: Summary of exact algorithms.

Algorithm	Time complexity
`LP-Exact` [5]	$\Omega(n^6)$
`Exact` [14]	$O(n^3 m \log n)$
`DC-Exact` **(Ours)**	$O(k \cdot nm \log n)$

Note: Theoretically, $k \le n^2$. But, $k \ll n^2$, in practice.

Table 2: Summary of approximation algorithms.

Algorithm	Approx. ratio	Time complexity
`KV-Approx` [13]	$O(\log n)$	$O(s^3 n)$
`PM-Approx` [2]	$2\delta(1+\epsilon)$	$O(\frac{\log n}{\log \delta} \log_{1+\epsilon} n(n+m))$
`KS-Approx` [14]	>2	$O(n+m)$
`BS-Approx` [5]	2	$O(n^2 \cdot (n+m))$
`Core-Approx` **(Ours)**	2	$O(\sqrt{m}(n+m))$

Note: s is the sample size; ϵ, δ are the error tolerance parameters.

To improve efficiency, approximation algorithms have been developed, the most efficient one being the algorithm in [14], which only costs $O(n+m)$ time, since it iteratively peels the vertex with the smallest in-degree or out-degree. However, it was misclaimed to achieve an approximation ratio of 2, as we will show in Section 3.2. Here, the approximation ratio is defined as the ratio of the density of the DDS (i.e., the optimal solution) over that of the subgraph returned. This makes the algorithm proposed by Charikar in [5] be the best available 2-approximation algorithm, and its time complexity is $O(n^2(n+m))$. Clearly, it is still very expensive, warranting more efficient algorithms. Tables 1 and 2 summarize the properties of the exact and approximation algorithms, respectively.

Our technical contributions. To improve the state-of-the-art exact algorithm [14], we optimize its inner and outer loops. Specifically, for the inner loop, we introduce a novel dense subgraph model on directed graphs, namely [x, y]-core, inspired by the k-core [22] on undirected graphs. That is, given two sets of vertices S and T of a graph G, the subgraph induced by S and T in G is an [x, y]-core, if each vertex in S has at least x outgoing edges to vertices in T, and each vertex in T has at least y incoming edges from vertices in S. Theoretically, we show that DDS can be accurately located through the [x, y]-cores, which are often much smaller than the entire graph. As a result, we can build the flow networks on some [x, y]-cores, rather than the entire graph, which greatly improves the efficiency of computing the maximum flow. For the outer loop, we propose a divide-and-conquer strategy, which dramatically reduces the number of values of $\frac{|S|}{|T|}$ examined from n^2 to k. Theoretically, $k \in O(n^2)$. But, $k \ll n^2$, in practice. Based on the two optimization techniques above, we develop an efficient exact algorithm `DC-Exact`.

We further show that theoretically, the [x^*, y^*]-core, where x^*y^* is the maximum value among the values of x and y for all the [x, y]-cores, is a 2-approximation solution to the DDS problem. To compute the [x^*, y^*]-core, we propose an efficient algorithm, called `Core-Approx`, which completes in $O(\sqrt{m} \cdot (n+m))$ time. Therefore, compared to existing 2-approximation algorithms, it has the lowest time complexity.

We have experimentally compared our proposed solutions with the state-of-the-art solutions on eight real graphs, where the largest one consists of around two billions edges. The results show that for the exact algorithms, our proposed `DC-Exact` is over six orders of magnitude faster than the baseline algorithm on a graph with around 6,500 vertices and 51,000 edges. Besides, for approximation algorithms, our proposed `Core-Approx` algorithm can scale well to billion-scale graphs, and is also up to six orders of magnitude faster than the existing 2-approximation algorithm [5].

Outline. The rest of the paper is organized as follows. In Section 2, we formally present the DDS problem. Section 3 reviews the state-of-the-art algorithms and discusses their limitations. We present our exact and approximation algorithm in Section 4. Experimental results are presented in Section 5. We conclude this paper in Section 6. Our full version [18] provides more complete details on our algorithms and technical and empirical results.

2. PROBLEM DEFINITION

Let G=(V, E) be a directed graph, $n = |V|$ and $m = |E|$ be the number of vertices and edges in G, respectively. Given two sets $S, T \subseteq V$ which are not necessarily disjoint, we use $E(S, T)$ to denote the set of all the edges linking their vertices, i.e., $E(S, T)$=$E \cap (S \times T)$. The subgraph induced by S, T, and $E(S, T)$ is called an (S, T)-induced subgraph, denoted by $G[S, T]$. For a vertex $v \in G$, we use $d^-_G(v)$ and $d^+_G(v)$ to denote its outdegree and indegree in G respectively. Next, we formally present the density of a directed graph [13] and the problem of Directed Densest Subgraph discovery, or DDS problem. Unless mentioned otherwise, all the graphs mentioned later in this paper are directed graphs.

Definition 1 (Density of a directed graph)**.** Given a directed graph G=(V, E) and two sets of vertices $S, T \subseteq V$, the density of the (S, T)-induced subgraph $G[S, T]$ is

$$\rho(S,T) = \frac{|E(S,T)|}{\sqrt{|S| \cdot |T|}}. \quad (1)$$

The reason why the directed density chooses $\sqrt{|S||T|}$ (instead of $|S||T|$) as the denominator is that a pair of vertices with an edge between them would have the highest density and become a trivial solution if we use $|S||T|$. In Figure 1a, if we use $|S||T|$, the densities of $G[S^*, T^*]$ and $G[\{a\}, \{c\}]$ are both equal to 1. The former subgraph is denser than the latter one. [13] provides more arguments about the directed density definition.

Definition 2 (DDS)**.** Given a directed graph G=(V, E), a directed densest subgraph (DDS) D is the (S^*, T^*)-induced subgraph, whose density is the highest among all the possible (S, T)-induced subgraphs. Here, the highest density is denoted by ρ^*.

***Problem* 1** (DDS problem [13, 8, 5, 14, 2])**:** Given a directed graph G=(V, E), find a DDS[1] D=$G[S^*, T^*]$ of G.

3. EXISTING ALGORITHMS

In this section, we review the state-of-the-art exact algorithm [14] and approximation algorithms [14, 5] for the DDS problem. We remark that for approximation algorithms, both the algorithms in [14] and [5] were claimed to achieve an approximation raito of 2, but the former one runs much faster than the latter one. However, we found that the approximation ratio of the former one was misclaimed, which will be illustrated by a counter-example. Note that in this paper, the approximation ratio is defined as the ratio of the maximum density over the density of the subgraph returned.

3.1 The Exact Algorithm

The state-of-the-art exact algorithm [14] computes the DDS by solving a maximum flow problem, which generally follows the same paradigm of the exact algorithm [9] of finding the densest subgraphs on undirected graphs. We denote this algorithm by `Exact`. A flow network [10] is a directed graph F=(V_F, E_F), where there is a source node[2] s, a sink node t, and some intermediate nodes; each edge has a capacity and the amount of flow on an edge cannot exceed the capacity of the edge. The maximum flow of a flow network equals the capacity of its minimum st-cut, $\langle \mathcal{S}, \mathcal{T} \rangle$, which partitions the node set V_F into two disjoint sets, $\mathcal{S}$ and $\mathcal{T}$, such that $s \in \mathcal{S}$ and $t \in \mathcal{T}$.

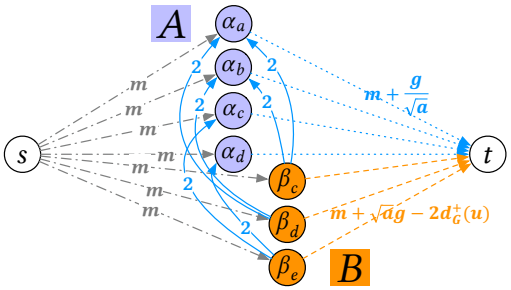


Figure 4: Illustrating the flow network.

We describe the general idea of `Exact`. Our full version [18] provides more details. It first enumerates all the possible values of a=$\frac{|S|}{|T|}$. Then, for each a, it guesses the value g of the maximum density via a binary search. After that, for each pair of a and g, it builds a flow network and runs the maximum flow algorithm to compute the minimum st-cut $\langle \mathcal{S}, \mathcal{T} \rangle$. Figure 4 depicts a flow network constructed in `Exact`. Note that if $\mathcal{S} \backslash \{s\} \neq \emptyset$, then there must be an (S, T)-induced subgraph such that its density is at least g. If such a subgraph exists and g is larger than ρ^*, we update the DDS D and its corresponding density ρ^*.

Limitations. In `Exact`, the number of possible values of a is n^2, and for each a, the while loop of binary search will have $O(\log n)$ iterations. Computing the minimum st-cut of a flow network takes $O(nm)$ time [19]. Consequently, the total time complexity of `Exact` is $O(n^3 m \log n)$, which is thus very inefficient on even small graphs.

3.2 Approximation Algorithms

The state-of-the-art approximation algorithm `KS-Approx` [14] follows the peeling paradigm. Specifically, it works in n rounds. In each round, it removes the vertex whose indegree or outdegree is the smallest, and recomputes the density of the residual graph. Finally, the subgraph whose density is the highest is returned.

[1]There might be several directed densest subgraphs of a graph, and our algorithm will find one of them.

[2]We use "node" to mean "flow network node" in this paper.

It was claimed in [14] that `KS-Approx` has an approximation ratio of 2. Unfortunately, as demonstrated by a counter-example in our full version paper [18], this claim is incorrect [3]. During our recent communication, the authors of [14] have proposed a fix which is a correct 2-approximation algorithm denoted by `FKS-Approx`, but costs $O(n \cdot (n+m))$ time.

Since `KS-Approx` is not a 2-approximation algorithm[4], the most accurate published approximation algorithm is `BS-Approx` [5], which is able to correctly find a 2-approximation result. Similar to `Exact`, `BS-Approx` enumerates all the possible values of $a=\frac{|S|}{|T|}$, and for each specific a, it iteratively removes the vertex with the minimum degree from S or T based on a predefined condition, and then updates S and T, as well as the approximate DDS $\widetilde{D}$.

Limitations. Clearly, the time complexity of `BS-Approx` is $O(n^2 \cdot (n+m))$, where the main overhead comes from the loop of enumerating all the n^2 values of a. Although it is much faster than `Exact`, it is still inefficient for large graphs. As shown in our experiments later, on a graph with about 3,000 vertices and 30,000 edges, it takes around 3 days to compute the DDS. Therefore, it is imperative to develop more efficient approximation algorithms.

4. CORE-BASED ALGORITHMS

In this section, we develop novel efficient exact and approximation algorithms for the DDS problem. Our algorithms rely on a new concept, namely $[x, y]$-core, which is an extension of the classic k-core [22] for directed graphs. In the following, we first introduce the $[x, y]$-core, then present our core-based exact algorithm, further optimize it by exploiting a divide-and-conquer strategy, and finally present our core-based 2-approximation algorithm. *Our full version [18] provides the proofs omitted in this section.*

4.1 k-core and $[x, y]$-core

We first review the definition of k-core on undirected graphs.

Definition 3 (k-core [22, 3])**.** Given an undirected graph G and an integer k ($k \geq 0$), the k-core, denoted by $\mathcal{H}_k$, is the largest subgraph of G, such that $\forall v \in \mathcal{H}_k$, $deg_{\mathcal{H}_k}(v) \geq k$.

Definition 4 ($[x, y]$-core)**.** Given a directed graph $G=(V, E)$, an (S, T)-induced subgraph $H=G[S, T]$ is called an **$[x, y]$-core**, if it satisfies:

1. $\forall u \in S, d^-_H(u) \geq x$ and $\forall v \in T, d^+_H(v) \geq y$;
2. $\nexists H' = G[S', T'] \neq H$, such that H is a subgraph of H', i.e., $S \subseteq S'$, $T \subseteq T'$, and H' satisfies (1).

We call $[x, y]$ the **core number pair** of the $[x, y]$-core, abbreviated as **cn-pair**.

Example 1. The subgraph induced by (S^*, T^*), i.e., $D = G[S^*, T^*]$ in Figure 1b is a $[2, 2]$-core. $H = G[\{a, b, c, d\}, \{c, d, e\}]$ is a $[1, 2]$-core, and D is contained in H. □

Similar to the classic k-core, the $[x, y]$-core also has some interesting properties, derived from Definition 4.

[3]The authors of [14] have confirmed that the approximation ratio of `KS-Approx` was misclaimed.

[4]We conduct an empirical study of various exact and approximation algorithms for DDS in Section 5, where we include a comparison with `FKS-Approx` and `KS-Approx`.

Lemma 1 (Nested property)**.** *An $[x, y]$-core is contained by an $[x', y']$-core, where $x \geq x' \geq 0$ and $y \geq y' \geq 0$. In other words, if $H=G[S, T]$ is an $[x, y]$-core, there must exist an $[x', y']$-core $H'=G[S', T']$, such that $S \subseteq S'$ and $T \subseteq T'$.*

Given a pair of x and y, to compute the $[x, y]$-core, we can borrow the idea of k-core decomposition [3]; that is, we can first initialize an (S, T)-induced subgraph such that $S=T=V$, then iteratively remove vertices whose indegrees (resp., outdegrees) are less than x (resp., y) from S (resp., T), and finally return the residual subgraph as the $[x, y]$-core. Clearly, computing a specific $[x, y]$-core takes $O(n+m)$ time by using the bin-sort technique in [3].

4.2 A Core-based Exact Algorithm

We first introduce an interesting lemma, then establish the relationship between the DDS and $[x, y]$-core, and finally present a core-based exact algorithm.

Lemma 2. *Given a directed graph $G=(V, E)$ and its DDS $D=G[S^*, T^*]$ with density ρ^*, we have following conclusions:*

1. for any subset U_S of S^*, removing U_S from S^* will result in the removal of at least $\frac{\rho^*}{2\sqrt{a}} \times |U_S|$ edges from D,

2. for any subset U_T of T^*, removing U_T from T^* will result in the removal of at least $\frac{\sqrt{a}\rho^*}{2} \times |U_T|$ edges from D,

where $a = \frac{|S^*|}{|T^*|}$.

Proof. We prove the lemma by contradiction. For (1), we assume that D is the DDS and removing U_S from D results in the removal of less than $\frac{\rho^*}{2\sqrt{a}} \times |U_S|$ edges from D. This implies that, after removing U_S from S^*, the density of the residual graph, denoted by $D_R=G[S^* \setminus U_S, T^*]$, will be

$$\begin{aligned}\rho(S^* \setminus U_S, T^*) = \frac{|E(S^* \setminus U_S, T^*)|}{\sqrt{|S^* \setminus U_S||T^*|}} &> \frac{\rho^*\sqrt{|S^*||T^*|} - \frac{\rho^*}{2\sqrt{a}}|U_S|}{\sqrt{(|S^*| - |U_S|)|T^*|}} \\ &= \rho^* \frac{|S^*| - \frac{|U_S|}{2}}{\sqrt{|S^*|^2 - |S^*||U_S|}} \\ &= \rho^* \frac{|S^*| - \frac{|U_S|}{2}}{\sqrt{(|S^*| - \frac{|U_S|}{2})^2 - \frac{|U_S|^2}{4}}} \\ &> \rho^*.\end{aligned}$$

However, this contradicts the assumption that D is the DDS, so the conclusion of (1) holds. Similarly, we can prove that the conclusion of (2) holds as well. Hence, the lemma holds. □

Theorem 1. *Given a graph $G=(V, E)$, the DDS $D=G[S^*, T^*]$ is contained in the $\left[\lceil\frac{\rho^*}{2\sqrt{a}}\rceil, \lceil\frac{\sqrt{a}\rho^*}{2}\rceil\right]$-core, where $a = \frac{|S^*|}{|T^*|}$.*

Since the value of ρ^* may not be known in advance, we can only locate the DDS in some cores based on $a=\frac{|S|}{|T|}$ and g guessed, by exploiting the nested property of cores. For example, given a specific a and a lower bound l of ρ^*, then we can locate the DDS in the $\left[\lceil\frac{l}{2\sqrt{a}}\rceil, \lceil\frac{\sqrt{a}l}{2}\rceil\right]$-core, since the $\left[\lceil\frac{\rho^*}{2\sqrt{a}}\rceil, \lceil\frac{\sqrt{a}\rho^*}{2}\rceil\right]$-core is nested within the $\left[\lceil\frac{l}{2\sqrt{a}}\rceil, \lceil\frac{\sqrt{a}l}{2}\rceil\right]$-core. Since the DDS is in some $[x, y]$-cores which are often

much smaller than G, we can build the flow network on these cores, rather than the entire graph G, which will significantly improve the overall efficiency.

Algorithm 1: `Core-Exact`

Input : G=(V, E)
Output: The exact DDS D=$G[S^*, T^*]$

```
1  ρ̃* ←run a 2-approximation algorithm;
2  ρ* ← ρ̃*;
3  foreach a ∈ {n1/n2 | 0 < n1, n2 <= n} do
4      l ← ρ*, r ← 2ρ̃*;
5      while r − l ≥ (√n − √(n−1)) / (n√(n−1)) do
6          g ← (l+r)/2, x ← ⌈l/(2√a)⌉, y ← ⌈√a·l/2⌉;
7          G_r ←Get-XY-Core(G, x, y);
8          F = (V_F, E_F) ← BuildFlowNetwok(G_r, a, g);
9          ⟨S, T⟩ ← Min-ST-Cut(F);
10         if S = {s} then r ← g ;
11         else
12             l ← g;
13             if g > ρ* then D ← G[S ∩ A, S ∩ B], ρ* = g;
14 return D;
```

Based on the above core-based optimization techniques, we develop a novel exact algorithm, called `Core-Exact`, which follows the same framework of `Exact`, as shown in Algorithm 1.

Analysis. To compute a specific [x, y]-core, we can complete in $O(n+m)$ time by using the idea of k-core decomposition [3]. Besides, computing the minimum st-cut takes $O(nm)$ time. Thus, the time complexity of `Core-Exact` is still $O(n^3 m \log n)$. Nevertheless, since we locate the DDS in some [x, y]-cores, the flow networks become smaller, so `Core-Exact` performs much faster than `Exact` in practice.

4.3 A Divide-and-conquer Exact Algorithm

In `Core-Exact`, we mainly optimize the inner loop of `Exact`, i.e., reducing cost of computing the minimum st-cut. A natural question comes: can we improve the outer loop of `Exact` so that we can enuemrate fewer values of a=$\frac{|S|}{|T|}$? In the following, we show that this is possible.

Our idea is based on a key observation that given a specific value of a, the results of the binary search (lines 5-13 in Algorithm 1) actually have provided insights for reducing the number of tries of a. As shown in [14], essentially the binary search solves the following optimization problem, where a is a pre-given value.

$$\max_{S,T\in V} g \quad \text{s.t. } \frac{|S|}{\sqrt{a}}\left(g-\frac{|E(S,T)|}{|S|/\sqrt{a}}\right)+|T|\sqrt{a}\left(g-\frac{|E(S,T)|}{|T|\sqrt{a}}\right)\le 0. \tag{2}$$

g is the maximum value the binary search can obtain when a is fixed. Then, we can derive the following lemma.

Lemma 3. *Given a graph G=(V, E) and a specific a, assume that S' and T' are the optimal choices for Equation (2). Let b=$\frac{|S'|}{|T'|}$ and c=$\frac{a^2}{b}$. Then, for any (S, T)-induced subgraph $G[S, T]$ of G, if $\min\{b,c\} \le \frac{|S|}{|T|} \le \max\{b,c\}$, we have $\rho(S,T)$≤$\rho(S',T')$.*

Algorithm 2: `DC-Exact`

Input : G=(V, E)
Output: The exact DDS D=$G[S^*, T^*]$

```
1  a_l ← 1/n, a_r ← n, ρ* ← 0, D ← ∅;
2  Divide-Conquer(a_l, a_r);
3  return D;
4  Function Divide-Conquer(a_l, a_r):
5      a_mid ← (a_l + a_r)/2;
6      run Lines 4-13 of Algorithm 1 (replace
         x ← ⌈l/(2√a)⌉, y ← ⌈√a·l/2⌉ with x ← ⌈l/(2√a_r)⌉,
         y ← ⌈√a_l·l/2⌉);
7      let G[S', T'] be the DDS found by binary search;
8      b ← |S'|/|T'|;
9      c ← a_mid^2 / b;
10     if b > c then Swap(b, c);
11     if a_l ≤ b then Divide-Conquer(a_l, b);
12     if c ≤ a_r then Divide-Conquer(c, a_r);
```

Proof. We prove the lemma by contradiction. In Equation (2), given a specific a, let $g^*(a)$ be the optimal value of g. Because S' and T' are the optimal choices for S and T in Equation (2), S', T', and $g^*(a)$ have the following relationship, according to [14]:

$$g^*(a)=\frac{2\rho(S',T')}{\frac{\sqrt{b}}{\sqrt{a}}+\frac{\sqrt{a}}{\sqrt{b}}}. \tag{3}$$

Let $h_a(x)=\frac{\sqrt{x}}{\sqrt{a}}+\frac{\sqrt{a}}{\sqrt{x}}$. Then, we have the derivative of $h_a(x)$,

$$h'_a(x)=\frac{-a+x}{2\sqrt{a}x^{3/2}},\ x>0. \tag{4}$$

It is easy to observe that when x=a, $h'_a(x)$=0; when $x\in(0,a)$, $h'_a(x)<0$; when $x\in(a,+\infty)$, $h'_a(x)>0$. Therefore, $h_a(x)$ is a convex function, and we can get its minimum value by setting x to a.

We now prove the lemma by contradiction. Assume that there exists an [S_x, T_x]-induced subgraph, which satisfies $\min\{b,c\} \le x=\frac{|S_x|}{|T_x|} \le \max\{b,c\}$, but it has $\rho(S_x,T_y) > \rho(S',T')$. Since $h_a(x) \le h_a(b)$ and $\rho(S_x,T_y)>\rho(S',T')$, we can conclude $\frac{2\rho(S_x,T_x)}{h_a(x)} > \frac{2\rho(S',T')}{h_a(b)}$. However, this contradicts the assumption that S' and T' are the optimal choice for Equation (2). Hence, the lemma holds. □

According to Lemma 3, after conducting the binary search for a specific value of a, we can skip performing binary search for all the possible values of a in the range ($\min\{b,c\}$, $\max\{b,c\}$), so the overall efficiency can be improved dramatically. Note that since a^2=bc, we always have $a\in(\min\{b,c\}, \max\{b,c\})$.

Based on the discussions above, we develop a novel divide-and-conquer algorithm, named `DC-Exact`, as shown in Algorithm 2. First, it initialize a_l to the smallest ratio $\frac{1}{n}$, a_r to the largest ratio n, ρ^* to 0, and D to $\emptyset$ (line 1).

Complexity. The time complexity of `DC-Exact` is $O(k\cdot nm\log n)$, where k denotes the number of times invoking the binary search, which is at most n^2 since the binary search is invoked at most n^2 times in the worst case. Nevertheless, in practice we have $k \ll n^2$. As shown by our experiments later, k is often orders of magnitude smaller than n^2.

4.4 A Core-based Approximation Algorithm

While our exact algorithm, `DC-Exact`, is significantly faster than the state-of-the-art algorithm `Exact`, we can further speed it up by trading accuracy: we develop an efficient approximation algorithm, called `Core-Approx`, which achieves an approximation ratio of 2, within $O(\sqrt{m}(m+n))$ time. In the following, we first show the lower bound of density of an $[x, y]$-core.

Lemma 4 (Lower bound of density of $[x,y]$-core)**.** *Given a graph G and an $[x, y]$-core, denoted by H=$G[S, T]$, in G, the density of H is*

$$\rho(S,T) \geq \sqrt{xy}. \tag{5}$$

Next, we derive an upper bound of ρ^*. Before showing this, we introduce a novel concept called the maximum cn-pair.

Definition 5 (Maximum cn-pair)**.** Given a graph G=(V, E), a cn-pair $[x, y]$ is called the **maximum cn-pair**, if $x \cdot y$ achieves the maximum value among all the possible $[x, y]$-cores. We denote the maximum cn-pair by $[x^*, y^*]$.

Lemma 5 (Upper bound of ρ^*)**.** *Given a graph G=(V, E) and its maximum cn-pair $[x^*, y^*]$, the density ρ^* of the DDS is*

$$\rho^* \leq 2\sqrt{x^*y^*}. \tag{6}$$

Combining Lemmas 4 and 5, we obtain:

Theorem 2. *Given a graph G=(V, E), the core whose cn-pair is the maximum cn-pair, i.e., $[x^*, y^*]$-core, is a 2-approximation solution to the DDS problem.*

To compute the $[x^*, y^*]$-core, a straightforward method is to compute all the cores of G and then return the one with maximum core number pair. It is easy to observe that this method takes $O(n(n+m))$ time, because for each specific x, it costs $O(n+m)$ time to compute all the $[x, y]$-cores where y ranges from 0 to its maximum value. This, however, is costly and unnecessary because we only need to find the $[x^*, y^*]$-core. To boost the efficiency, we propose a more efficient algorithm which takes only $O(\sqrt{m}(n+m))$ time. Next, we introduce two concepts to facilitate the elaboration.

Definition 6 (Maximum equal cn-pair)**.** Given a graph G=(V, E), a cn-pair $[x, x]$ is the **maximum equal cn-pair**, if x achives the maximum value among all the possible $[x, x]$-cores. We denote the maximum equal cn-pair by $[\gamma, \gamma]$.

Lemma 6. *Given a graph G=(V, E) and its maximum equal cn-pair $[\gamma, \gamma]$, for any cn-pair $[x, y]$, we have either $x \leq \gamma$ or $y \leq \gamma$, or both of them.*

Definition 7 (Key cn-pair)**.** Given a graph G=(V, E) and its maximum equal cn-pair $[\gamma, \gamma]$, the cn-pair of an $[x, y]$-core is a **key cn-pair**, if one of the following conditions is satisfied:

1. if $x \leq \gamma$, $\nexists$ $[x, y']$-core in G, such that $y' > y$;
2. if $y \leq \gamma$, $\nexists$ $[x', y]$-core in G, such that $x' > x$.

Clearly, the maximum cn-pair is also a key cn-pair. We illustrate these concepts by Example 2.

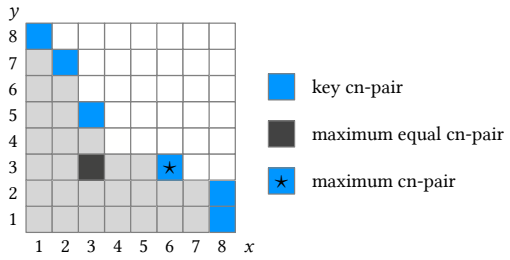


Figure 5: Illustrating the concepts of cn-pairs.

Algorithm 3: `Core-Approx`

```
Input  : G=(V, E)
Output: An approximate DDS D̃, i.e., the [x*, y*]-core
1  x* ← 0, y* ← 0;
2  [γ, γ] ← compute the [γ, γ]-core;
3  for x ← 1 to γ do
4      y ←GetMaxY(G, x);
5      if xy > x*y* then x* ← x, y* ← y ;
6  for y ← 1 to γ do
7      x ←GetMaxX(G, y);
8      if xy > x*y* then x* ← x, y* ← y ;
9  return compute the [x*, y*]-core;
10 Function GetMaxY(G, x):
11     S ← V, T ← V, y_max ← 0, y ← ⌊x*y*/x⌋ + 1;
12     if y > max_{u∈T}{d+_G(u)} then return y_max;
13     while |E| > 0 do
14         while ∃u ∈ T, d+_G(u) < y do
15             E ← E \ {(v,u)|v ∈ S}, T ← T \ {u};
16             while ∃v ∈ S, d−_G(v) < x do
17                 E ← E \ {(v,u)|u ∈ T}, S ← S \ {v};
18         if |E| > 0 then y_max ← y;
19         y ← y + 1;
20     return y_max;
21 Function GetMaxX(G, y):
22     reuse lines 11-20 by interchanging u with v, S with T,
       x with y, and changing y_max to x_max;
```

Example 2. Suppose that we have a graph whose cn-pairs are presented in Figure 5, where each colored cell (x, y) denotes the cn-pair of the $[x, y]$-core. Note that the blank cells do not correspond any $[x, y]$-cores. Then, the cn-pairs of the blue cells are key cn-pairs, in which the one with a star is the maximum cn-pair. The cn-pair of the black cell, i.e., [3, 3], is the maximum equal cn-pair.

Lemma 7. *Given a graph G=(V, E) and its maximum equal cn-pair $[\gamma, \gamma]$, we have $\gamma \leq \sqrt{m}$.*

By combining Lemmas 6 and 7, we get Lemma 8.

Lemma 8. *Given a graph G=(V, E), there are at most $2\sqrt{m}$ key cn-pairs in G.*

Based on the above discussions, we develop `Core-Approx`, which returns the $[x^*, y^*]$-core as an approximate DDS.

We further illustrate `Core-Approx` by Example 3.

Example 3. Reconsider the graph and its cn-pairs in Example 2. `Core-Approx` will run steps as follows: (1) finds the maximum equal cn-pair $[3, 3]$; (2) iterates x from 1 to 3 to compute the key cn-pairs whose first elements are x, i.e., $[1, 8]$, $[2, 7]$, and $[3, 5]$; (3) iterates y from 1 to 3 to

search the key cn-pairs whose second elements are y, i.e., $[8, 1]$, $[8, 2]$ and $[6, 3]$; and (4) returns the $[x^*, y^*]$-core, i.e., $[6, 3]$-core. □

Complexity. Computing the $[\gamma, \gamma]$-core takes $O(m + n)$ time as it iteratively peels vertices with the minimum indegrees or outdegrees. Similarly, functions `GetMaxY` and `GetMaxX` also complete in $O(m+n)$ time. Since there are at most $2\sqrt{m}$ key cn-pairs by Lemma 8, the total time cost of `Core-Approx` is bounded by $O(\sqrt{m}(n+m))$.

5. EXPERIMENTS

We now present the experimental results. We first discuss the setup in Section 5.1, then describe the results of exact and approximation algorithms in Sections 5.2 and 5.3.

5.1 Setup

Datasets. We use eight real datasets[5] (named MO, TC, OF, AD, AM, AR, BA, and TW in ascending order w.r.t. graph size) up to billion scale [16]. These graphs cover various domains, including social network (e.g., Twitter and Advogato), e-commerce (e.g., Amazon), and infrastructures (e.g., flight network).

Algorithms. In the experiments, we used our newly proposed exact algorithms `Core-Exact` and `DC-Exact`, and approximation algorithm `Core-Approx` to compute the DDS. Besides, we tested the following existing algorithms: `Exact` [14], `KS-Approx` [14], `FKS-Approx` [18], `PM-Approx` [2], and `BS-Approx` [5].

All the algorithms above are implemented in C++ with STL used. We run all the experiments on a machine having an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz processor, and 256GB memory, with Ubuntu installed.

5.2 Exact Algorithms

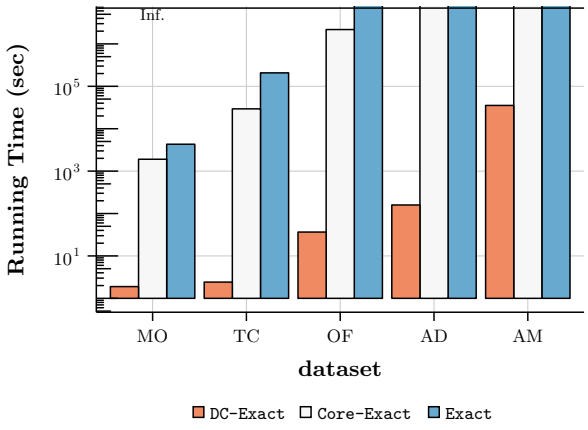


Figure 6: Efficiency of exact algorithms.

In Figure 6, we report the efficiency results of exact algorithms on the first five datasets (i.e., MO, TC, OF, AD, and AM). As these solutions cannot finish in a reasonable time on larger datasets, we do not report their results here.

[5]The datasets are available online at http://konect.uni-koblenz.de/networks/

Note that `Exact` and `Core-Exact` cannot compute the DDS within 600 hours on OF, AD, and AM.

We can observe that `Core-Exact` is at least 2× and up to 100× faster than the state-of-the-art exact algorithm `Exact`. This is mainly because `Core-Exact` locates the DDS in some $[x, y]$-cores, which are often much smaller than the entire graph, so the flow networks built on these cores become much smaller, resulting in less time cost on computing the minimum st-cut of the flow networks. Our full version [18] provides extra results about the flow network sizes during the binary search iterations.

Meanwhile, from Figure 6, we can see that `DC-Exact` is up to six and five orders of magnitude faster than `Exact` and `Core-Exact`, respectively. The main reason is that `DC-Exact` exploits a divide-and-conquer strategy, which dramatically reduces the number of $a=\frac{|S|}{|T|}$ examined. To analyze the speedup of `DC-Exact` over `Exact`, we compare the number of values of a examined in these two algorithms, which is essentially the number of times the loop of binary search is invoked. As discussed in Section 4, a is examined n^2 times in `Exact`, and k times in `DC-Exact`. Empirically, we find that k is less than 100 in the five smaller datasets. Clearly, n^2 is much larger than k. Thus, `DC-Exact` runs much faster than `Exact`.

5.3 Approximation Algorithms

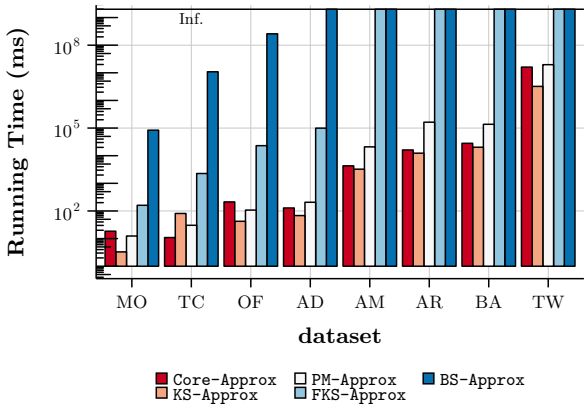


Figure 7: Efficiency of approximation algorithms.

In Figure 7, we show the running time of approximation algorithms on all the eight datasets, where bars touching the upper boundaries mean that the corresponding algorithms cannot finish within 200 hours. We can make the following observations: (1) `BS-Approx` and `FKS-Approx` are the two most inefficient algorithms, because their time complexities, i.e., $O(n^2(n+m))$ and $O(n(n+m))$, are higher than those of other algorithms. (2) `KS-Approx` is the most efficient one almost on all the datsets, since it takes only linear time cost, i.e., $O(n+m)$. However, its approximation ratio could be larger than 2, as analyzed in Section 3. (3) `Core-Approx` is the second most efficient one on almost all the datasets, followed by `PM-Approx`. (4) Among all the 2-approximation algorithms, `Core-Approx` is the fastest one, since it is up to six orders of magnitude faster than `BS-Approx`, and three orders of magnitude faster than `FKS-Approx`. Moreover, it can process billion-scale graphs. Thus, it not only obtains

high quality results, but also achieves high efficiency.

6. CONCLUSION AND FUTURE WORK

This paper studies the densest subgraph discovery on directed graphs. We show that a previous algorithm [14], which was claimed to achieve an approximation of 2, fails to satisfy the approximation guarantee. To boost the efficiency of finding DDS, we introduce a novel dense subgraph model, namely $[x, y]$-core, on directed graphs, and estalish bounds on the density of the $[x, y]$-core. We then propose a core-based exact algorithm, and further optimize it by incorporating a divide-and-conquer strategy. Besides, we find that the $[x^*, y^*]$-core, where x^*y^* is the maximum value of xy for all the $[x, y]$-cores, is a good approximation solution to the DDS problem, with theoretical guarantee. To compute the $[x^*, y^*]$-core, we develop an efficient algorithm. Extensive experiments on eight real large datasets show that both our exact and approximation algorithms are up to six orders of magnitude faster than state-of-the-art approaches.

In the future, there are several interesting research directions: (1) It would be interesting to investigate how to efficiently find the DDS with size constraints on directed graphs, such as finding a subgraph with exactly k vertices such that its density is the highest among possible subgraphs with k vertices. This problem is even more challenging than the DDS problem because of its NP-hardness. (2) In real applications, the real-world graphs often change as the time goes on, where vertices and edges may be inserted or deleted frequently. Thus, it is desirable to study how to efficiently maintain the DDS dynamically without recomputing it from scratch. (3) Recently, some researchers have developed elegant algorithms [23, 6] to decompose an undirected graph into a chain of dense subgraphs such that each one is nested within the next one and the former one has higher density than the latter one. Therefore, another exciting research direction is to perform the graph decomposition according to the directed density. (4) To the best knowledge, almost all the existing works of densest subgraph discovery model the density purely based on graph vertices and edges. However, real-world graphs are often associated with labels or attributes. As a result, it would be interesting to formulate a novel definition of graph density by considering both the links and labels or attributes, and then study how to efficiently find the densest subgraphs under this definition.

Acknowledgement

Chenhao Ma and Reynold Cheng were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 106150091), University of Hong Kong (Projects 104005858, 104005994), the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18), and the HKU-TCL Joint Research Center for Artificial Intelligence (200009430). Lakshmanan's research was supported in part by a discovery grant and a discovery accelerator supplement grant from NSERC (Canada). Wenjie Zhang was supported by PS53783, DP200101116 and DP180103096. Xuemin Lin was supported by NSFC61232006, 2018YFB1003504, U1636215, DP200101338, DP180103096, and DP170101628.

7. REFERENCES

[1] R. Albert, H. Jeong, and A.-L. Barabási. Internet: Diameter of the world-wide web. *nature*, 401(6749):130, 1999.

[2] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012.

[3] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. 2003.

[4] A. Capocci, V. D. Servedio, F. Colaiori, L. S. Buriol, D. Donato, S. Leonardi, and G. Caldarelli. Preferential attachment in the growth of social networks: The internet encyclopedia wikipedia. *Physical Review E*, 74(3):036116, 2006.

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

[6] M. Danisch, T.-H. H. Chan, and M. Sozio. Large scale density-friendly graph decomposition via convex programming. In *WWW*, pages 233–242, 2017.

[7] Y. Fang, K. Yu, R. Cheng, L. V. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *PVLDB*, 12(11):1719 – 1732, 2019.

[8] A. Gionis and C. E. Tsourakakis. Dense subgraph discovery: Kdd 2015 tutorial. In *KDD*, pages 2313–2314, 2015.

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

[10] G. Heineman, G. Pollice, and S. Selkow. Network flow algorithms. algorithms in a nutshell, 2008.

[11] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *KDD*, pages 895–904, 2016.

[12] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: understanding microblogging usage and communities. In *WebKDD*, pages 56–65, 2007.

[13] R. Kannan and V. Vinay. *Analyzing the structure of large graphs.* University of Bonn, 1999.

[14] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, pages 597–608. Springer, 2009.

[15] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.

[16] J. Kunegis. KONECT – The Koblenz Network Collection. In *WWW*, pages 1343–1350, 2013.

[17] C. Ma, R. Cheng, L. V. Lakshmanan, T. Grubenmann, Y. Fang, and X. Li. Linc: a motif counting algorithm for uncertain graphs. *PVLDB*, 13(2):155–168, 2019.

[18] C. Ma, Y. Fang, R. Cheng, L. V. Lakshmanan, W. Zhang, and X. Lin. Efficient algorithms for densest subgraph discovery on large directed graphs. In *SIGMOD*, pages 1051–1066, 2020.

[19] J. B. Orlin. Max flows in o (nm) time, or better. In *STOC*, pages 765–774, 2013.

[20] B. A. Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. Faloutsos. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In *PAKDD*, pages 435–448, 2010.

[21] S. Sawlani and J. Wang. Near-optimal fully dynamic densest subgraph. In *STOC*, pages 181–193, 2020.

[22] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

[23] N. Tatti and A. Gionis. Density-friendly graph decomposition. In *WWW*, pages 1089–1099, 2015.