Efficient Algorithms for Densest Subgraph Discovery on Large Directed Graphs

Chenhao Ma
The University of Hong Kong
chma2@cs.hku.hk

Yixiang Fang*
University of New South Wales
yixiang.fang@unsw.edu.au

Reynold Cheng
The University of Hong Kong
ckcheng@cs.hku.hk

Laks V.S. Lakshmanan
The University of British Columbia
laks@cs.ubc.ca

Wenjie Zhang
University of New South Wales
wenjie.zhang@unsw.edu.au

Xuemin Lin
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 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.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; • **Mathematics of computing → Graph algorithms**; *Network flows.*

KEYWORDS

directed graph, densest subgraph discovery

ACM Reference Format:
Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V.S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2020. Efficient Algorithms for Densest Subgraph Discovery on Large Directed Graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3389697

*Yixiang Fang is the corresponding author.

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3389697

1 INTRODUCTION

In emerging systems that manage complex relationship among objects, directed graphs are often used to model their relationships [3, 8, 30, 36]. For example, in online microblogging services (e.g., Twitter and Weibo), the "following" relationships between users can be captured as directed edges [30]. Figure 3a 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 [8]. As yet another example, the Web can also be viewed as a huge directed graph [3].

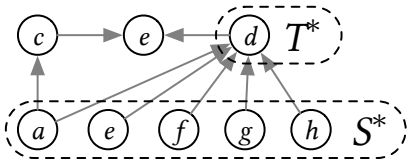


Figure 1: An example of fake follower detection.

In this paper, we study the problem of finding the densest subgraph from a directed graph G, which was first proposed in [31]. Conceptually, this *directed densest subgraph* (DDS) problem aims to find two sets of vertices, S^* and T^*, from 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 [26, 44] and community mining [33]:

• *Fake follower detection* [26, 44] aims to identify fraudulent actions in social networks [26]. Figure 1 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.

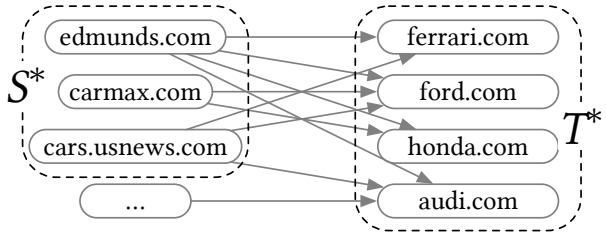
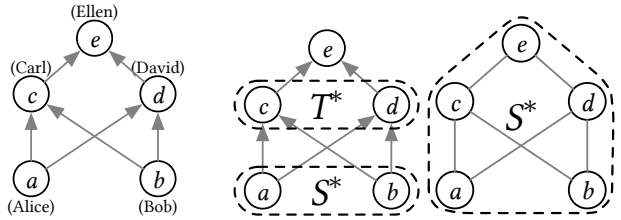


Figure 2: An example of web community.

• *Community mining* [33]. In [33], 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 2, for example, websites in S^* can be viewed as hubs providing car rankings and recommendations, while websites in T^* play the roles of authorities, as the official websites for well-known automakers.

DDS queries are also useful for *graph compression*, as discussed in [7]. Paricularly, Buehrer and Chellapilla [7] proposed to reduce the number of edges by introducing virtual nodes linking to S^* and T^* without sacrificing the connectivity of G.

Now let us give more details about the DDS query [4, 10, 31, 32][31]. 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 3a, the DDS is the subgraph induced by S^*={a, b} and T^*={c, d}, (see Figure 3b). Its density is $\rho^* = \frac{4}{\sqrt{2\times2}} = 2$. The DDS is exactly what the above applications need.



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

Figure 3: Illustrating the problem of densest subgraph discovery (or DDS problem) on the directed graph.

In undirected graphs, the density of a graph $G = (V, E)$ is defined to be $\rho(G) = \frac{|E|}{|V|}$ [24], 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 [24]. For example, for the undirected graph G in Figure 3c, 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 DDS problem returns two sets, S^* and T^*, which provide the advantage to distinguish different roles of vertices in the above applications.

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 [10, 32] and approximation algorithms [1, 4, 10, 31] have been studied. The state-of-the-art exact algorithm is a flow-based algorithm [32], 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.

To improve efficiency, approximation algorithms have been developed, the most efficient one being the algorithm in [32], 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 4.2. Here, the approximation ratio is defined as the ratio of the density of the DDS over that of the subgraph returned. This makes the algorithm proposed by Charikar in [10] be the best available 2-approximation algorithm, and its time complexity is $O(n^2(n+m))$. Clearly, it

Table 1: Summary of exact algorithms.

Algorithm	Time complexity
LP-Exact [10]	$\Omega(n^6)$
Exact [32]	$O(n^3 m \log n)$
DC-Exact **(Ours)**	$O(k \cdot nm \log n)$

Table 2: Summary of approximation algorithms.

Algorithm	Approx. ratio	Time complexity
KV-Approx [31]	$O(\log n)$	$O(s^3 n)$
PM-Approx [4]	$2\delta(1+\epsilon)$	$O(\frac{\log n}{\log \delta} \log_{1+\epsilon} n(n+m))$
KS-Approx [32]	>2	$O(n+m)$
BS-Approx [10]	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.

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 [32], 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 [47] 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 the $[x, y]$-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, $k \leq n^2$. 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 can scale well on billion-scale graphs, and is also up to six orders of magnitude faster than the existing 2-approximation algorithm.

Outline. The rest of the paper is organized as follows. We review the related work in Section 2. In Section 3, we formally present the DDS problem. Section 4 reviews the state-of-the-art algorithms and discusses their limitations. We present our exact algorithms in Section 5 and approximation algorithm in Section 6. Experimental results are presented in Section 7. We conclude this paper in Section 8.

2 RELATED WORK

The densest subgraph can be regarded as one type of cohesive subgraphs. Other related topics contain k-core [47, 52], k-truss [12, 29], cliques and motifs [27, 37], as well as community search [13–19, 28, 51] based on k-core and k-truss. In the following, we focus on two groups of work on densest subgraph discovery on undirected graphs [24] and directed graphs [31] respectively.

Densest subgraph discovery on undirected graphs. In [24], Goldberg introduced the densest subgraph discovery problem (DS problem), which aims to find the subgraph whose edge-density is the highest among all the subgraphs where the edge-density of a graph G=(V, E) is defined as $\frac{|E|}{|V|}$, and proposed a max-flow-based algorithm to compute the exact densest subgraph. Tsourakakis [49] and Mitzenmacher et al. [39] generalized the above edge-density as clique-density and developed efficient exact algorithms for finding the corresponding DS. Recently, Fang et al. [20] have proposed efficient DS algorithms by exploiting k-cores which are able to find the densest subgraphs for a wide range of graph density definitions such as edge-density, clique-density, and pattern-density. Generally, the exact DS algorithms [20, 24, 49] work well on small or moderate-size graphs, but they are inefficient for processing large graphs as shown in [20]. To remedy this issue, several efficient approximation algorithms have been developed. In [10], Charikar developed a 2-approximation algorithm which takes linear time cost. Fang et al. [20] improved the algorithm by exploiting k-cores. In [4], Bahmani et al. designed a parameterized approximation algorithm, which achives an approximation of $2(1+\epsilon)$ where $\epsilon > 0$. Besides, many variants of the DS problem have been studied. In [6], Bhaskara et al. studied the densest k-subgraph problem, where a k-subgraph means a subgraph consisting of k vertices. Qin et al. [45] developed solutions for finding the top-k locally densest subgraphs. In [39], Mitzenmacher et al. studied the (p, q)-biclique densest subgraph problem on bipartite graphs. Tsourakakis et al. [50]

developed algorithms for discovering quasi-cliques with quality guarantees. Recently, Tatti and Gionis [48] and Danisch et al. [11] have studied the topic of density-friendly graph decomposition, which decomposes a graph into a chain of subgraphs, where each subgraph is nested within the next one and the inner one is denser than the outer ones.

Densest subgraph discovery on directed graphs. Kannan and Vinay [31] were the first to introduce the notion of density and the problem of the densest subgraph on directed graphs (DDS problem). In [10], Charikar developed an exact algorithm for this problem, which completes in polynomial time cost by solving $O(n^2)$ linear programs. Later in [32], Khuller and Saha proposed a flow-based algorithm, which also takes polynomial time cost. Table 1 summarizes the time complexities of these exact algorithms. Nevertheless, all these exact algorithms are computationally expensive for large graphs, so researchers have turned to develop efficient approximation algorithms. In [31], Kannan and Vinay proposed an $O(\log n)$-approximation algorithm to compute the densest subgraph. In [10], Charikar designed a 2-approximation algorithm with a time complexity of $O(n^2 \cdot (n+m))$. In [32], Khuller and Saha presented a linear approximation algorithm, and claimed that it achieves an approximation of 2. Unfortunately, as we shall show in Section 4.2 (Example 4.1), the claim is incorrect. In [4], Bahmani et al. developed a $2(1+\epsilon)$-approximation algorithm based on the streaming model ($\epsilon > 0$). Table 2 summarizes the approximation ratios and time complexities of these approximation algorithms.

3 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 [31] 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 3.1 (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

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

Definition 3.2 (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.

Table 3: Notations and meanings.

Notation	Meaning
G=(V, E)	a directed graph with vertex set V and edge set E
n, m	$n = \|V\|$, $m = \|E\|$
H=$G[S, T]$	the subgraph induced by S and T in G
$E(S, T)$	the edges induced by S and T in G
$d^-_G(v)$, $d^+_G(v)$	the outdegree and indegree of a vertex $v \in G$ resp.
$\rho(S, T)$	the density of the (S, T)-induced subgraph
$D = G[S^*, T^*]$	the densest subgraph D in G
ρ^*	$\rho^* = \max_{S,T \subseteq V}\{\rho(S, T)\} = \rho(S^*, T^*)$
$\widetilde{D} = G[\widetilde{S^*}, \widetilde{T^*}]$	the approximate densest subgraph in G
$\widetilde{\rho^*} = \rho(\widetilde{S^*}, \widetilde{T^*})$	the density of $\widetilde{D}$
$F = (V_F, E_F)$	a flow network with node set V_F and edge set E_F

***Problem* 1** (DDS problem [4, 10, 23, 31, 32])**:** Given a directed graph G=(V, E), return a DDS[1] D=$G[S^*, T^*]$ of G.

Example 3.3. Consider the directed graph in Figure 3a. Its DDS D=$G[S^*, T^*]$ is the subgraph highlighted in Figure 3b, where S^*=$\{a, b\}$and T^*=$\{c, d\}$, since its density $\rho(S^*, T^*)$=$\frac{4}{\sqrt{2\times2}}$ =2 is higher than the density of any other (S, T)-induced subgraphs. For instance, if we let S=V and T=V, then we get a (V, V)-induced subgraph H=$G[V, V]$, and its density is $\rho(V, V)$=$\frac{6}{\sqrt{5\times5}}$=$\frac{6}{5}$, which is less than 2. □

4 EXISTING ALGORITHMS

In this section, we review the state-of-the-art exact algorithm [32] and approximation algorithms [10, 32] for the DDS problem. We remark that for approximation algorithms, both the algorithms in [32] and [10] 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 illusrated 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.

4.1 The Exact Algorithm

The state-of-the-art exact algorithm [32] computes the DDS by solving a maximum flow problem, which generally follows the same paradigm of the exact algorithm [24] of finding the densest subgraphs on undirected graphs. We denote this algorithm by `Exact`. A flow network [25] 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

[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.

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 $t \in \mathcal{T}$.

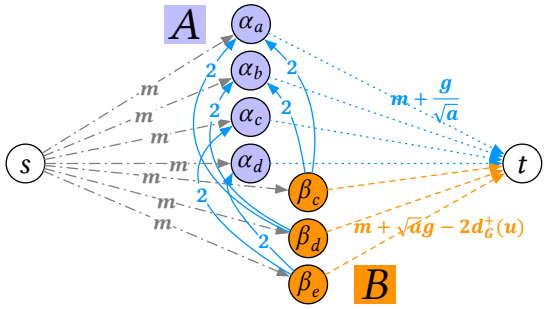


Figure 4: Illustrating the flow network.

Algorithm 1 presents Exact. It first enumerates all the possible values of $a = \frac{|S|}{|T|}$ (line 2). Then, for each a, it guesses the value g of the maximum density via a binary search (lines 2-5). 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 $\langle \mathcal{S}, \mathcal{T} \rangle$ (lines 6-11). 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^* (line 11). Note A and B are two node sets contained in V_F (cf. line 15 and Figure 4). To build the flow network, it first creates a set V_F of nodes (lines 14-15), and then adds directed edges with different capacities between these nodes (lines 16-20). For example, for the direct graph depicted in Figure 3a, we can build a flow network as shown in Figure 4.

Limitations. In Algorithm 1, 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 [43]. Consequently, the total time complexity of Exact is $O(n^3 m \log n)$, which is thus very inefficient on even small graphs. For example, our later experiments show that Exact takes more than 2 days to find the DDS on a graph with ~1,200 vertices and ~2,600 edges. The sources of inefficiency are three-fold: First, it needs to check all the n^2 values of a, which is very costly. Second, the flow network F is always built on the entire graph in each iteration, while the DDS is often a small subgraph of G. Third, the initial lower and upper bounds of ρ^* are not very tight. Therefore, there is room for improving its efficiency.

4.2 Approximation Algorithms

The state-of-the-art approximation algorithm KS-Approx [32] 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. Algorithm 2 outlines these steps.

Algorithm 1: Exact [32]

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

1 $\rho^* \leftarrow 0$;
2 **foreach** $a \in \{\frac{n_1}{n_2} \mid 0 < n_1, n_2 <= n\}$ **do**
3 $\quad l \leftarrow 0, r \leftarrow \max_{u \in V}\{d_G^+(u), d_G^-(u)\}$;
4 $\quad$ **while** $r - l \geq \frac{\sqrt{n}-\sqrt{n-1}}{n\sqrt{n-1}}$ **do**
5 $\quad\quad g \leftarrow \frac{l+r}{2}$;
6 $\quad\quad F=(V_F, E_F) \leftarrow$ BuildFlowNetwok(G, a, g);
7 $\quad\quad \langle \mathcal{S}, \mathcal{T} \rangle \leftarrow$ Min-ST-Cut(F);
8 $\quad\quad$ **if** $\mathcal{S}=\{s\}$ **then** $r \leftarrow g$;
9 $\quad\quad$ **else**
10 $\quad\quad\quad l \leftarrow g$;
11 $\quad\quad\quad$ **if** $g > \rho^*$ **then** $D \leftarrow G[\mathcal{S} \cap A, \mathcal{S} \cap B]$, $\rho^* = g$;
12 **return** D;
13 **Function** BuildFlowNetwok$(G = (V, E), a, g)$:
14 $\quad A \leftarrow \{\alpha_u | u \in V\}$, $B \leftarrow \{\beta_u | u \in V\}$, $E_F \leftarrow \emptyset$;
15 $\quad V_F \leftarrow \{s\} \cup A \cup B \cup \{t\}$;
16 $\quad$ **for** $\alpha_u \in A$ **do** add (s, α_u) to E_F with capacity m;
17 $\quad$ **for** $\beta_u \in B$ **do** add (s, β_u) to E_F with capacity m;
18 $\quad$ **for** $\alpha_u \in A$ **do** add (α_u, t) to E_F with capacity $m + \frac{g}{\sqrt{a}}$;
19 $\quad$ **for** $\beta_u \in B$ **do** add (β_u, t) to E_F with capacity $m + \sqrt{a}g - 2d_G^+(u)$;
20 $\quad$ **for** $(u, v) \in E$ **do** add (β_v, α_u) to E_F with capacity 2;
21 $\quad$ **return** $F = (V_F, E_F)$

Algorithm 2: KS-Approx [32]

Input : $G=(V, E)$
Output: An approximate DDS $\widetilde{D}$

1 $\widetilde{\rho}^* \leftarrow 0, \widetilde{D} \leftarrow \emptyset, S \leftarrow V, T \leftarrow V$;
2 **while** $|E|>0$ **do**
3 $\quad$ **if** $\rho(S, T) > \widetilde{\rho}^*$ **then**
4 $\quad\quad \widetilde{\rho}^* \leftarrow \rho(S, T), \widetilde{D} \leftarrow (S, T)$;
5 $\quad u_+ \leftarrow \arg\min_u d_G^+(u), u_- \leftarrow \arg\min_u d_G^-(u)$;
6 $\quad$ **if** $d_G^+(u_+) \leq d_G^-(u_-)$ **then**
7 $\quad\quad E \leftarrow E \setminus \{(v, u_+) | v \in S\}, T \leftarrow T \setminus \{u_+\}$;
8 $\quad$ **else**
9 $\quad\quad E \leftarrow E \setminus \{(u_-, v) | v \in T\}, S \leftarrow S \setminus \{u_-\}$;
10 **return** $\widetilde{D}$;

It was claimed in [32] that KS-Approx achieves an approximation ratio of 2. Unfortunately, as shown in the following

counter-example, their claim is incorrect [3]. Specifically, Example 4.1 shows that KS-Approx may report results whose approximation ratios are larger (i.e., worse) than 2.

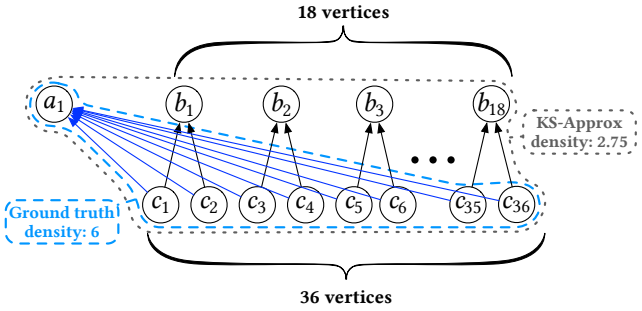


Figure 5: A counter-example for KS-Approx.

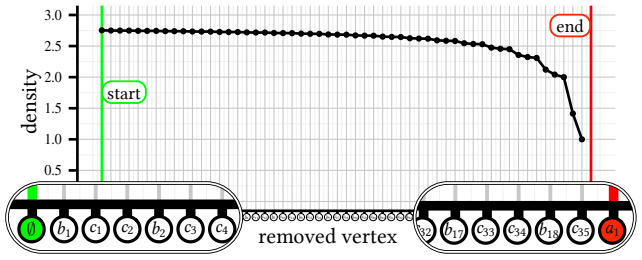


Figure 6: Running steps by KS-Approx.

Example 4.1. In Figure 5, the graph has 3 sets of vertices, i.e., $\{a_1\}$, $\{b_i|1 \le i \le 18\}$, and $\{c_i|1 \le i \le 36\}$. Note that a_1 has 36 incoming edges from $c_1, c_2, \cdots, c_{36}$. For each vertex b_i ($i \in [1, 18]$), it has 2 incoming edges from c_{2i-1} and c_{2i}. The exact DDS is the subgraph induced by $a_1, c_1, c_2, \cdots, c_{36}$, and its density is 6.

Figure 6 provides a step-by-step breakdown for KS-Approx on the graph in Figure 5. In the beginning, $S = \{c_i|1 \le i \le 36\}$ and $T = \{a_1\} \cup \{b_i|1 \le i \le 18\}$ (the vertices with no outgoing edge in S and no incoming edge in T are eliminated for simplicity). $\rho(S,T) = 2.7530$. Then, b_1 is removed from T based on the condition in Line 6 of Algorithm 2, and $\rho(S,T) = 2.7499$. Next c_1 is removed from S, and $\rho(S,T)$ becomes 2.7490. c_2 is removed from S afterward, and $\rho(S,T)$ becomes 2.7487. Along with more vertices deleted, the density $\rho(S,T)$ declines gradually. After a_1 is removed from T, the algorithm ends as T becomes empty, and no edge is left in the graph. Thus, KS-Approx will return the whole graph as the approximate DDS, whose density is 2.75. Hence, the actual approximation ratio is $\frac{6}{2.75} > 2$, which contradicts the claim that it is a 2-approximation algorithm. □

[3]The authors of [32] have confirmed that the approximation ratio of KS-Approx was misclaimed.

Why KS-Approx fails? KS-Approx is supported by Theorem 2 in [32]. Theorem 2 requires that there is a iteration that $\forall u \in S, d^-_{G[S,T]}(u) \ge \lambda_o = |E(S^*,T^*)| \cdot \left(1 - \sqrt{1 - \frac{1}{|S^*|}}\right)$ and $\forall v \in T, d^+_{G[S,T]}(v) \ge \lambda_i = |E(S^*,T^*)| \cdot \left(1 - \sqrt{1 - \frac{1}{|T^*|}}\right)$. In Example 4.1, $S^* = \{c_i|1 \le i \le 36\}$ and $T^* = \{a_1\}$. Thus, $\lambda_o = 0.5035$ and $\lambda_i = 36$. By reviewing the iterations of KS-Approx over the counter-example, we can find such condition cannot be guaranteed simultaneously. During our recent communication, they have proposed a fix (called FKS-Approx) which is a correct 2-approximation algorithm, but costs $O(n \cdot (n+m))$ time. The details about FKS-Approx is provided in [1].

Since KS-Approx is not a 2-approximation algorithm, the most accurate published approximation algorithm is BS-Approx [10], which is able to correctly find a 2-approximation result. We outline its steps in Algorithm 3. Similar to Exact, BS-Approx enumerates all the possible values of $a = \frac{|S|}{|T|}$ (line 2), and for each specific a, it iteratively removes the vertex with the minimum degree from S or T based on a predefined condition (line 8), and then updates S and T, as well as the approximate DDS $\widetilde{D}$ (lines 4-9).

Algorithm 3: BS-Approx [10]

Input : $G=(V, E)$
Output: An approximate DDS $\widetilde{D}$

1 $\widetilde{\rho^*} \leftarrow 0, \widetilde{D} \leftarrow \emptyset$;
2 **foreach** $a \in \{\frac{n_1}{n_2} | 0 < n_1, n_2 <= n\}$ **do**
3 $S \leftarrow V, T \leftarrow V$;
4 **while** $S \neq \emptyset \wedge T \neq \emptyset$ **do**
5 **if** $\rho(S,T) > \widetilde{\rho^*}$ **then** $\widetilde{D} \leftarrow G[S,T]$, $\widetilde{\rho^*} \leftarrow \rho(S,T)$;
6 $u \leftarrow \arg\min_{u \in S} d^-_G(u)$;
7 $v \leftarrow \arg\min_{v \in T} d^+_G(v)$;
8 **if** $\sqrt{a} \cdot d^-_G(u) \le \frac{1}{\sqrt{a}} \cdot d^+_G(v)$ **then** $S \leftarrow S \setminus \{u\}$;
9 **else** $T \leftarrow T \setminus \{v\}$;
10 **return** $\widetilde{D}$;

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.

5 EXACT ALGORITHMS

In this section, we develop novel efficient exact 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 [47] for directed graphs. In the following, we first introduce the $[x, y]$-core, then present our core-based exact algorithm, and further optimize it by exploiting a divide-and-conquer strategy.

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

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

Definition 5.1 (k-core [5, 47]). 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 $\forall v \in \mathcal{H}_k$, $deg_{\mathcal{H}_k}(v) \geq k$.

A k-core has some interesting properties [5]: (1) k-cores are "nested": given two nonnegative integers i and j, if $i < j$, then $\mathcal{H}_j \subseteq \mathcal{H}_i$; (2) a k-core may not be connected; and (3) computing all the k-cores of a graph, known as k-core decomposition, can be done in linear time [5].

Definition 5.2 ($[x, y]$-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) $\forall u \in S, d_H^- \geq x$ and $\forall v \in T, d_H^+ \geq y$;
(2) $\nexists H'$, s.t. $H \subset H'$ and H' satisfies (1).

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

Example 5.3. The subgraph induced by (S^*, T^*), i.e., $D = G[S^*, T^*]$ in Figure 3b 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 5.2.

Lemma 5.4 (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 [5]; 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 [5].

Remark. In [22], Giatsidis et al. introduced another kind of core model for a directed graph G=(V, E), called (k, l)-core, which is the largest subgraph of G such that each vertex's indegree and outdegree are at least k and l respectively. This is different with our core model because our $[x, y]$-core is an (S, T)-induced subgraph such that each vertex of S has an indegree of x and each vertex of T has an outdegree of y. Moreover, S and T are not necessarily disjoint. When $S = T$, our $[x, y]$-core is reduced to the (k, l)-core by letting $k = x$ and $l = y$. Hence, the $[x, y]$-core naturally generalizes the (k, l)-core.

5.2 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 5.5. *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 5.6. *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 $a = \frac{|S^*|}{|T^*|}$.*

Proof. By Lemma 5.5, removing any single vertex u from S^* will result in the removal of $\lceil\frac{\rho^*}{2\sqrt{a}}\rceil$ edges from D, so we conclude that for each vertx $u \in S^*$, $d_D^-(u) \geq \lceil\frac{\rho^*}{2\sqrt{a}}\rceil$. Similarly, for each vertex $v \in T^*$, we have $d_D^+(v) \geq \lceil\frac{\sqrt{a}\rho^*}{2}\rceil$. Thus, by the definition of $[x, y]$-core, we conclude that the DDS is 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.

Moreover, during the binary search, the lower bound l of ρ^* is gradually enlarged, so we can further locate the DDS in the $[x, y]$-cores with larger values of cn-pairs. As the values of cn-pairs increase, the sizes of $[x, y]$-cores become smaller, so the flow networks built become smaller gradually and the cost of computing the minimum st-cut is greatly reduced.

Algorithm 4: `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 4. Specifically, it first runs a 2-approximation algorithm[4] and initializes ρ^* as the density of the approximate DDS (lines 1-2). Then, for each value of a, it sets the lower and upper bounds of ρ^* using $\tilde{\rho}^*$ (lines 3-4). After that, it performs binary search to compute the DDS, where the flow networks are built based on the $[x, y]$-cores (lines 5-13).

Analysis. To compute a specific $[x, y]$-core, we can complete in $O(n + m)$ time by using the idea of k-core decomposition [5]. 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]Note that any 2-approximation algorithm can be applied here; in this paper, we use our core-based approximation algorithm developed in Section 6.

5.3 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 4-11 in Algorithm 1) actually have provided insights for reducing the number of tries of a. As shown in [32], 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 5.7. *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 $\rho(S, T)\le\rho(S', T')$.*

Proof. We first introduce more details regarding Equation (2), and then 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 [32]:

$$\frac{|S'|}{\sqrt{a}}\left(g^*(a)-\frac{|E(S',T')|}{|S'|/\sqrt{a}}\right)+|T'|\sqrt{a}\left(g^*(a)-\frac{|E(S',T')|}{|T'|\sqrt{a}}\right)=0$$

$$\sqrt{|S'||T'|}\frac{\sqrt{b}}{\sqrt{a}}\left(g^*(a)-\frac{\rho(S',T')}{\sqrt{b}/\sqrt{a}}\right)+\sqrt{|S'||T'|}\frac{\sqrt{a}}{\sqrt{b}}\left(g^*(a)-\frac{\rho(S',T')}{\sqrt{a}/\sqrt{b}}\right)=0$$

$$\sqrt{|S'||T'|}\left(\left(\frac{\sqrt{b}}{\sqrt{a}}+\frac{\sqrt{a}}{\sqrt{b}}\right)g^*(a)-2\rho(S',T')\right)=0 \tag{3}$$

Then, $g^*(a)$ can be written as

$$g^*(a)=\frac{2\rho(S',T')}{\frac{\sqrt{b}}{\sqrt{a}}+\frac{\sqrt{a}}{\sqrt{b}}}. \tag{4}$$

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

$$h'_a(x) = \frac{-a+x}{2\sqrt{a}x^{3/2}},\ x > 0. \quad (5)$$

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 5.7, 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 $a \in (\min\{b, c\}, \max\{b, c\})$.

To further illustrate the prunning effectiveness of Lemma 5.7, we conduct an experiment on a small real graph and discuss the results in Example 5.8.

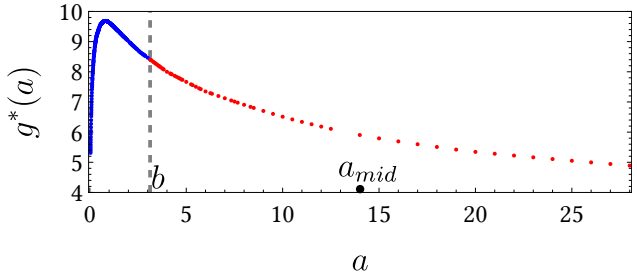


Figure 7: The prunning effectiveness of Lemma 5.7.

Example 5.8. We consider a small dataset MA [46] which consists of 28 vertices and 217 edges; this implies that the values of a are in the range $[\frac{1}{28}, 28]$. We plot the values of $g^*(a)$ for $a \in [\frac{1}{28}, 28]$ in Figure 7. Let a_{mid}=$(\frac{1}{28}$+28)/2. After applying the binary search for a_{mid}, we get a=14.02, b=3.125, and c=62.88, by Lemma 5.7. Therefore, we can skip the binary search for all the 78 values of $a \in (3.125, 62.88)$, which are marked in red in Figure 7. □

Based on the discussions above, we develop a novel divide-and-conquer algorithm, named DC-Exact, as shown in Algorithm 5. 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).

Algorithm 5: DC-Exact

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

1 $a_l \leftarrow \frac{1}{n}, a_r \leftarrow n, \rho^* \leftarrow 0, D \leftarrow \emptyset$;
2 Divide-Conquer(a_l, a_r);
3 **return** D;
4 **Function** Divide-Conquer(a_l, a_r):
5 $\quad a_{mid} \leftarrow \frac{a_l+a_r}{2}$;
6 $\quad$ run Lines 4-13 of Algorithm 4 (replace $x \leftarrow \lceil \frac{l}{2\sqrt{a}} \rceil, y \leftarrow \lceil \frac{\sqrt{a}l}{2} \rceil$ with $x \leftarrow \lceil \frac{l}{2\sqrt{a_r}} \rceil$, $y \leftarrow \lceil \frac{\sqrt{a_r}l}{2} \rceil$);
7 $\quad$ let $G[S', T']$ be the DDS found by binary search;
8 $\quad b \leftarrow \frac{|S'|}{|T'|}$;
9 $\quad c \leftarrow \frac{a_{mid}^2}{b}$;
10 $\quad$ **if** $b > c$ **then** Swap(b, c);
11 $\quad$ **if** $a_l \le b$ **then** Divide-Conquer(a_l, b);
12 $\quad$ **if** $c \le a_r$ **then** Divide-Conquer(c, a_r);

Then, it applies (line 2) the function Divide-Conquer to check the possible values of a (Lines 4-12). Specifically, in Divide-Conquer, it first picks the middle point a_{mid} in range $[a_l, a_r]$ (line 5). Then, it uses the binary search process (similar to the one in Core-Exact) to find the (S', T')-induced subgraph $G[S', T']$ which maximizes Equation (2) (line 6). Afterwards, it computes b and c according to Lemma 5.7 (lines 8-10). Finally, it skips the whole range (b, c) of the value of a, and conducts search on the two interals which are split by (b, c) recursively to compute the DDS. Note that to exploit the $[x, y]$-cores for improving the efficiency, we build the flow networks on the union of $[x, y]$-cores for all possible values of a in the range $[a_l, a_r]$, i.e., the $\left[\lceil \frac{l}{2\sqrt{a_r}} \rceil, \lceil \frac{\sqrt{a_l}l}{2} \rceil\right]$-core (line 6).

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.

6 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 6.1 (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*

$$\rho(S, T) \geq \sqrt{xy}. \tag{6}$$

Proof. For each vertex $u \in S$, since it participates at least x edges in H, we have $|E(S, T)| \geq x|S|$. Similarly, we can obtain $|E(S, T)| \geq y|T|$. Thus, we conclude

$$\rho(S, T) = \frac{|E(S, T)|}{\sqrt{|S||T|}} = \sqrt{\frac{|E(S, T)|^2}{|S||T|}} \geq \sqrt{\frac{x|S| \cdot y|T|}{|S||T|}} = \sqrt{xy}.$$

Thus, Lemma 6.1 holds. □

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

Definition 6.2 (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 6.3 (Upper bound of ρ^*). *Given a graph G=(V, E) and its maximum cn-pair $[x^*, y^*]$, the density ρ^* of the DDS is*

$$\rho^* \leq 2\sqrt{x^* y^*}. \tag{7}$$

Proof. We prove the lemma by contradiction. Assume that $\rho^* > 2\sqrt{x^* y^*}$. Let $a^* = \frac{|S^*|}{|T^*|}$. Then, by Theorem 5.6, we conclude that the DDS is in the $[x', y']$-core, where $x' > \frac{\sqrt{x^* y^*}}{\sqrt{a^*}}$ and $y' > \sqrt{a^*}\sqrt{x^* y^*}$, so $x'y' > x^* y^*$, which contradicts the fact that $[x^*, y^*]$ is the maximum cn-pair of G. Therefore, ρ^* is at most $2\sqrt{x^* y^*}$. □

Theorem 6.4. *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.*

Proof. Let the $[x^*, y^*]$-core be an (S, T)-induced subgraph. By Lemma 6.1, we have $\rho(S, T) \geq \sqrt{x^* y^*}$. According to Lemma 6.3, we have $\rho^* \leq 2\sqrt{x^* y^*}$, so we conclude

$$\frac{\rho^*}{\rho(S, T)} \leq \frac{2\sqrt{x^* y^*}}{\sqrt{x^* y^*}} = 2. \tag{8}$$

Hence, the theorem holds. □

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 6.5 (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 $[\gamma, \gamma]$.

Remarks. The approximation algorithm `KS-Approx` [32] actually returns the $[\gamma, \gamma]$-core in the graph. However, the $[\gamma, \gamma]$-core may not be necessarily the $[x^*, y^*]$-core; hence, it is not guaranteed to be the 2-approximation DDS.

Lemma 6.6. *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.*

Proof. We prove this lemma by contradiction. Assume there is a cn-pair $[x, y]$ where $x > \gamma$ and $y > \gamma$. Then, let $\gamma' = \min\{x, y\} > \gamma$, so there exists a $[\gamma', \gamma']$-core in G, which contradicts $[\gamma, \gamma]$ is the maximum equal cn-pair. □

Definition 6.7 (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 $x \leq \gamma$, there does not exist any $[x, y']$-core in G, such that $y' > y$;
(2) if $y \leq \gamma$, there does not exist any $[x', y]$-core in G, such that $x' > x$.

Clearly, the maximum cn-pair is also a key cn-pair. We illustrate these concepts by Example 6.8.

Example 6.8. Suppose that we have a graph whose cn-pairs are presented in Figure 8, 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.

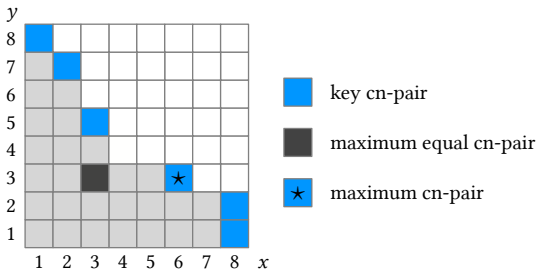


Figure 8: Illustrating the concepts of cn-pairs.

Lemma 6.9. *Given a graph G=(V, E) and its maximum equal cn-pair $[\gamma, \gamma]$, we have $\gamma \leq \sqrt{m}$.*

Algorithm 6: Core-Approx

Input : G=(V, E)
Output: An approximate DDS $\widetilde{D}$, i.e., the $[x^*, y^*]$-core

1 $x^* \leftarrow 0, y^* \leftarrow 0$;
2 $[\gamma, \gamma] \leftarrow$ compute the $[\gamma, \gamma]$-core by iteratively peeling vertices which have the minimum indegrees or outdegrees;
3 **for** $x \leftarrow 1$ **to** γ **do**
4 $\quad y \leftarrow$GetMaxY(G, x);
5 $\quad$ **if** $xy > x^*y^*$ **then** $x^* \leftarrow x, y^* \leftarrow y$;
6 **for** $y \leftarrow 1$ **to** γ **do**
7 $\quad x \leftarrow$GetMaxX(G, y);
8 $\quad$ **if** $xy > x^*y^*$ **then** $x^* \leftarrow x, y^* \leftarrow y$;
9 **return** compute the $[x^*, y^*]$-core;
10 **Function** GetMaxY(G, x)**:**
11 $\quad S \leftarrow V, T \leftarrow V, y_{\max} \leftarrow 0, y \leftarrow \lfloor \frac{x^*y^*}{x} \rfloor + 1$;
12 $\quad$ **if** $y > \max_{u \in T}\{d^+_G(u)\}$ **then return** $y_{\max}$;
13 $\quad$ **while** $|E| > 0$ **do**
14 $\quad\quad$ **while** $\exists u \in T, d^+_G(u) < y$ **do**
15 $\quad\quad\quad E \leftarrow E \setminus \{(v,u) | v \in S\}, T \leftarrow T \setminus \{u\}$;
16 $\quad\quad\quad$ **while** $\exists v \in S, d^-_G(v) < x$ **do**
17 $\quad\quad\quad\quad E \leftarrow E \setminus \{(v,u) | u \in T\}, S \leftarrow S \setminus \{v\}$;
18 $\quad\quad$ **if** $|E| > 0$ **then** $y_{\max} \leftarrow y$;
19 $\quad\quad y \leftarrow y + 1$;
20 $\quad$ **return** $y_{\max}$;
21 **Function** GetMaxX(G, y)**:**
22 $\quad$ reuse lines 11-20 by interchanging u with v, S with T, x with y, and changing $y_{\max}$ to $x_{\max}$;

Proof. A $[\gamma, \gamma]$-core contains at least γ vertices whose out-degrees are at least γ. Meanwhile, there are at most m edges in the $[\gamma, \gamma]$-core. Hence, $\gamma \cdot \gamma \leq m$. □

By combining Lemmas 6.9 and 6.6, we get Lemma 6.10.

Lemma 6.10. *Given a graph G=(V, E), there are at most* $2\sqrt{m}$ *key cn-pairs in G.*

Proof. According to Lemma 6.6 and Definition 6.7, there are at most 2γ key cn-pairs in G. Since we have $\gamma \leq \sqrt{m}$ by Lemma 6.9, 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. Specifically, we first compute the maximum equal cn-pair, then enumerate all the key cn-pairs, and finally return the core with the maximum cn-pair. Algorithm 6 presents Core-Approx. First, it obtains the maximum equal cn-pair (line 2). Then, it enumerates x and y from 1 to γ to search all the key cn-pairs (lines 3-8). Finally, the $[x^*, y^*]$-core is returned.

Given an input x, the function GetMaxY computes the key cn-pair whose first element is x. In GetMaxY, we first initialize S, T, $y_{\max}$, and y, where y is set to $\lfloor \frac{x^*y^*}{x} \rfloor + 1$. Then, in the loop, if there is a vertex $u \in T$ with in-degree less than y, we remove it (lines 14-15); this may make some vertices' out-degrees become less than x, so we have to remove these vertices as well (lines 16-17). After that, we update $y_{\max}$ and increase y by 1 (lines 18-19). Finally, we get the maximum value of y. Similarly, we have a function GetMaxX to get the key cn-pair whose second element is a given y.

We further illustrate Core-Approx by Example 6.11.

Example 6.11. Reconsider the graph and its cn-pairs in Example 6.8. 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 in-degrees 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 6.10, the total time cost of Core-Approx is bounded by $O(\sqrt{m}(n + m))$.

7 EXPERIMENTS

We now present the experimental results. We first discuss the setup in Section 7.1, then describe the results of exact and approximation algorithms in Sections 7.2-7.4.

7.1 Setup

Datasets. We use eight real datasets[5] [34], and report the numbers of vertices and edges of each dataset in Table 4. These graphs cover various domains, including social network (e.g., Twitter and Advogato), e-commerce (e.g., Amazon), and infrastructures (e.g., flight network).

Table 4: Datasets used in our experiments.

Dataset	Category	$\|V\|$	$\|E\|$
MO [21]	Human Social	217	2,672
TC [2]	Infrastructure	1,226	2,615
OF [42]	Infrastructure	2,939	30,501
AD [38]	Social	6,541	51,127
AM [35]	E-commerce	403,394	3,387,388
AR [40]	E-commerce	3,376,972	5,838,041
BA [41]	Hyperlink	2,141,300	17,794,839
TW [9]	Social	52,579,682	1,963,263,821

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:

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

- `Exact` [32] is the state-of-the-art exact algorithm, which is also recapped in Section 4.1.
- `KS-Approx` [32] is an approximation algorithm whose appproximation ratio was misclaimed, which is also recapped in Section 4.2.
- `FKS-Approx` [1] is the fixed version provided by the authors of [32]. Its time complexity is $O(n \cdot (n+m))$, with an approximation ratio of 2.
- `PM-Approx` [4] is a parameterized approximation algorithm. Note that we use its default parameters in [4] in our experiments (δ=2, ϵ=1).
- `BS-Approx` [10] is a 2-approximation algorithm.

All the algorithms above are implemented in C++ with STL support. 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.

7.2 Exact Algorithms

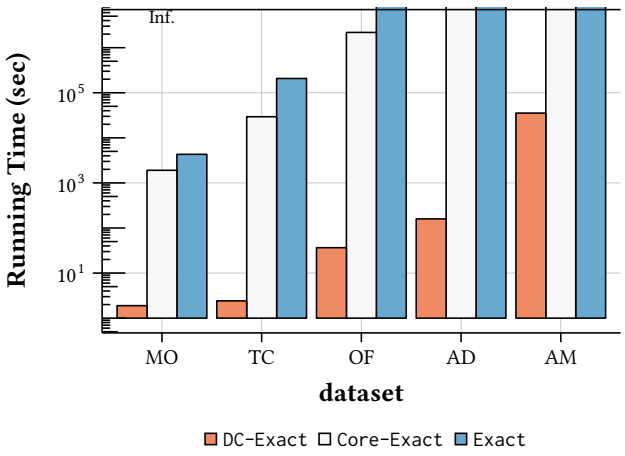


Figure 9: Efficiency of exact algorithms.

In Figure 9, 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. 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. We further investigate how the flow network size (number of edges) changes in the first ten iterations of the binary search in exact algorithms. Figure 10 reports the flow network size of these three algorithms on two datasets, i.e., AD and AM. Clearly, we can observe that the size of the flow network in `Core-Exact` is reduced significantly as the iteration goes on, while the flow network size of `Exact` does not change during these iterations. Thus, we conclude that the $[x, y]$-cores are indeed effectivce for locating the DDS in some smaller subgraphs, allowing the exact DDS to be computed more efficiently. The sizes of flow networks created in `DC-Exact` are larger than those in `Core-Exact` because, in `DC-Exact`, the flow network is built based on the union of $[x, y]$-cores for all possible values of a in the range of $[a_l, a_r]$.

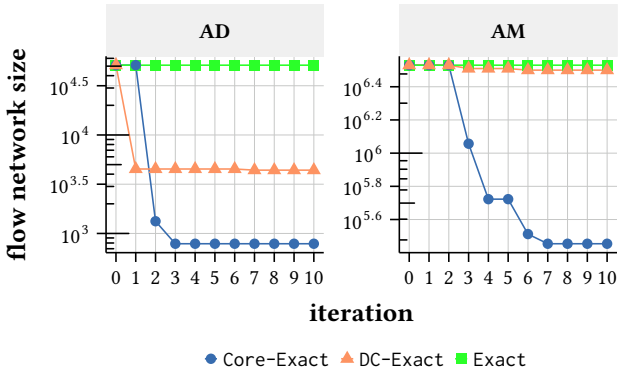


Figure 10: Flow network sizes in exact algorithms.

Meanwhile, from Figure 9, 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, as illustrated in Figure 7. To further analyze the speedup of `DC-Exact` over `Exact`, we report the numbers of values of a examined in `DC-Exact` and `Exact`, which equal to the numbers of times of inovking the loop of binary search. As discussed in Section 5, the total numbers of values of a examined in `Exact` and `DC-Exact` are n^2 and k, respectively. The values of n^2, k, and $\frac{n^2}{k}$ on the first five datasets are reported in Table 5. Clearly, n^2 is much larger than k. For example, on the dataset AM, n^2 is over 10 orders of magnitude larger than k. Thus, `DC-Exact` runs much faster than `Exact`.

7.3 Approximation Algorithms

In Figure 11, we show the running time of approximation algorithms on all the eight datasets, where bars touching the

Table 5: The total numbers of values of a examined in DC-Exact and Exact.

Dataset	n^2 (Exact)	k (DC-Exact)	$\frac{n^2}{k}$
MO	4.71×10^4	16	2.94×10^3
TC	1.50×10^6	23	6.54×10^4
OF	8.64×10^6	35	2.47×10^5
AD	4.28×10^7	81	5.28×10^5
AM	1.63×10^{11}	13	1.25×10^{10}

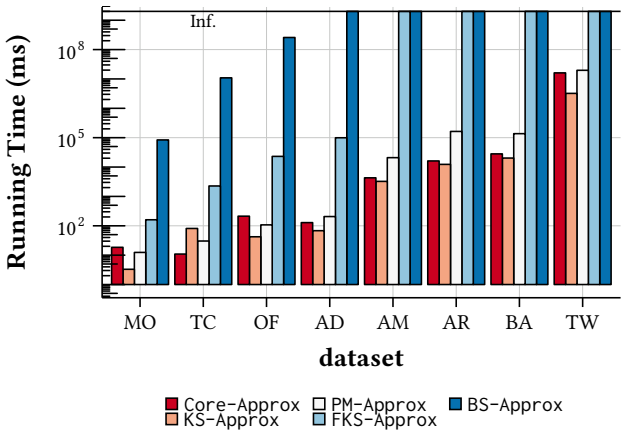


Figure 11: Efficiency of approximation algorithms.

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 4. (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.

Next, we focus on the 2-approximation algorithms and perform a deeper investigation on why Core-Approx is significantly faster than others. Recall that the time complexities of BS-Approx, FKS-Approx, and Core-Approx are $O(n^2(n+m))$, $O(n\cdot(n+m))$, and $O(\delta(n+m))$, respectively, so we can roughly use $\frac{n^2}{\delta}$ and $\frac{n}{\delta}$ to explain the speedup of Core-Approx over BS-Approx and FKS-Approx, respectivecly. In Table 6, we report the values of $\frac{n}{\delta}$ and $\frac{n^2}{\delta}$ on all the eight datasets. Clearly, on the first four datasets, the values of $\frac{n^2}{\delta}$ and $\frac{n}{\delta}$ are roughly the same as the times of speedup in Figure 11. Based on this observation, we conjecture that for other larger datasets (e.g., AR), Core-Approx could be up to ten and five orders of magnitude faster than BS-Approx and FKS-Approx respectively, although we did not get the actual running time of BS-Approx and FKS-Approx in our experiments.

Besides, we compare the actual approximation ratios of all the five approximation algorithms. Specifically, for each graph, we first obtain the exact DDS using DC-Exact, then compute the approximate DDS's using these approximation algorithms, and get the actual approximation ratios (i.e., the density of the exact DDS over those of approximate DDS's). Note that a smaller actual approximation ratio indicates that the corresponding approximate DDS has higher accuracy. We report the actual approximation ratios of each algorithm on the first five datasets in Figure 12. Clearly, the actual approximation ratios of Core-Approx, BS-Approx, and FKS-Approx are indeed smaller than their theoretial approximation ratios (i.e., 2). Besides, we can clearly see that KS-Approx cannot provide 2-approximation results on some datasets (e.g., AM). This well confirms our finding that KS-Approx may fail to report a 2-approximation result in some cases, as discussed in Section 4.2. In addition, the actual approximation ratios of PM-Approx could be larger than 2 but less than 5, since its theoretical approximation ratio is $2\delta(1+\epsilon)$=8.

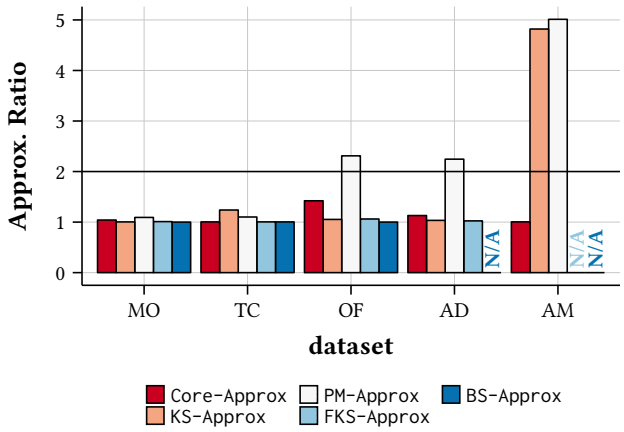


Figure 12: The actual approximation ratios of all the five approximation algorithms.

To further analyze why KS-Approx leads to very high approximation ratios, we revisit its algorithm steps and find that actually, it restricts the search on the subgraph, in which

Table 6: Analyzing 2-approximation algorithms.

Dataset	MO	TC	OF	AD	AM	AR	BA	TW
n	217	1,226	2,939	6,541	4.03×10^5	3.38×10^6	2.14×10^6	5.26×10^7
n^2	4.71×10^4	1.50×10^6	8.64×10^6	4.28×10^7	1.63×10^{11}	1.14×10^{13}	4.59×10^{12}	2.76×10^{15}
δ	8	3	27	18	10	26	60	2221
$\frac{n}{\delta}$	27	408	108	363	4.03×10^4	1.30×10^5	3.57×10^4	2.37×10^4
$\frac{n^2}{\delta}$	5.89×10^3	5.01×10^5	3.20×10^5	2.38×10^6	1.63×10^{10}	4.39×10^{11}	7.64×10^{10}	1.24×10^{12}

Table 7: The values of $\frac{|S^*|}{|T^*|}$ on the first five datasets.

Dataset	MO	TC	OF	AD	AM
$\frac{\lvert S^*\rvert}{\lvert T^*\rvert}$	1.04	5.67×10^{-2}	1.01	2.32	2.47×10^3

the mininum out-degree and minimum in-degree of all vertices are close to each other. In other words, when $\frac{|S^*|}{|T^*|}$=1, `KS-Approx` tends to find an approximate DDS with higher accuracy. In Table 7, we report the ratio of $|S^*|$ over $|T^*|$ on the first five datasets. We can observe that on most of the datasets, when $\frac{|S^*|}{|T^*|}$ is close to 1, `KS-Approx` tends to find DDS's with higher accuracy than `Core-Approx`, while when $\frac{|S^*|}{|T^*|}$ largely deviates from 1, it will perform worse.

7.4 Comparing `DC-Exact` and `Core-Approx`

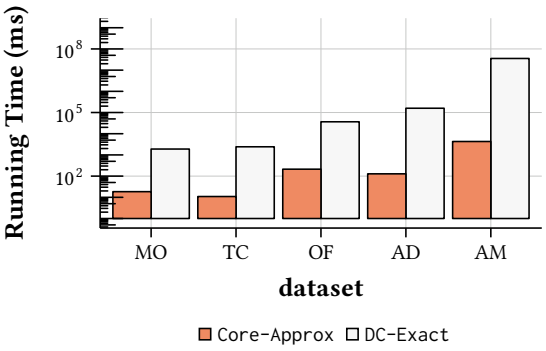


Figure 13: Efficiency of `Core-Approx` and `DC-Exact`.

In Figure 13, we report the running time of `Core-Approx` and `DC-Exact` on the first five datasets. Clearly, we can observe that `Core-Approx` is up to four orders of magnitude than `DC-Exact`. This is because `DC-Exact` computes the exact DDS by enumerating all the possible values of a=$\frac{|S|}{|T|}$ and solving a maximum flow problem for each of them, which are very computationally expensive, while `Core-Approx` only needs to extract the $[x^*, y^*]$-core from the graph.

In summary, for small-to-moderate-sized graphs (e.g., AD), `DC-Exact` is the best choice, because it computes an exact result in a reasonable time. For large-scale graphs (e.g., TW), `Core-Approx` is a much better option, since it achieves both high accuracy and high efficiency.

8 CONCLUSION

In this paper, we study the problem of densest subgraph discovery on directed graphs (DDS problem for short). We first review existing algorithms and discuss their limitations. We show that a previous algorithm [32], 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, which is more efficient than all the existing 2-approximation algorithms. 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, we will investigate how to efficiently find the DDS with size constraints on directed graphs.

ACKNOWLEDGEMENT

We would like to thank Prof. Barna Saha for her kind help. Reynold Cheng and Chenhao Ma were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116, 106150091, and 17205115), the University of Hong Kong (Projects 104004572, 102009508, and 104004129), and the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18). Lakshmanan's research was supported in part by a discovery grant and a discovery accelerator supplement grant fromÂăNSERCÂă(Canada). Wenjie Zhang was supported by PS53783, DP200101116 and DP180103096. Xuemin Lin was supported by NSFC61232006, 2018YFB1003504, U1636215, DP200101338, DP180103096, and DP170101628.

REFERENCES

[1] 2019. Supplementary Note. https://i.cs.hku.hk/~chma2/sup-sigmod2020.pdf. (2019).

[2] Federal Aviation Administration. 2019. Air Traffic Control System Command Center. https://www.faa.gov. (2019).

[3] Réka Albert, Hawoong Jeong, and Albert-László Barabási. 1999. Internet: Diameter of the world-wide web. *nature* 401, 6749 (1999), 130.

[4] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment* 5, 5 (2012), 454–465.

[5] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[6] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. 2010. Detecting high log-densities: an O (n 1/4) approximation for densest k-subgraph. In *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM, 201–210.

[7] Gregory Buehrer and Kumar Chellapilla. 2008. A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 95–106.

[8] Andrea Capocci, Vito DP Servedio, Francesca Colaiori, Luciana S Buriol, Debora Donato, Stefano Leonardi, and Guido Caldarelli. 2006. Preferential attachment in the growth of social networks: The internet encyclopedia Wikipedia. *Physical review E* 74, 3 (2006), 036116.

[9] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. 2010. Measuring User Influence in Twitter: The Million Follower Fallacy. In *Proc. Int. Conf. on Weblogs and Social Media*. 10–17.

[10] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.

[11] Maximilien Danisch, T-H Hubert Chan, and Mauro Sozio. 2017. Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 233–242.

[12] Soroush Ebadian and Xin Huang. 2019. Fast Algorithm for K-Truss Discovery on Public-Private Graphs. (2019), 2258–2264.

[13] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu. 2017. Effective and efficient attributed community search. *The VLDB Journal* 26, 6 (2017), 803–828.

[14] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *PVLDB* 10, 6 (2017), 709–720.

[15] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *PVLDB* 9, 12 (2016), 1233–1244.

[16] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2019. A survey of community search over big graphs. *The VLDB Journal* (2019), 1–40.

[17] Yixiang Fang, Zheng Wang, Reynold Cheng, Xiaodong Li, Siqiang Luo, Jiafeng Hu, and Xiaojun Chen. 2019. On spatial-aware community search. *TKDE* 31, 4 (2019), 783–798.

[18] Yixiang Fang, Zhongran Wang, Reynold Cheng, Hongzhi Wang, and Jiafeng Hu. 2019. Effective and efficient community search over large directed graphs. *TKDE* 31, 11 (2019), 2093–2107.

[19] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and Efficient Community Search over Large Heterogeneous Information Networks. *PVLDB* 13, 6 (Feb. 2020).

[20] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks VS Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1719 – 1732.

[21] Linton Clarke Freeman, Cynthia Marie Webster, and Deirdre M Kirke. 1998. Exploring Social Structure using Dynamic Three-dimensional Color Images. *Social Networks* 20, 2 (1998), 109–118.

[22] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2013. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems* 35, 2 (2013), 311–343.

[23] Aristides Gionis and Charalampos E Tsourakakis. 2015. Dense subgraph discovery: Kdd 2015 tutorial. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2313–2314.

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

[25] GT Heineman, G Pollice, and S Selkow. 2008. Network Flow Algorithms. Algorithms in a Nutshell. (2008).

[26] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 895–904.

[27] Jiafeng Hu, Reynold Cheng, Kevin Chen-Chuan Chang, Aravind Sankar, Yixiang Fang, and Brian YH Lam. 2019. Discovering Maximal Motif Cliques in Large Heterogeneous Information Networks. In *International Conference on Data Engineering (ICDE)*. IEEE, 746–757.

[28] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2019. *Community Search over Big Graphs*. Morgan & Claypool Publishers.

[29] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *PVLDB* 9, 4 (2015), 276–287.

[30] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. 2007. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*. ACM, 56–65.

[31] Ravi Kannan and V Vinay. 1999. *Analyzing the structure of large graphs*. Rheinische Friedrich-Wilhelms-Universität Bonn Bonn.

[32] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

[33] Jon M Kleinberg. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)* 46, 5 (1999), 604–632.

[34] Jérôme Kunegis. 2013. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*. 1343–1350. http://userpages.uni-koblenz.de/~kunegis/paper/kunegis-koblenz-network-collection.pdf

[35] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. 2007. The Dynamics of Viral Marketing. *ACM Transaction on the Web* 1, 1 (2007).

[36] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based Community Search over Large Directed Graphs. In *SIGMOD*.

[37] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, Tobias Grubenmann, Yixiang Fang, and Xiaodong Li. 2019. LINC: a motif counting algorithm for uncertain graphs. *Proceedings of the VLDB Endowment* 13, 2 (2019), 155–168.

[38] Paolo Massa, Martino Salvetti, and Danilo Tomasoni. 2009. Bowling Alone and Trust Decline in Social Network Sites. In *Proc. Int. Conf. Dependable, Autonomic and Secure Computing*. 658–663.

[39] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. 2015. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 815–824.

[40] Arjun Mukherjee, Bing Liu, and Natalie Glance. 2012. Spotting Fake Reviewer Groups in Consumer Reviews. In *WWW*. 191–200.

[41] Xing Niu, Xinruo Sun, Haofen Wang, Shu Rong, Guilin Qi, and Yong Yu. 2011. Zhishi.me – Weaving Chinese Linking Open Data. In *Proc. Int. Semantic Web Conf.* 205–220.

[42] Tore Opsahl, Filip Agneessens, and John Skvoretz. 2010. Node Centrality in Weighted Networks: Generalizing Degree and Shortest Paths. *Social Networks* 3, 32 (2010), 245–251.

[43] James B Orlin. 2013. Max flows in O (nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 765–774.

[44] B Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. 2010. Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 435–448.

[45] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally densest subgraph discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 965–974.

[46] Martin W. Schein and Milton H. Fohrman. 1955. Social Dominance Relationships in a Herd of Dairy Cattle. *The British J. of Animal Behaviour* 3, 2 (1955), 45–55.

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

[48] Nikolaj Tatti and Aristides Gionis. 2015. Density-friendly graph decomposition. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1089–1099.

[49] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 1122–1132.

[50] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 104–112.

[51] Zhiwei Zhang, Xin Huang, Jianliang Xu, Byron Choi, and Zechao Shang. 2019. Keyword-Centric Community Search. In *ICDE*. 422–433.

[52] Dong Zheng, Jianquan Liu, Rong-Hua Li, Cigdem Aslay, Yi-Cheng Chen, and Xin Huang. 2017. Querying intimate-core groups in weighted graphs. In *IEEE International Conference on Semantic Computing*. 156–163.