A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery

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

Yixiang Fang*
Chinese University of Hong Kong, Shenzhen
Shenzhen, China
fangyixiang@cuhk.edu.cn

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

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

Xiaolin Han
The University of Hong Kong
Hong Kong SAR, China
xlhan@cs.hku.hk

ABSTRACT

Given a directed graph G, the directed densest subgraph (DDS) problem refers to finding a subgraph from G, whose density is the highest among all subgraphs of G. The DDS problem is fundamental to a wide range of applications, such as fake follower detection and community mining. Theoretically, the DDS problem closely connects to other essential graph problems, such as network flow and bipartite matching. However, existing DDS solutions suffer from efficiency and scalability issues. In this paper, we develop a convex-programming-based solution by transforming the DDS problem into a set of linear programs. Based on the duality of linear programs, we develop efficient exact and approximation algorithms. Especially, our approximation algorithm can support flexible parameterized approximation guarantees. We have performed an extensive empirical evaluation of our approaches on eight real large datasets. The results show that our proposed algorithms are up to five orders of magnitude faster than the state-of-the-art.

CCS CONCEPTS

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

KEYWORDS

densest subgraph discovery, directed graphs, convex programming

ACM Reference Format:
Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V.S. Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3514221.3517837

*Yixiang Fang is the corresponding author.

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517837

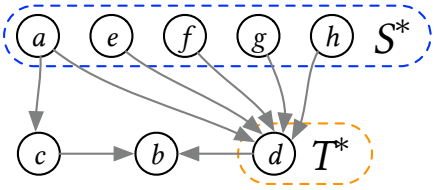


Figure 1: An example of fake follower detection [37].

1 INTRODUCTION

As one of the most representative kinds of graph data [9, 19–22, 27, 34, 35], directed graphs have been widely used to model complex relationships among objects [2, 9, 27]. For example, in Twitter, a directed edge can represent the "following" relationship between two users [27]; the Wikipedia article network can be modeled as a directed graph by mapping articles to vertices and links among articles to edges [9]; the Web network can also be modeled as a vast directed graph [2]; in gene regulatory networks, a link from gene A to gene B represents the regulatory relationship between those genes [29].

In this work, we study efficient solutions of the *directed densest subgraph* (DDS) problem, which aims to find the subgraph of a given directed graph having the highest density. This problem was first introduced by Kannan and Vinay [28], and has since received significant research interest [4, 10, 17, 30, 37, 50]. Essentially, the DDS problem aims to find two sets of vertices, S^* and T^*, from G, where (1) vertices in S^* have a large proportion of outgoing edges to those in T^*, and (2) vertices in T^* receive a large proportion of edges from those in S^* [28, 37]. The DDS has been widely used in many real applications [17], such as fake follower detection [24, 46], community mining [31], link spam detection [16], and graph compression [8]. For example, Figure 1 illustrates the application of fake follower detection [24, 46], which aims to identify fraudulent actions in a microblogging network, with edges representing the "following" relationships among users. By issuing a DDS query, we can find two sets of users S^* and T^*. Since compared with other users, the user d (in T^*) has unusually numerous followers (i.e., a, e, f, g, h) in S^*; it may be worth investigating whether d has bribed the users in S^* for following him/her.

Given a directed graph $G = (V, E)$ and two sets of (not necessarily disjoint) vertices $S, T \subseteq V$, the density of the directed subgraph

induced by S and T is the number $|E(S,T)|$ of edges linking vertices in S to 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 the density definition, the DDS problem [4, 10, 28, 30, 37] is defined as finding two sets of vertices, S^* and T^*, such that $\rho(S^*,T^*)$ is the largest among all the possible choices of $S, T \subseteq V$. For example, for the directed graph in Figure 1, the DDS is the subgraph induced by $S^* = \{a, e, f, g, h\}$ and $T^* = \{d\}$, whose density is $\rho^* = \frac{5}{\sqrt{5\times1}} = \sqrt{5}$, and there is no other subgraph whose density is larger than $\sqrt{5}$.

In undirected graphs, the density of a graph $G = (V, E)$ is defined to be $\rho(G) = \frac{|E|}{|V|}$ [18], which is different from that in directed graphs. In other words, finding the densest subgraph in undirected graphs (DS problem for short) amounts to finding the subgraph with the highest average degree [18]. For example, suppose we treat the graph in Figure 1 as an undirected graph by ignoring the directions of the edges. In that case, the densest subgraph will be the graph itself, with density 1, since there is no subgraph with a higher density. Compared to the DS problem, the DDS problem asks for two sets, S^* and T^*, which provides the advantage to distinguish different roles of vertices in the above application. On the other hand, if we restrict $S = T$, the density of a directed graph reduces to the classical notion of the density of undirected graphs. Hence, it naturally generalizes the density of undirected graphs and provides more information specific to directed graphs.

Prior works. In the literature, both exact [10, 30, 37] and approximation algorithms [4, 10, 28, 37, 50] have been developed for solving the DDS problem. The state-of-the-art exact algorithm is `DC-Exact` [37], which improves the flow-based algorithm proposed by Khuller and Saha [30] via the divide-and-conquer strategy and elegant core-based pruning techniques. Nevertheless, `DC-Exact` [37] is still inefficient on large datasets since it involves heavy cost of max-flow computation. For example, as we will show later, on a graph with 2.14M vertices and 17.6M edges, `DC-Exact` takes more than eight days to find the DDS.

Among approximation algorithms, the most efficient one is `Core-Approx` [37], which takes $O(\sqrt{m}(n+m))$ time, where n and m denote the numbers of vertices and edges in a directed graph $G = (V, E)$. However, it can only achieve a theoretical approximation ratio of 2, where the approximation ratio is the ratio of the density of the DDS to that of the subgraph returned. As a result, it does not afford the flexibility to control the approximation guarantee of the subgraph returned, e.g., to be better than 2. To alleviate this issue, recently Sawlani and Wang [50] have presented an algorithm with approximation ratio of $(1+\varepsilon)$, where $\varepsilon > 0$. However, as shown by our experiments later, it may perform even slower than the exact algorithms in some scenarios. Thus, the question of *whether we can design efficient algorithms that can provide an approximation guarantee that is parameterizable is open.*

Contributions. Our contributions are summarized as follows:

(1) *An extended linear programming (LP) formulation of the DDS problem.* We present an extended LP formulation of the DDS problem based on the LP formulation in [10], in which the DDS problem is converted as a set of linear programs. Based on convex programming, we derive the dual program for each linear program. We further exploit the duality of the primal and dual problems to avoid the overhead of computing the max-flow of the whole graph by leveraging the iterative Frank-Wolfe algorithm [14].

(2) *A divide-and-conquer algorithmic framework.* The above LP formulation needs to solve $O(n^2)$ linear programs by enumerating all $O(n^2)$ possible values of $\frac{|S|}{|T|}$, which is impractical for large graphs. To address this issue, we establish a connection between optimal values of LPs and the density of the DDS. We use these results to develop a divide-and-conquer strategy for reducing the number of LPs to solve.

(3) *An efficient $(1+\varepsilon)$-approximation algorithm.* Based on the framework above, we first develop an efficient approximation algorithm, `CP-Approx`, which can produce a $(1+\varepsilon)$-approximate DDS by exploiting the duality gap between the primal and dual programs, where $\varepsilon > 0$. In particular, we devise an efficient strategy to extract the approximate DDS candidate from the feasible solutions of the LPs and evaluate whether the candidate satisfies the approximation guarantee.

(4) *An efficient exact algorithm.* We further develop an efficient exact algorithm, namely `CP-Exact`, which similarly extracts the DDS candidates with that of `CP-Approx`. Given this, we first present the approximation algorithm and then introduce the exact algorithm. Besides, we introduce a novel concept, namely *stable subgraph*, based on the feasible solution of the dual program, which can help locate the DDS candidate and reduce the computation cost of DDS verification. We also propose a verification strategy based on max-flow on the stable subgraph.

(5) *Extensive experiments.* We have experimentally compared our proposed DDS algorithms with the state-of-the-art algorithms on eight real large datasets, where the largest one contains around two billion edges. The results show that for exact DDS algorithms, our `CP-Exact` is up to three orders of magnitude faster than the state-of-the-art exact algorithm. To the best of our knowledge, `CP-Exact` is the first exact algorithm that scales to billion-scale graphs. Besides, for the $(1+\varepsilon)$-approximation algorithms, our proposed `CP-Approx` is up to five orders of magnitude faster than the existing one [50].

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 discusses the linear programming formulation of the DDS problem and its dual program. We present our exact and approximation algorithms in Section 5 and experimental results in Section 6. Section 7 concludes the paper.

2 RELATED WORK

Densest subgraph discovery is a fundamental problem in network science [4, 6]. In the following, we mainly review the works of densest subgraph discovery on undirected graphs and directed graphs, respectively. A more comprehensive tutorial can be found in [17].

Densest subgraph discovery on undirected graphs. Given an undirected graph G=(V, E), its density is defined as $\frac{|E|}{|V|}$. Goldberg [18] first introduced the densest subgraph problem on undirected graphs, which aims to find the subgraph with the highest density among all the subgraphs, and designed a max-flow-based exact algorithm. Later, more efficient exact algorithms were developed [13, 41, 51, 53]. Generally, the algorithms above work well on small or

moderate-size graphs but are still inefficient to handle large graphs, as shown in [13]. Thus, researchers turned to develop efficient approximation algorithms [4, 7, 10, 13], which often run much faster by sacrificing some accuracy.

Besides, many variants, such as densest k-subgraph problem [5], locally densest subgraph problem [47], k-clique-densest subgraph problem [13, 41, 51, 53], and density-based graph decomposition [11, 52], have been extensively studied. Furthermore, some researchers studied how to efficiently maintain the densest subgraph on dynamic graphs [3, 6, 12, 25, 49, 50], where graph edges are inserted and deleted frequently. Among those, [50] also studied the densest subgraph problem on directed graphs, which will be introduced later. Nevertheless, the undirected solutions cannot be directly applied to solving the DDS problem since the definitions of density on undirected graphs and directed graphs are different.

Densest subgraph discovery on directed graphs (DDS problem). Kannan and Vinay [28] were the first to define a notion of density for directed graphs and propose the DDS problem. They also presented a polynomial-time algorithm based on max-flow. Charikar [10] developed an exact polynomial-time DDS algorithm by solving $O(n^2)$ linear programs. As a preview, we would like to remark that its linear program formulation is different from ours, and our formulation allows us to reduce the number of linear programs to be solved. Recently, Ma et al. [37] have introduced a novel exact algorithm by introducing the notion of $[x, y]$-core and exploiting a divide-and-conquer strategy.

Unfortunately, all the algorithms above are still inefficient, so some efficient approximation algorithms were developed. Kannan and Vinay [28] proposed an $O(\log n)$-approximation algorithm. Charikar [10] designed a 2-approximation algorithm taking time $O(n^2 \cdot (n+m))$. Khuller and Saha updated their algorithm in [30] to a 2-approximation algorithm with time complexity of $O(n(n+m))$ (see [37]). Bahmani et al. [4] provided a $2(1+\varepsilon)$-approximation algorithm ($\varepsilon > 0$), based on a streaming model. Ma et al. [37–39] developed an $[x, y]$-core-based 2-approximation algorithm with a time complexity of $O(\sqrt{m}(n+m))$. Sawlani and Wang [50] provided an algorithm for maintaining the $(1+\varepsilon)$-approximation densest subgraphs over dynamic directed graphs, and developed an approximation algorithm for static graphs with a time complexity of $O(\log_{1+\varepsilon} n \cdot t_{\mathsf{LP}})$, where t_{LP} is the time complexity for solving a linear program and $\varepsilon > 0$. The static version of [50] is the main competitor of our approximation algorithm.

3 PROBLEM DEFINITION

Consider a directed graph G=(V, E) with vertex set V, $|V| = n$, and edge set E, $|E| = m$. Given two sets $S, T \subseteq V$ which are not necessarily disjoint, we use $E(S, T)$ to denote the set of all edges from S to T, i.e., $E(S, T)$=$E \cap (S \times T)$. The subgraph induced by vertices S, T, and edges $E(S, T)$ is called an (S, T)-induced subgraph, denoted by $G[S, T]$. For each 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 definitions of density and the DDS problem. Unless mentioned otherwise, all the graphs mentioned later in this paper are directed graphs.

Definition 3.1 (DDS). Given a directed graph G=(V, E) and vertices $S, T \subset V$, the density of the subgraph $G[S, T]$ is defined as $\rho(S, T) = \frac{|E(S,T)|}{\sqrt{|S||T|}}$. A *directed densest subgraph* (DDS) of G is the (S^*, T^*)-induced subgraph $D = G[S^*, T^*]$, whose density $\rho(S^*, T^*)$ is the highest among all possible (S, T)-induced subgraphs, for $S, T \subset V$. We use $\rho^* = \rho(S^*, T^*)$ to denote the density of the DDS.

***Problem* 1** (DDS problem [4, 10, 17, 28, 30, 37])**:** Given a directed graph G=(V, E), return a DDS D=$G[S^*, T^*]$ of G.[1]

4 FROM DDS TO LP

In this section, we first introduce a linear programming (LP) formulation of the DDS problem (Section 4.1), in which we formulate the DDS problem as a set of LPs. Next, we present the dual program (DP) of the LP formulation (Section 4.2). Finally, we develop a Frank-Wolfe-based iterative algorithm to solve the DP (Section 4.3).

4.1 An LP formulation of DDS

Recall that ρ^* is the maximum value of $\rho(S, T)$ over all subsets S, T of vertices. Inspired by the linear programming (LP) relaxation in [10], we present another LP relaxation of ρ^*. Specifically, we consider all the possible ratios of $\frac{|S|}{|T|}$, and for each particular ratio $\frac{|S|}{|T|}$=c, we formulate an LP(c) as follows:

$$
\begin{aligned}
\mathsf{LP}(c) \quad \max \quad & x_{\text{sum}} = \sum_{(u,v)\in E} x_{u,v} & \\
\text{s.t.} \quad & x_{u,v} \geq 0, & \forall (u,v) \in E \\
& x_{u,v} \leq s_u, & \forall (u,v) \in E \\
& x_{u,v} \leq t_v, & \forall (u,v) \in E \\
& \sum_{u\in V} s_u = a\sqrt{c}, & \\
& \sum_{v\in V} t_v = \frac{b}{\sqrt{c}}, & \\
& a + b = 2. &
\end{aligned}
$$

Our LP relaxation is similar to the LP relaxation in [10], but they are different since we have an additional constraint $a+b=2$. When $a=1$ and $b=1$, our LP formulation is exactly the same as the one in [10]. We will show later that this additional constraint allows us to establish the connection between the optimal value of the LP(c) for a fixed c, denoted by OPT(LP(c)), and the density of the DDS, and the connection will play a key role in reducing the number of LPs examined. For other variables, s_u, t_v, and $x_{u,v}$ indicate the inclusion of a vertex u/vertex v/edge (u, v) in an optimal densest subgraph according to whether the variable value is larger than 0, when $c = \frac{|S^*|}{|T^*|}$.

Next, we show that our LP relaxation is correct for the DDS problem by establishing the lower and upper bounds of OPT(LP(c)).

Lemma 4.1 (Lower bound of OPT(LP(c))). *For a fixed c, consider two arbitrary sets of vertices $P, Q \subseteq V$, and let $c' = \frac{|P|}{|Q|}$. Then,* $\mathsf{OPT}(\mathsf{LP}(c)) \geq \frac{2\sqrt{c}\sqrt{c'}}{c+c'}\rho(P, Q)$.

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

By Lemma 4.1, it is easy to observe that if we set $c = c' = \frac{|S^*|}{|T^*|}$, then we have $\text{OPT}(\text{LP}(c)) \geq \rho(S^*, T^*)$.

Lemma 4.2 (Upper bound of $\text{OPT}(\text{LP}(c))$). *Given a feasible solution (x, s, t, a, b) of $\text{LP}(c)$ with value x_{sum}, we can construct an (S, T)-induced subgraph $G[S, T]$ such that $\sqrt{ab}\rho(S, T) \geq x_{\text{sum}}$.*

Lemma 4.2 implies that given a fixed c, we have a subgraph satisfying $\sqrt{ab}\rho(S, T) \geq \text{OPT}(\text{LP}(c))$, where a, b are from the optimal solution of $\text{LP}(c)$.

The proofs of Lemmas 4.1 and 4.2 can be obtained by following the proofs of Lemma 5 and 6 in [10], respectively. We provide the detailed proofs in the technical report [36].

Combining Lemmas 4.1 and 4.2, we get Theorem 4.3.

Theorem 4.3. $\rho^* = \rho(S^*, T^*) = \max_c\{\text{OPT}(\text{LP}(c))\}$.

Proof. According to Lemma 4.1, by setting $c = c' = \frac{|S^*|}{|T^*|}$, we can get $\max_c\{\text{OPT}(\text{LP}(c))\} \geq \rho(S^*, T^*)$, From Lemma 4.2, there exists an (S, T)-induced subgraph $G[S, T]$ such that $\sqrt{ab}\rho(S, T) \geq \max_c\{\text{OPT}(\text{LP}(c))\}$, where a and b are from the optimal solution to $\text{LP}(c^*)$ where c^* is the value that maximizes $\text{OPT}(\text{LP}(c))$. Since $a + b = 2$ and $a, b \geq 0$, we have $\sqrt{ab}\rho(S, T) \leq \rho(S, T) \leq \rho(S^*, T^*)$. Hence, $\rho(S^*, T^*) = \max_c\{\text{OPT}(\text{LP}(c))\}$. □

Theorem 4.3 establishes the connection between the DDS and the maximum value among the optimal values of all linear programs, which means our LP formulation is correct for the DDS problem.

4.2 The dual program

To solve $\text{LP}(c)$ for a fixed c, we use the Frank-Wolfe method [14], which is one of the simplest and earliest known iterative optimizers. However, for $\text{LP}(c)$, it is hard to derive the gradient of all variables w.r.t. $\sum_{(u,v)\in E} x_{u,v}$. Thus, we resort to solving the dual program $\text{DP}(c)$ of $\text{LP}(c)$. Hence, we first introduce the dual program $\text{DP}(c)$ of $\text{LP}(c)$. Then, based on the duality of $\text{DP}(c)$, we can figure out the connection between the DDS and $\text{OPT}(\text{LP}(c))$ (which is also the optimal value of $\text{DP}(c)$, denoted by $\text{OPT}(\text{DP}(c))$) when c is fixed. In the next section, we will further show that this connection enables a divide-and-conquer strategy for reducing the number of LPs to be solved.

Now, we present the Lagrangian dual $\text{DP}(c)$ of $\text{LP}(c)$,

$$
\begin{aligned}
\text{DP}(c) \quad \min \quad & \phi \\
\text{s.t.} \quad & \alpha_{u,v} + \beta_{v,u} \geq 1, && \forall (u,v) \in E \\
& \zeta \geq \sum_{(u,v)\in E} \alpha_{u,v}, && \forall u \in V \\
& \eta \geq \sum_{(u,v)\in E} \beta_{v,u}, && \forall v \in V \\
& \phi \geq 2\sqrt{c}\zeta, \\
& \phi \geq \frac{2}{\sqrt{c}}\eta, \\
& \alpha_{u,v}, \beta_{v,u} \geq 0. && \forall (u,v) \in E
\end{aligned}
$$

Before analyzing the properties of $\text{DP}(c)$, we propose a novel concept called *c-biased density* and the corresponding *c-biased DDS* to facilitate the following derivation of $\text{OPT}(\text{DP}(c))$.

Definition 4.4 (c-biased density). Given a directed graph $G = (V, E)$, a fixed $c \in \mathbb{R}_+$, and two sets of vertices $P, Q \subseteq V$, the c-biased density of the (P, Q)-induced subgraph $G[P, Q]$ is defined as

$$\rho_c(P, Q) = \frac{2\sqrt{c}\sqrt{c'}}{c + c'}\rho(P, Q) = \frac{2\sqrt{c}\sqrt{c'}}{c + c'}\frac{|E(P, Q)|}{\sqrt{|P| \cdot |Q|}}, \tag{1}$$

where $c' = \frac{|P|}{|Q|}$. Note when $c' = c$, $\rho_c(P, Q) = \rho(P, Q)$.

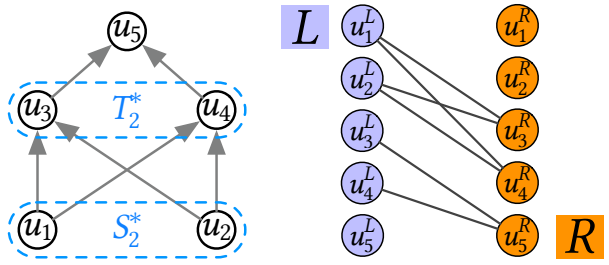
Definition 4.5 (c-biased DDS). Given a directed graph $G = (V, E)$ and a fixed c, the c-biased directed densest subgraph (c-biased DDS) is the (S_c^*, T_c^*)-induced subgraph, i.e., $G[S_c^*, T_c^*]$, whose c-biased density is the highest among all the possible (S, T)-induced subgraphs. Let $\rho_c^* = \rho_c(S_c^*, T_c^*)$ be the density of the c-biased DDS.

Example 4.6. For the directed graph G shown in Figure 2a, if c is fixed to 2, the 2-biased DDS will be the subgraph induced by $(S_2^* = \{u_1, u_2\}, T_2^* = \{u_3, u_4\})$. Its 2-biased density is $\frac{2\sqrt{2}\sqrt{c'}}{2+c'}\rho(S_2^*, T_2^*) = \frac{4\sqrt{2}}{3}$, where $c' = \frac{|S_2^*|}{|T_2^*|} = 1$. □

By analyzing the feasible solution of $\text{DP}(c)$, we can derive an upper bound of $\text{OPT}(\text{LP}(c))$, by exploiting the weak duality.

Lemma 4.7 (Upper bound of $\text{OPT}(\text{DP}(c))$). *For a fixed c, let S_c^*, T_c^* be the two subsets that maximize $\rho_c(S, T)$ (i.e., $G[S_c^*, T_c^*]$ is the c-biased DDS). Then, there exists a feasible solution to $\text{DP}(c)$ whose value is $\rho_c(S_c^*, T_c^*)$.*

To facilitate the proof of Lemma 4.7, we introduce an auxiliary bipartite graph B and *propagable paths* defined on B.



(a) A directed graph G **(b) The auxiliary bipartite graph B**

Figure 2: A directed graph and its auxiliary bipartite graph.

Definition 4.8 (Auxiliary bipartite graph). Given a directed graph $G = (V, E)$, its auxiliary bipartite graph B is a triplet, i.e., $B = (L, R, E_B)$, where $L = \{u^L | u \in V\}$, $R = \{u^R | u \in V\}$, $E_B = \{(u^L, v^R) | (u, v) \in E\} \subseteq L \times R$.

Figure 2 shows an auxiliary bipartite graph of a directed graph. Note that the auxiliary bipartite is only used to explain the design; it is not materialized in the implementation.

Definition 4.9 (Propagable path). Given a feasible solution $(\alpha, \beta, \zeta, \eta, \phi)$ of $\text{DP}(c)$, which satisfies that $\forall (u, v) \in E, \alpha_{u,v} + \beta_{v,u} = 1$. A path $u_0^{\mathcal{I}} \to u_1^{\mathcal{I}} \to \cdots \to u_k^{\mathcal{I}}$ in B is called a *propagable path*, denoted as $P_{u_0^{\mathcal{I}} \rightsquigarrow u_k^{\mathcal{I}}}$, where $\mathcal{I}$ is a binary variable and can be L or R indicating that the corresponding vertex belongs to L or R, respectively, if the following conditions are fulfilled,

(1) $\alpha_{u_i,u_{i+1}} > 0, 0 \le i < k$, if $u_i^I \in L$,
(2) $\beta_{u_i,u_{i+1}} > 0, 0 \le i < k$, if $u_i^I \in R$.

The *weight* of the propagable path is defined as,

$$w(P_{u_0^I \rightsquigarrow u_k^I}) = \min(\{\alpha_{u_i,u_{i+1}} | u_i^I \in L\} \cup \{\beta_{u_i,u_{i+1}} | u_i^I \in R\}). \quad (2)$$

Proof of Lemma 4.7. We claim that there exists a feasible solution $(\alpha, \beta, \zeta, \eta, \phi)$ to DP(c) with objective value $\rho_c(S_c^*, T_c^*)$, where $\zeta = \frac{1}{2\sqrt{c}}\rho_c(S_c^*, T_c^*)$, $\eta = \frac{\sqrt{c}}{2}\rho_c(S_c^*, T_c^*)$. We prove the claim by contradiction.

Suppose there were no feasible α and β which satisfy the first three conditions in DP(c). In other words, for any α and β satisfying $\forall(u,v) \in E, \alpha_{u,v} + \beta_{u,v} = 1$, there exists a vertex $u \in V$ such that $\sum_{v \in V} \alpha_{u,v} > \zeta$ or a vertex $v \in V$ such that $\sum_{u \in V} \beta_{u,v} > \eta$. Without loss of generality, we assume $\sum_{v \in V} \alpha_{u_0,v} > \zeta$. Meanwhile, none of the following cases exists,

(1) $\exists P_{u_0^L \rightsquigarrow u_k^R} \in B$ and $\sum_v \beta_{u_k,v} < \eta$,
(2) $\exists P_{u_0^L \rightsquigarrow u_k^L} \in B$ and $\sum_v \alpha_{u_k,v} < \zeta$.

Otherwise, assuming case (1) exists, we can propagate the value of $\min\{\sum_{v \in V} \alpha_{u,v} - \zeta, w(P_{u_0^L \rightsquigarrow u_k^R}), \eta - \sum_v \beta_{u_k,v}\}$ from $\sum_{v \in V} \alpha_{u,v}$ to $\sum_v \beta_{u_k,v}$ by changing the α and β values along the propagable path, until no such case exists.

For u_0 such that $\sum_{v \in V} \alpha_{u_0,v} > \zeta$, we construct two sets $S_c = \{v | \exists P_{u_0^L \rightsquigarrow v^L} \in B\} \cup \{u_0\}$ and $T_c = \{v | \exists P_{u_0^L \rightsquigarrow v^R} \in B\}$. Thus,

$$\begin{aligned} |E(S_c, T_c)| &= \sum_{(u,v) \in E(S_c,T_c)} (\alpha_{u,v} + \beta_{v,u}) \\ &> \zeta|S_c| + \eta|T_c| \\ &= \left(\frac{|S_c|}{\sqrt{c}} + \sqrt{c}|T_c|\right)\frac{\rho_c(S_c^*, T_c^*)}{2}. \end{aligned} \quad (3)$$

Further, we have

$$|E(S_c, T_c)| = \left(\frac{|S_c|}{\sqrt{c}} + \sqrt{c}|T_c|\right)\frac{\rho_c(S_c, T_c)}{2}. \quad (4)$$

Combining Equations (3) and (4), we have $\rho_c(S_c, T_c) > \rho_c(S_c^*, T_c^*)$, which contradicts with the assumption made in Lemma 4.7 that $G[S_c^*, T_c^*]$ is the c-biased DDS. Hence, the lemma holds. □

Combining Lemmas 4.1 and 4.7, we can establish the connection between the c-biased DDS and OPT(LP(c)) by Theorem 4.10.

Theorem 4.10. *For a fixed c, let $G[S_c^*, T_c^*]$ be the c-biased DDS. Then, we have* OPT(LP(c)) = OPT(DP(c)) = $\rho_c(S_c^*, T_c^*)$.

Proof. We have OPT(LP(c)) $\ge \rho_c(S_c^*, T_c^*)$ by Lemma 4.1, and OPT(LP(c)) $\le \rho_c(S_c^*, T_c^*)$ by Lemma 4.7 and weak duality. Thus, Theorem 4.10 holds by strong duality. □

Here, we use an example to illustrate further the correctness of Theorem 4.10.

Example 4.11. For $c = 2$, we can construct the optimal solutions for LP(c) and DP(c), whose value is exactly the c-biased density of c-biased DDS discussed in Example 4.6.

For LP(c), by setting $a = \frac{2}{3}$ and $b = \frac{4}{3}$, we can get $s_1 = s_2 = \frac{\sqrt{2}}{3}$ and $t_3 = t_4 = \frac{\sqrt{2}}{3}$. Then, $x_{u_1,u_3} = x_{u_1,u_4} = x_{u_2,u_3} = x_{u_2,u_4} = \frac{\sqrt{2}}{3}$. Hence, the value of this solution is $\frac{4\sqrt{2}}{3}$. (ref. the proof of Lemma 4.1)

For DP(c), by setting $\forall(u,v) \in E, \alpha_{u,v} = \frac{1}{3}$, $\forall(u,v) \in E, \beta_{v,u} = \frac{2}{3}$, we can get $\zeta = \frac{2}{3}$, $\eta = \frac{4}{3}$, and $\phi = \frac{4\sqrt{2}}{3}$.

Because both LP(2) and DP(2) have solutions with value of $\frac{4\sqrt{2}}{3}$, OPT(LP(2)) = OPT(DP(2)) = $\rho_c(S_2^*, T_2^*) = \frac{4\sqrt{2}}{3}$. □

Comparison with the LP formulation in [10]. After a detailed analysis of our LP(c) and DP(c), we provide an in-depth comparison between the two LP formulations (i.e., ours and the one in [10]) from two perspectives:

(1) *From the perspective of* LP(c). When $a = 1$ and $b = 1$, our LP formulation (LP(c)) is the same as the one in [10]. Hence, $a + b = 2$ is a relaxation, which allows slightly larger search space for a fixed $c = \frac{|S|}{|T|}$. Intuitively, because the search space of LP(c) is enlarged, it is quite possible that the subgraph corresponding to the optimal value for a fixed c has a different $\frac{|S|}{|T|}$ ratio from c. We can observe this difference from Example 4.11, when $c = 2$, the c-biased DDS is the subgraph induced by $(S_2^* = \{u_1, u_2\}, T_2^* = \{u_3, u_4\})$, whose $\frac{|S|}{|T|}$ ratio is actually 1. In next section, we will show how to use this difference to reduce the number of c values to be examined.

(2) *From the perspective of* DP(c). The dual program in [10] minimizes $2\sqrt{c}\zeta + \frac{2}{\sqrt{c}}\eta$, while our DP($c$) minimizes $\max(2\sqrt{c}\zeta, \frac{2}{\sqrt{c}}\eta)$. Hence, it can be treated that our DP(c) is equivalent to the dual program in [10] with one more constraint that $2\sqrt{c}\zeta = \frac{2}{\sqrt{c}}\eta$, because our DP($c$) reaches the optimal when $2\sqrt{c}\zeta = \frac{2}{\sqrt{c}}\eta$ according to Lemma 4.7 and its proof. Meanwhile, this constraint helps us to derive the equivalence between the optimal value of DP and the density the c-biased DDS via the propagable path.

4.3 Solving the dual program DP(c)

In this subsection, we introduce the Frank-Wolfe-based method for solving DP(c), when c is fixed. To do this, we first simplify the DP(c) as follows:

(1) $\phi = \max\begin{cases} \max\limits_{u \in V}\{2\sqrt{c}\sum\limits_{(u,v) \in E} \alpha_{u,v}\}, \\ \max\limits_{v \in V}\{\frac{2}{\sqrt{c}}\sum\limits_{(u,v) \in E} \beta_{v,u}\}. \end{cases}$
(2) $\forall(u,v) \in E, \alpha_{u,v} + \beta_{v,u} = 1$.

The second item holds, because we are trying to minimize ϕ and if there exist an edge (u,v) such that $\alpha_{u,v} + \beta_{v,u} > 1$, then we might further minimize ϕ by decreasing the value of $\alpha_{u,v}$ or $\beta_{v,u}$.

Next, we introduce a new vector $\vec{r}$:

$$\vec{r} = \langle r_\alpha(1), r_\alpha(2), \cdots, r_\alpha(n), r_\beta(1), r_\beta(2), \cdots, r_\beta(n)\rangle, \quad (5)$$

where $r_\alpha(u) = 2\sqrt{c}\sum_{(u,v) \in E} \alpha_{u,v}$ denotes the outgoing weight defined on u and $r_\beta(v) = \frac{2}{\sqrt{c}}\sum_{(u,v) \in E} \beta_{v,u}$ denotes the incoming weight defined on v. As a result, the dual program DP(c) can be

re-written as

$$\begin{aligned} \text{DP}(c) \quad \min \quad & \|\vec{r}\|_\infty \\ \text{s.t.} \quad & \alpha_{u,v} + \beta_{v,u} = 1, && \forall (u,v) \in E \\ & 2\sqrt{c} \sum_{(u,v)\in E} \alpha_{u,v} = r_\alpha(u), && \forall u \in V \\ & \frac{2}{\sqrt{c}} \sum_{(u,v)\in E} \beta_{v,u} = r_\beta(v), && \forall v \in V \\ & \alpha_{u,v}, \beta_{v,u} \geq 0. && \forall (u,v) \in E \end{aligned} \tag{6}$$

Notice that $\|\vec{r}\|_\infty = \max_{u\in V}\{|r_\alpha(u)|, |r_\beta(u)|\}$.

Combining Theorem 4.10 and Equation (6), we can claim that it is possible to distribute the weight of each edge such that there exist two vertex sets S_c^* and T_c^* satisfying that the outgoing weight of each vertex u in S_c^* and the incoming weight of each vertex v in T_c^* are exactly the c-biased density of the c-biased DDS, i.e., $r_\alpha(u) = r_\beta(v) = \rho_c(S_c^*, T_c^*)$. After solving the DP($c$) and getting $\vec{r}$, we can get $G[S_c^*, T_c^*]$ by the following c-biased DDS construction method: (1) select the vertices of $\vec{r}$ with the same highest values; (2) let S_c^* include vertices with the highest outgoing weights; and (3) let T_c^* include vertices with the highest incoming weights.

We then adopt the Frank-Wolfe method to solve DP(c) above in an iterative manner. In each iteration, the algorithm considers the linearization of the objective function at the current position and moves towards a minimizer of this function [26]. To linearize $\|\vec{r}\|_\infty$ at (α, β), we need the subgradient of $\|\vec{r}\|_\infty$, as $\|\vec{r}\|_\infty$ is convex but not differentiable. Equation (7) gives a subgradient of $\|\vec{r}\|_\infty$.

$$\begin{aligned} \frac{\partial \|\vec{r}\|_\infty}{\partial \alpha_{u,v}} &= \frac{2\sqrt{c}}{|M|} \cdot \mathbb{1}_{r_\alpha(u)=\|\vec{r}\|_\infty}, && \forall (u,v) \in E; \\ \frac{\partial \|\vec{r}\|_\infty}{\partial \beta_{v,u}} &= \frac{2}{\sqrt{c}\cdot|M|} \cdot \mathbb{1}_{r_\beta(v)=\|\vec{r}\|_\infty}, && \forall (u,v) \in E; \end{aligned} \tag{7}$$

where $M = \{u | r_\alpha(u) = \|\vec{r}\|_\infty\} \cup \{v | r_\beta(v) = \|\vec{r}\|_\infty\}$, $\mathbb{1}_{\texttt{expr}}$ is the indicator function. More precisely, $\mathbb{1}_{\texttt{expr}} = 1$ if the condition expr is satisfied; otherwise $\mathbb{1}_{\texttt{expr}} = 0$.

$$\begin{aligned} \widehat{\alpha}_{u,v} &= \mathbb{1}_{r_\alpha(u)<r_\beta(v) \vee r_\alpha(u)=r_\beta(v) \wedge c<1}, && \forall (u,v) \in E; \\ \widehat{\beta}_{u,v} &= \mathbb{1}_{r_\alpha(u)>r_\beta(v) \vee r_\alpha(u)=r_\beta(v) \wedge c\geq 1}, && \forall (u,v) \in E. \end{aligned} \tag{8}$$

Equation (8) gives $(\widehat{\alpha}, \widehat{\beta})$, which is the minimizer of the linear function given by $\partial\|\vec{r}\|_\infty$ among the feasible area of DP(c).

Based on Equations (7) and (8), we can develop a variant of the Frank-Wolfe method [26], called Frank-Wolfe-DDS, to optimize DP(c) in Equation (6). Algorithm 1 presents the details, which takes input a directed graph G, the number of iterations N, and the ratio c, and outputs $(\vec{r}^{(N)}, \alpha^{(N)}, \beta^{(N)})$ after N iterations. First, it initializes $\alpha^{(0)}$, $\beta^{(0)}$, and $\vec{r}^{(0)}$ (lines 2-4). Then, it repeats N iterations to update α, β, and r (lines 5-12). In detail, the minimizer of the linearization of $\|\vec{r}\|_\infty$ at $(\alpha^{(i-1)}, \beta^{(i-1)})$, denoted as $(\widehat{\alpha}, \widehat{\beta})$, is computed via Equation (8) (lines 7-8); $\alpha^{(i)}$ (resp. $\beta^{(i)}$) is calculated based on $\alpha^{(i-1)}$ (resp. $\beta^{(i-1)}$) and $\widehat{\alpha}$ (resp. $\widehat{\beta}$) in line 9 (resp. line 10); the algorithm aggregates $\alpha^{(i)}$ and $\beta^{(i)}$ to obtain $\vec{r}^{(i)}$ (lines 11-12).

Theorem 4.12 (Convergence of Algorithm 1). *Suppose $d_{\max}^+$ (resp. $d_{\max}^-$) is the maximum outdegree (resp. indegree) of G and c is fixed. In Algorithm 1, for $i > 16(\sqrt{c} + \frac{1}{\sqrt{c}})\frac{|E|\max\{\sqrt{c}d_{\max}^+, \frac{1}{\sqrt{c}}d_{\max}^-\}}{\varepsilon^2}$, we have $\|\vec{r}^{(i)}\|_\infty - \rho_c^* \leq \varepsilon$.*

Algorithm 1: A Frank-Wolfe-based algorithm.

```
1  Function Frank-Wolfe-DDS(G = (V, E), N ∈ ℤ+, c):
2      foreach (u, v) ∈ E do α_{u,v}^(0) ← 1/2, β_{v,u}^(0) ← 1/2 ;
3      foreach u ∈ V do r_α^(0)(u) ← 2√c Σ_{(u,v)∈E} α_{u,v}^(0) ;
4      foreach v ∈ V do r_β^(0)(v) ← 2/√c Σ_{(u,v)∈E} β_{v,u}^(0) ;
5      for i = 1, ..., N do
6          γ_i ← 2/(i+2);
7          foreach (u, v) ∈ E do
8              compute α̂_{u,v}, β̂_{v,u} via Equation (8);
9          α^(i) ← (1 − γ_i) · α^(i−1) + γ_i · α̂;
10         β^(i) ← (1 − γ_i) · β^(i−1) + γ_i · β̂;
11         foreach u ∈ V do r_α^(i)(u) ← 2√c Σ_{(u,v)∈E} α_{u,v}^(i) ;
12         foreach v ∈ V do r_β^(i)(v) ← 2/√c Σ_{(u,v)∈E} β_{v,u}^(i) ;
13     return (r^(N), α^(N), β^(N));
```

Proof. For lack of space, we present the detailed proof in the technical report [36]. □

5 FAST LP SOLUTIONS FOR DDS

In Section 4, we transform the DDS problem into a set of LPs LP(c), w.r.t. different values of $c = \frac{|S|}{|T|}$, and develop a Frank-Wolfe-based algorithm to optimize LP(c) via solving its dual DP(c) when c is fixed. However, the straightforward method to find the DDS needs to solve all linear programs LP(c), w.r.t. $O(n^2)$ possible c values, which is prohibitively expensive. To reduce the number of LPs to be solved, we build the connection between the c-biased DDS and the DDS and develop a convex-programming-based algorithm framework according to the connection we establish in Section 5.1. Under this framework, we design approximation and exact algorithms in Sections 5.2 and 5.3, respectively.

5.1 Algorithm framework

Our proposed approximation and exact algorithms share the same framework, as depicted in Figure 3. Specifically, given a fixed c, we first optimize the dual program DP(c) via the Frank-Wolfe-based algorithm (Algorithm 1). Then, we extract the c-biased DDS from the near-optimal solution of DP(c) (briefed in Section 4.3). Afterward, we establish the connection between the c-biased DDS and the DDS and use it to devise a divide-and-conquer strategy to reduce the number of different c values to be examined.

To reduce the number of c values to be examined, we derive the following lemmas to compute (c_o, c_p).

Lemma 5.1. *For a fixed c, let $G[S_c^*, T_c^*]$ be the c-biased DDS. Let $c_o = \frac{|S_c^*|}{|T_c^*|}$ and $c_p = \frac{c^2}{c_o}$. For any (S,T)-induced subgraph $G[S,T]$ of G, if $\min\{c_o, c_p\} \leq \frac{|S|}{|T|} \leq \max\{c_o, c_p\}$, we have $\rho(S,T) \leq \rho(S_c^*, T_c^*)$.*

Proof. The proof is similar to the proof of Lemma 4.7 in [37]. We prove the lemma by contradiction. Let $h_c(x) = \frac{2\sqrt{c}\sqrt{x}}{c+x}$, which is a concave function, and its maximum value can be obtained by

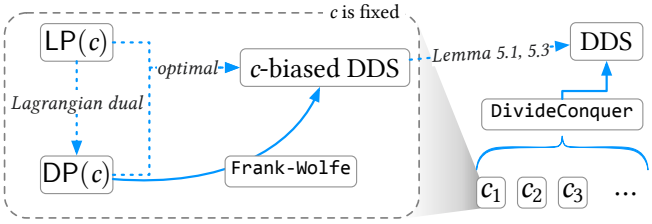


Figure 3: Our algorithm framework.

setting x to c. Assume that there exists an $[S_x, T_x]$-induced subgraph, which satisfies $\min\{c_o, c_p\} \le x = \frac{|S_x|}{|T_x|} \le \max\{c_o, c_p\}$, but it has $\rho(S_x, T_x) > \rho(S_c^*, T_c^*)$. Since $h_c(x) \le h_c(c_o)$ and $\rho(S_x, T_x) > \rho(S_c^*, T_c^*)$, we have $h_c(x)\rho(S_x, T_x) > h_c(c_o)\rho(S_c^*, T_c^*)$. This gives a contradiction to our assumption that S_c^*, T_c^* are the two subsets which maximize $\frac{2\sqrt{c}\sqrt{c'}}{c+c'}\rho(S,T)$, where $c' = \frac{|S|}{|T|}$. □

We illustrate Lemma 5.1 by Example 5.2.

Example 5.2. Reconsider Example 4.6. If we fix $c = 2, c_o = \frac{|S_c^*|}{|T_c^*|} = 1$ and $c_p = \frac{c^2}{c_o} = 4$, then for any (S,T)-induced subgraph $G[S,T]$ satisfying $1 \le \frac{|S|}{|T|} \le 4$, its density will be at most $\rho(S_c^*, T_c^*)$. This implies if we first compute the c-biased DDS for $c = 2$, then the values of c in $[1, 4]$ can be skipped safely by Lemma 5.1. □

According to Lemma 5.1, we can apply a divide-and-conquer strategy to reduce the number of values of c to be checked. That is, for a range of c values (c_l, c_r) to be examined, we pick the middle value c in the range, find the c-biased DDS, and compute (c_o, c_p) via Lemma 5.1. Then, all the values in (c_o, c_p) can be skipped safely, and the remaining intervals of c can be processed recursively.

Before presenting the details of the algorithm, we introduce the $[x, y]$-core, a kind of cohesive subgraphs on directed graphs [37], which is helpful to reduce the size of the graph to be processed by `Frank-Wolfe-DDS`.

Definition 5.3 ($[x, y]$-core [37]). Given a directed graph G=(V, E), the $[x, y]$-**core** is the largest (S, T)-induced subgraph $G[S,T]$, which satisfies:

(1) $\forall u \in S, d^+_{G[S,T]}(u) \ge x$ and $\forall v \in T, d^-_{G[S,T]}(v) \ge y$;
(2) $\nexists G[S', T'] \ne G[S,T]$, such that $G[S,T]$ is a subgraph of $G[S', T']$, i.e., $S \subseteq S'$, $T \subseteq T'$, and $G[S', T']$ satisfies (1).

Theorem 5.4 ([37]). *Given a graph G=(V, E), its DDS D=$G[S^*, T^*]$ is contained in the $\left[\lceil \frac{\rho^*}{2\sqrt{c}} \rceil, \lceil \frac{\sqrt{c}\rho^*}{2} \rceil\right]$-core, where c=$\frac{|S^*|}{|T^*|}$.*

By Theorem 5.4, we only need to run the `Frank-Wolfe-DDS` algorithm on the $\left[\frac{\tilde{\rho}^*}{2\sqrt{c_r}}, \frac{\sqrt{c_l}\tilde{\rho}^*}{2}\right]$-core, where (c_l, c_r) is the interval of c values to be examined and $\tilde{\rho}^*$ is the density of the densest subgraph found so far.

Based on `Frank-Wolfe-DDS` and the divide-and-conquer strategy, we design an algorithm framework, as shown in Algorithm 2. Given the range (c_l, c_r) of c to be checked, we first assign the middle value of c_l and c_r to c (line 2), and prune the graph via the $[x, y]$-core (line 3).

Algorithm 2: Our algorithm framework.

```
1  Function CP-DDS(G, c_l, c_r, ε, N):
2      c ← (c_l + c_r)/2;
3      G ← prune G via [x, y]-core;                    // Theorem 5.4
4      repeat
5          (r⃗, α, β) ← Frank-Wolfe-DDS(G, N, c);
6          if ε > 0 then (S_c, T_c, c_o, c_p, f) ← App-cDDS(G, r, ε, c);
7          else (S_c, T_c, c_o, c_p, f) ← Exact-cDDS(G, r, α, β, c);
8      until f = True;
9      if ρ(S_c, T_c) > ρ̃* then ρ̃* ← ρ(S_c, T_c), D̃ ← G[S_c, T_c];
10     if c_l ≤ c_o then
11         (S, T) ← CP-DDS(G, c_l, c_o, ε);
12         if ρ(S, T) > ρ̃* then ρ̃* ← ρ(S, T), D̃ ← G[S, T];
13     if c_p ≤ c_r then
14         (S, T) ← CP-DDS(G, c_o, c_r, ε);
15         if ρ(S, T) > ρ̃* then ρ̃* ← ρ(S, T), D̃ ← G[S, T];
16     return D̃;
```

Then, the function repeats calling `Frank-Wolfe-DDS` with N iterations (line 5) and extracting the approximate (resp. exact) DDS candidate as well as the c value range to be skipped via `App-cDDS` (resp. `Exact-cDDS`) in line 6 (resp. line 7) until the accuracy requirement (noted as f) is fulfilled (lines 4-8). Next, we check whether the current DDS needs to be updated; if so, update the DDS (line 9). Finally, the whole range (c_o, c_p) is skipped and we conduct search on the two intervals which are split by (c_o, c_p) to compute the approximate DDS (lines 10-15).

The detailed functions of extracting the approximate and exact DDS's and skipping the range of c values, i.e., `App-cDDS` and `Exact-cDDS`, will be discussed extensively in Sections 5.2 and 5.3 respectively.

Under the convex-programming-based framework (Algorithm 2), to compute the $(1+\varepsilon)$-approximation DDS, we can directly invoke `CP-DDS(`G, $\frac{1}{n}$, n, ε, N`)` and term it as `CP-Approx`. Similarly, to compute the exact DDS, we can directly invoke `CP-DDS(`G, $\frac{1}{n}$, n, 0, N`)` and call it `CP-Exact`.

5.2 The $(1+\varepsilon)$-approximation algorithm

We begin with an interesting Lemma:

Lemma 5.5. *Given a directed graph $G = (V, E)$, a positive real value ε, and $c^* = \frac{|S^*|}{|T^*|}$, if c satisfies that $\sqrt{c^*} \cdot \frac{1}{1+\varepsilon} \le \sqrt{c} \le \sqrt{c^*} \cdot (1+\varepsilon)$, we have*

$$\frac{\rho^*}{\rho_c^*} \le 1 + \varepsilon, \tag{9}$$

where the DDS of G is $G[S^*, T^*]$ and $c^* = \frac{|S^*|}{|T^*|}$.

Proof. According to the definition of the c-biased DDS, we have $\rho_c^* \ge \frac{2}{\frac{\sqrt{c}}{\sqrt{c^*}} + \frac{\sqrt{c^*}}{\sqrt{c}}}\rho(S^*, T^*)$. Since c satisfies $\frac{\sqrt{c}}{\sqrt{c^*}} \le 1+\varepsilon$ and $\frac{\sqrt{c^*}}{\sqrt{c}} \le 1+\varepsilon$, we can easily conclude that $\rho_c^* \ge \frac{2}{\frac{\sqrt{c}}{\sqrt{c^*}} + \frac{\sqrt{c^*}}{\sqrt{c}}}\rho(S^*, T^*) \ge \frac{1}{1+\varepsilon}\rho^*$. Hence, Lemma 5.5 holds. □

Clearly, Lemma 5.5 states that if the value of c is close to c^*, then the c-biased DDS provides a good approximation solution with

theoretical approximation guarantee. However, the value of c^* is unknown in advance, so a straightforward approximation algorithm needs to split the whole range of c, i.e., $[\frac{1}{n}, n]$, into a list consecutive intervals, i.e., $[\frac{1}{n}, \frac{1}{n}(1+\varepsilon)^2]$, $[\frac{1}{n}(1+\varepsilon)^2, \frac{1}{n}(1+\varepsilon)^4]$, $\cdots$, $[\frac{1}{(1+\varepsilon)^2}n, n]$, then compute the exact c-biased DDS for a value of c from each interval, and return the one with the highest density. This algorithm needs to compute the exact c-biased DDS for a c selected from each interval, which is very costly, and examine many such intervals. We introduce two corollaries to tackle these issues, which allow us to compute the approximate c-biased DDS and prune some intervals of the c values.

Corollary 5.6. *For a fixed c, let $(\alpha, \beta, \vec{r})$ be a feasible solution of* DP(c). *For $G[S_c, T_c]$ satisfying $\frac{\|\vec{r}\|_\infty}{\rho_c(S_c,T_c)} \le 1+\varepsilon$, let $c_o = \frac{|S_c|}{|T_c|}$ and $c_p = \frac{c^2}{c_o}$. For any (S,T)-induced subgraph $G[S,T]$, if $\min\{c_o, c_p\} \le \frac{|S|}{|T|} \le \max\{c_o, c_p\}$, then $\rho(S,T) \le (1+\varepsilon)\cdot\rho(S_c,T_c)$, where $\varepsilon \in \mathbb{R}_+$.*

Proof. As $\|\vec{r}\|_\infty$ is the upper bound of ρ_c^*, $\rho_c^* \le (1+\varepsilon)\rho_c(S_c,T_c)$. For any $G[S,T]$ satisfying $\min\{c_o,c_p\} \le \frac{|S|}{|T|} \le \max\{c_o,c_p\}$, we have $\rho(S,T) \le \frac{c+c_o}{2\sqrt{c}\sqrt{c_o}}\rho_c^* \le (1+\varepsilon)\cdot\rho(S_c,T_c)$. □

Corollary 5.7. *For a fixed c, let $(\alpha, \beta, \vec{r})$ be a feasible solution of* DP(c). *Suppose $G[S_c,T_c]$ satisfies $\frac{\|\vec{r}\|_\infty}{\rho_c(S_c,T_c)} \le \sqrt{1+\varepsilon}$. For any (S,T)-induced subgraph $G[S,T]$, if $\frac{c}{1+\varepsilon} \le \frac{|S|}{|T|} \le c\cdot(1+\varepsilon)$, then $\rho(S,T) \le (1+\varepsilon)\cdot\rho(S_c,T_c)$.*

Proof. According to Lemma 5.5, we have $\frac{\rho(S,T)}{\rho_c^*} \le \sqrt{1+\varepsilon}$, where $\frac{c}{1+\varepsilon} \le \frac{|S|}{|T|} \le c\cdot(1+\varepsilon)$. Further, we have $\frac{\rho_c^*}{\rho(S_c,T_c)} \le \frac{\|r\|_\infty}{\rho_c(S_c,T_c)} \le \sqrt{1+\varepsilon}$. Multiplying the two inequalities, we have $\frac{\rho(S,T)}{\rho_c^*}\cdot\frac{\rho_c^*}{\rho(S_c,T_c)} \le 1+\varepsilon$. Hence, the corollary holds. □

Based on Corollaries 5.6 and 5.7, we propose a strategy for reducing the number of c values to be examined. We use Figure 4 to illustrate the strategy: When the interval $[c_o, c_p]$ covers $[\frac{c}{1+\varepsilon}, c\cdot(1+\varepsilon)]$, we can skip the c values by using both Corollary 5.6 and Corollary 5.7. When the interval $[\frac{c}{1+\varepsilon}, c\cdot(1+\varepsilon)]$ covers $[c_o, c_p]$, then only Corollary 5.7 will be used. Note that these two intervals never partially intersect with each other, since $c_o c_p = c^2 = \frac{c}{1+\varepsilon}c(1+\varepsilon)$. In other words, the intervals fulfill that either $c_o \le \frac{c}{1+\varepsilon} \le c(1+\varepsilon) \le c_p$ or $\frac{c}{1+\varepsilon} \le c_o \le c_p \le c(1+\varepsilon)$. In the two cases, the number of trials of c is bounded by $O(\log_{1+\varepsilon} n)$, since the size of the interval increases exponentially with $(1+\varepsilon)$.

After approximately solving the DP(c) and getting $\vec{r}$, we can get the approximate c-biased DDS by slightly modifying the construction method in Section 4.3. That is, we sort the vertices of $\vec{r}$ and then construct the approximate c-biased DDS using vertices with higher incoming weights and outgoing weights.

App-cDDS (Algorithm 3) presents the detailed steps of computing an approximate c-biased DDS. It first initializes ρ_c^* to 0, S_c^*, T_c^* to $\emptyset$, and S_c, T_c to $\emptyset$ (line 1). Then, the vertices of $\vec{r}$ are sorted in descending order to their corresponding values (line 2). We put vertices with outgoing weight $r_\alpha(u)$ into set L and vertices with incoming weight $r_\beta(v)$ into set R (line 4). Afterwards, each vertex is inserted into S_c (resp. T_c) if its corresponding vertex is contained in

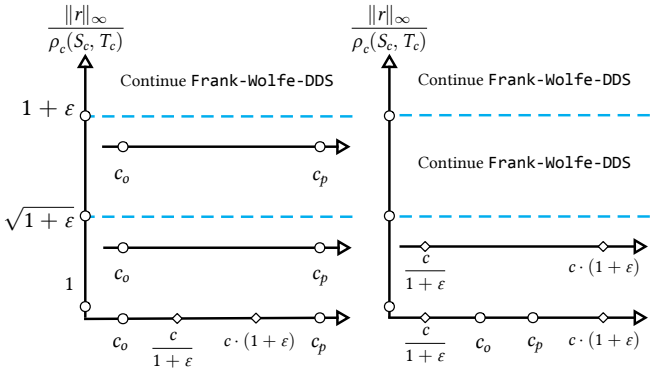


Figure 4: The strategy of reducing the number of c values.

L (resp. R) (in lines 6-7). Once S_c or T_c is updated, App-cDDS checks whether ρ_c^* can be updated by $\rho_c(S_c,T_c)$ (line 9); if yes, updates ρ_c^*, S_c^*, and T_c^* (line 10). Next, it computes c_o and c_p according to Corollary 5.6 (lines 11-12). Finally, it checks whether the approximate DDS candidate satisfies the conditions in Corollaries 5.6 and 5.7 and returns the DDS candidate as well as the range of c to be skipped (lines 14-16).

Algorithm 3: Extract approximate c-biased DDS.

```
1  Function App-cDDS(G = (V, E), r, ε, c):
2      ρ_c^* ← 0, S_c^*, T_c^* ← ∅, S_c, T_c ← ∅;
3      sort the nodes according to r⃗: r(u_1) ≥ r(u_2) ≥ ··· ≥ r(u_2n);
4      L ← {u | r_α(u) ∈ r⃗}, R ← {v | r_β(v) ∈ r⃗};
5      for i = 1, ..., 2n do
6          if u_i ∈ L then S_c ← S_c ∪ {u_i};
7          else T_c ← T_c ∪ {u_i};
8          if S_c = ∅ or T_c = ∅ then continue;
9          if ρ_c(S_c, T_c) > ρ_c^* then
10             ρ_c^* ← ρ_c(S_c, T_c), S_c^* ← S_c, T_c^* ← T_c;
11     c_o ← |S_c^*|/|T_c^*|, c_p ← c^2/c_o;          // Corollary 5.6
12     if c_o > c_p then Swap(c_o, c_p);
13     δ ← r_{u_1^I} / ρ_c^*;
14     if δ ≤ √(1+ε) then return
           (S_c^*, T_c^*, min{c_o, c/(1+ε)}, max{c_p, c·(1+ε)}, True);
15     else if δ ≤ 1+ε ∧ c_o < c/(1+ε) ∧ c·(1+ε) < c_p then return
           (S_c^*, T_c^*, c_o, c_p, False);
16     else return (S_c^*, T_c^*, c_o, c_p, False);
```

Complexity. The time complexity of CP-Approx is $O(\log_{1+\varepsilon} n \cdot t_{\mathsf{FW}})$. t_{FW} denotes the complexity of Frank-Wolfe-DDS, and its convergence rate is provided by Theorem 4.12.

Comparison with the state-of-the-art. VW-Approx [50] is the state-of-the-art $(1+\varepsilon)$-approximation algorithm. VW-Approx transforms the DDS problem into $O(\log_{1+\varepsilon})$ vertex-weighted undirected densest subgraph problems, where the vertex weights are set according to $O(\log_{1+\varepsilon})$ different guesses of $\frac{|S|}{|T|}$. We summarize the reasons on why CP-Approx is more efficient than VW-Approx:

(1) *Less values of $\frac{|S|}{|T|}$ to be examined.* Both algorithms need to select several different values of $\frac{|S|}{|T|}$ for inner-loop computation, but the strategies of choosing values of $\frac{|S|}{|T|}$ are different, which can explain the efficiency improvement. VW-Approx select $O(\log_{1+\varepsilon} n)$ values, i.e., the powers of $1+\varepsilon$ over the range $[\frac{1}{n}, n]$, while CP-Approx uses the divide-and-conquer strategy to prune the values of $\frac{|S|}{|T|}$ based on the optimization result in the inner-loop (Corollary 5.6). The worst case of the number of values of $\frac{|S|}{|T|}$ examined in CP-Approx is also $O(\log_{1+\varepsilon} n)$ (Corollary 5.7), but Corollary 5.6 allows more values to be skipped.

(2) *Tighter error estimation in the inner loop.* When transforming the DDS problem to a set of vertex weighted undirected densest subgraph problems, VW-Approx applies a relaxation of AM-GM inequality. In contrast, in CP-Approx, we build the equivalence between the optimal solution of LP(c) and the c-biased DDS (Theorem 4.10). We conjecture that the relaxation of AM-GM inequality causes extra overhead to satisfy the approximation guarantee for VW-Approx, especially when ε is small.

(3) *Smaller size of the graph to be processed.* The $[x, y]$-core-based pruning strategy (Theorem 5.4) helps prune the vertices, which are certainly not contained in the DDS, and further reduce the size of the graph to be processed by the Frank-Wolfe computation in CP-Approx, while VW-Approx needs to process the whole graph each time.

5.3 The exact algorithm

To obtain the exact c-biased DDS, a straightforward method is to compute the optimal solution of DP(c) using the Frank-Wolfe-based algorithm, and then compute the c-biased DDS using the construction method in Algorithm 1. However, this method is very costly since the Frank-Wolfe-based algorithm needs many iterations to derive the optimal solution of DP(c) as shown by Theorem 4.12. To reduce the number of such iterations, we introduce some novel techniques such that the Frank-Wolfe-based algorithm can be stopped earlier, but it can still output the optimal solution.

In the following, we first introduce a novel concept called *stable (S,T)-induced subgraph* inspired by [11], and then present the necessary and sufficient conditions of verifying whether a stable (S, T)-induced subgraph is the exact c-biased DDS.

Definition 5.8 (Stable (S,T)-induced subgraph). Given a directed graph G and a fixed c, an (S,T)-induced subgraph $G[S,T]$ of G is a stable (S,T)-induced subgraph with respect to a feasible solution $(\vec{r}, \alpha, \beta)$ to DP(c), if the following conditions hold:

(1) $\min\{\min_{u\in S}\{r_\alpha(u)\}, \min_{v\in T}\{r_\beta(v)\}\} > \max\{\max_{u\in V\setminus S}\{r_\alpha(u)\}, \max_{v\in V\setminus T}\{r_\beta(v)\}\}$;
(2) for each $(u,v) \in E \setminus E(S,T)$ such that $\alpha_{u,v} = 0$ if $u \in S$, or $\beta_{v,u} = 0$ if $v \in T$.

Essentially, in Definition 5.8, the first condition requires that vertices in the stable (S,T)-induced subgraph are with higher incoming weights or outgoing weights, while the second one states that the edges of the stable (S,T)-induced subgraph are denser because the incoming weights or outgoing weights received by vertices in the subgraph only come from the edges in the subgraph. Then, we give an example to explain further the stable (S,T)-induced subgraph.

Algorithm 4: Verify c-biased DDS.

```
1 Function Is-cDDS(G[S, T], c):
2     L ← {u^L | u ∈ S}, R ← {u^R | u ∈ T};
3     V_F ← {s} ∪ L ∪ R ∪ {t};
4     for u^R ∈ R do add (s, u^R) to E_F with capacity d+_{G[S,T]}(u);
5     for u^L ∈ L do add (u^L, t) to E_F with capacity ρ_c(S,T) / (2√c);
6     for u^R ∈ R do add (u^R, t) to E_F with capacity √c ρ_c(S,T) / 2;
7     for (u, v) ∈ E(S, T) do add (u^R, u^L) to E_F with capacity 2;
8     f ← maximum flow from s to t;
9     return f = |E(S, T)|;
```

Example 5.9. Reconsider the graph G in Figure 2a. Given c=2, $G[S=\{u_1,u_2\}, T=\{u_3,u_4\}]$ is stable with respect to the feasible solution $(\alpha, \beta, \vec{r})$ to DP(c), where $\forall(u,v) \in E(S,T)$, $\alpha_{u,v} = \frac{1}{3}$, $\beta_{u,v} = \frac{2}{3}$, and $\forall(u,v) \in E \setminus E(S,T)$, $\alpha_{u,v} = \beta_{u,v} = \frac{1}{2}$. The first condition in Definition 5.8 is fulfilled since $r_\alpha(u_1) = r_\alpha(u_2) = r_\beta(u_3) = r_\beta(u_4) = \frac{4\sqrt{2}}{3}$ is the highest value in $\vec{r}$. The second condition is also fulfilled as $\forall(u,v) \in E \setminus E(S,T)$ satisfies $u \notin S$ and $v \notin T$.

We now theoretically show that for a fixed c, the c-biased DDS must be contained in some stable (S,T)-induced subgraphs:

Lemma 5.10. *For a fixed c, suppose an (S,T)-induced subgraph $G[S,T]$ is stable with respect to some feasible solution $(\vec{r}, \alpha, \beta)$ to* DP(c)*, and $G[S_c^*, T_c^*]$ is the c-biased DDS. Then, $G[S_c^*, T_c^*]$ is contained in $G[S,T]$, i.e., $S_c^* \subseteq S$ and $T_c^* \subseteq T$.*

Proof sketch. We prove the lemma by contradiction via assuming $G[S_c^*, T_c^*]$ is not contained the stable subgraph $G[S,T]$. Then, we derive the contradiction by considering two cases according to whether $G[S_c^*, T_c^*]$ and $G[S,T]$ overlap with each other. The detailed proof can be found in our technical report [36]. □

Lemma 5.10 implies that for a fixed c, the constraint that $G[S,T]$ is a stable (S,T)-induced subgraph is the necessary condition of that $G[S,T]$ is the c-biased DDS, so the c-biased DDS verification process can be stopped earlier by checking this condition.

Next, we introduce the verification procedure for checking whether a stable (S,T)-induced subgraph is the c-biased DDS, inspired by [37, 51], which is based on the max-flow algorithm, as shown in Algorithm 4. To build the flow network, it first creates two sets L and R of nodes (lines 2), initializes the flow network with node set $\{s\} \cup L \cup R \cup \{t\}$ (line 3), and then adds directed edges with different capacities between these nodes (lines 4-7). Afterward, it computes the max-flow (line 8) and uses the value of the max-flow to verify the optimality (line 9).

The correctness of Algorithm 4 is guaranteed by Theorem 5.11.

Theorem 5.11 (Optimality test by max-flow). *Given a directed graph G, a stable (S,T)-induced subgraph $G[S,T]$ of G, a fixed c, the max-flow f in Algorithm 4 equals the edge number $|E(S,T)|$, if and only if $G[S,T]$ is the c-biased DDS.*

Before proving the theorem, we introduce a support lemma, which gives the upper bound of $|E(S,T)|$.

Lemma 5.12. *Given a feasible vector $\vec{r}$ in* DP(c) *with* $r_\alpha(u_1) \geq r_\alpha(u_2) \geq \cdots \geq r_\alpha(u_n)$ *and* $r_\beta(u_1) \geq r_\beta(u_2) \geq \cdots \geq v_\beta(u_n)$, *any* (S,T)*-induced subgraph in* G *satisfies*

$$|E(S,T)| \leq \left\lfloor \frac{1}{2\sqrt{c}} \sum_{i=1}^{|S|} r_\alpha(u_i) + \frac{\sqrt{c}}{2} \sum_{i=1}^{|T|} r_\beta(u_i) \right\rfloor. \tag{10}$$

Proof. For each edge (u,v), $\alpha_{u,v}$ and $\beta_{v,u}$ can be considered as the weights distributed from the edge to its two endpoints. As a result, $|E(S,T)| \leq \left\lfloor \frac{1}{2\sqrt{c}} \sum_{v\in S} r_\alpha(v) + \frac{\sqrt{c}}{2} \sum_{v\in T} r_\beta(v) \right\rfloor \leq \left\lfloor \frac{1}{2\sqrt{c}} \sum_{i=1}^{|S|} r_\alpha(u_i) + \frac{\sqrt{c}}{2} \sum_{i=1}^{|T|} r_\beta(u_i) \right\rfloor$, where the last inequality holds because of the fact that $u_1, u_2, \ldots, u_{|S|}$ are the $|S|$ nodes with largest r_α values and $u_1, u_2, \ldots, u_{|T|}$ are the $|T|$ nodes with largest r_β values. □

Proof of Theorem 5.11. Suppose f equals to $|E(S,T)|$, i.e., there exists a feasible flow with value $|E(S,T)|$ in the constructed network. The feasible flow induces $(\alpha,\beta) \in$ DP(c) for $G[S,T]$: for each edge $(u,v) \in E(S,T)$, $\alpha_{u,v}$ is the flow on the edge (v^R, u^L) (i.e., f_{v^R,u^L}) and $\beta_{v,u} = 1 - f_{v^R,u^L}$. This (α,β) induces $\vec{r}$ where $r_\alpha(u) = \rho_c(S,T), \forall u \in S$ and $r_\beta(v) = \rho_c(S,T), \forall v \in T$. In other words, each item in $\vec{r}$ is equal to $\rho_c(S,T)$. Then, Lemma 5.12 shows that there is no subgraph in $G[S,T]$ with strictly higher c-biased density, because for any subgraph $G[X,Y] \subset G[S,T]$ we have $\rho_c(X,Y) \leq \frac{2\sqrt{c\cdot c'}}{c+c'} \frac{\frac{|X|}{2\sqrt{c}} + \frac{\sqrt{c}|Y|}{2}}{\sqrt{|X|\cdot|Y|}} \rho_c(S,T) = \rho_c(S,T)$, where $c' = \frac{|X|}{|Y|}$. By Lemma 5.10, the c-biased DDS is within $G[S,T]$. Hence, $G[S,T]$ is the c-biased DDS.

Conversely, if $G[S,T]$ is the c-biased DDS, there is a feasible $(\alpha,\beta,\vec{r}) \in$ DP(c) for $G[S,T]$ such that $r_\alpha(u) = \rho_c(S,T), \forall u \in S$ and $r_\beta(v) = \rho_c(S,T), \forall v \in T$, following Lemma 4.7 and its proof. From α, we can construct a feasible flow with value $|E_H|$ by setting the flow on the edge (v^R, u^L) to $\alpha_{u,v}$, for each $(u,v) \in E(S,T)$. □

Example 5.13. Following Example 5.9, we can validate that given $c = 2$, the flow network generated based on the stable subgraph $G[S = \{u_1,u_2\}, T = \{u_3,u_4\}]$ has the maximum flow with value of $2 = |E(S,T)|$ by assigning the flows $f_{u_3^R,u_1^L}$, $f_{u_3^R,u_2^L}$, $f_{u_4^R,u_1^L}$ and $f_{u_4^R,u_2^L}$ to $\frac{1}{3}$. Hence, the stable subgraph is a c-biased DDS.

Based on the above discussions, we develop the whole algorithm of extracting and verifying the exact c-biased DDS in Algorithm 5. Precisely, we first extract a tentative c-biased DDS following the method used in `App-cDDS` (line 2). Then, we compute c_o and c_p according to Lemma 5.1 (lines 3-4). Afterward, we check whether the extracted subgraph is a stable (S,T)-induced subgraph via Definition 5.8 (line 5). If yes, we will continue to check its optimality by Algorithm 4 (line 6). If yes, the c-biased DDS is found (line 7). If the subgraph is stable but not the c-biased DDS, we use the subgraph to replace the graph G. For the current c, the following `Frank-Wolfe-DDS` computation will be conducted on the updated G, as the c-biased DDS is contained in the subgraph according to Lemma 5.10 (line 8). If the subgraph is not the c-biased DDS, the algorithm returns `False`, meaning that `Frank-Wolfe-DDS` needs to be invoked again (line 9).

Algorithm 5: Extract exact c-biased DDS.

```
1 Function Exact-cDDS(G = (V, E), r, α, β, c):
2     run lines 2-9 in Algorithm 3 to get G[S*_c, T*_c];
3     c_o ← |S*_c|/|T*_c|, c_p ← c²/c_o;
4     if c_o > c_p then Swap(c_o, c_p);
5     if Is-Stable(G, S*_c, T*_c, α, β) then      // Definition 5.8
6         if Is-cDDS(G[S*_c, T*_c], c) then       // Theorem 5.11
7             return (S*_c, T*_c, c_o, c_p, True);
8         update G as G[S*_c, T*_c];              // Lemma 5.10
9     return (S*_c, T*_c, c_o, c_p, False);
```

Complexity. The time complexity of `CP-Exact` is $O(h \cdot t_{\mathsf{FW}})$. h is the number of LPs to solve. Theoretically, $h \leq n^2$, but $h \ll n^2$ in practice. t_{FW} denotes the complexity of `Frank-Wolfe-DDS`, and its convergence rate is provided by Theorem 4.12.

Comparison with the state-of-the-art. `DC-Exact` [37] is the state-of-the-art exact DDS algorithm enhanced with the divide-and-conquer strategy and elegant core-based pruning techniques. Both `CP-Exact` and `DC-Exact` adopt the divide-and-conquer strategy to reduce the number of different c values to be examined, but they are derived based on different paradigms. The one in [37] is based on the output of the max-flow-based algorithm, and it needs to finish the max-flow-based binary search to skip the c values. In contrast, the one in our algorithm is based on the optimal value of the LP/DP formulation. The feasible solutions of DP(c) and LP(c) provide the upper and lower bound for the optimal value, respectively. Hence, our divide-and-conquer strategy also works for our approximation algorithm via the bounds. We further summarize the reasons why `CP-Exact` is more efficient than `DC-Exact`:

(1) *Avoiding the repeated max-flow computation.* `CP-Exact` uses the iterative `Frank-Wolfe-DDS` algorithm to avoid the heavy time cost of computing the max-flow many times, where computing the max-flow on a flow network takes at least $O(nm)$ [45]. Instead, it only uses the max-flow algorithm for the optimal validation on a small subgraph.

(2) *Early stop of the inner loop.* The stable subgraph (Lemma 5.10) and the optimality test by max-flow (Theorem 5.11) can help terminate the inner loop iterations early.

(3) *Smaller size of the graph to be processed.* First, we borrow the $[x,y]$-core (line 3 of Algorithm 2) from [37] to keep the graph to be computed as small as possible before the Frank-Wolfe iterations. Next, our proposed stable subgraph (line 8 of Algorithm 5) helps shrink the graph size further to be processed during the Frank-Wolfe iterations.

6 EXPERIMENTS

In this section, we first introduce the experimental setup in Section 6.1, and then present the experimental results of approximation algorithms and exact algorithms in Sections 6.2 and 6.3 respectively.

Table 1: Directed graphs used in our experiments.

Dataset	Full name	Category	$\|V\|$	$\|E\|$
MO [15]	moreno-oz	Human Social	217	2,672
TC [1]	maayan-faa	Infrastructure	1,226	2,615
OF [44]	openflights	Infrastructure	2,939	30.5K
AD [40]	advogato	Social	6,541	51K
AM [33]	amazon	E-commerce	403K	3.38M
AR [42]	amazon-ratings	E-commerce	3.38M	5.84M
BA [43]	baidu-zhishi	Hyperlink	2.14M	17.6M
SK [48]	web-sk-2005-all	Web	50.6M	1.95B

6.1 Setup

We use eight real datasets [32] which are publicly available [2]. These graphs cover various domains, including social networks (e.g., Twitter and Advogato), e-commerce (e.g., Amazon), and infrastructures (e.g., flight networks). Table 1 summarizes their statistics.

We compare the following approximation DDS algorithms:

- `CP-Approx` is our proposed approximation algorithm (Section 5.2).
- `Core-Approx` [37] is the state-of-the-art 2-approximation algorithm.
- `VW-Approx` [50] is the state-of-the-art $(1+\varepsilon)$-approximation algorithm (briefed in Section 5.2).

We also compare the following exact DDS algorithms:

- `CP-Exact` is our proposed exact algorithm (Section 5.3).
- `DC-Exact` [37] is the state-of-the-art exact algorithm enhanced with the divide-and-conquer strategy and elegant core-based pruning techniques.
- `Core-Exact` [37] is simplified version of `DC-Exact` without using the divide-and-conquer strategy.
- `Flow-Exact` [30] is the first max-flow-based exact algorithm.
- `LP-Exact` [10] is the LP-based exact algorithm.

Note that the parameter N of `Frank-Wolfe-DDS` is set to 100 in `CP-Exact` and `CP-Approx`. All the algorithms above are implemented in C++ with STL used. Our source code is publicly available[3]. 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.

6.2 Approximation algorithms

In this section, we mainly compare our approximation algorithm `CP-Approx` with the state-of-the-art $(1+\varepsilon)$-approximation algorithm `VW-Approx` [50] and the state-of-the-art 2-approximation algorithm `Core-Approx` [37].

6.2.1 Efficiency comparison. In this experiment, we evaluate the efficiency of `CP-Approx`, `VW-Approx`, and `Core-Approx`, with $\varepsilon = 1$. Note that `Core-Approx` only provides the 2-approximation DDS, and the efficiency result of `CP-Approx` and `VW-Approx` w.r.t. different values of ε will be presented later. Figure 5 reports the efficiency result of the three algorithms. The datasets are ordered by graph size on the x-axis. Notice that for some datasets, the bars of `VW-Approx` touch the solid upper line, which means `VW-Approx` cannot finish within one week on those datasets. From Figure 5, we can make

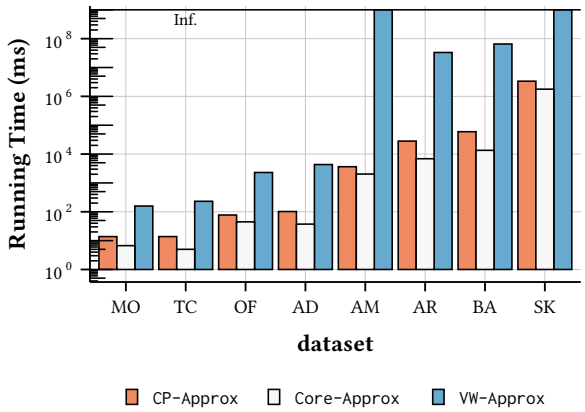


Figure 5: Efficiency of approximation algorithms.

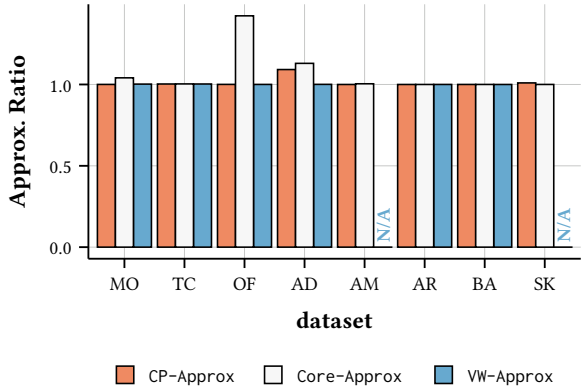


Figure 6: Actual approx. ratios of approx. algorithms.

the following observations: First, `CP-Approx` is slightly slower than `Core-Approx`. On average, the running time of `CP-Approx` is 2.68× of that of `Core-Approx` over all datasets. Second, `CP-Approx` is at least 10× and up to five orders of magnitude faster than `VW-Approx`. We have listed three reasons on why `CP-Approx` is faster than `VW-Approx` at the end of Section 5.2: fewer trials of different $\frac{|S|}{|T|}$, tighter error estimation, and the smaller size of graph to be processed.

6.2.2 Accuracy comparison. We present the actual approximation ratios of all the three approximation algorithms in Figure 6 with $\varepsilon = 1$. Specifically, for each dataset, we first obtain the exact DDS via `CP-Exact`, then compute the approximate DDS's using those approximation algorithms and get the actual approximation ratio (i.e., the density of the exact DDS over those of approximate DDS's). For `VW-Approx`, some bars are missing, as they cannot finish within one week on those datasets. From Figure 6, we can observe that actual approximation ratios are quite close to each other on most datasets, except that on the AD dataset, the ratio of `CP-Approx` are slightly larger than that of `VW-Approx`. Besides, most ratios of `CP-Approx` are smaller than those of `Core-Approx` (except SK dataset), and `CP-Approx` offers more flexibility on the approximation guarantee

[2] http://konect.uni-koblenz.de/networks/
[3] https://github.com/chenhao-ma/DDS-convex-code

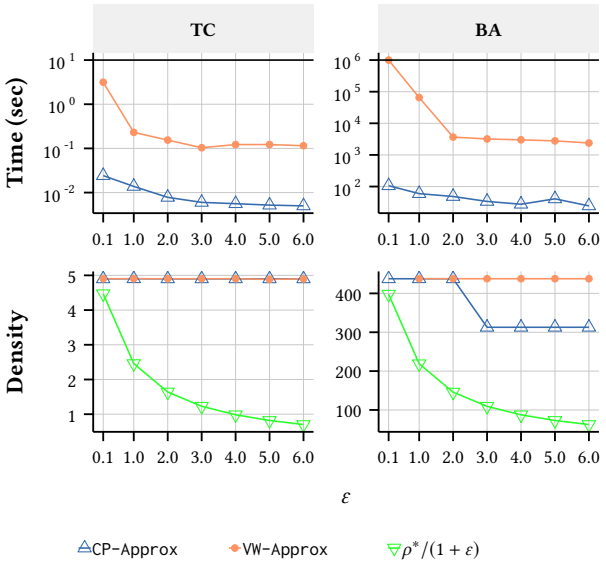


Figure 7: Effect of ε.

Table 2: Statistics of DDS's w.r.t. different ε values on AD.

ε	Density	$\lvert S\rvert$	$\lvert T\rvert$	Similarity w.r.t. $G[S^*, T^*]$
0	31.6811	453	195	1
0.1	31.6299	443	197	0.98
1	29.0183	913	2	0.43
2	28.0357	1	786	0.16

since ϵ can be any positive real values. The flexibility further helps explore DDS of higher or lower density.

6.2.3 Effect of ε. We evaluate the effect of ε on the efficiency and accuracy of the two $(1+\varepsilon)$-algorithms, i.e., `CP-Approx` and `VW-Approx`. Figure 7 presents the running time and the densities of the subgraphs returned by the algorithms over different ε values from 0.1 to 6 on the two datasets. We also add the plot of $\frac{\rho^*}{1+\varepsilon}$ to gauge how well the two algorithms satisfy the accuracy requirement. When ε is larger, `CP-Approx` can provide less dense subgraphs, which shows that `CP-Approx` can provide subgraphs with higher or lower density via smaller or larger ε. Note the running time plot of `VW-Approx` touches the solid upper line when $\varepsilon = 0.1$ on dataset BA, which means `VW-Approx` cannot finish within one week on that case. Hence, the density plot of `VW-Approx` is also missing in that case. From Figure 7, we can observe: First, for both `CP-Approx` and `VW-Approx`, their running time decreases along with the growth of ε. This is reasonable since computing a more accurate result often takes a longer time cost. Second, the improvement of `CP-Approx` over `VW-Approx` is more significant when ε is set smaller. One reason is that `CP-Approx` examines fewer LPs with different c values. Another reason is that the relaxation via AM-GM inequality in `VW-Approx` causes extra overhead to satisfy the approximation guarantee, especially when ε is small.

Hence, we conclude that our `CP-Approx` makes better use of the error tolerance to gain the efficiency speedup over `VW-Approx`.

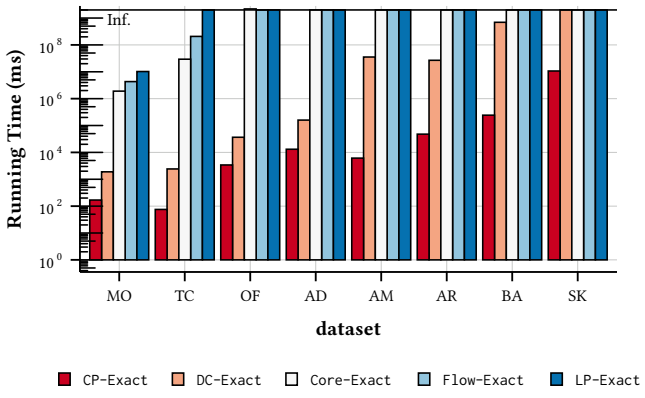


Figure 8: Efficiency of exact algorithms.

6.2.4 A case study: parameter selection of ε. In this case study, we investigate the approximate DDS's returned by `CP-Approx` under different ε values compared to the exact DDS. Table 2 reports the statistics of the exact DDS and three approximate DDS's (with $\varepsilon = 0.1$, 1 and 2, respectively) on the AD dataset. In terms of the density, we observe that all the three approximate DDS's have relatively high densities, but the approximate DDS with $\varepsilon = 0.1$ tends to have higher overlap than the subgraphs with $\varepsilon = 1$ and 2 since the former one's vertices and structures are very close to the exact DDS while the latter subgraphs look quite different from the exact DDS. Furthermore, we have computed the similarity between the approximate DDS's and the exact DDS w.r.t. the sets of vertices in the subgraphs. The similarity of the approximate DDS with $\varepsilon = 0.1$ is 0.98 while the one with $\varepsilon = 1$ is 0.43. Hence, we can conclude that if the users want to find a dense subgraph quickly, they can choose larger ε values (e.g., $\varepsilon = 1$). On the other hand, if the users want to find denser subgraphs that are highly overlapped with and similar to the exact DDS, it is better to set ε to smaller values (e.g., $\varepsilon = 0.1$). Further, users can also explore the DDS's returned with different values of ε in downstream applications (e.g., fake follower detection), as our `CP-Approx` is quite efficient.

6.3 Exact algorithms

6.3.1 Efficiency comparison. In Figure 8, we report the running time of exact algorithms on all eight datasets scaling from thousands to billions (ordered by the graph size on the x-axis). We can observe that `CP-Exact` is at least 10× and up to 5000× faster than the state-of-the-art exact algorithm `DC-Exact`. We have summarized three reasons why `CP-Exact` is more efficient than `DC-Exact` at the end of Section 5.3: avoiding the repeated max-flow computation; early stop in the inner loop; the smaller size of the graph to be processed.

To further investigate the performance improvement of `CP-Exact`, we collect some statistics of `CP-Exact` in Table 3, including the number of LPs with different c values examined (noted as "#c"), the average number of iterations that `Frank-Wolfe-DDS` runs for #c LPs (noted as "Avg #iterations"), the average number of edges that `Frank-Wolfe-DDS` processes for #c LPs after pruned via the techniques in Section 5.1 (noted as "Avg #edges"), and the product

Table 3: Statistics of `CP-Exact` over different datasets.

Datasets	#c	Avg #iterations	Avg #edges	Product
MO	17	158.82	1707.35	4.61×10^6
TC	18	177.78	588.72	1.88×10^6
OF	39	300	9146.62	1.07×10^8
AD	49	542.86	11687.8	3.11×10^8
AM	9	144.44	6982	9.08×10^6
AR	20	70	12426.5	1.74×10^7
BA	18	61.11	288142	3.17×10^8
SK	23	121.74	407×10^7	1.14×10^{11}

Table 4: The running time of `CP-Exact` and `CP-Exact-ab`

Dataset	`CP-Exact`	`CP-Exact-ab`	Speedup
MO	0.17s	22.26s	131.95
TC	0.08s	1.15s	15.29
OF	3.40s	439.06s	128.99
AD	13.12s	1208.11s	92.09
AM	6.13s	505.47s	82.46
AR	47.78s	321.27s	6.72
BA	242.66s	114709.50s	472.71
SK	10687.5s	NA	NA

for the three items (noted as "Product"). We can see that the number of c values examined on each dataset is much smaller than the possible values of c ($O(n^2)$), which demonstrates that the divide-and-conquer strategy is indeed effective. Besides, as `Frank-Wolfe-DDS` consumes the major running time of `CP-Exact`, the product explains why its time cost on AM is less than that on AD, although AM is larger than AD. Similarly, its time cost on TC is less than that on MO due to the same reason.

6.3.2 Ablation study of `Is-Stable` *and* `Is-cDDS`. Here, we conduct an ablation study on `CP-Exact` to understand the effectiveness of the early stop strategies (`Is-Stable` and `Is-cDDS`). We name the variant without `Is-Stable` and `Is-cDDS` as `CP-Exact-ab` (ablation). In this variant, the `Frank-Wolfe-DDS` computation needs to keep running until the optimal value reaches. Table 4 reports the running time of `CP-Exact` and `CP-Exact-ab` over different datasets. Note that `CP-Exact-ab` cannot finish reasonably on SK, and its running time is marked as "NA" in the table. We can observe the speedup provided by `Is-cDDS` and `Is-Stable` is from 6× to 472×. Hence, the early stop strategies based on the stable (S, T)-induced subgraph and the max-flow are effective to reduce the `Frank-Wolfe-DDS` iterations.

6.4 Memory usage

We evaluate the memory usage of all algorithms. We observe that the memory costs of all algorithms are around the same scale because all algorithms take linear memory usage w.r.t. the graph size. The details are ommited here.

6.5 Comparing `CP-Exact` and `CP-Approx`

In Table 5, we report the average speedup of `CP-Approx` compared to `CP-Exact` with respect to different values of ε over all datasets. We can observe that the speedup provided by `CP-Approx` increases along with the increase of ε, because `CP-Approx` can tolerate larger errors when ε is larger. Besides, when $\varepsilon = 0.001/0.005$, the speedup is less than 1, which means `CP-Approx` is slower than `CP-Exact`. The reason is that the number of iterations for Frank-Wolfe is proportional to ε^{-2} according to Theorem 4.12. Further, for `CP-Exact`, we introduced effective early stop strategies via stable subgraphs and optimality test by max-flow. To summarize, for small-to-moderate-sized graphs (e.g., AM), `CP-Exact` is the best choice, as it computes an exact DDS in a reasonable time. For large-scale graphs (e.g., SK), `CP-Approx` allows the users to efficiently explore the different approximate DDS's via different ε.

Table 5: Average speedup of `CP-Approx` compared to `CP-Exact`

ε	0.001	0.005	0.007	0.01	0.05	0.07
Avg Speedup	0.45	0.90	1.03	1.24	2.82	3.67
ε	0.1	0.5	1	1.5	2	2.5
Avg Speedup	4.67	9.37	25.13	26.87	37.39	38.26

7 CONCLUSION

This paper studies efficient solutions of the directed densest subgraph (DDS) problem via convex programming. We first review and discuss the limitations of existing algorithms. To efficiently find the DDS, we formulate the DDS problem as a set of linear programs and derive their dual programs. We use a Frank-Wolfe-based algorithm to iteratively solve the dual program and construct the DDS candidates based on their duality. Next, we apply a divide-and-conquer strategy to reduce the number of linear programs to be solved and develop both efficient exact and $(1+\varepsilon)$-approximation algorithms, respectively, where ε is an arbitrary positive value. Finally, we perform extensive experiments on eight real datasets (up to 2 billion edges) to evaluate the proposed algorithms. The experimental results show that our exact and approximation DDS algorithms are up to three and five orders of magnitude faster than their state-of-the-art competitors, respectively.

The excellent performance of applying the Frank-Wolfe-based algorithm on the DDS problem (this work), the densest subgraph problem on undirected graphs [11], and the k-clique-based densest subgraph problem [51], inspires the possibility that the convex programming theory and Frank-Wolfe-based algorithms may also bring improvement to other graph optimization problems, e.g., the max-flow problem [23]. Furthermore, it is worth investigating how to define and study the top-k DDS's problem, because users might be interested in finding more than one densest subgraph.

ACKNOWLEDGMENTS

Chenhao Ma and Xiaolin Han were supported by the University of Hong Kong (Projects 104005858, 10400599, KE award 2100048) and the Innovation and Technology Commission of Hong Kong (ITF project MRP/029/18). Reynold Cheng was supported by the HKU-TCL Joint Research Center for Artificial Intelligence (Project no. 200009430). Yixiang Fang was supported by NSFC under Grant 62102341 and CUHK-SZ grant UDF01002139. Lakshmanan's research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

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

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

[3] Albert Angel, Nick Koudas, Nikos Sarkas, Divesh Srivastava, Michael Svendsen, and Srikanta Tirthapura. 2014. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *The VLDB journal* 23, 2 (2014), 175–199.

[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] 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.

[6] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. 2015. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 173–182.

[7] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. 2020. Flowless: Extracting Densest Subgraphs Without Flow Computations. In *Proceedings of The Web Conference 2020* (Taipei, Taiwan) *(WWW '20)*. Association for Computing Machinery, New York, NY, USA, 573–583. https://doi.org/10.1145/3366423.3380140

[8] 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.

[9] 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.

[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] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference on World Wide Web*. 300–310.

[13] 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.

[14] Marguerite Frank, Philip Wolfe, et al. 1956. An algorithm for quadratic programming. *Naval research logistics quarterly* 3, 1-2 (1956), 95–110.

[15] 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.

[16] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering large dense subgraphs in massive graphs. In *PVLDB*. VLDB Endowment, 721–732.

[17] 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.

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

[19] Xiaolin Han, Reynold Cheng, Tobias Grubenmann, Silviu Maniu, Chenhao Ma, and Xiaodong Li. 2022. Leveraging Contextual Graphs for Stochastic Weight Completion in Sparse Road Networks. In *SIAM International Conference on Data Mining*. SIAM.

[20] Xiaolin Han, Reynold Cheng, Chenhao Ma, and Tobias Grubenmann. 2022. DeepTEA: Effective and Efficient Online Time-dependent Trajectory Outlier Detection. In *Proceedings of the VLDB Endowment*.

[21] Xiaolin Han, Daniele DellâĂŹAglio, Tobias Grubenmann, Reynold Cheng, and Abraham Bernstein. 2022. A framework for differentially-private knowledge graph embeddings. *Journal of Web Semantics* 72 (2022), 100696.

[22] Xiaolin Han, Tobias Grubenmann, Reynold Cheng, Sze Chun Wong, Xiaodong Li, and Wenya Sun. 2020. Traffic incident detection: A trajectory-based approach. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1866–1869.

[23] TE Harris and FS Ross. 1955. *Fundamentals of a method for evaluating rail net capacities.* Technical Report. RAND CORP SANTA MONICA CA.

[24] 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.

[25] Shuguang Hu, Xiaowei Wu, and TH Hubert Chan. 2017. Maintaining densest subsets efficiently in evolving hypergraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 929–938.

[26] Martin Jaggi. 2013. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th international conference on machine learning*. 427–435.

[27] 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.

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

[29] Shamir R. Karlebach, G. 2008. Modelling and analysis of gene regulatory networks. https://doi.org/10.1038/nrm2503. *Nat Rev Mol Cell Biol.* 9 (2008), 770âĂŞ780.

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

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

[32] 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

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

[34] Xiaodong Li, Reynold Cheng, Kevin Chen-Chuan Chang, Caihua Shan, Chenhao Ma, and Hongtai Cao. 2021. On analyzing graphs with motif-paths. *Proceedings of the VLDB Endowment* 14, 6 (2021), 1111–1123.

[35] 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.

[36] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, and Xiaolin Han. 2022. A Convex-Programming Approach for Efficient Directed Densest Subgraph Discovery [Technical Report]. https://github.com/chenhao-ma/DDS-convex-code/main.pdf. (2022).

[37] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS 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*. 1051–1066.

[38] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. Efficient directed densest subgraph discovery. *ACM SIGMOD Record* 50, 1 (2021), 33–40.

[39] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks VS Lakshmanan, Wenjie Zhang, and Xuemin Lin. 2021. On Directed Densest Subgraph Discovery. *TODS* 46, 4 (2021), 1–45.

[40] 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.

[41] 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.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] Ryan Rossi and Nesreen Ahmed. 2013. Network Repository. http://networkrepository.com

[49] Atish Das Sarma, Ashwin Lall, Danupon Nanongkai, and Amitabh Trehan. 2012. Dense subgraphs on dynamic networks. In *International Symposium on Distributed Computing*. Springer, 151–165.

[50] Saurabh Sawlani and Junxing Wang. 2020. Near-optimal fully dynamic densest subgraph. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 181–193.

[51] Bintao Sun, Maximilien Dansich, Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. *Proceedings of the VLDB Endowment* 13, 10 (2020), 1628 – 1640.

[52] 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.

[53] 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.