The VLDB Journal
https://doi.org/10.1007/s00778-023-00805-0

REGULAR PAPER

Accelerating directed densest subgraph queries with software and hardware approaches

Chenhao Ma[1] · Yixiang Fang[1] · Reynold Cheng[2] · Laks V. S. Lakshmanan[3] · Xiaolin Han[4] · Xiaodong Li[5]

Received: 13 October 2022 / Revised: 17 April 2023 / Accepted: 19 June 2023

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. Particularly, our approximation algorithm can support flexible parameterized approximation guarantees. We further investigate using GPU to speed up the solution of convex programs in parallel and achieve hundreds of times speedup compared to the original Frank–Wolfe computation. 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.

Keywords Directed graph · Densest subgraph discovery · Convex programming · GPU

1 Introduction

As one of the most representative kinds of graph data [9, 21–24, 28, 36, 38], directed graphs have been widely used to model complex relationships among objects [2, 9, 28]. For example, in Twitter, a directed edge can represent the "following" relationship between two users [28]; 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]; and in gene regulatory networks, a link from gene A to gene B represents the regulatory relationship between those genes [30].

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 [29] and has since received significant research interest [4, 10, 19, 31, 40, 53]. 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^* [29, 40]. The DDS has been widely used in many real applications [19], such as echo chamber and

✉ Xiaolin Han
xiaolinh@nwpu.edu.cn

Chenhao Ma
machenhao@cuhk.edu.cn

Yixiang Fang
fangyixiang@cuhk.edu.cn

Reynold Cheng
ckcheng@cs.hku.hk

Laks V. S. Lakshmanan
laks@cs.ubc.ca

Xiaodong Li
xdli@cs.hku.hk

1 The Chinese University of Hong Kong, Shenzhen, China

2 Department of Computer Science and Guangdong-Hong Kong-Macau Joint Laboratory and HKU Musketeers Foundation Institute of Data Science, The University of Hong Kong, Hong Kong, China

3 The University of British Columbia, Vancouver, Canada

4 Northwestern Polytechnical University, Xi'an, China

5 The University of Hong Kong, Hong Kong, China

Published online: 31 July 2023

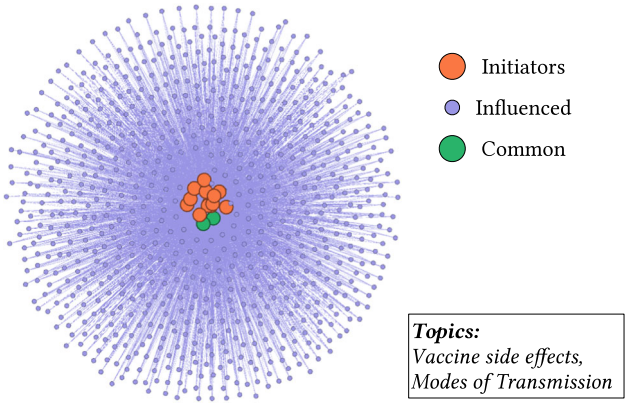


Fig. 1 Illustration of echo chamber

filter bubble detection [34], fake follower detection [25, 49], community mining [32], link spam detection [18], and graph compression [8].

Figure 1 illustrates an echo chamber detected via finding an approximate DDS from a dataset[1] with 660,730 nodes and 835,193 edges extracted from around one million retweets about Covid-19 [34]. From the figure, we can find that hundreds of users are "influenced" by 15 "initiators." Among those 15 initiators, there are two users that are also influenced by other initiators (marked as "Common"). The topics with this echo chamber consist of vaccine side effects and modes of transmission.

Figure 2 illustrates another application of fake follower detection [25, 49], 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, 29, 31, 40] 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 Fig. 2, the DDS is the subgraph induced by $S^* = \{a, e, f, g, h\}$ and $T^* = \{d\}$, whose density is $\rho^* = \frac{5}{\sqrt{5\times 1}} = \sqrt{5}$, and there is no other subgraph whose density is larger than $\sqrt{5}$.

[1] The dataset is due to Dr. Saravanan Thirumuruganathan from QCRI, HBKU.

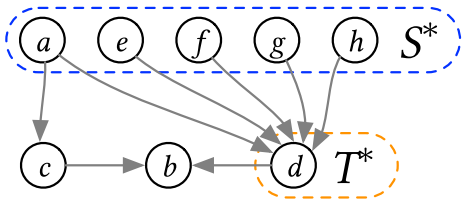


Fig. 2 An example of fake follower detection [40]

In undirected graphs, the density of a graph $G = (V, E)$ is defined to be $\rho(G) = \frac{|E|}{|V|}$ [20], 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 [20]. For example, suppose we treat the graph in Fig. 2 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, 31, 40] and approximation algorithms [4, 10, 29, 40, 53] have been developed for solving the DDS problem. The state-of-the-art exact algorithm is `DC-Exact` [40], which improves the flow-based algorithm proposed by Khuller and Saha [31] via the divide-and-conquer strategy and elegant core-based pruning techniques. Yet, `DC-Exact` [40] 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 2.14 M vertices and 17.6 M edges, `DC-Exact` takes more than eight days to find the DDS.

Among approximation algorithms, the most efficient one is `Core-Approx` [40], 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 et al. [53] and Chekuri et al. [11] have presented algorithms with approximation ratio of $(1+\varepsilon)$, where $\varepsilon > 0$. However, as shown by our experiments later, these two algorithms 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 [16].
2. *A divide-and-conquer algorithm 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, `CP-Exact`, which extracts DDS candidates similarly to `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. *Parallel Frank–Wolfe computation on GPU.* In our experiments, we found that Frank–Wolfe computations can be a bottleneck and usually take up most of the runtime. To mitigate this, we can perform computations within an iteration in parallel; however, large degree differences between vertices can cause load imbalance. To compensate for this imbalance, we propose an adaptive strategy to efficiently perform Frank–Wolfe computations on GPUs, which can further reduce the total runtime on billion-scale graphs by over 70%.
6. *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, `CP-Exact` is up to three orders of magnitude faster than the state-of-the-art exact algorithm. To 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 [53]. Furthermore, our GPU-based parallel strategy can further speed up Frank–Wolfe computations by hundreds of times.

Outline The rest of the paper is organized as follows. We review the related work in Sect. 2. In Sect. 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 Sect. 5. Section 6 presents our GPU-based parallel strategies for Frank–Wolfe computations. Experimental results are presented in Sect. 7. Section 8 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 the densest subgraph discovery on undirected graphs and directed graphs, respectively. A more comprehensive survey can be found in [37].

Densest subgraph on undirected graphs Given an undirected graph $G=(V, E)$, its density is defined as $\frac{|E|}{|V|}$. Goldberg [20] 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 [15, 44, 56, 58]. Generally, the algorithms above work well on small or moderate-size graphs but are still inefficient to handle large graphs, as shown in [15]. Thus, researchers turned to develop efficient approximation algorithms [4, 7, 10, 15, 59], which often run much faster by sacrificing some accuracy.

Besides, many variants, such as densest k-subgraph problem [5], locally densest subgraph problem [39, 50], k-clique-densest subgraph problem [15, 44, 56, 58], and density-based graph decomposition [13, 57], have been extensively studied. Furthermore, some researchers studied how to efficiently maintain the densest subgraph on dynamic graphs [3, 6, 14, 26, 52, 53], where graph edges are inserted and deleted frequently. Among those, [53] 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 on directed graphs (DDS problem) Kannan and Vinay [29] 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. [40] 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 [29] 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 [31] to a 2-approximation algorithm with time complexity of $O(n(n+m))$ (see [40]). Bahmani et al. [4] provided a $2(1+\varepsilon)$-approximation algorithm ($\varepsilon > 0$), based on a streaming model. Ma et al. [40–42] developed an $[x, y]$-core-based 2-approximation algorithm with a time complexity of $O(\sqrt{m}(n+m))$. Sawlani and Wang [53] provided an algorithm for maintaining the $(1+\varepsilon)$-approximation densest subgraphs over dynamic directed graphs and developed an approximation algorithm for static graphs with complexity of $O(\log_{1+\varepsilon} n \cdot t_{\mathsf{LP}})$, where t_{LP} is the time complexity for solving a linear program and $\varepsilon > 0$. Recently, Chekuri et al. [11] proposed another $(1+\varepsilon)$-approximation algorithm, `Flow-Approx`, by performing a limited number of blocking flows on the flow networks. However, in our experiments, we find that the flow-based approximation algorithm is not efficient enough. `Flow-Approx` [11] and the static version of [53] are the main competitors of our approximation algorithm.

3 Preliminaries

3.1 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 3.1 *(DDS problem* [4, 10, 19, 29, 31, 40]) Given a directed graph G=(V, E), return a DDS D=$G[S^*, T^*]$ of G.[2]

3.2 GPU architecture

A GPU card is comprised of many streaming multiprocessors (SMs) and device memory. Each SM is an independent hardware unit, which contains many cores and local fast-access memory, i.e., shared memory. The device memory (or GPU global memory) has longer latency but is accessible to all threads with high bandwidth. CUDA (Compute Unified Device Architecture) is the most popular GPU programming language created by Nvidia. It uses block as the programmable unit for programmers, which has many warps. A block will be assigned to only one SM in runtime. A warp, consisting of 32 threads, is the minimum granularity of instruction scheduling. Specifically, the scheduler can issue an eligible warp to compute units for executing the next instruction strictly following the lock-step rule. Hence branches inside of it will lead to some threads to idle. This phenomenon is called warp divergence, which is a major concern of the performance optimization on GPUs [55]. In Sect. 6, we propose an adaptive strategy to address the warp divergence issue.

4 From DDS to LP

In this section, we first introduce a linear programming (LP) formulation of the DDS problem (Sect. 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 (Sect. 4.2). Finally, we develop a Frank–Wolfe-based iterative algorithm to solve the DP (Sect. 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 $\frac{|S|}{|T|}$,

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

and for each particular ratio $\frac{|S|}{|T|}$=c, we formulate an $\mathsf{LP}(c)$ as follows:

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

The LP formulation tries to maximize the total weight of the edges subject to multiple constraints. The constraints include: (a) nonnegativity, whereby all variables must be nonnegative; (b) capacity constraints, which dictate that the weight of an edge (u, v) cannot exceed the minimum of the capacities of its endpoints, i.e., $x_{u,v} \leq s_u$ and $x_{u,v} \leq t_v$ (c) capacity sum constraints, which require that the sum of the capacities of the vertices must be fixed at a value related to the ratio of the sizes of two sets.

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 $\mathsf{LP}(c)$ for a fixed c, denoted by $\mathsf{OPT}(\mathsf{LP}(c))$, and the density of the DDS, and the connection will play a key role in reducing the number of LPs examined. The 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 $\mathsf{OPT}(\mathsf{LP}(c))$.

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

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

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

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

The proofs of Lemmas 4.1,4.2 can be obtained by following the proofs of Lemma 5 and 6 in [10], respectively. We provide the detailed proofs in Appendix (Sect. 9).

Combining Lemmas 4.1,4.2, we get Theorem 4.1.

Theorem 4.1 $\rho^* = \rho(S^*, T^*) = \max_c\{\mathsf{OPT}(\mathsf{LP}(c))\}$.

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

Theorem 4.1 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 $\mathsf{LP}(c)$ for a fixed c, we use the Frank–Wolfe method [16], which is one of the simplest and earliest known iterative optimizers. However, for $\mathsf{LP}(c)$, 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 $\mathsf{DP}(c)$ of $\mathsf{LP}(c)$. Hence, we first introduce the dual program $\mathsf{DP}(c)$ of $\mathsf{LP}(c)$. Then, based on the duality of $\mathsf{DP}(c)$, we can figure out the connection between the DDS and $\mathsf{OPT}(\mathsf{LP}(c))$ (which is also the optimal value of $\mathsf{DP}(c)$, denoted by $\mathsf{OPT}(\mathsf{DP}(c))$) 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 $\mathsf{DP}(c)$ of $\mathsf{LP}(c)$,

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

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

Definition 4.1 (*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

$$\rho_c(P, Q) = \frac{2\sqrt{c}\sqrt{c'}}{c + c'}\rho(P, Q) = \frac{2\sqrt{c}\sqrt{c'}}{c + c'}\frac{|E(P, Q)|}{\sqrt{|P| \cdot |Q|}}, \tag{4.1}$$

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

Definition 4.2 (*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 4.1 For the directed graph G shown in Fig. 3a, 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 $\mathsf{DP}(c)$, we can derive an upper bound of $\mathsf{OPT}(\mathsf{LP}(c))$, by exploiting the weak duality.

Lemma 4.3 (Upper bound of $\mathsf{OPT}(\mathsf{DP}(c))$) *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 $\mathsf{DP}(c)$ whose value is $\rho_c(S_c^*, T_c^*)$.*

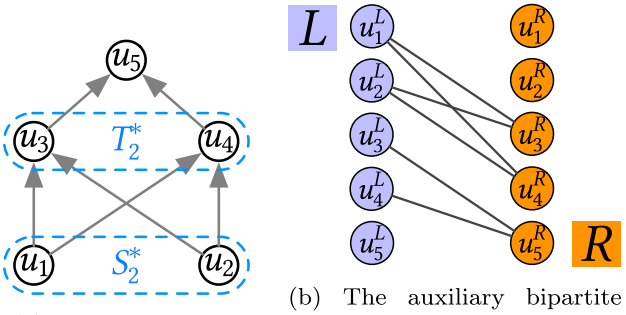
To facilitate the proof of Lemma 4.3, we introduce an auxiliary bipartite graph B and *propagable paths* defined on B.

Definition 4.3 *(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 3 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 4.4 *(Propagable path)* Given a feasible solution $(\alpha, \beta, \zeta, \eta, \phi)$ of $\mathsf{DP}(c)$, 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 $u_i^{\mathcal{I}} \in L$,
2. $\beta_{u_i, u_{i+1}} > 0, 0 \le i < k$, if $u_i^{\mathcal{I}} \in R$.



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

Fig. 3 A directed graph and its auxiliary bipartite graph

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

$$w(P_{u_0^{\mathcal{I}} \rightsquigarrow u_k^{\mathcal{I}}}) = \min(\{\alpha_{u_i, u_{i+1}} | u_i^{\mathcal{I}} \in L\} \cup \{\beta_{u_i, u_{i+1}} | u_i^{\mathcal{I}} \in R\}). \tag{4.2}$$

Proof of Lemma 4.3 We claim that there exists a feasible solution $(\alpha, \beta, \zeta, \eta, \phi)$ to $\mathsf{DP}(c)$ 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 $\mathsf{DP}(c)$. 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 that 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,

$$\begin{aligned} |E(S_c, T_c)| &= \sum_{(u,v) \in E(S_c, T_c)} (\alpha_{u,v} + \beta_{v,u}) \\ &> \zeta|S_c| + \eta|T_c| \\ &= \left(\frac{|S_c|}{\sqrt{c}} + \sqrt{c}|T_c|\right)\frac{\rho_c(S_c^*, T_c^*)}{2}. \end{aligned} \tag{4.3}$$

Further, we have

$$|E(S_c, T_c)| = \left(\frac{|S_c|}{\sqrt{c}} + \sqrt{c}|T_c|\right)\frac{\rho_c(S_c, T_c)}{2}. \tag{4.4}$$

Combining Equation (4.3) and Equation (4.4), we have $\rho_c(S_c, T_c) > \rho_c(S_c^*, T_c^*)$, which contradicts with the assumption made in Lemma 4.3 that $G[S_c^*, T_c^*]$ is the c-biased DDS. Hence, the lemma holds. □

Combining Lemmas 4.1,4.3, we can establish a connection between the c-biased DDS and $\mathsf{OPT}(\mathsf{LP}(c))$ by Theorem 4.2.

Theorem 4.2 *For a fixed c, let $G[S_c^*, T_c^*]$ be the c-biased DDS. We have* $\mathsf{OPT}(\mathsf{LP}(c)) = \mathsf{OPT}(\mathsf{DP}(c)) = \rho_c(S_c^*, T_c^*)$.

Proof We have $\mathsf{OPT}(\mathsf{LP}(c)) \geq \rho_c(S_c^*, T_c^*)$ by Lemma 4.1, and $\mathsf{OPT}(\mathsf{LP}(c)) \leq \rho_c(S_c^*, T_c^*)$ by Lemma 4.3 and weak duality. Thus, Theorem 4.2 holds by strong duality. □

Here, we use an example to illustrate further the correctness of Theorem 4.2.

Example 4.2 For $c = 2$, we can construct the optimal solutions for $\mathsf{LP}(c)$ and $\mathsf{DP}(c)$, whose value is exactly the c-biased density of c-biased DDS discussed in Example 4.1.

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

Comparison with the LP formulation in [10] After a detailed analysis of our $\mathsf{LP}(c)$ and $\mathsf{DP}(c)$, 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* $\mathsf{LP}(c)$. When $a = 1$ and $b = 1$, our LP formulation ($\mathsf{LP}(c)$) 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 $\mathsf{LP}(c)$ 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 4.2; 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 the 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* $\mathsf{DP}(c)$. The dual program in [10] minimizes $2\sqrt{c}\zeta + \frac{2}{\sqrt{c}}\eta$, while our $\mathsf{DP}(c)$ minimizes $\max(2\sqrt{c}\zeta, \frac{2}{\sqrt{c}}\eta)$. Hence, it can be treated that our $\mathsf{DP}(c)$ is equivalent to the dual program in [10] with one more constraint that $2\sqrt{c}\zeta = \frac{2}{\sqrt{c}}\eta$, because our $\mathsf{DP}(c)$ reaches the optimal when $2\sqrt{c}\zeta = \frac{2}{\sqrt{c}}\eta$ according to Lemma 4.3 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 $\mathsf{DP}(c)$, when c is fixed. To do this, we first simplify the $\mathsf{DP}(c)$ 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 $\mathbf{r}$:

$$\mathbf{r} = \langle r_\alpha(1), r_\alpha(2), \ldots, r_\alpha(n), r_\beta(1), r_\beta(2), \ldots, r_\beta(n)\rangle, \tag{4.5}$$

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 $\mathsf{DP}(c)$ can be rewritten as

$$\begin{aligned} \mathsf{DP}(c) \quad \min \quad & \|\mathbf{r}\|_\infty \\ \text{s.t.} \quad & \alpha_{u,v} + \beta_{v,u} = 1, && \forall(u, v) \in E \\ & 2\sqrt{c} \sum_{(u,v)\in E} \alpha_{u,v} = r_\alpha(u), && \forall u \in V \\ & \frac{2}{\sqrt{c}} \sum_{(u,v)\in E} \beta_{v,u} = r_\beta(v), && \forall v \in V \\ & \alpha_{u,v}, \beta_{v,u} \geq 0. && \forall(u, v) \in E \end{aligned} \tag{4.6}$$

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

Combining Theorems 4.2 and (4.6), 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 $\mathsf{DP}(c)$ and getting $\mathbf{r}$, we can get $G[S_c^*, T_c^*]$ by the following c-biased DDS construction method: (1) select the vertices of $\mathbf{r}$ 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 toward a minimizer of this function [27]. To linearize $\|\mathbf{r}\|_\infty$ at (α, β), we need the subgradient of $\|\mathbf{r}\|_\infty$, as $\|\mathbf{r}\|_\infty$ is convex but not differentiable. (4.7) gives a subgradient of $\|\mathbf{r}\|_\infty$.

$$\begin{aligned} \frac{\partial \|\mathbf{r}\|_\infty}{\partial \alpha_{u,v}} &= \frac{2\sqrt{c}}{|M|} \cdot \mathbb{1}_{r_\alpha(u)=\|\mathbf{r}\|_\infty}, && \forall (u,v) \in E; \\ \frac{\partial \|\mathbf{r}\|_\infty}{\partial \beta_{v,u}} &= \frac{2}{\sqrt{c} \cdot |M|} \cdot \mathbb{1}_{r_\beta(v)=\|\mathbf{r}\|_\infty}, && \forall (u,v) \in E; \end{aligned} \tag{4.7}$$

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

$$\begin{aligned} \widehat{\alpha}_{u,v} &= \mathbb{1}_{r_\alpha(u)<r_\beta(v) \vee r_\alpha(u)=r_\beta(v) \wedge c<1}, && \forall (u,v) \in E; \\ \widehat{\beta}_{u,v} &= \mathbb{1}_{r_\alpha(u)>r_\beta(v) \vee r_\alpha(u)=r_\beta(v) \wedge c\geq 1}, && \forall (u,v) \in E. \end{aligned} \tag{4.8}$$

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

Based on (4.7),(4.8), we can develop `Frank-Wolfe-DDS`, a variant of the Frank–Wolfe method [27], to optimize DP(c) in (4.6). Algorithm 1 presents the details, which takes input a directed graph G, the number of iterations N, and the ratio c, and outputs $(\mathbf{r}^{(N)}, \alpha^{(N)}, \beta^{(N)})$ after N iterations. First, it initializes $\alpha^{(0)}$, $\beta^{(0)}$, and $\mathbf{r}^{(0)}$ (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 $\|\mathbf{r}\|_\infty$ at $(\alpha^{(i-1)}, \beta^{(i-1)})$, denoted as $(\widehat{\alpha}, \widehat{\beta})$, is computed via (4.8) (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 $\mathbf{r}^{(i)}$ (lines 11–12).

Algorithm 1: A Frank-Wolfe-based algorithm.

1 **Function** `Frank-Wolfe-DDS`$(G = (V, E), N \in \mathbb{Z}_+, c)$**:**
2 **foreach** $(u, v) \in E$ **do** $\alpha^{(0)}_{u,v} \leftarrow \frac{1}{2}, \beta^{(0)}_{v,u} \leftarrow \frac{1}{2}$;
3 **foreach** $u \in V$ **do** $r^{(0)}_\alpha(u) \leftarrow 2\sqrt{c} \sum_{(u,v)\in E} \alpha^{(0)}_{u,v}$;
4 **foreach** $v \in V$ **do** $r^{(0)}_\beta(v) \leftarrow \frac{2}{\sqrt{c}} \sum_{(u,v)\in E} \beta^{(0)}_{v,u}$;
5 **for** $i = 1, \ldots, N$ **do**
6 $\gamma_i \leftarrow \frac{2}{i+2}$;
7 **foreach** $(u, v) \in E$ **do**
8 compute $\widehat{\alpha}_{u,v}, \widehat{\beta}_{v,u}$ via (4.8);
9 $\alpha^{(i)} \leftarrow (1 - \gamma_i) \cdot \alpha^{(i-1)} + \gamma_i \cdot \widehat{\alpha}$;
10 $\beta^{(i)} \leftarrow (1 - \gamma_i) \cdot \beta^{(i-1)} + \gamma_i \cdot \widehat{\beta}$;
11 **foreach** $u \in V$ **do** $r^{(i)}_\alpha(u) \leftarrow 2\sqrt{c} \sum_{(u,v)\in E} \alpha^{(i)}_{u,v}$;
12 **foreach** $v \in V$ **do** $r^{(i)}_\beta(v) \leftarrow \frac{2}{\sqrt{c}} \sum_{(u,v)\in E} \beta^{(i)}_{v,u}$;
13 **return** $(\mathbf{r}^{(N)}, \alpha^{(N)}, \beta^{(N)})$;

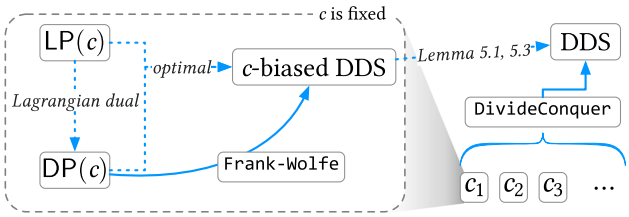


Fig. 4 Our algorithm framework

Theorem 4.3 (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 $\|\mathbf{r}^{(i)}\|_\infty - \rho^*_c \leq \varepsilon$.*

Proof For lack of space, we present the detailed proof in Appendix (Sect. 9). □

5 Fast LP solutions for DDS

In Sect. 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 Sect. 5.1. Under this framework, we design approximation and exact algorithms in Sects. 5.2,5.3, respectively.

5.1 Algorithm framework

Our proposed approximation and exact algorithms share the same framework, as depicted in Fig. 4. 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 Sect. 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 [40]. 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 setting x to c. Assume that there exists an $[S_x, T_x]$-induced subgraph, which satisfies $\min\{c_o, c_p\} \leq x = \frac{|S_x|}{|T_x|} \leq \max\{c_o, c_p\}$, but it has $\rho(S_x, T_x) > \rho(S_c^*, T_c^*)$. Since $h_c(x) \geq h_c(c_o)$ and $\rho(S_x, T_x) > \rho(S_c^*, T_c^*)$, we will 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 5.1.

Example 5.1 Reconsider Example 4.1. 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 $1 \leq \frac{|S|}{|T|} \leq 4$, 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 [40], which is helpful to reduce the size of the graph to be processed by `Frank-Wolfe-DDS`.

Definition 5.1 ($[x, y]$-*core* [40]) Given a directed graph G=(V, E), the $[x, y]$**-core** is the largest (S, T)-induced subgraph $G[S, T]$, which satisfies:

1. $\forall u \in S, d^+_{G[S,T]}(u) \geq x$ and $\forall v \in T, d^-_{G[S,T]}(v) \geq y$;
2. $\nexists G[S', T'] \neq G[S, T]$, 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 5.1 [40] *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 5.1, we can 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 5.1
4      repeat
5          (r, 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 DDSs and skipping the range of c values, i.e., `App-cDDS` and `Exact-cDDS`, will be discussed extensively in Sects. 5.2,5.3, respectively.

Under the convex-programming-based framework (Algorithm 2), to compute the (1+ε)-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+ε)-approximation algorithm

We begin with an interesting Lemma:

Lemma 5.2 *Given a directed graph $G = (V, E)$, a positive real value ε, and $c^* = \frac{|S^*|}{|T^*|}$, if c satisfies that $\sqrt{c^*} \cdot \frac{1}{1+\varepsilon} \leq \sqrt{c} \leq \sqrt{c^*} \cdot (1 + \varepsilon)$, we have*

$$\frac{\rho^*}{\rho_c^*} \leq 1 + \varepsilon, \tag{5.1}$$

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 $\rho_c^* \geq \frac{2}{\frac{\sqrt{c}}{\sqrt{c^*}}+\frac{\sqrt{c^*}}{\sqrt{c}}}\rho(S^*, T^*)$. Since c satisfies $\frac{\sqrt{c}}{\sqrt{c^*}} \leq 1+\varepsilon$ and $\frac{\sqrt{c^*}}{\sqrt{c}} \leq 1+\varepsilon$, we can easily conclude that $\rho_c^* \geq \frac{2}{\frac{\sqrt{c}}{\sqrt{c^*}}+\frac{\sqrt{c^*}}{\sqrt{c}}}\rho(S^*, T^*) \geq \frac{1}{1+\varepsilon}\rho^*$. Hence, Lemma 5.2 holds. □

Clearly, Lemma 5.2 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., $[\frac{1}{n}, \frac{1}{n}(1+\varepsilon)^2]$, $[\frac{1}{n}(1+\varepsilon)^2, \frac{1}{n}(1+\varepsilon)^4]$, $\ldots$, $[\frac{1}{(1+\varepsilon)^2}n, n]$, 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 5.1 *For a fixed c, let $(\alpha, \beta, \mathbf{r})$ be a feasible solution of $\mathsf{DP}(c)$. For $G[S_c, T_c]$ satisfying $\frac{\|\mathbf{r}\|_\infty}{\rho_c(S_c,T_c)} \leq 1+\varepsilon$, 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 $\min\{c_o, c_p\} \leq \frac{|S|}{|T|} \leq \max\{c_o, c_p\}$, then $\rho(S, T) \leq (1+\varepsilon) \cdot \rho(S_c, T_c)$, where $\varepsilon \in \mathbb{R}_+$.*

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

Corollary 5.2 *For a fixed c, let $(\alpha, \beta, \mathbf{r})$ be a feasible solution of $\mathsf{DP}(c)$. Suppose $G[S_c, T_c]$ satisfies $\frac{\|\mathbf{r}\|_\infty}{\rho_c(S_c,T_c)} \leq \sqrt{1+\varepsilon}$. For any (S, T)-induced subgraph $G[S, T]$, if $\frac{c}{1+\varepsilon} \leq \frac{|S|}{|T|} \leq c \cdot (1+\varepsilon)$, then $\rho(S, T) \leq (1+\varepsilon) \cdot \rho(S_c, T_c)$.*

Proof According to Lemma 5.2, we have $\frac{\rho(S,T)}{\rho_c^*} \leq \sqrt{1+\varepsilon}$, where $\frac{c}{1+\varepsilon} \leq \frac{|S|}{|T|} \leq c \cdot (1+\varepsilon)$. Further, we have $\frac{\rho_c^*}{\rho(S_c,T_c)} \leq \frac{\|r\|_\infty}{\rho_c(S_c,T_c)} \leq \sqrt{1+\varepsilon}$. Multiplying the two inequalities, we have $\frac{\rho(S,T)}{\rho_c^*} \cdot \frac{\rho_c^*}{\rho(S_c,T_c)} \leq 1+\varepsilon$. Hence, the corollary holds. □

Based on Corollaries 5.1,5.2, we propose a strategy for reducing the number of c values to be examined. We use Fig. 5 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 5.1 and Corollary 5.2. When the interval $[\frac{c}{1+\varepsilon}, c \cdot (1+\varepsilon)]$ covers $[c_o, c_p]$, then only Corollary 5.2 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 $c_o \leq \frac{c}{1+\varepsilon} \leq c(1+\varepsilon) \leq c_p$ or $\frac{c}{1+\varepsilon} \leq c_o \leq c_p \leq c(1+\varepsilon)$. In the two cases, the number of

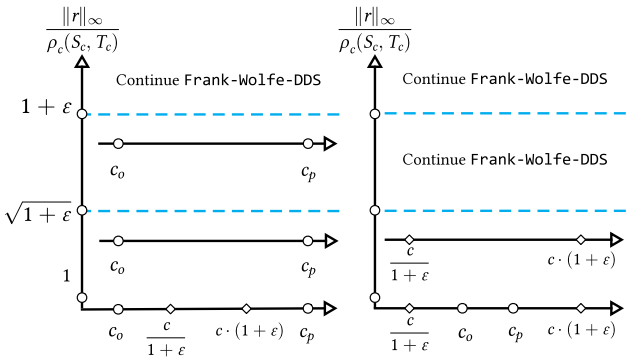


Fig. 5 The strategy of reducing the number of c values

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 $\mathsf{DP}(c)$ and getting $\mathbf{r}$, we can get the approximate c-biased DDS by slightly modifying the construction method in Sect. 4.3. That is, we sort the vertices of $\mathbf{r}$ 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 $\mathbf{r}$ 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). Afterward, each vertex is inserted into S_c (resp. T_c) if its corresponding vertex is contained in 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 5.1 (lines 11–12). Finally, it checks whether the approximate DDS candidate satisfies the conditions in Corollaries 5.1,5.2 and returns the DDS candidate as well as the range of c to be skipped (lines 14–16).

Complexity The time complexity of `CP-Approx` is $O(\log_{1+\varepsilon} n \cdot t_{\mathsf{FW}})$. t_{FW} denotes the time complexity of `Frank-Wolfe-DDS`, and its convergence rate is provided by Theorem 4.3.

Comparison with state-of-the-art `VW-Approx` [53] is the state-of-the-art $(1+\varepsilon)$-approximation algorithm. `VW-Approx` transforms the DDS problem into $O(\log_{1+\varepsilon} n)$ vertex-weighted undirected densest subgraph problems, where the vertex weights are set according to $O(\log_{1+\varepsilon} n)$ 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.

Algorithm 3: Extract approximate c-biased DDS.

Function App-cDDS$(G = (V, E), r, \varepsilon, c)$**:**
  $\rho_c^* \leftarrow 0$, $S_c^*, T_c^* \leftarrow \emptyset$, $S_c, T_c \leftarrow \emptyset$;
  sort the nodes according to $\mathbf{r}$: $r(u_1) \geq r(u_2) \geq \cdots \geq r(u_{2n})$;
  $L \leftarrow \{u | r_\alpha(u) \in \mathbf{r}\}$, $R \leftarrow \{v | r_\beta(v) \in \mathbf{r}\}$;
  for $i = 1, \ldots, 2n$ **do**
    if $u_i \in L$ **then** $S_c \leftarrow S_c \cup \{u_i\}$;
    else $T_c \leftarrow T_c \cup \{u_i\}$;
    if $S_c = \emptyset$ *or* $T_c = \emptyset$ **then continue**;
    if $\rho_c(S_c, T_c) > \rho_c^*$ **then**
      $\rho_c^* \leftarrow \rho_c(S_c, T_c)$, $S_c^* \leftarrow S_c$, $T_c^* \leftarrow T_c$;
  $c_o \leftarrow \frac{|S_c^*|}{|T_c^*|}$, $c_p \leftarrow \frac{c^2}{c_o}$; // Corollary 5.1
  if $c_o > c_p$ **then** Swap(c_o, c_p);
  $\delta \leftarrow \frac{r_{u_1}}{\rho_c^*}$;
  if $\delta \leq \sqrt{1+\varepsilon}$ **then return** $(S_c^*, T_c^*, \min\{c_o, \frac{c}{1+\varepsilon}\}, \max\{c_p, c \cdot (1+\varepsilon)\}, \text{True})$;
  else if $\delta \leq 1 + \varepsilon \wedge c_o < \frac{c}{1+\varepsilon} \wedge c \cdot (1+\varepsilon) < c_p$ **then return** $(S_c^*, T_c^*, c_o, c_p, \text{False})$;
  else return $(S_c^*, T_c^*, c_o, c_p, \text{False})$;

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 5.1). The worst case of the number of values of $\frac{|S|}{|T|}$ examined in CP-Approx is also $O(\log_{1+\varepsilon} n)$ (Corollary 5.2), but Corollary 5.1 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 4.2). 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 5.1) 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 4.3. 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 [13] and then present the necessary and sufficient conditions of verifying whether a stable (S, T)-induced subgraph is the exact c-biased DDS.

Definition 5.2 *(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 $(\mathbf{r}, \alpha, \beta)$ to DP(c), if the following conditions hold:

1. $\min\left\{\min_{u \in S}\{r_\alpha(u)\}, \min_{v \in T}\{r_\beta(v)\}\right\} > \max\left\{\max_{u \in V \setminus S}\{r_\alpha(u)\}, \max_{v \in V \setminus T}\{r_\beta(v)\}\right\}$;
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 5.2, 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.

Example 5.2 Reconsider the graph G in Fig. 3a. Given c=2, $G[S = \{u_1, u_2\}, T = \{u_3, u_4\}]$ is stable with respect to the feasible solution $(\alpha, \beta, \mathbf{r})$ 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 5.2 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 $\mathbf{r}$. 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 5.3 *For a fixed c, suppose an (S, T)-induced subgraph $G[S, T]$ is stable with respect to some feasible solution $(\mathbf{r}, \alpha, \beta)$ 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 (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

Algorithm 4: Verify c-biased DDS.

1 **Function** `Is-cDDS` $(G[S,T], c)$:
2 $L \leftarrow \{u^L | u \in S\}$, $R \leftarrow \{u^R | u \in T\}$;
3 $V_F \leftarrow \{s\} \cup L \cup R \cup \{t\}$;
4 **for** $u^R \in R$ **do** add (s, u^R) to E_F with capacity $d^+_{G[S,T]}(u)$;
5 **for** $u^L \in L$ **do** add (u^L, t) to E_F with capacity $\frac{\rho_c(S,T)}{2\sqrt{c}}$;
6 **for** $u^R \in R$ **do** add (u^R, t) to E_F with capacity $\frac{\sqrt{c}\rho_c(S,T)}{2}$;
7 **for** $(u, v) \in E(S, T)$ **do** add (u^R, u^L) to E_F with capacity 2;
8 $f \leftarrow$ maximum flow from s to t;
9 **return** $f = |E(S, T)|$;

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 Appendix (Sect. 9). □

Lemma 5.3 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 [40, 56], 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 5.2.

Theorem 5.2 (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 5.4 *Given a feasible vector $\mathbf{r}$ in $\mathsf{DP}(c)$ 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*

$$|E(S,T)| \leq \left\lfloor \frac{1}{2\sqrt{c}} \sum_{i=1}^{|S|} r_\alpha(u_i) + \frac{\sqrt{c}}{2} \sum_{i=1}^{|T|} r_\beta(u_i) \right\rfloor. \quad (5.2)$$

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, $|E(S,T)| \leq \lfloor \frac{1}{2\sqrt{c}} \sum_{v \in S} r_\alpha(v) + \frac{\sqrt{c}}{2} \sum_{v \in T} r_\beta(v) \rfloor \leq \left\lfloor \frac{1}{2\sqrt{c}} \sum_{i=1}^{|S|} r_\alpha(u_i) + \frac{\sqrt{c}}{2} \sum_{i=1}^{|T|} r_\beta(u_i) \right\rfloor$, 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 5.2 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 $(\alpha, \beta) \in \mathsf{DP}(c)$ 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 $\mathbf{r}$ 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 $\mathbf{r}$ is equal to $\rho_c(S, T)$. Then, Lemma 5.4 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 $c' = \frac{|X|}{|Y|}$. According to Lemma 5.3, 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 $(\alpha, \beta, \mathbf{r}) \in \mathsf{DP}(c)$ 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 4.3 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 5.3 Following Example 5.2, 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 based on Lemma 5.1 (lines 3–4). Afterward, we check whether the extracted subgraph is a stable (S, T)-induced subgraph via Definition 5.2 (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 5.3 (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).

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 4.3.

Comparison with the state-of-the-art `DC-Exact` [40] is the state-of-the-art exact DDS algorithm enhanced with the

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 5.2
6          if Is-cDDS(G[S*_c, T*_c], c) then    // Theorem 5.2
7              return (S*_c, T*_c, c_o, c_p, True);
8          update G as G[S*_c, T*_c];            // Lemma 5.3
9      return (S*_c, T*_c, c_o, c_p, False);
```

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 [40] 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 $\mathsf{DP}(c)$ and $\mathsf{LP}(c)$ 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 $O(nm)$ [48]. 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 5.3) and the optimality test by max-flow (Theorem 5.2) 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 [40] 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 GPU-enabled DDS algorithms

With careful design and elegant pruning strategies, we are able to ensure that our algorithms are much faster than the state-of-the-art algorithms. However, for very large-scale graphs, users may still need to wait for a long time to obtain the DDS. For example, we need to wait for around three hours to obtain the DDS for dataset SK, a graph with around 2 billion edges (dataset characteristics in Table 2). After profiling our DDS algorithms, we find that Frank–Wolfe computation takes the majority of the total running time. For example, `Frank-Wolfe-DDS` takes more than 70% of the whole processing time to compute the exact DDS on dataset SK. Fortunately, the updates of edge weights assigned to vertices (via $\hat{\alpha}$, $\hat{\beta}$, and $\mathbf{r}$) in each Frank–Wolfe iteration are amenable to parallelization, which can be achieved via GPU computing. Nevertheless, the trivial parallelization strategy of mapping each vertex to one GPU thread may cause workload imbalance and computing resource underutilization due to the skewed vertex degree distribution present in many real-world graphs. Next, we will study parallelization for Frank–Wolfe iterations, analyze the workload imbalance caused by the skewed degree distribution, and design strategies to remedy the imbalance issue.

In the i-th Frank–Wolfe iteration (Algorithm 1), there are two major steps:

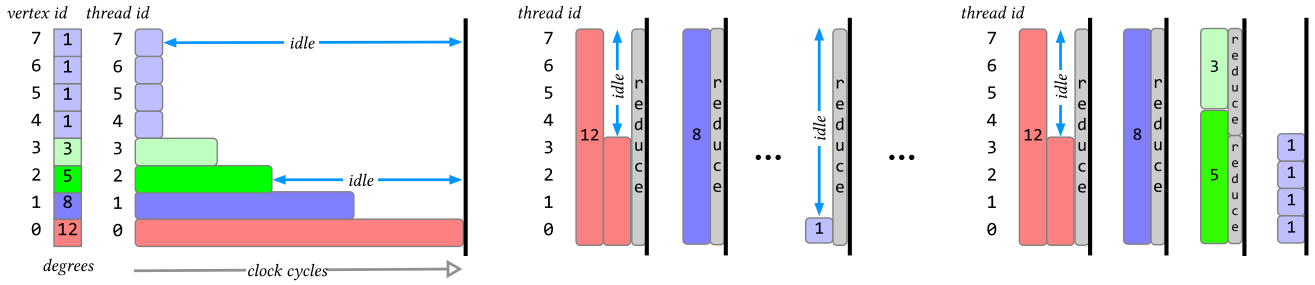
1. update $\alpha^{(i)}$ and $\beta^{(i)}$ via $\hat{\alpha}$ and $\hat{\beta}$;
2. compute $r^{(i)}$ based on $\alpha^{(i)}$ and $\beta^{(i)}$.

To update $\alpha^{(i)}$ and $\beta^{(i)}$ in parallel, we can map each edge to a thread, because there is no dependence and conflict when we parallelize by edges. Algorithm 6 gives the pseudo-code for updating $\alpha^{(i)}$. As the threads are organized in blocks on GPU, we use the block information (i.e., `block.id` and `block.size`) and the thread ID within the block (i.e., `thread.id`) to globally identify each thread and locate the edge mapped to the thread (line 1). Next, we update $\alpha^{(i)}_{u,v}$ based on the values of $r^{(i-1)}_{\alpha}(u)$ and $r^{(i-1)}_{\beta}(v)$ (lines 2–4). Similarly, we can also update the $\beta^{(i)}$ values.

Algorithm 6: Update $\alpha^{(i)}$ via GPU

```
Input  : r^(i−1), γ_i
Output : α^(i)
// map edge to thread
1  e = (u, v) ← block.id×block.size+thread.id;
2  if r_α^(i−1)(u) < r_β^(i−1)(v) then
3      α^(i−1)_{u,v} ← (1 − γ_t) · α^(i)_{u,v} + γ_t
4  else α^(i)_{u,v} ← (1 − γ_t) · α^(i−1)_{u,v};
```

To compute $r^{(i)}$ in parallel, a straightforward solution is to assign each vertex to a thread. Each thread is responsible for aggregating the $\alpha^{(i)}$ or $\beta^{(i)}$ values corresponding to the adjacent edges of the assigned vertex. Algorithm 7 presents the pseudo-code. The algorithm first assigns each vertex to a thread (line 1). Next, each thread aggregates the $\alpha^{(i)}$ values



(a) thread (b) block (c) adaptive

Fig. 6 GPU parallel strategies

associated with the edges incident on u to compute $r_u^{(i)}$ (lines 2–4).

Algorithm 7: Compute $r_\alpha^{(i)}$ by thread→vertex

Input : $\alpha^{(i)}, c$
Output : $r_\alpha^{(i)}$

```
// map vertex to thread
1 u ← block.id×block.size+thread.id;
2 foreach (u, v) ∈ E do
3     r_α^(i)(u) ← r_α^(i)(u) + 2√c · α_{u,v}^(i)
```

However, the degree distributions of most real networks are not uniform, and even follow a power law, at least asymptotically [12]. Meanwhile, the threads within a warp follow the same instruction in each step. Hence, there may be some idle threads in the block, as the number of clock cycles in the block is determined by the vertex with the largest degree. Our next example illustrates the idle threads in Fig. 6a.

Example 6.1 Assume that a block has one warp consisting of 8 threads,[3] and the degrees of the vertices assigned to the threads are as listed on the left of Fig. 6a. We can find that many threads are idle after a few cycles, as the corresponding vertices have low degrees while vertex 0 has a large degree (i.e., 12).

Another strategy is to assign each vertex u to a block, and aggregate $\alpha_{u,v}^{(i)}$, where $(u, v) \in E$, to obtain $r_\alpha^{(i)}(u)$ in parallel within the block. Algorithm 8 gives the pseudo-code. We first map each vertex to a block (line 1). Next, each thread in the block sums up a partition of α values into *sum* (a local variable in each thread) according to their thread id in the block (lines 2–6). Next, we aggregate *sum* from all threads (line 7) and compute $r_\alpha^{(i)}(u)$ (line 8).

By assigning each vertex to a block, we can reduce the idle clock cycles within one block, as shown in Fig. 6b. However,

[3] In a real GPU, a block can have several warps, and a warp contains 32 threads. Here, we use small numbers for illustration.

Algorithm 8: Compute $r_\alpha^{(i)}$ by block→vertex

Input : $\alpha^{(i)}, c$
Output : $r_\alpha^{(i)}$

```
1 u ← block.id;                    // map vertex to block
2 k ← thread.id, sum ← 0;
3 while k ≤ d_G^+(u) do
4     v ← k-th out-neighbor of u;
5     sum ← sum + α_{u,v}^(i);
6     k = k + block.size;
  // reduction in the block
7 r_α^(i)(u) ← aggregate sum via all threads within the block in
    parallel;
8 r_α^(i)(u) ← 2√c · r_α^(i)(u);
```

for the vertices with very small degrees, it is extravagant to assign each of them to a block. For example, in the 4 blocks handling vertices 4–7 of Fig. 6b, each block only has one working thread, which means that 87.5% threads in those blocks are idle. Inspired by the block strategy, we also implement a warp strategy, i.e., assigning one vertex to a warp. Empirically, the warp strategy performs better than the block strategy. This means the warp strategy can reduce the computing resource waste compared to the block strategy. However, it still suffers from the imbalance among vertex degrees, as the warp size is 32 and there can be many vertices with degrees much less than 32.

To reduce the idle resources caused by the above three strategies, we propose a new adaptive strategy, depicted in Fig. 6c, to allocate computing resources to each vertex according to its degree. To achieve this goal, we first build an auxiliary array `b2v` to keep track of which vertices are assigned to a thread block. If the degree of a vertex is not smaller than the block size, the vertex can occupy a block alone; otherwise, it needs to share a block with other vertices. For example, Fig. 7 depicts the vertex assignment on blocks for the vertices we presented in Fig. 6. We can find that block 0 only has vertex 0 with degree 12, but vertices 2 and 3 need to share block 2 together. Vertices 4 to 7 are assigned to block 3, as 8 in `b2v[4]` is outside the range.

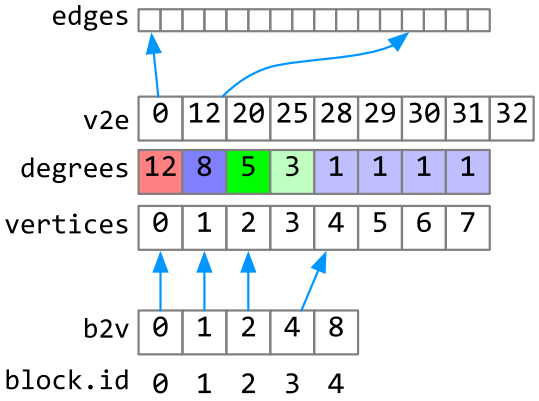


Fig. 7 Vertex assignment on blocks

Algorithm 9: Compute $r_\alpha^{(i)}$ adaptively

Input : $\alpha^{(i)}$, c, b2v, v2e
Output : $r_\alpha^{(i)}$

1 $\delta \leftarrow$ b2v[block.id + 1] - b2v[block.id];
2 **if** $\delta = 1$ **then**
3 $u \leftarrow$ b2v[block.id];
4 reuse lines 2-8 from Algorithm 8;
5 **else if** $\delta \geq$ block.size / 2 **then**
6 $u \leftarrow$ b2v[block.id]+thread.id;
7 reuse lines 2-3 from Algorithm 7;
8 **else**
9 $\tau \leftarrow$ block.size/δ;
10 round τ to the previous power of 2;
11 $u_l \leftarrow$ b2v[block.id], $u_r \leftarrow$ b2v[block.id+1];
12 $e_l \leftarrow$ v2e[u_l], $e_r \leftarrow$ v2e[u_r];
  // *sum* is shared in block
13 **if** e_l+thread.id< e_r **then**
14 *sum*[thread.id] $\leftarrow \alpha^{(i)}_{e_l+\text{thread.id}}$;
15 $u \leftarrow u_l$+thread.id/τ;
16 **if** $u < u_r$ **then**
17 $k \leftarrow u - u_l$;
18 $i_l \leftarrow$v2e[u]$-$v2e[u_l], $i_r \leftarrow$v2e[$u+1$]$-$v2e[u_l];
19 $r_\alpha^{(i)}(u) \leftarrow$ aggregate *sum*$[i_l : i_r - 1]$ via k-th τ threads in the block in parallel;
20 $r_\alpha^{(i)}(u) \leftarrow 2\sqrt{c} \cdot r_\alpha^{(i)}(u)$;

Based on b2v, the vertex assignment array, Algorithm 9 presents the pseudo-code for the adaptive strategy. For the current block, we first obtain δ, the number of vertices assigned to this block, via b2v (line 1). If only one vertex v is assigned, we compute the r_v value in parallel by all threads within the block, following the strategy in Algorithm 8 (lines 2–4). Similarly, if the number of vertices allocated is at least half of the block size, then we assign each vertex to a thread within the block, following the strategy in Algorithm 7 (lines 5–7). If the number of assigned vertices is within the range $[2, block.id/2)$, we first read all related α values from global memory into *sum*, which resides in the shared memory of the block (lines 11–14), and allocate τ threads to compute the r value for each vertex in parallel (lines 15–20). In detail, we first obtain τ, the number of threads assigned to each vertex (lines 9–10). Here, we round τ to the previous power of 2 to facilitate the later aggregation. Next, we obtain the index range $[e_l, e_r)$ of the related α values via the auxiliary arrays (lines 11–12). Afterward, for each assigned vertex u, we denote u as the k-th vertex in the block (line 17), identify the index range of the related *sum* values (line 18), allocate k-th τ threads to aggregate the *sum* values in parallel (line 19), and compute the final r_u value (line 20).

Example 6.2 From Fig. 6, we can find that the adaptive strategy has the advantages of both the thread and the block strategies. By allowing vertices with large degrees to occupy blocks alone, we reduce the idle computing resources in the thread strategy. For example, vertex 0 with a degree of 12 occupies a block alone. By assigning multiple vertices with smaller degrees to a block, we reduce the idle resources in the block strategy. For example, vertices 2 (with degree 5) and 3 (with degree 3) are assigned to one block, and the four vertices with a degree of 1 are also assigned to one block.

We can find that in the adaptive strategy, threads in the same block share the same control flow, while different blocks can have different computing strategies. The former ensures we will not suffer from warp divergence, and the latter employs different methods for different degree distributions.

Algorithms 7,8,9 focus on computing $r_\alpha^{(i)}$ from $\alpha^{(i)}$. Similarly, the algorithms can also be used to compute $r_\beta^{(i)}$ with minor changes.

Complexity We follow the work-span framework of [54] to analyze the complexity of a Frank–Wolfe iteration, which is performed in parallel here.

Compared to serial computation, parallel computation has two components of extra work, i.e., the graph data copy operation between CPU memory and GPU memory and the reduction process in each block, for the block and adaptive strategies. The time cost of the former is linear in the graph size, and the latter one is linear in the number of blocks, which is also linear in the graph size. Hence, the work of a Frank–Wolfe iteration is still $O(m)$. The span (or depth) differs for different strategies, as summarized in Table 1.

7 Experiments

In this section, we first introduce the experimental setup in Sect. 7.1 and then present the experimental results of approximation algorithms and exact algorithms in Sects. 7.2 and 7.3, respectively.

Table 1 Work and span per Frank–Wolfe iteration

Strategy	Work	Span		
		Per block	# blocks	Per iter
Thread	$O(m)$	$O(d_{\max})$	$O(\frac{n}{s_b})$	$O(\frac{n \cdot d_{\max}}{s_b \cdot s_g})$
Block		$O(\frac{d_{max}}{s_b})$	$O(n)$	$O(\frac{n \cdot d_{\max}}{s_b \cdot s_g})$
Adaptive		$O(\frac{d_{max}}{s_b})$	$O(b)$	$O(\frac{b \cdot d_{\max}}{s_b \cdot s_g})$

s_b denotes the size of a thread block. s_g denotes the number of concurrent blocks in the GPU. b is the number of blocks in the adaptive strategy, where $\frac{n}{s_k} \leq b \leq n$

Table 2 Directed graphs used in our experiments

Dataset	Full name	Category	$\|V\|$	$\|E\|$
MO [17]	moreno-oz	Human Social	217	2672
TC [1]	maayan-faa	Infrastructure	1226	2615
OF [47]	openflights	Infrastructure	2939	30.5K
AD [43]	advogato	Social	6541	51K
AM [35]	amazon	E-commerce	403K	3.38 M
AR [45]	amazon-ratings	E-commerce	3.38 M	5.84 M
BA [46]	baidu-zhishi	Hyperlink	2.14 M	17.6 M
SK [51]	web-sk-2005-all	Web	50.6 M	1.95B

7.1 Setup

We use eight real datasets [33] which are publicly available.[4] 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 2 summarizes their statistics.

We compare the following approximation DDS algorithms:

- `CP-Approx` is our proposed approximation algorithm (Sect. 5.2).
- `Core-Approx` [40] is the state-of-the-art 2-approximation algorithm.
- `VW-Approx` [53] is the state-of-the-art $(1+\varepsilon)$-approximation algorithm (briefed in Sect. 5.2).
- `Flow-Approx` [11] is the max-flow-based state-of-the-art $(1+\varepsilon)$-approximation algorithm.

We also compare the following exact DDS algorithms:

- `CP-Exact` is our proposed exact algorithm (Sect. 5.3).
- `DC-Exact` [40] is the state-of-the-art exact algorithm enhanced with the divide-and-conquer strategy and elegant core-based pruning techniques.

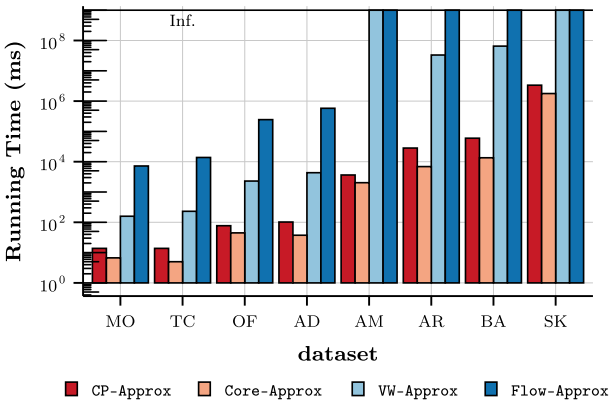


Fig. 8 Efficiency of approximation algorithms

- `Core-Exact` [40] is simplified version of `DC-Exact` without using the divide-and-conquer strategy.
- `Flow-Exact` [31] 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. [40] provides code for `Flow-Exact`, `DC-Exact`, `Core-Exact`, and `Core-Approx`. No source code is available for other algorithms, we implement code for them. Our source code is publicly available.[5] Due to space limit, additional experimental results can also be found in the code repo. We ran all the experiments on a machine having an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz processor, a GeForce GTX 2080 Ti 11GB GPU, and 256GB memory, with Ubuntu installed.

7.2 Approximation algorithms

In this section, we mainly compare our approximation algorithm `CP-Approx` with the state-of-the-art $(1+\varepsilon)$-approximation algorithms `VW-Approx` [53], `Flow-Approx` [11], and the state-of-the-art 2-approximation algorithm `Core-Approx` [40].

7.2.1 Efficiency comparison

In this experiment, we evaluate the efficiency of four approximation algorithms, i.e., `CP-Approx`, `VW-Approx`, `Flow-Approx` and `Core-Approx`, with $\varepsilon = 1$. Note that `Core-Approx` only provides the 2-approximation DDS, and the efficiency result of `CP-Approx`, `VW-Approx`, and `Flow-Approx` w.r.t. different values of ε will be presented later. Figure 8 reports the efficiency result of the four algorithms. The datasets are ordered by graph size on the x-axis.

[4] http://konect.uni-koblenz.de/networks/.

[5] https://github.com/chenhao-ma/DDS-convex-code.

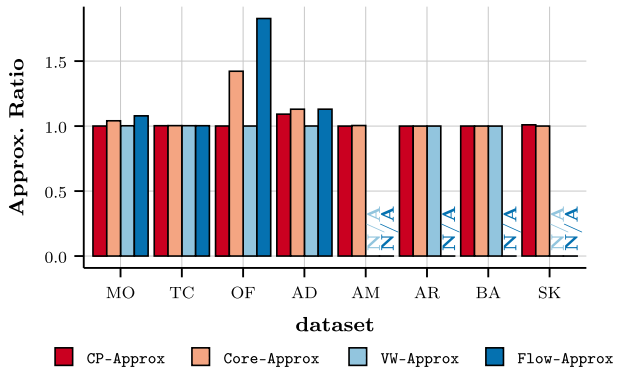


Fig. 9 Actual approx. ratios of approx. algorithms

Notice that for some datasets, the bars of VW-Approx and Flow-Approx touch the solid upper line, which means they cannot finish within one week on those datasets. From Fig. 8, we can make 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. CP-Approx offers more flexibility to control the accuracy guarantee, and it is reasonable to pay for the flexibility. The time complexity of CP-Approx is also slightly higher than Core-Approx when $\varepsilon = 1$. Second, CP-Approx is at least 10× and up to five orders of magnitude faster than VW-Approx and Flow-Approx. We have listed three reasons on why CP-Approx is faster than VW-Approx at the end of Sect. 5.2: fewer trials of different $\frac{|S|}{|T|}$, tighter error estimation, and the smaller size of graph to be processed. Flow-Approx is slow because it typically needs to perform the blocking flow computations on the whole graph.

7.2.2 Accuracy comparison

We present the actual approximation ratios of all the four approximation algorithms in Fig. 9 with $\varepsilon = 1$. Specifically, for each dataset, we first obtain the exact DDS via CP-Exact, then compute the approximate DDSs using those approximation algorithms, and get the actual approximation ratio (i.e., the density of the exact DDS over those of approximate DDSs). Some bars are missing for VW-Approx and Flow-Approx, as they cannot finish within one week on those datasets. From Fig. 9, 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 is slightly larger than that of VW-Approx. Flow-Approx provides the worst actual approximation ratios on dataset OF. Besides, most ratios of CP-Approx are smaller than those of Core-Approx (except on SK), and CP-Approx offers more flexibility on the approximation guarantee since ϵ can be any positive real values. The flexibility further helps explore DDS of higher or lower density.

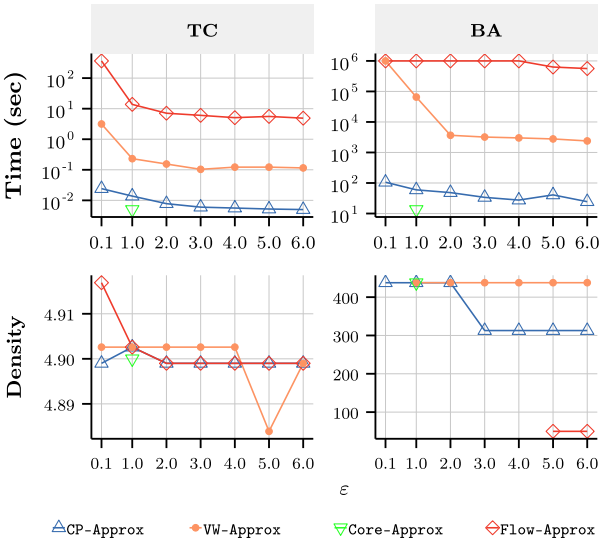


Fig. 10 Effect of ε

7.2.3 Effect of ε

We evaluate the effect of ε on the efficiency and accuracy of the three $(1+\varepsilon)$-algorithms, i.e., CP-Approx, VW-Approx and Flow-Approx. Figure 10 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. 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 and Flow-Approx touches the solid upper line for some cases, which means the corresponding algorithm cannot finish within one week on those cases. Hence, the density plot of VW-Approx is also missing in that case. From Fig. 10, we can observe: First, for all three algorithms, 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. Third, the improvement of CP-Approx over Flow-Approx is quite significant in terms of both running time and accuracy.

Table 3 Statistics of DDSs w.r.t. different ε values on AD

ε	Density	$\|S\|$	$\|T\|$	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

7.2.4 A case study: parameter selection of ε.

In this case study, we investigate the approximate DDSs returned by `CP-Approx` under different ε values compared to the exact DDS. Table 3 reports the statistics of the exact DDS and three approximate DDSs (with $\varepsilon = 0.1$, 1 and 2, respectively) on the AD dataset. In terms of the density, we observe that all the three approximate DDSs 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 DDSs 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 DDSs returned with different values of ε in downstream applications (e.g., fake follower detection), as our `CP-Approx` is quite efficient.

7.3 Exact algorithms

7.3.1 Efficiency comparison

In Fig. 11, 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 Sect. 5.3: avoiding the repeated max-flow computation; early stop in the inner loop; and 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 4, 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 Sect. 5.1 (noted as "Avg #edges"), and the product 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.

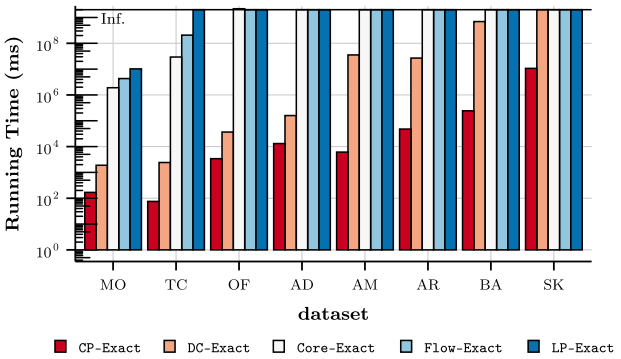


Fig. 11 Efficiency of exact algorithms

Table 4 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	288,142	3.17×10^8
SK	23	121.74	407×10^7	1.14×10^{11}

7.3.2 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 5 reports the running time of `CP-Exact` and `CP-Exact-ab` over different datasets. `CP-Exact-ab` cannot finish reasonably on SK, so 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

Table 5 The running time of `CP-Exact` and `CP-Exact-ab`

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

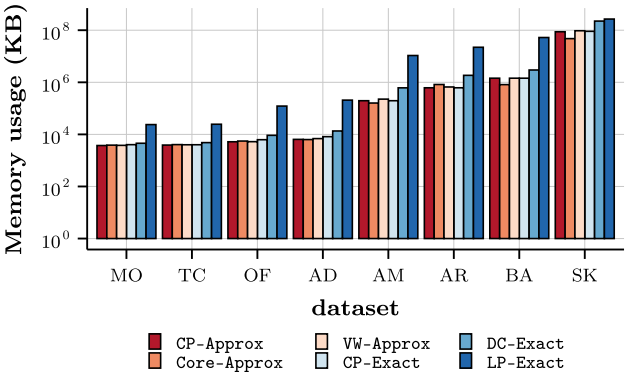


Fig. 12 Memory usage of algorithms

Table 6 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

subgraph and the max-flow are effective to reduce the `Frank-Wolfe-DDS` iterations.

7.4 Memory usage

We report the maximum memory usage of all algorithms in Fig. 12. The memory usage of `Flow-Exact` and `Core-Exact` is omitted because the results are very similar to that of `DC-Exact`. 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. Among those algorithms, `LP-Exact` needs more memory than others because we implemented `LP-Exact` via Google OR-Tools, which materializes all constraints in the LPs.

7.5 Comparing `CP-Exact` and `CP-Approx`

In Table 6, 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 4.3. 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 DDSs via different ε.

7.6 GPU parallel evaluation

To examine the effects of our different parallelization strategies, we evaluate the running time of the Frank–Wolfe iterations for each strategy with varying block sizes on all eight datasets. Figure 13 reports the running time results under these settings. We make the following observations from the figure. (1) In most cases, especially on large graphs, the adaptive strategy is the most efficient compared to the other two strategies. On large datasets, the adaptive strategy can be around 10× faster than the other two strategies. (2) Usually, a larger thread block size is preferable in the adaptive strategy, especially for large graphs. We can find that a block size of 1024 has the best performance on the three largest datasets and has a similar running time to the other two methods on the remaining datasets. (3) In the block strategy and the thread strategy, a smaller block size is desirable in almost all cases. Besides, the block strategy with a block size of 64 usually performs better than the thread strategy with the same block size. We reckon the reason is that the vertices with larger degrees can receive more computing resources, and meanwhile, the smaller block size will not cause too many idle threads for vertices with small degrees. (4) The warp strategy usually outperforms the block strategy, and is outperformed by the adaptive strategy, as the warp size is 32 and there can be many vertices with degrees much less than 32.

To gain further insights into the performance differences among the various strategies, we report two metrics in Table 7: *Avg. Active Threads Per Warp*, which is the average number of active threads in a warp, and *Achieved Active Warps Per SM*, which is the average number of active warps per SM during kernel execution with block size 1024. We observe that the adaptive and block strategies have similar

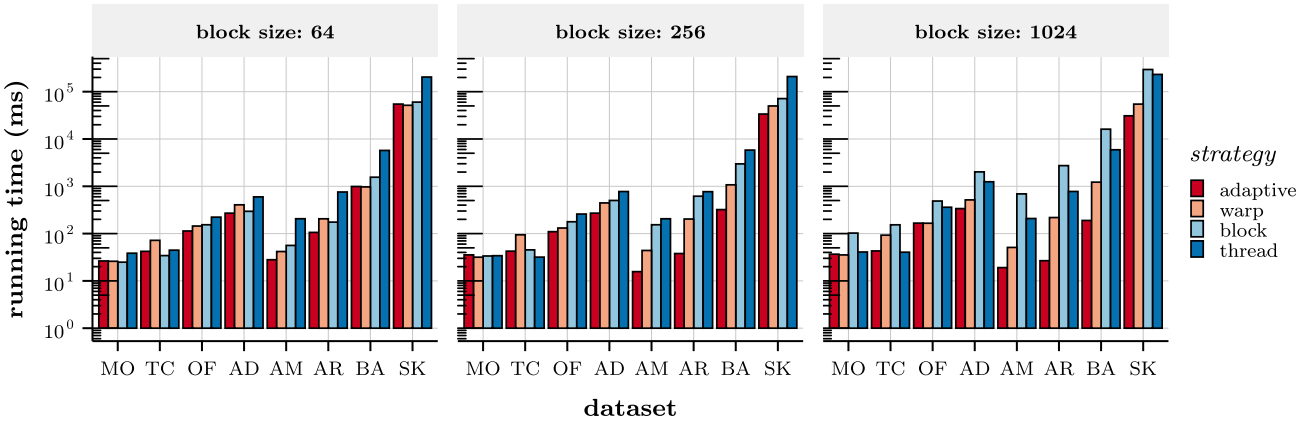


Fig. 13 Running time of Frank–Wolfe iterations under different strategies

Table 7 Compute workload analysis

Strategy	Avg. active threads / Warp	Achieved active warps / SM
Adaptive	31.19875	36.38
Block	31.8475	31.445
Thread	20.74125	14.32125
Warp	26.1425	33.62125

Table 8 Speedup by the adaptive strategy on Frank–Wolfe computation

Dataset	MO	TC	OF	AD	AM	AR	BA	SK
Speedup	4.0	1.8	18.8	39.2	50.3	456	517	232

levels of warp divergence, as the average number of active threads per warp is close to 32. However, the adaptive strategy achieves better performance than the block strategy, as it can utilize the GPU resources more efficiently, leading to a higher number of active warps per SM.

Next, we evaluate the speedup provided by the adaptive strategy with a block size of 1024 on GPU compared to the original CPU Frank–Wolfe computation for optimizing the dual programs, as this setting is almost the best choice. Table 8 reports the speedup results. We can find that the speedup is more significant on large datasets. Notice that these speedups relate to the Frank–Wolfe computation, which, as we mentioned earlier, takes up around 75% of the whole processing time on dataset SK. Thus, the theoretical speedup provided by the GPU parallel Frank–Wolfe computation is limited to at most $\frac{1}{1-75\%} = 4$ times. We further report the end-to-end speedup provided by GPU acceleration in Table 9. The end-to-end speedup on SK is around 3.88, which is close to the theoretical limit. However, for smaller datasets such as MO, TC, and OF, the speedup can be smaller than 1 due to the running time on those datasets being less than 0.1 s, while the GPU initialization can incur up to 3 s of latency.[6] Thus, the GPU speedup is more suitable for large-scale datasets. The maximum GPU memory usage and the size of data transferred between GPU and CPU memory can be found in Table 10.

Table 9 Breakdown of total time (in seconds) for GPU-enabled version and end-to-end speedup

Dataset	Data move	GPU comp	Pre/post process	Speedup
MO	0.0081	0.0367	3.337	0.05
TC	0.0079	0.043	3.2798	0.03
OF	0.0374	0.1664	3.5683	0.87
AD	0.1209	0.3378	3.5789	3.36
AM	0.0228	0.0191	5.8581	0.62
AR	0.0881	0.0267	13.0513	1.73
BA	0.9461	0.1894	118.0355	1.80
SK	118.765	30.9194	2390.4377	3.88

We have also evaluated the CPU parallel implementation of Frank–Wolfe with 32 threads. However, the speedup provided by CPU parallel is much smaller than our GPU implementation. The average speedup provided by GPU implementation is about 164×, while the 32-thread CPU-only version can provide about 3× speedup in terms of Frank–Wolfe computation. For details, please refer to our code repo.

8 Conclusion

This paper studies efficient solutions of the directed densest subgraph (DDS) problem via convex programming. We

[6] https://deci.ai/blog/measure-inference-time-deep-neural-networks/.

Table 10 Maximum GPU memory used and data transferred between CPU and GPU

Memory (in MB)	MO	TC	OF	AD	AM	AR	BA	SK
Max usage	207	207	207	207	213	257	257	5,585
Transferred	1.93	2.01	35.8	177	6.06	29.7	241	44,662

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. We further study parallelization strategies for computing Frank–Wolfe iterations on GPU. 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 best GPU parallelization strategy can also speed up the Frank–Wolfe computation by up to two orders of magnitude.

Acknowledgements This work was supported in part by NSFC under Grant 62102341, Basic and Applied Basic Research Fund in Guangdong Province under Grant 2023A1515011280 and 2022A1515010166, Guangdong Talent Program under Grant 2021QN02X826, Shenzhen Science and Technology Program under Grants JCYJ20220530143602006 and ZDSYS20211021111415025, the Fundamental Research Funds for the Central Universities (No. D5000230191), the University of Hong Kong (Projects 104005858 and 10400599), the Guangdong-Hong Kong-Macau Joint Laboratory Program 2020 (Project No: 2020B1212030009), The Hong Kong Jockey Club Charities Trust (HKJC), No. 260920140, and a grant from the Natural Sciences and Engineering Research Council of Canada.

9 Appendix

9.1 Convergence rate of `Frank-Wolfe-DDS`

To perform the convergence analysis of `Frank-Wolfe-DDS` (Algorithm 1), it would be easier if the objective function is differentiable [27], which, however, is not the case for $\|\mathbf{r}\|_\infty$ in $\mathsf{DP}(c)$ ((4.6)). Hence, we construct a convex program with a differentiable objective function, which shares the same optimal solution and minimizer of the linearization of the objective function at a specific position ((4.8)) with $\mathsf{DP}(c)$.

$$\mathsf{CP}(c)\ \min \quad f(\alpha,\beta)=\frac{1}{4\sqrt{c}}\sum_{u\in V} r_\alpha(u)^2+\frac{\sqrt{c}}{4}\sum_{v\in V} r_\beta(v)^2 \qquad (9.1)$$
$$\text{s.t.} \quad \alpha,\beta,\mathbf{r} \text{ satisfy the constraints in } \mathsf{DP}(c).$$

We can verify that (4.8) is also the minimizer of the linear function given by $\partial f(\alpha,\beta)$. Hence, `Frank-Wolfe-DDS` (Algorithm 1) applies to both $\mathsf{DP}(c)$ and $\mathsf{CP}(c)$. Further, the following two lemmas indicate that the optimal solution of $\mathsf{CP}(c)$ induces the c-biased DDS, which is also the objective of $\mathsf{DP}(c)$.

Lemma 9.1 *Suppose that an optimal solution* (α,β) *of* $\mathsf{CP}(c)$ *induces the density vector* $\mathbf{r}\in\mathbb{R}_+^{2|V|}$. *Then, we have*

1. $\exists(u,v)\in E, r_\alpha(u)>r_\beta(v)\Rightarrow \alpha_{u,v}=0,\beta_{v,u}=1$;
2. $\exists(u,v)\in E, r_\alpha(u)<r_\beta(v)\Rightarrow \beta_{v,u}=0,\alpha_{u,v}=1$.

Proof We prove the lemma by contradiction. For (1), suppose $\alpha_{u,v}>0$. There exists $\epsilon>0$ such that we could decrease $\alpha_{u,v}$ by ϵ and increase $\beta_{v,u}$ by ϵ to strictly decrease the objective function because $\frac{\partial f}{\partial\alpha_{u,v}}=r_\alpha(u)>\frac{\partial f}{\partial\beta_{v,u}}=r_\beta(v)$. This contradicts the optimal assumption. Similarly, we can also prove (2). □

To simplify the notations, we denote $\mathcal{D}_c$ as the feasible set of $\mathsf{DP}(c)$ and $\mathsf{CP}(c)$, as $\mathsf{DP}(c)$ and $\mathsf{CP}(c)$ share the same constraints.

Lemma 9.2 *Suppose a non-empty subset pair* (S,T), *where* $S,T\subseteq V$, *is stable with respect to a pair* $(\alpha,\beta,\mathbf{r})\in\mathcal{D}_c$. *Suppose that* $\exists\rho_c^*\in\mathbb{R}$ *such that* $\forall u\in S, r_\alpha(u)=\rho_c^*$ *and* $\forall v\in T, r_\beta(v)=\rho_c^*$. *Then,* $G[S,T]$ *is the c-biased DDS and has c-biased density* ρ_c^*.

Proof As $(\alpha,\beta,\mathbf{r})$ is a feasible solution of $\mathsf{DP}(c)$ and (S,T) is stable, the objective value of $(\alpha,\beta,\mathbf{r})$ is $\|\mathbf{r}\|_\infty=\rho_c^*$. Moreover, since (S,T) is stable, $\rho_c(S,T)=\frac{2\sqrt{cc'}}{c+c'}\cdot\frac{|E(S,T)|}{\sqrt{|S||T|}}=\rho_c^*$, where $c'=\frac{|S|}{|T|}$. This comes from $|E(S,T)|=(\frac{|S|}{2\sqrt{c}}+\frac{\sqrt{c}|T|}{2})\rho_c(S,T)$. By Lemma 4.1, (S,T) gives a feasible primal solution in $\mathsf{LP}(c)$ with objective value ρ_c^*. Hence, ρ_c^* is the optimal value for both $\mathsf{LP}(c)$ and $\mathsf{DP}(c)$, which means that $G[S,T]$ is the c-biased DDS. □

Lemmas 9.1,9.2 imply that an optimal solution $(\alpha,\beta,\mathbf{r})$ of $\mathsf{CP}(c)$ induces the c-biased DDS $G[S_c^*,T_c^*]$ in G, where $S_c^*=\{u|r_\alpha(u)=\|\mathbf{r}\|_\infty\}$ and $T_c^*=\{v|r_\beta(v)=\|\mathbf{r}\|_\infty\}$.

Hence, we can confirm that `Frank-Wolfe-DDS` (Algorithm 1) applies to both $\mathsf{DP}(c)$ and $\mathsf{CP}(c)$ and the optimal solutions of both programs induce the c-biased DDS. Hence, we use $\mathsf{CP}(c)$ to analyze the convergence rate of `Frank-Wolfe-DDS`. According to the previous convergence analysis of the Frank–Wolfe-based algorithms in [13,

27], the convergence rate of our `Frank-Wolfe-DDS` algorithm can be described by a value related to the graph, $Q_c = \frac{1}{2}\mathsf{Diam}(\mathcal{D}_c)^2 \sup_{(\alpha,\beta)\in\mathcal{D}_c} \|\nabla^2 f(\alpha,\beta)\|_2$, where $\mathsf{Diam}(\mathcal{D}_c)$ is the diameter of $\mathcal{D}_c$, $\nabla^2 f(\alpha,\beta)$ is the Hessian, and $\|\cdot\|_2$ is the spectral norm of a matrix.

Theorem 9.1 (Convergence Rate of Frank–Wolfe [27]) *Suppose $(\alpha^*,\beta^*) \in \mathcal{D}_c$ is an optimal solution of $\mathsf{CP}(c)$. Then, for all $i \geq 1$, $f(\alpha^{(i)},\beta^{(i)}) - f(\alpha^*,\beta^*) \leq \frac{2Q_c}{i+2}$.*

Lemma 9.3 (Bounding Q_c) *Given a directed graph $G = (V,E)$ with maximum outdegree $d^+_{\max}$ and maximum indegree $d^-_{\max}$ and a given c, we have that $Q_c \leq 2|E|\max\{\sqrt{c}d^+_{\max}, \frac{1}{\sqrt{c}}d^-_{\max})\}$.*

Proof First, we have $\mathsf{Diam}(\mathcal{D}_c) = \sqrt{2|E|}$. The Hessian of $f(\alpha,\beta)$ is irrelevant to the value of (α,β), and it is a nonnegative symmetric matrix. Therefore, $\sup_{(\alpha,\beta)\in\mathcal{D}_c} \|\nabla^2 f(\alpha,\beta)\|_2$ is the maximum singular value of $\nabla^2 f(\alpha,\beta)$. Let $A = \nabla^2 f(\alpha,\beta)$, λ_1 be the maximum singular value (also the maximum eigenvalue) of A, x be the eigenvector associated with λ_1, and p be the component in which x has maximum absolute value. Without loss of generality, we assume x_p is positive. We have

$$\lambda_1 x_p = (Ax)_p = \sum_{q=1}^{2n} A_{p,q}x_q \leq \sum_{q=1}^{2n} A_{p,q}x_p \leq x_p \max\{2\sqrt{c}d^+_{\max}, \frac{2}{\sqrt{c}}d^-_{\max}\}.$$

Therefore, $Q_c \leq 2|E|\max\{\sqrt{c}d^+_{\max}, \frac{1}{\sqrt{c}}d^-_{\max})\}$. □

Lemma 9.4 *Suppose $(\alpha,\beta,r) \in \mathcal{D}_c$ such that $\varepsilon := \|\mathbf{r}\|_\infty - \rho^*_c$, where $\rho^*_c = \|\mathbf{r}^*\|_\infty$ and $(\alpha^*,\beta^*,\mathbf{r}^*)$ is the optimal solution of $\mathsf{DP}(c)$. Then, we have that $(4\sqrt{c} + \frac{4}{\sqrt{c}}) \cdot (f(\alpha,\beta) - f(\alpha^*,\beta^*)) \geq \varepsilon^2$.*

Proof First, we have $f(\alpha,\beta) - f(\alpha^*,\beta^*) \geq f(\alpha-\alpha^*,\beta-\beta^*)$, because $f(\alpha,\beta) - f(\alpha^*,\beta^*) - f(\alpha-\alpha^*,\beta-\beta^*)$ is an affine function on $\mathcal{D}_c$ and obtains its minimum value 0 at (α^*,β^*). Second, $f(\alpha-\alpha^*,\beta-\beta^*)$ can be bounded by the l^2-norm of $\mathbf{r}-\mathbf{r}^*$, i.e., $(4\sqrt{c} + \frac{4}{\sqrt{c}}) f(\alpha-\alpha^*,\beta-\beta^*) \geq \|\mathbf{r}-\mathbf{r}^*\|_2^2$. For the infinity norm and the l^2-norm, we have $\|\mathbf{r}\|_\infty - \rho^*_c \leq \|\mathbf{r}-\mathbf{r}^*\|_\infty \leq \|\mathbf{r}-\mathbf{r}^*\|_2$. Combining the above inequalities, we will have the lemma. □

Corollary 9.1 (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 $\|\mathbf{r}^{(i)}\|_\infty - \rho^*_c \leq \varepsilon$.*

9.2 Proofs

Proof of Lemma 4.1 We prove the lemma by showing a feasible solution (x,s,t,a,b) of $\mathsf{LP}(c)$. Let $a = \frac{2c'}{c+c'}$ and $b = \frac{2c}{c+c'}$. For each vertex $u \in P$, set $s_u = \frac{a\sqrt{c}}{|P|} = \frac{2c'\sqrt{c}}{(c+c')|P|}$. For each vertex $v \in Q$, set $t_v = \frac{b}{\sqrt{c}|Q|} = \frac{2c}{(c+c')\sqrt{c}|Q|} = \frac{2c'\sqrt{c}}{(c+c')|P|}$. For each edge $(u,v) \in E(P,Q)$, set $x_{u,v} = s_u = t_v$. All the remaining variables are set to 0. Now, $\sum_{u\in V} s_u = a\sqrt{c}$ and $\sum_{v\in V} t_v = \frac{b}{\sqrt{c}}$. Hence, this is a feasible solution to $\mathsf{LP}(c)$. The value of this solution is

$$\frac{2c'\sqrt{c}}{(c+c')|P|}|E(P,Q)| = \frac{2\sqrt{c}c'\sqrt{|Q|}}{(c+c')\sqrt{|P|}}\frac{|E(P,Q)|}{\sqrt{|P||Q|}} = \frac{2\sqrt{c}\sqrt{c'}}{c+c'}\rho(P,Q).$$

Thus, the lemma holds. □

Proof of Lemma 4.2 Without loss of generality, we can assume that for each $(u,v) \in E$, $x_{u,v} = \min\{s_u, t_v\}$. We define a collection of sets S,T indexed by a parameter $r \geq 0$. Let $S(r) = \{u | s_u \geq r\}$, $T(r) = \{v | t_v \geq r\}$, and $E(r) = \{(u,v) | x_{u,v} = \min\{s_u,t_v\}\}$. Hence, $E(r)$ is precisely the set of edges that go from $S(r)$ to $T(r)$.

Now, $\int_0^\infty |S(r)|\mathrm{d}r = \sum_{u\in V} s_u = a\sqrt{c}$. Similarly, $\int_0^\infty |T(r)|\mathrm{d}r = \sum_{v\in V} t_v = \frac{b}{\sqrt{c}}$. By the Cauchy–Schwarz inequality,

$$\int_0^\infty \sqrt{|S(r)||T(r)|}\mathrm{d}r \leq \sqrt{\left(\int_0^\infty |S(r)|\mathrm{d}r\right)\left(\int_0^\infty |T(r)|\mathrm{d}r\right)} = \sqrt{ab}.$$

Note that $\int_0^\infty |E(r)|\mathrm{d}r = \sum_{(u,v)\in E} x_{u,v}$. This is the objective function value of the solution. Let this value be x_{sum}.

We claim that there exists r such that $\frac{E(r)}{\sqrt{|S(r)||T(r)|}} \geq \frac{x_{\text{sum}}}{\sqrt{ab}}$. Suppose there was no such r. Then,

$$\int_0^\infty |E(r)|\mathrm{d}r < \frac{x_{\text{sum}}}{\sqrt{ab}}\int_0^\infty \sqrt{|S(r)||T(r)|}\mathrm{d}r \leq x_{\text{sum}}.$$

This gives a contradiction. Thus, the lemma holds. □

Proof of Lemma5.3 As $G[S^*_c,T^*_c]$ is the c-biased DDS with c-biased density $\rho_c(S^*,T^*)$, there must exist $u \in S$ satisfying $r_\alpha(u) \leq \rho_c(S^*_c,T^*_c)$, or $v \in T$ satisfying $r_\beta(v) \leq \rho_c(S^*_c,T^*_c)$. Otherwise, $G[S,T]$ is a subgraph with a higher c-biased density than $G[S^*_c,T^*_c]$.

Now, we prove the lemma by contradiction. Assume $G[S^*_c,T^*_c]$ is not contained in $G[S,T]$. Based on whether $G[S^*_c,T^*_c]$ overlaps $G[S,T]$, there are two cases.

1. $S_c^* \cap S = \emptyset$ and $T_c^* \cap T = \emptyset$. Since $|E(S_c^*, T_c^*)| = (\frac{|S_c^*|}{\sqrt{c}} + \sqrt{c}|T_c^*|)\rho_c(S_c^*, T_c^*)$, there exists $u \in S_c^*, r_\alpha(u) \geq \rho_c(S_c^*, T_c^*)$ or $v \in T_c^*, r_\beta(v) \geq \rho_c(S_c^*, T_c^*)$.
2. $S_c^* \cap S \neq \emptyset$ or $T_c^* \cap T \neq \emptyset$.

$$
\begin{aligned}
&|E(S_c^*, T_c^*)| \\
=&|E(S_c^* \cap S, T_c^* \cap T)| + |E(S_c^*, T_c^*) \setminus E(S, T)| \\
=&\left(\frac{|S_c^* \cap S|}{\sqrt{c}} + \sqrt{c}|T_c^* \cap T|\right)\rho_c(S_c^* \cap S, T_c^* \cap T) \\
&+ \left(\frac{|S_c^* \setminus S|}{\sqrt{c}} + \sqrt{c}|T_c^* \setminus T|\right)\rho_c'.
\end{aligned}
$$

Since $\rho_c(S_c^* \cap S, T_c^* \cap T) \leq \rho_c(S_c^*, T_c^*)$, we have $\rho_c' \geq \rho_c(S_c^*, T_c^*)$. Thus, there exists $u \in S_c^* \setminus S, r_\alpha(u) \geq \rho_c(S_c^*, T_c^*)$ or $v \in T_c^* \setminus T, r_\beta(v) \geq \rho_c(S_c^*, T_c^*)$.

Consequently, for each case above, combining the inequalities will give us a contradiction to the first condition of the stable (S, T)-induced subgraph definition. Hence, the lemma holds. □

References

1. Administration, F.A.: Air traffic control system command center. https://www.faa.gov (2019)
2. Albert, R., Jeong, H., Barabási, A.L.: Internet: diameter of the world-wide web. Nature **401**(6749), 130 (1999)
3. Angel, A., Koudas, N., Sarkas, N., Srivastava, D., Svendsen, M., Tirthapura, S.: Dense subgraph maintenance under streaming edge weight updates for real-time story identification. VLDB J. **23**(2), 175–199 (2014)
4. Bahmani, B., Kumar, R., Vassilvitskii, S.: Densest subgraph in streaming and mapreduce. Proc. VLDB Endowm. **5**(5), 454–465 (2012)
5. Bhaskara, A., Charikar, M., Chlamtac, E., Feige, U., Vijayaraghavan, A.: Detecting high log-densities: an o (n 1/4) approximation for densest k-subgraph. In: STOC, pp. 201–210. ACM (2010)
6. Bhattacharya, S., Henzinger, M., Nanongkai, D., Tsourakakis, C.: Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In: STOC, pp. 173–182 (2015)
7. Boob, D., Gao, Y., Peng, R., Sawlani, S., Tsourakakis, C., Wang, D., Wang, J.: Flowless: Extracting densest subgraphs without flow computations. In: Proceedings of The Web Conference 2020, WWW '20, pp. 573–583. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3366423.3380140
8. Buehrer, G., Chellapilla, K.: A scalable pattern mining approach to web graph compression with communities. In: WSDM, pp. 95–106. ACM (2008)
9. Capocci, A., Servedio, V.D., Colaiori, F., Buriol, L.S., Donato, D., Leonardi, S., Caldarelli, G.: Preferential attachment in the growth of social networks: the internet encyclopedia wikipedia. Phys Review E **74**(3), 0360116 (2006)
10. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: International Workshop on Approximation Algorithms for Combinatorial Optimization, pp. 84–95. Springer (2000)
11. Chekuri, C., Quanrud, K., Torres, M.R.: Densest subgraph: Supermodularity, iterative peeling, and flow. In: Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 1531–1555. SIAM (2022)
12. Clauset, A., Shalizi, C.R., Newman, M.E.: Power-law distributions in empirical data. SIAM Rev. **51**(4), 661–703 (2009)
13. Danisch, M., Chan, T.H.H., Sozio, M.: Large scale density-friendly graph decomposition via convex programming. In: WWW, pp. 233–242. International World Wide Web Conferences Steering Committee (2017)
14. Epasto, A., Lattanzi, S., Sozio, M.: Efficient densest subgraph computation in evolving graphs. In: Proceedings of the 24th International Conference on World Wide Web, pp. 300–310 (2015)
15. Fang, Y., Yu, K., Cheng, R., Lakshmanan, L.V., Lin, X.: Efficient algorithms for densest subgraph discovery. Proc. VLDB Endowm. **12**(11), 1719–1732 (2019)
16. Frank, M., Wolfe, P., et al.: An algorithm for quadratic programming. Naval Res. Logist. Q. **3**(1–2), 95–110 (1956)
17. Freeman, L.C., Webster, C.M., Kirke, D.M.: Exploring social structure using dynamic three-dimensional color images. Soc. Netw. **20**(2), 109–118 (1998)
18. Gibson, D., Kumar, R., Tomkins, A.: Discovering large dense subgraphs in massive graphs. In: PVLDB, pp. 721–732. VLDB Endowment (2005)
19. Gionis, A., Tsourakakis, C.E.: Dense subgraph discovery: Kdd 2015 tutorial. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2313–2314. ACM (2015)
20. Goldberg, A.V.: Finding a Maximum Density Subgraph. University of California Berkeley, CA (1984)
21. Han, X., Cheng, R., Grubenmann, T., Maniu, S., Ma, C., Li, X.: Leveraging contextual graphs for stochastic weight completion in sparse road networks. In: SIAM International Conference on Data Mining. SIAM (2022)
22. Han, X., Cheng, R., Ma, C., Grubenmann, T.: Deeptea: Effective and efficient online time-dependent trajectory outlier detection. In: Proceedings of the VLDB Endowment (2022)
23. Han, X., Dell'Aglio, D., Grubenmann, T., Cheng, R., Bernstein, A.: A framework for differentially-private knowledge graph embeddings. J. Web Semant. **72**, 100696 (2022)
24. Han, X., Grubenmann, T., Cheng, R., Wong, S.C., Li, X., Sun, W.: Traffic incident detection: A trajectory-based approach. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1866–1869. IEEE (2020)
25. Hooi, B., Song, H.A., Beutel, A., Shah, N., Shin, K., Faloutsos, C.: Fraudar: Bounding graph fraud in the face of camouflage. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 895–904. ACM (2016)
26. Hu, S., Wu, X., Chan, T.H.: Maintaining densest subsets efficiently in evolving hypergraphs. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 929–938 (2017)
27. Jaggi, M.: Revisiting frank-wolfe: Projection-free sparse convex optimization. In: Proceedings of the 30th International Conference on Machine Learning, CONF, pp. 427–435 (2013)
28. Java, A., Song, X., Finin, T., Tseng, B.: 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, pp. 56–65. ACM (2007)
29. Kannan, R., Vinay, V.: Analyzing the structure of large graphs. Rheinische Friedrich-Wilhelms-Universität Bonn Bonn (1999)
30. Karlebach, G., Shamir, R.: Modelling and analysis of gene regulatory networks. Nat. Rev. Mol. Cell Biol. **9**(10), 770–780 (2008)

31. Khuller, S., Saha, B.: On finding dense subgraphs. In: International Colloquium on Automata, Languages, and Programming, pp. 597–608. Springer (2009)
32. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM (JACM) **46**(5), 604–632 (1999)
33. Kunegis, J.: KONECT – The Koblenz Network Collection. In: WWW, pp. 1343–1350 (2013). http://userpages.uni-koblenz.de/~kunegis/paper/kunegis-koblenz-network-collection.pdf
34. Lakshmanan, L.V.: On a quest for combating filter bubbles and misinformation. In: Proceedings of the 2022 International Conference on Management of Data, pp. 2–2 (2022)
35. Leskovec, J., Adamic, L.A., Huberman, B.A.: The dynamics of viral marketing. ACM Trans. Web **1**(1) (2007)
36. Li, X., Cheng, R., Chang, K.C.C., Shan, C., Ma, C., Cao, H.: On analyzing graphs with motif-paths. Proc. VLDB Endowm. **14**(6), 1111–1123 (2021)
37. Luo, W., Ma, C., Fang, Y., Lakshmanan, L.V.S.: A survey of densest subgraph discovery on large graphs (2023)
38. Ma, C., Cheng, R., Lakshmanan, L.V., Grubenmann, T., Fang, Y., Li, X.: Linc: a motif counting algorithm for uncertain graphs. Proc. VLDB Endowm. **13**(2), 155–168 (2019)
39. Ma, C., Cheng, R., Lakshmanan, L.V., Han, X.: Finding locally densest subgraphs: a convex programming approach. Proc. VLDB Endowm. **15**(11), 2719–2732 (2022)
40. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Zhang, W., Lin, X.: Efficient algorithms for densest subgraph discovery on large directed graphs. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 1051–1066 (2020)
41. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Zhang, W., Lin, X.: Efficient directed densest subgraph discovery. ACM SIGMOD Rec. **50**(1), 33–40 (2021)
42. Ma, C., Fang, Y., Cheng, R., Lakshmanan, L.V., Zhang, W., Lin, X.: On directed densest subgraph discovery. TODS **46**(4), 1–45 (2021)
43. Massa, P., Salvetti, M., Tomasoni, D.: Bowling alone and trust decline in social network sites. In: Proc. Int. Conf. Dependable, Autonomic and Secure Computing, pp. 658–663 (2009)
44. Mitzenmacher, M., Pachocki, J., Peng, R., Tsourakakis, C., Xu, S.C.: 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, pp. 815–824. ACM (2015)
45. Mukherjee, A., Liu, B., Glance, N.: Spotting fake reviewer groups in consumer reviews. In: WWW, pp. 191–200 (2012)
46. Niu, X., Sun, X., Wang, H., Rong, S., Qi, G., Yu, Y.: Zhishi.me – weaving Chinese linking open data. In: Proc. Int. Semantic Web Conf., pp. 205–220 (2011)
47. Opsahl, T., Agneessens, F., Skvoretz, J.: Node centrality in weighted networks: generalizing degree and shortest paths. Soc. Netw. **3**(32), 245–251 (2010)
48. Orlin, J.B.: Max flows in o (nm) time, or better. In: STOC, pp. 765–774 (2013)
49. Prakash, B.A., Sridharan, A., Seshadri, M., Machiraju, S., Faloutsos, C.: Eigenspokes: Surprising patterns and scalable community chipping in large graphs. In: PAKDD, pp. 435–448. Springer (2010)
50. Qin, L., Li, R.H., Chang, L., Zhang, C.: Locally densest subgraph discovery. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 965–974. ACM (2015)
51. Rossi, R., Ahmed, N.: Network repository (2013). http://networkrepository.com
52. Sarma, A.D., Lall, A., Nanongkai, D., Trehan, A.: Dense subgraphs on dynamic networks. In: International Symposium on Distributed Computing, pp. 151–165. Springer (2012)
53. Sawlani, S., Wang, J.: Near-optimal fully dynamic densest subgraph. In: STOC, pp. 181–193 (2020)
54. Shiloach, Y., Vishkin, U.: An o (n2log n) parallel max-flow algorithm. J. Algorithms **3**(2), 128–146 (1982)
55. Stratton, J.A., Anssari, N., Rodrigues, C., Sung, I.J., Obeid, N., Chang, L., Liu, G.D., Hwu, W.m.: Optimization and architecture effects on gpu computing workload performance. In: 2012 Innovative Parallel Computing (InPar), pp. 1–10. IEEE (2012)
56. Sun, B., Dansich, M., Chan, H., Sozio, M.: Kclist++: a simple algorithm for finding k-clique densest subgraphs in large graphs. Proc. VLDB Endowm. **13**(10), 1628–1640 (2020)
57. Tatti, N., Gionis, A.: Density-friendly graph decomposition. In: WWW, pp. 1089–1099. International World Wide Web Conferences Steering Committee (2015)
58. Tsourakakis, C.: The k-clique densest subgraph problem. In: WWW, pp. 1122–1132. International World Wide Web Conferences Steering Committee (2015)
59. Xu, Y., Ma, C., Fang, Y., Bao, Z.: Efficient and effective algorithms for generalized densest subgraph discovery. Proc. ACM Manag. Data **1**(2), 1–27 (2023)