Densest k-Subgraph Mining via a Provably Tight Relaxation

Qiheng Lu[1], **Nicholas D. Sidiropoulos**[1], **Aritra Konar**[2]

[1] University of Virginia
[2] KU Leuven
luqh@virginia.edu, nikos@virginia.edu, aritra.konar@kuleuven.be

Abstract

Given an unweighted, undirected, and simple graph, the Densest k-Subgraph (DkS) problem aims to find a subgraph of k vertices that has the maximum average induced degree. In this paper, we consider an equivalent reformulation of the DkS problem via diagonal loading. On relaxing the combinatorial constraint of the reformulated problem, we show that the resulting non-convex, continuous relaxation is tight under certain conditions by leveraging an extension of the Motzkin-Straus theorem. We utilize two projection-free approaches to solve the relaxed problem: one based on the Frank-Wolfe algorithm and the other on explicit constraint parameterization. We compare their performance to state-of-the-art baselines across various benchmarks. Our empirical results show that the Frank-Wolfe-based algorithm proposed in this paper outperforms existing baselines in terms of subgraph density and computational complexity.

Code —
https://github.com/luqh357/DkS-Diagonal-Loading

Introduction

Dense subgraph detection (DSD) aims to extract highly interconnected subsets of vertices from a graph. DSD is a key primitive in graph mining (see the surveys (Lanciano et al. 2024; Luo et al. 2023) and references therein), and finds applications in diverse settings ranging from discovering DNA motifs (Fratkin et al. 2006) to mining trending topics in social media (Angel et al. 2014) and discovering complex patterns in gene annotation graphs (Saha et al. 2010). In recent years, DSD has also been employed for detecting fraudulent activities occurring in e-commerce and financial transaction networks (Hooi et al. 2016; Li et al. 2020; Ji et al. 2022; Chen and Tsourakakis 2022).

Owing to its relevance, the development of algorithmic approaches for DSD has received extensive attention. A widely-used formulation for DSD is the Densest Subgraph (DSG) problem (Goldberg 1984), which aims to extract the subgraph with the maximum average induced degree. DSG can be solved exactly in polynomial time via maximum-flow, while a lower complexity alternative is a greedy peeling algorithm that runs in linear time and provides a $1/2$-factor approximation guarantee (Charikar 2000). Recent work (Boob et al. 2020; Chekuri, Quanrud, and Torres 2022) has developed multi-stage extensions of the basic greedy algorithm, which provide improved approximation guarantees for DSG at roughly the same complexity. In addition, several recent works, including (Danisch, Chan, and Sozio 2017; Harb, Quanrud, and Chekuri 2022, 2023; Nguyen and Ene 2024; Chandrasekaran et al. 2024) have proposed algorithms based on methods such as FISTA, Frank-Wolfe, and coordinate descent to further enhance the performance of algorithms for solving DSG. Another popular approach for DSD is to compute the k-core of a graph (Seidman 1983), which corresponds to the subgraph that maximizes the minimum induced degree. The greedy algorithm for DSG (with a slight tweak) optimally computes the k-core in linear-time. Recently, (Veldt, Benson, and Kleinberg 2021) provided a more general framework for computing solutions of degree-based density objectives, which contain DSG and k-core as special cases.

An inherent limitation of the aforementioned approaches is that the size of the extracted subgraph cannot be explicitly controlled, which raises the possibility that the solution may be a large subgraph that is too sparsely connected to be practically useful. These observations have been made on real-world graphs for DSG (Tsourakakis et al. 2013) and the k-core (Shin, Eliassi-Rad, and Faloutsos 2016). A natural remedy is to add an explicit size constraint to DSG, leading to the Densest k-Subgraph (DkS) problem (Feige, Peleg, and Kortsarz 2001). The objective of DkS is to find a size-k vertex subset with the maximum average induced degree. By varying the size parameter k, DkS can extract a spectrum of dense subgraphs, providing flexibility for users to select a solution with the desired density when the outputs of the aforementioned approaches are unsatisfactory. However, this simple modification makes the resulting DkS problem NP-hard. Furthermore, DkS has been shown to be difficult to approximate (Khot 2006; Manurangsi 2017) in the worst case. Hence, developing effective algorithmic tools for DkS (on practical instances) is substantially more challenging compared to DSG or k-core extraction. It is also worth noting that there are variants of DkS, such as the Densest at-least-k Subgraph problem (Dal$k$S) and the Densest at-most-k Subgraph problem (Dam$k$S) (Andersen and Chellapilla 2009; Khuller and Saha 2009; Kawase and Miyauchi 2018; Chekuri, Quanrud, and Torres 2022). These variants

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

impose size constraints on the extracted subgraph. However, using these formulations does not guarantee that the entire spectrum of densest subgraphs (i.e., of every size) can be explored.

Related Work: The best polynomial-time approximation algorithm for DkS can provide a $O(n^{1/4+\epsilon})$-approximation solution in time $O(n^{1/\epsilon})$ for every $\epsilon > 0$ (Bhaskara et al. 2010), which is generally quite pessimistic. The work of (Bhaskara et al. 2012) proposed using a hierarchy of linear programming (LP) relaxations of increasing complexity to approximate DkS, which makes it unattractive for large datasets. Even implementing the lowest level of the hierarchy is computationally expensive, as a large number of extra variables and linear inequalities have to be introduced in order to successfully linearize the quadratic objective function of DkS. Meanwhile, (Ames 2015) formulated DkS as a rank-constrained optimization problem, which was then relaxed and solved using the Alternating Direction Method of Multipliers (ADMM) (Eckstein and Bertsekas 1992). However, implementing the approach requires using matrix lifting, which results in a quadratic number of variables and thus makes it unsuitable for even moderately sized problem instances. A different relaxation of DkS which uses semidefinite programming (SDP) was proposed in (Bombina and Ames 2020). The relaxation provides optimality guarantees for DkS when applied on a certain class of random graphs. In addition to the high complexity entailed in solving the SDP, the optimality claims of the proposed algorithm in (Bombina and Ames 2020) do not extend straightforwardly to real-world graphs.

Since the aforementioned algorithms are sophisticated and can incur high computational complexity, efforts have also focused on developing more practical algorithms for the DkS problem. The work of (Yuan and Zhang 2013) proposed using the Truncated Power Method (TPM) for tackling DkS and proved the convergence of this algorithm under certain conditions. However, in practice, the subgraphs obtained by TPM can be highly sub-optimal. (Papailiopoulos et al. 2014) employed the Spannogram, an exact solver for certain low-rank bilinear optimization problems, to approximately solve the DkS problem. If the graph adjacency matrix has a constant rank, it was shown that the Spannogram can approximate the solution to the DkS problem in polynomial time. Improving the approximation quality may require choosing a relatively high rank, which limits scalability due to the exponential dependence of the algorithm's time complexity on the rank. Even performing a rank-2 approximation using the Spannogram framework incurs $O(n^3)$ complexity, making it challenging to apply to large-scale problems.

A separate line of research has considered applying continuous relaxations of DkS followed by application of gradient-based optimization algorithms. The work of (Hager, Phan, and Zhu 2016) introduced a relaxation of DkS and applied a pair of gradient-based optimization algorithms for solving it. A different relaxation was introduced in (Sotirov 2020), and coordinate descent algorithms were applied to solve it. However, neither (Hager, Phan, and Zhu 2016) nor (Sotirov 2020) analyzed whether the relaxations are tight or not. In (Konar and Sidiropoulos 2021), an alternate convex relaxation was proposed based on using the Lovaśz extension of the objective function of DkS as a convex surrogate. A customized algorithm based on an inexact version of ADMM (Condat 2013) was put forth to solve the relaxed formulation. Nonetheless, (Konar and Sidiropoulos 2021) also did not provide an analysis of whether the relaxation is tight and the computational cost of the proposed algorithm can also be relatively high. Recently, (Liu, Liu, and Ma 2024) introduced Extreme Point Pursuit (EXPP), which relaxes the discrete constraint of DkS, and adopts a quadratic penalty based approach combined with a homotopy optimization technique to tackle the relaxed problem. They employed an empirical strategy for selecting the penalty parameter for homotopy optimization, and it remains unclear whether their relaxation is tight.

Contributions: Our contributions can be summarized as follows:

• We adopt a continuous relaxation of the DkS problem with diagonal loading. Although the resulting problem is nonconvex, we establish that for a suitable choice of the diagonal loading parameter λ, the relaxation is guaranteed to be tight for all subgraph sizes less than or equal to the maximum clique size, *without imposing any restrictions on the graph*. We derive this surprising result via a non-trivial extension of the classic Motzkin-Straus theorem (Motzkin and Straus 1965). Furthermore, our result reveals that the value of λ required to establish tightness of the diagonal loading based relaxation is much smaller than that obtained using classical results in optimization (Rockafellar 1970). The upshot is that the optimization landscape of such a "lightly" loaded relaxation can be more amenable to gradient-based optimization approaches.

• To efficiently obtain (approximate) solutions of the relaxed optimization problem, we utilize two iterative gradient-based approaches: one based on the Frank-Wolfe algorithm (Frank and Wolfe 1956; Jaggi 2013) and the other based on explicit constraint parameterization. A key feature of these methods is that they are *projection-free*, meaning they avoid the need to compute computationally expensive projections onto the constraint set of the proposed relaxation, which enhances their scalability. We provide a complexity analysis demonstrating that the computational cost of the proposed parameterization approach is not prohibitive, while that of the Frank-Wolfe approach is surprisingly low.

• We validate the effectiveness of diagonal loading empirically and demonstrate that the proposed algorithms, especially the Frank-Wolfe algorithm, achieve state-of-the-art results across several real-world datasets. We also show that the Frank-Wolfe algorithm achieves good subgraph density while maintaining low computational complexity.

It is important to point out that the Frank-Wolfe framework has been previously applied for solving DSG in (Danisch, Chan, and Sozio 2017; Harb, Quanrud, and Chekuri 2023). However, their formulations are significantly different from the relaxation we consider in (5), as the problems in these works are convex, whereas problem (5) is not. Consequently, the Frank-Wolfe algorithm applied to (5) exhibits different convergence properties, step size selection,

and implementation requirements.

Notation: In this paper, lowercase regular font is used to represent scalars, lowercase bold font is used to represent vectors, and uppercase bold font is used to represent matrices. x_i denotes the i-the element of the vector $\boldsymbol{x}$ and $\boldsymbol{x}^t$ denotes the vector $\boldsymbol{x}$ at the t-th iteration. $[n]$ denotes the set $\{1, 2, \cdots, n\}$. $(\cdot)^\mathsf{T}$ denotes the transpose of a vector or a matrix. $\boldsymbol{A} \succeq \boldsymbol{0}$ denotes $\boldsymbol{A}$ is a positive semi-definite matrix. $\|\cdot\|_2$ denotes the Euclidean norm of a vector or the spectral norm of a matrix. $\text{top}_k(\boldsymbol{x})$ denotes the indices of the largest k elements in $\boldsymbol{x}$ and $\text{supp}(\text{top}_k(\boldsymbol{x}))$ denotes the support of the largest k elements in $\boldsymbol{x}$.

Problem Formulation

Consider an unweighted, undirected, and simple graph $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ with n vertices and m edges, where $\mathcal{V} := [n] = \{1, \cdots, n\}$ represents the vertex set and $\mathcal{E}$ is the edge set. Given a positive integer $k \in [n]$, the DkS problem aims to find a vertex subset $S \subseteq \mathcal{V}$ of size k such that the subgraph by S has the maximum average degree. Formally, the problem can be expressed as

$$\begin{aligned} \max_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{x}^\mathsf{T} \boldsymbol{A} \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{B}_k^n, \end{aligned} \tag{1}$$

where $\boldsymbol{A}$ is the symmetric $n \times n$ adjacency matrix of $\mathcal{G}$ and the constraint set $\mathcal{B}_k^n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \in \{0,1\}^n, \sum_{i \in [n]} x_i = k\}$ captures the subgraph size restriction. The binary vector $\boldsymbol{x} \in \{0,1\}^n$ is an indicator vector corresponding to the selected vertices in the subset S, i.e.,

$$x_i = \begin{cases} 1, & \text{if } i \in S, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The combinatorial sum-to-k constraint in (1) makes the DkS problem hard to solve. A natural choice to alleviate the difficulty is to relax the constraint to its convex hull, a strategy also employed in (Sotirov 2020; Konar and Sidiropoulos 2021; Liu, Liu, and Ma 2024). After relaxation, the problem can be formulated as

$$\begin{aligned} \max_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{x}^\mathsf{T} \boldsymbol{A} \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{C}_k^n, \end{aligned} \tag{3}$$

where $\mathcal{C}_k^n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \in [0,1]^n, \sum_{i \in [n]} x_i = k\}$ is a polytope that represents the convex hull of $\mathcal{B}_k^n$. As the problem is now continuous, it is amenable to (approximate) maximization via gradient-based methods. However, we cannot guarantee that the optimal solution of (1) will remain optimal for (3), indicating that the relaxation from (1) to (3) might not be tight. One way to address this issue is to introduce a regularization term in the objective function to promote sparsity in the solution. This can be achieved by equivalently reformulating (1) as

$$\begin{aligned} \max_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{x}^\mathsf{T} (\boldsymbol{A} + \lambda \boldsymbol{I}) \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{B}_k^n, \end{aligned} \tag{4}$$

where λ is a constant and $\boldsymbol{I}$ is the identity matrix of size n. We refer to this trick as "diagonal loading" with parameter λ. Similar to the relaxation from (1) to (3), the combinatorial constraint $\mathcal{B}_k^n$ in (4) can be relaxed to $\mathcal{C}_k^n$, yielding the following relaxation

$$\begin{aligned} \max_{\boldsymbol{x} \in \mathbb{R}^n} \quad & \boldsymbol{x}^\mathsf{T} (\boldsymbol{A} + \lambda \boldsymbol{I}) \boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{C}_k^n. \end{aligned} \tag{5}$$

If λ is selected large enough so that the matrix $\boldsymbol{A} + \lambda \boldsymbol{I} \succeq \boldsymbol{0}$, (i.e., $\lambda \geq -\lambda_n(\boldsymbol{A})$, where $\lambda_n(\boldsymbol{A})$ denotes the smallest eigenvalue of $\boldsymbol{A}$), then the objective function of (5) becomes convex. Since the maximum of a convex function over a polytope is attained at one of its extreme points, directly invoking (Rockafellar 1970, Theorem 32.2) shows that

$$\arg\max_{\boldsymbol{x} \in \mathcal{B}_k^n} \boldsymbol{x}^\mathsf{T} (\boldsymbol{A} + \lambda I) \boldsymbol{x} \subseteq \arg\max_{\boldsymbol{x} \in \mathcal{C}_k^n} \boldsymbol{x}^\mathsf{T} (\boldsymbol{A} + \lambda \boldsymbol{I}) \boldsymbol{x}. \tag{6}$$

This implies that the maximum of $\boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda \boldsymbol{I})\boldsymbol{x}$ over the combinatorial sum-to-k constraint $\boldsymbol{x} \in \mathcal{B}_k^n$ (i.e., the solution of DkS) is attained only when $\boldsymbol{x}$ is also the maximum of $\boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda \boldsymbol{I})\boldsymbol{x}$ over the continuous sum-to-k constraint $\boldsymbol{x} \in \mathcal{C}_k^n$. Hence, for such a choice of λ, the relaxation from (4) to (5) is tight. However, when gradient-based optimization algorithms are applied on (5), they can get stuck in sub-optimal solutions if the diagonal loading parameter λ is too large. Therefore, choosing the diagonal loading parameter λ properly has a significant impact on the quality of the solution that can be achieved.

Previous works (Yuan and Zhang 2013; Hager, Phan, and Zhu 2016; Liu, Liu, and Ma 2024) also use diagonal loading to equivalently reformulate the DkS problem as (4). The main differences between existing works and this paper are twofold. First, (Yuan and Zhang 2013) does not apply relaxation on (4), while (Hager, Phan, and Zhu 2016) applies a relaxation that is different from (5). Specifically, the relaxed problem is recast as sparse PCA with a non-convex constraint set. Second, (Hager, Phan, and Zhu 2016) only mentions choosing λ sufficiently large to ensure that $\boldsymbol{A} + \lambda \boldsymbol{I}$ is a positive definite matrix, without analyzing whether their sparse PCA-based relaxation is tight. This can also lead the algorithm to converge to a sub-optimal result. Meanwhile, (Yuan and Zhang 2013; Liu, Liu, and Ma 2024) only provide empirical strategies to avoid premature convergence to sub-optimal solutions by gradually increasing λ. None of these works provide theoretical justification for the choices of λ employed in their algorithms.

In this paper, we consider the continuous relaxation (5) and aim to derive an alternate set of sufficient conditions for λ so that the relaxation (5) is tight. If we can establish such a result for values of $\lambda \ll -\lambda_n(\boldsymbol{A})$, then it can make the optimization landscape of (5) "friendlier" for gradient-based approaches, so that higher quality solutions of DkS can be attained in practice while using principled choices of λ. The key technical idea is based on an extension of the classic Motzkin-Straus theorem, which forges a link between the global maxima of a non-convex quadratic program and the maximum clique problem.

Theorem 1 ((Motzkin and Straus 1965))**.** *Let $\mathcal{G}$ be an unweighted, undirected, and simple graph of n vertices. Let ω*

be the size of the largest clique of $\mathcal{G}$ and $\boldsymbol{A}$ be the adjacency matrix of $\mathcal{G}$. Let

$$f(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}\boldsymbol{A}\boldsymbol{x}, \tag{7}$$

then

$$f^* = \max_{\boldsymbol{x}\in\Delta_n} f(\boldsymbol{x}) = 1 - \frac{1}{\omega}, \tag{8}$$

where

$$\Delta_n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \sum_{i\in[n]} x_i = 1, x_i \geq 0, \forall i \in [n]\} \tag{9}$$

is the n-dimensional probability simplex.

We now present our extended version of the Motzkin-Straus theorem, which we will later use to analyze the choice of the requisite diagonal loading parameter λ.

Theorem 2 (Extended Motzkin-Straus Theorem)**.** *Let $\mathcal{G}$ be an unweighted, undirected, and simple graph of n vertices. Let ω be the size of the largest clique of $\mathcal{G}$ and $\boldsymbol{A}$ be the adjacency matrix of $\mathcal{G}$. Let*

$$g(\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x}, \tag{10}$$

where $\lambda \in [0, 1]$ and $\boldsymbol{I}$ is the identity matrix of size n, then

$$g^* = \max_{\boldsymbol{x}\in\Delta_n} g(\boldsymbol{x}) = 1 + \frac{\lambda - 1}{\omega}, \tag{11}$$

where

$$\Delta_n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \sum_{i\in[n]} x_i = 1, x_i \geq 0, \forall i \in [n]\}. \tag{12}$$

Proof. Please refer to Appendix B in (Lu, Sidiropoulos, and Konar 2024). □

With the above result in hand, consider the following problem

$$\begin{aligned} \max_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \Delta_k^n, \end{aligned} \tag{13}$$

where $\Delta_k^n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \sum_{i\in[n]} x_i = k, x_i \geq 0, \forall i \in [n]\}$. Observe that (13) can be viewed as a relaxation of (5). This yields the following chain of inequalities.

$$\begin{aligned} \max_{\boldsymbol{x}\in\mathcal{B}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} &\leq \max_{\boldsymbol{x}\in\mathcal{C}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} \\ &\leq \max_{\boldsymbol{x}\in\Delta_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x}. \end{aligned} \tag{14}$$

From Theorem 2, we know that if $\lambda \in [0, 1]$, then
• If $k < \omega$, then $\max_{\boldsymbol{x}\in\mathcal{B}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} = k(k+\lambda-1)$ and $\max_{\boldsymbol{x}\in\mathcal{C}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} = \max_{\boldsymbol{x}\in\Delta_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} = k^2 + \frac{k^2(\lambda-1)}{\omega}$. When λ is restricted to lie in $[0, 1]$, the relaxation from (4) to (5) is tight if and only if $\lambda = 1$.
• If $k = \omega$, then $\max_{\boldsymbol{x}\in\mathcal{B}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} = k(k + \lambda - 1)$ and $\max_{\boldsymbol{x}\in\mathcal{C}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} = \max_{\boldsymbol{x}\in\Delta_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} = k(k + \lambda - 1)$. The relaxation from (4) to (5) is tight for all $\lambda \in [0, 1]$.
• If $k > \omega$, then $\max_{\boldsymbol{x}\in\mathcal{B}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} \geq \omega(\omega - 1) + k\lambda = \omega^2(1 - \frac{1}{\omega}) + k\lambda$ and $\max_{\boldsymbol{x}\in\mathcal{C}_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x} \leq \max_{\boldsymbol{x}\in\Delta_k^n} \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} = k^2 + \frac{k^2(\lambda-1)}{\omega} = k^2(1-\frac{1}{\omega}) + \frac{k^2}{\omega}\lambda$. Since $\omega^2(1 - \frac{1}{\omega}) + k\lambda < k^2(1 - \frac{1}{\omega}) + \frac{k^2}{\omega}\lambda$, we cannot theoretically guarantee that the relaxation from (4) to (5) is tight for any $\lambda \in [0, 1]$.

To conclude, we have shown that setting the diagonal loading parameter $\lambda = 1$ in (5) suffices to guarantee that the relaxation from (4) to (5) is tight for subgraph sizes k no greater than the maximum clique size ω. Note that this result is guaranteed to hold for *any* graph. Furthermore, in contrast to using the value $\lambda \geq -\lambda_n(\boldsymbol{A})$ (as suggested by using standard results in optimization (Rockafellar 1970)), it is surprising that we can establish tightness of the non-convex relaxation (5) for the significantly smaller and data independent value of $\lambda = 1$. Experimental results in this paper will further reveal that the diagonal loading parameter $\lambda = 1$ effectively balances the trade-off between the quality of solutions and the convergence rate.

Proposed Approaches

To solve the optimization problem (5), projected gradient ascent (PGA) is a natural choice. However, ensuring that the iterates of PGA remain feasible requires computing the projection onto the polytope $\boldsymbol{x} \in \mathcal{C}_k^n$ in every iteration. This can prove to be a computational burden since the projection problem does not admit a closed-form solution. To bypass the projection step altogether, we employ two projection-free approaches in this paper to solve the optimization problem (5): one based on the Frank-Wolfe algorithm (Frank and Wolfe 1956; Jaggi 2013), and the other based on explicit constraint parameterization.

The Frank-Wolfe Algorithm: The Frank-Wolfe algorithm (Frank and Wolfe 1956; Jaggi 2013) is a first-order projection-free method that can be employed to maximize a continuously differentiable function with a Lipschitz continuous gradient over a nonempty, closed, and convex set (Bertsekas 2016, Section 3.2). Pseudocode for the algorithm is provided in Algorithm 1.

Algorithm 1: Frank-Wolfe for problem (5)

Input: The adjacency matrix $\boldsymbol{A}$, the subgraph size k, and the diagonal loading parameter λ.
Output: A solution of the optimization problem (5).

$L = \|\boldsymbol{A} + \lambda\boldsymbol{I}\|_2$;
while *the convergence criterion is not met* **do**
  $\boldsymbol{g}^t \leftarrow (\boldsymbol{A} + \lambda\boldsymbol{I})\boldsymbol{x}^t$;
  $\boldsymbol{s}^t \leftarrow \text{supp}(\text{top}_k(\boldsymbol{g}^t))$;
  $\boldsymbol{d}^t \leftarrow \boldsymbol{s}^t - \boldsymbol{x}^t$;
  Option I: $\gamma^t \leftarrow \min\left\{1, \frac{(\boldsymbol{g}^t)^\mathsf{T}\boldsymbol{d}^t}{L\|\boldsymbol{d}^t\|_2^2}\right\}$;
  Option II: $\gamma^t \leftarrow \min\left\{1, \frac{(\boldsymbol{g}^t)^\mathsf{T}\boldsymbol{d}^t}{2kL}\right\}$;
  $\boldsymbol{x}^{t+1} \leftarrow \boldsymbol{x}^t + \gamma^t\boldsymbol{d}^t$;
  $t \leftarrow t + 1$;

The algorithm first requires finding an initial feasible solution, which is easy for the optimization problem (5). After

initialization, at each iteration, Line 3 in Algorithm 1 computes the gradient $\boldsymbol{g}^t$ at the current iterate $\boldsymbol{x}^t$. Line 4 uses the gradient direction to determine an ascent direction $\boldsymbol{s}^t$ by solving the following linear maximization problem (LMP)

$$\boldsymbol{s}^t \in \arg\max_{\boldsymbol{x}\in\mathcal{C}_k^n} (\boldsymbol{g}^t)^\mathsf{T}\boldsymbol{x}. \tag{15}$$

The LMP admits a simple closed-form solution - simply extract the support of the largest k elements of the gradient of the objective function. Lines 6 and 7 in Algorithm 1 offer two choices to calculate the step size in each iteration: Option I is taken from (Bertsekas 2016, p. 268) whereas Option II is mentioned in (Lacoste-Julien 2016). Both step size schedules guarantee convergence of the generated iterates to a stationary point of (5), which satisfies necessary conditions for optimality. Additionally, the convergence rate of the step size provided by Option II is $O(1/\sqrt{t})$ (Lacoste-Julien 2016). Line 8 in Algorithm 1 updates the feasible solution $\boldsymbol{x}^t$ through a convex combination using the step size obtained from one of the previous options.

Parameterization: Another approach to bypass the projection onto the set $\mathcal{C}_k^n$ is to equivalently reformulate the constrained problem (5) as an unconstrained optimization problem through an appropriate change of variables. As a result, we can apply algorithms such as Adam (Kingma and Ba 2015) or AdamW (Loshchilov and Hutter 2019) to solve this new unconstrained problem.

We now outline our second approach. Since the adjacency matrix $\boldsymbol{A}$ is non-negative, the optimization problem (5) is equivalent to

$$\begin{aligned} \max_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{x}^\mathsf{T}(\boldsymbol{A}+\lambda\boldsymbol{I})\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{x} \in \mathcal{D}_k^n. \end{aligned} \tag{16}$$

where $\mathcal{D}_k^n := \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \in [0,1]^n, \sum_{i\in[n]} x_i \le k\}$. For the constraints $\boldsymbol{x} \in [0,1]^n$, consider the parameterization

$$x_i = \frac{1}{(1+\mathrm{e}^{\phi})(1+\mathrm{e}^{-\theta_i})} = \frac{\sigma(\theta_i)}{1+\mathrm{e}^{\phi}}, \forall i \in [n], \tag{17}$$

where $\sigma(\cdot)$ is the sigmoid function. Clearly, $x_i \in [0,1], \forall \phi \in \mathbb{R}, \forall \theta_i \in \mathbb{R}, \forall i \in [n]$. Furthermore, the remaining constraint

$$\sum_{i\in[n]} x_i = \frac{1}{1+\mathrm{e}^{\phi}} \sum_{i\in[n]} \sigma(\theta_i) \le k \tag{18}$$

can be parameterized as

$$\mathrm{e}^{\phi} = \max\left\{ \frac{1}{k}\sum_{i\in[n]} \sigma(\theta_i) - 1, 0 \right\}. \tag{19}$$

Hence, the constrained variables $\boldsymbol{x} \in \mathcal{C}_k^n$ in problem (16) can be parameterized in terms of the free variables $\{\theta_i\}_{i=1}^n$ as

$$x_i = \frac{\sigma(\theta_i)}{1+\max\left\{\frac{1}{k}\sum_{i\in[n]}\sigma(\theta_i) - 1, 0\right\}}, \forall i \in [n]. \tag{20}$$

Changing the variables to $\{\theta_i\}_{i=1}^n$ in (16) results in an unconstrained problem. Notice that when $k = 1$, the variable ϕ is not needed, reducing to the well-known softmax parameterization. To the best of our knowledge, this is the first time that the more general constraint $\mathcal{D}_k^n$ has been parameterized.

Computational Complexity

We first consider the computational complexity of the Frank-Wolfe algorithm. The addition of $\boldsymbol{A}$ and $\lambda\boldsymbol{I}$ takes $O(m+n) = O(m)$ time because $\boldsymbol{A}$ and $\lambda\boldsymbol{I}$ have $O(m)$ and $O(n)$ non-zeros elements, respectively. The multiplication of $\boldsymbol{A}+\lambda\boldsymbol{I}$ and $\boldsymbol{x}$ takes $O(m)$ time because $\boldsymbol{A}+\lambda\boldsymbol{I}$ has $O(m)$ non-zeros elements. If we use the Power method and consider the number of iterations for it as a constant, then we need $O(m)$ time to obtain the Lipschitz constant L. For Line 3 in Algorithm 1, as mentioned earlier, it takes $O(m)$ time. For Line 4, we can build a heap by Floyd's algorithm in $O(n)$ time and it takes $O(k\log n)$ time to select the largest k elements from the heap. For Lines 5 to 9, it takes $O(n)$ time. Therefore, the per-iteration computational complexity of the Frank-Wolfe algorithm is $O(m + k\log n)$.

For parameterization, the per-iteration computational complexity depends on the cost of computing the (sub)gradient of the objective function $f(\boldsymbol{\theta})$ of the parameterized optimization problem. Since f is not differentiable only when $\frac{1}{k}\sum_{i\in[n]}\sigma(\theta_i) = 1$, we consider two cases: $\frac{1}{k}\sum_{i\in[n]}\sigma(\theta_i) > 1$ and $\frac{1}{k}\sum_{i\in[n]}\sigma(\theta_i) < 1$.

• If $\frac{1}{k}\sum_{i\in[n]}\sigma(\theta_i) > 1$, then $x_j = \frac{k\sigma(\theta_j)}{\sum_{i\in[n]}\sigma(\theta_i)}, \forall j \in [n]$. For every $j \in [n]$, we have

$$\frac{\partial f}{\partial \theta_j} = \frac{\partial f}{\partial x_j}\cdot\frac{\partial x_j}{\partial \theta_j} + \sum_{l\neq j}\frac{\partial f}{\partial x_l}\cdot\frac{\partial x_l}{\partial \theta_j}. \tag{21}$$

Since $\boldsymbol{A}+\lambda\boldsymbol{I}$ has $O(m)$ non-zeros elements, all partial derivatives in the tuple $\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$ can be obtained in $O(m)$ time. Since

$$\frac{\partial x_j}{\partial \theta_j} = \frac{k\sigma(\theta_j)(1-\sigma(\theta_j))(\sum_{i\neq j}\sigma(\theta_i))}{(\sum_{i\in[n]}\sigma(\theta_i))^2}, \tag{22}$$

and for every $l \neq j$

$$\frac{\partial x_l}{\partial \theta_j} = -\frac{k\sigma(\theta_j)(1-\sigma(\theta_j))\sigma(\theta_l)}{(\sum_{i\in[n]}\sigma(\theta_i))^2}, \tag{23}$$

we can obtain all partial derivatives in the Jacobian matrix

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial x_1}{\partial \theta_1} & \frac{\partial x_1}{\partial \theta_2} & \cdots & \frac{\partial x_1}{\partial \theta_n} \\ \frac{\partial x_2}{\partial \theta_1} & \frac{\partial x_2}{\partial \theta_2} & \cdots & \frac{\partial x_2}{\partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial \theta_1} & \frac{\partial x_n}{\partial \theta_2} & \cdots & \frac{\partial x_n}{\partial \theta_n} \end{bmatrix} \tag{24}$$

in $O(n^2)$ time by precomputing $\sum_{i\in[n]}\sigma(\theta_i)$. Hence, it takes $O(m+n^2) = O(n^2)$ time to compute the gradient.

• If $\frac{1}{k}\sum_{i\in[n]}\sigma(\theta_i) < 1$, then $x_j = \sigma(\theta_j), \forall j \in [n]$. For every $j \in [n]$, we have

$$\frac{\partial f}{\partial \theta_j} = \frac{\partial f}{\partial x_j}\cdot\frac{\partial x_j}{\partial \theta_j}. \tag{25}$$

It is clear that we can obtain all partial derivatives in the following tuples $\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right)$ and $\left(\frac{\partial x_1}{\partial \theta_1}, \frac{\partial x_2}{\partial \theta_2}, \cdots, \frac{\partial x_n}{\partial \theta_n}\right)$ in $O(m)$ and $O(n)$ time, respectively. Hence, it takes $O(m+n) = O(m)$ time to compute the gradient.

Network	n	m	ω
Facebook	4,039	88,234	69
web-Stanford	281,903	1,992,636	61
com-Youtube	1,134,890	2,987,624	17
soc-Pokec	1,632,803	22,301,964	29
as-Skitter	1,696,415	11,095,298	67
com-Orkut	3,072,441	117,185,083	51

Table 1: The details of the pre-processed datasets (n denotes the number of vertices, m denotes the number of edges, and ω denotes the size of the largest clique).

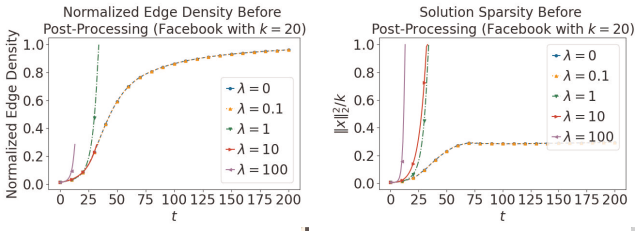

Figure 1: The comparison of different diagonal loading parameters for the Facebook dataset.

Experimental Results

In this section, we first compare different choices of the diagonal loading parameter λ and various step sizes for the Frank-Wolfe algorithm. Subsequently, we compare the proposed algorithms with existing algorithms to illustrate the effectiveness of our approaches.

Datasets: We pre-processed the datasets by converting directed graphs to undirected graphs and removing self-loops. A summary of the datasets used in this paper is provided in Table 1. All datasets were downloaded from (Leskovec and Krevl 2014), and the sizes of the largest cliques were obtained from (Jursa 2016; Jain and Seshadhri 2020).

Baselines: The following baselines are used for comparison with the algorithms proposed in this paper.

• **Greedy:** The greedy approximation algorithm was proposed in (Feige, Peleg, and Kortsarz 2001, Procedure 2). This algorithm first selects $k/2$ vertices with the maximum degrees from the vertex set $\mathcal{V}$, and these selected vertices form a subset of the vertices $\mathcal{H} \subseteq \mathcal{V}$. This algorithm then selects $k/2$ vertices with the maximum number of neighbors in $\mathcal{H}$ from the set $\mathcal{V}\backslash\mathcal{H}$, and these selected vertices form a subset of the vertices $\mathcal{C} \subseteq \mathcal{V}\backslash\mathcal{H}$. The output of this algorithm is $\mathcal{H} \cup \mathcal{C}$.

• **Low-Rank Bilinear Optimization (LRBO):** The low-rank bilinear optimization algorithm was proposed in (Papailiopoulos et al. 2014). This algorithm first performs a rank-d approximation on the adjacency matrix $\boldsymbol{A}$ to obtain $\boldsymbol{A}_d$. This algorithm then uses the Spannogram, an exact bilinear optimization solver, to solve the DkS problem approximately. The time complexity of this algorithm is $O(n^{d+1})$. The experimental results in (Konar and Sidiropoulos 2021) demonstrate that even performing a rank-2 approximation on a small dataset, the execution time of this algorithm significantly exceeds that of other algorithms. Therefore, in the experiments conducted in this paper, we only consider rank-1 approximation.

• **L-ADMM:** The L-ADMM algorithm was proposed in (Konar and Sidiropoulos 2021) to solve the Lovaśz relaxation optimization problem. Since the solution obtained by this algorithm is not guaranteed to be an integer vector in general, the solution will be projected onto the combinatorial sum-to-k constraint $\boldsymbol{x} \in \mathcal{B}_k^n$ as a post-processing step.

• **EXPP:** The EXPP algorithm was proposed in (Liu, Liu, and Ma 2024). This algorithm first relaxes the binary constraint and adds a proper penalty function, then utilizes a homotopy optimization method to solve the problem. This algorithm requires the same post-processing as L-ADMM.

• **Upper Bound (UB):** The upper bound of the edge density was introduced in (Papailiopoulos et al. 2014, Lemma 3). In this paper, we use the same normalized edge density as in (Konar and Sidiropoulos 2021), which is defined by

$$\min\left\{1, \frac{\mathbf{1}_{\mathcal{S}_1^*}^\top \boldsymbol{A}_1 \mathbf{1}_{\mathcal{S}_1^*}}{k(k-1)} + \frac{\sigma_2(\boldsymbol{A})}{k-1}, \frac{\sigma_1(\boldsymbol{A})}{k-1}\right\}, \tag{26}$$

where $\sigma_i(\boldsymbol{A})$ denotes the i-th largest singular value of $\boldsymbol{A}$, $\boldsymbol{A}_1$ denotes the rank-1 approximation on $\boldsymbol{A}$, and $\mathcal{S}_1^*$ is the optimal solution of the rank-1 approximate DkS problem.

Implementation: In this paper, all the code is implemented in Python, and all the experiments were conducted on a workstation with 256GB RAM and an AMD Ryzen Threadripper PRO 5975WX processor. We use the built-in AdamW optimizer in PyTorch (Paszke et al. 2019) to solve the unconstrained optimization problem after parameterization. The initialization of the Frank-Wolfe algorithm is chosen to be $x_i = \frac{k}{n}, \forall i \in [n]$ and the initialization of AdamW is chosen to be $\theta_i = 0, \forall i \in [n]$. Unless specifically stated, the diagonal loading parameter $\lambda = 1$ and the Frank-Wolfe algorithm uses Option I in Algorithm 1 for the step size. AdamW uses learning rates of 3. The maximum number of iterations for both the Frank-Wolfe algorithm and AdamW is set to 200. Similar to L-ADMM, the proposed algorithms also perform a projection onto the combinatorial sum-to-k constraint $\boldsymbol{x} \in \mathcal{B}_k^n$ as a post-processing step. For baselines, we adopt the parameters used in the original papers.

Selecting the Diagonal Loading Parameter: Theorem 2 proves that if the diagonal loading parameter $\lambda = 1$, the relaxation from (4) to (5) is tight when $k \leq \omega$, where ω is the size of the largest clique of $\mathcal{G}$. Figure 1 shows that in comparison to $\lambda = 10$ and $\lambda = 100$, the Frank-Wolfe algorithm with $\lambda = 1$ achieves better subgraph density. In comparison to $\lambda = 0$ and $\lambda = 0.1$, the Frank-Wolfe algorithm with $\lambda = 1$ exhibits a faster convergence rate and also is more prone to converge to integer solutions. The experimental results in Figure 1 indicate that the diagonal loading parameter setting $\lambda = 1$ effectively achieves a balance between the quality of solutions and the convergence rate.

Step size selection for Frank-Wolfe: Figure 2 demonstrates that although Option I in Algorithm 1 does not guarantee a convergence rate of $O(1/\sqrt{t})$ like Option II, Option I offers much faster convergence rate than Option II. Therefore, we

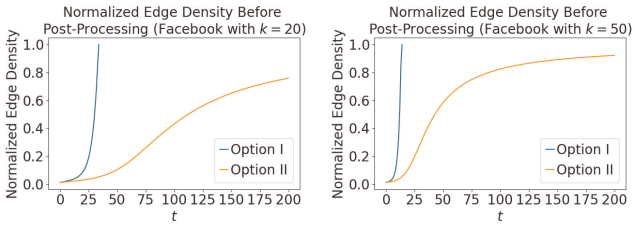

Figure 2: The comparison of the Frank-Wolfe algorithm with different step sizes for the Facebook dataset.

adopt Option I as the step size for the Frank-Wolfe algorithm.

Comparison with Baselines: Figures 3 and 4 show the normalized edge density and the execution time of the two proposed algorithms and baselines on various datasets and different subgraph sizes. Through Figures 3 and 4, we have the following observations:

• The proposed Frank-Wolfe algorithm generally yields better subgraph density in the vast majority of cases, while also exhibiting lower computational cost compared to other algorithms that achieve similar subgraph density.

• Using the AdamW optimizer to solve the proposed parameterized optimization problem yields better subgraph density in certain cases, but overall it does not outperform the proposed Frank-Wolfe algorithm. This may be related to the landscape of the parameterized optimization problem, which may not be as favorable as that of the original optimization problem. The high computational cost of AdamW is partially attributable to the fact that matrix computations in PyTorch are more costly than those in NumPy.

• Theoretical analysis only guarantees the tightness of the relaxation from (4) to (5) when the size of subgraph k is smaller than or equal to the size of the largest clique ω; however, experimental results demonstrate its good performance in a wide range of scenarios.

Conclusion

In this paper, we consider the DkS problem. We first reformulate the DkS problem through diagonal loading, and then relax the combinatorial sum-to-k constraint $\boldsymbol{x} \in \mathcal{B}_k^n$ to its convex hull. We prove the tightness of this relaxation under certain conditions by leveraging an extension of the Motzkin-Straus theorem that we also prove herein. We propose two projection-free approaches to solve the relaxed optimization problem. The first approach is based on the Frank-Wolfe algorithm, while the second approach involves first reformulating the relaxed optimization problem to a new unconstrained optimization problem and then solving the new optimization problem using AdamW. We experimentally verify the effectiveness of diagonal loading with parameter $\lambda = 1$ and the relaxation. Experimental results show that the Frank-Wolfe algorithm is relatively lightweight and a top performer in most cases considered. It also scales well to large networks with millions of nodes.

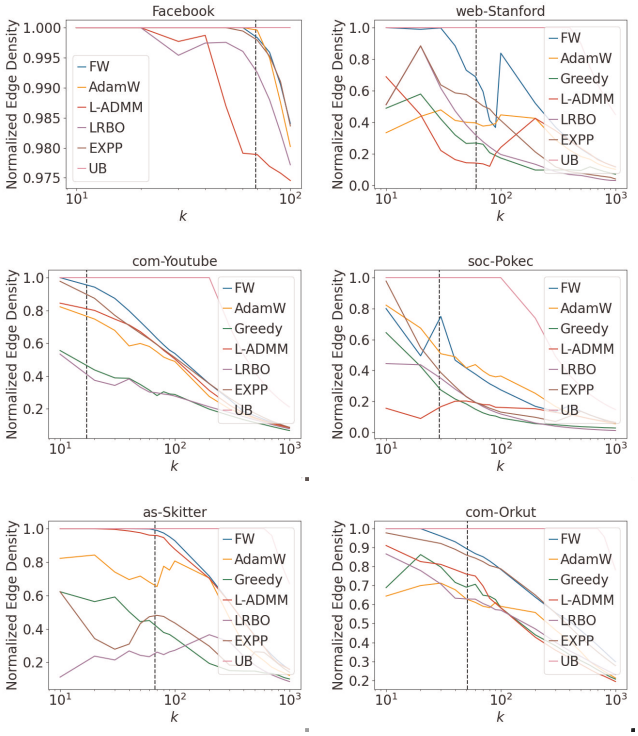

Figure 3: Normalized edge density versus subgraph size (the dashed line indicates the size of the largest clique. The greedy algorithm is omitted in Facebook to bring out the differences among the other algorithms).

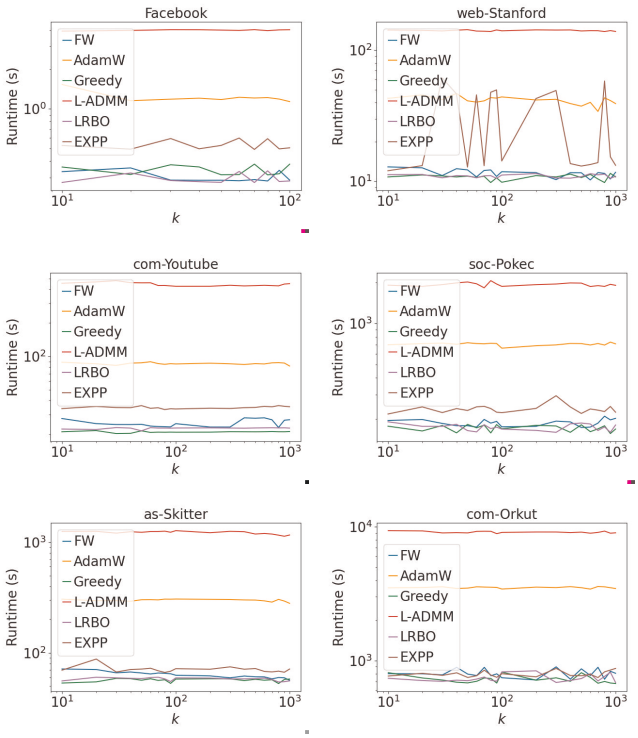

Figure 4: Execution time versus subgraph size.

Acknowledgements

Supported by the KU Leuven Special Research Fund (BOF/STG-22-040) and the National Science Foundation, USA (IIS-1908070). The authors would like to thank Ya Liu, Junbin Liu, and Professor Wing-Kin Ma for sharing their code for (Liu, Liu, and Ma 2024) as a baseline for comparison.

References

Ames, B. P. 2015. Guaranteed Recovery of Planted Cliques and Dense Subgraphs by Convex Relaxation. *Journal of Optimization Theory and Applications*, 167: 653–675.

Andersen, R.; and Chellapilla, K. 2009. Finding Dense Subgraphs with Size Bounds. In *International Workshop on Algorithms and Models for the Web-Graph*, 25–37. Springer.

Angel, A.; Koudas, N.; Sarkas, N.; Srivastava, D.; Svendsen, M.; and Tirthapura, S. 2014. Dense Subgraph Maintenance under Streaming Edge Weight Updates for Real-time Story Identification. *The VLDB journal*, 23: 175–199.

Bertsekas, D. 2016. *Nonlinear Programming*. Belmont, MA: Athena Scientific, 3rd edition.

Bhaskara, A.; Charikar, M.; Chlamtac, E.; Feige, U.; and Vijayaraghavan, A. 2010. Detecting High Log-Densities: an $O(n^{1/4})$ Approximation for Densest k-Subgraph. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, 201–210. ACM.

Bhaskara, A.; Charikar, M.; Guruswami, V.; Vijayaraghavan, A.; and Zhou, Y. 2012. Polynomial Integrality Gaps for Strong SDP Relaxations of Densest k-Subgraph. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, 388–405. SIAM.

Bombina, P.; and Ames, B. 2020. Convex Optimization for the Densest Subgraph and Densest Submatrix Problems. In *SN Operations Research Forum*, volume 1, 1–24. Springer.

Boob, D.; Gao, Y.; Peng, R.; Sawlani, S.; Tsourakakis, C.; Wang, D.; and Wang, J. 2020. Flowless: Extracting Densest Subgraphs Without Flow Computations. In *Proceedings of The Web Conference 2020*, 573–583. ACM.

Chandrasekaran, K.; Chekuri, C.; Torres, M. R.; and Zhu, W. 2024. On the Generalized Mean Densest Subgraph Problem: Complexity and Algorithms. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024)*, volume 317, 9:1–9:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Charikar, M. 2000. Greedy Approximation Algorithms for Finding Dense Components in a Graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, 84–95. Springer.

Chekuri, C.; Quanrud, K.; and Torres, M. R. 2022. Densest Subgraph: Supermodularity, Iterative Peeling, and Flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1531–1555. SIAM.

Chen, T.; and Tsourakakis, C. 2022. Antibenford Subgraphs: Unsupervised Anomaly Detection in Financial Networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2762–2770. ACM.

Condat, L. 2013. A Primal-Dual Splitting Method for Convex Optimization Involving Lipschitzian, Proximable and Linear Composite Terms. *Journal of Optimization Theory and Applications*, 158(2): 460–479.

Danisch, M.; Chan, T.-H. H.; and Sozio, M. 2017. Large Scale Density-Friendly Graph Decomposition via Convex Programming. In *Proceedings of the 26th International Conference on World Wide Web*, 233–242.

Eckstein, J.; and Bertsekas, D. P. 1992. On the Douglas—Rachford Splitting Method and the Proximal Point Algorithm for Maximal Monotone Operators. *Mathematical Programming*, 55: 293–318.

Feige, U.; Peleg, D.; and Kortsarz, G. 2001. The Dense k-Subgraph Problem. *Algorithmica*, 29: 410–421.

Frank, M.; and Wolfe, P. 1956. An Algorithm for Quadratic Programming. *Naval Research Logistics Quarterly*, 3(1-2): 95–110.

Fratkin, E.; Naughton, B. T.; Brutlag, D. L.; and Batzoglou, S. 2006. MotifCut: Regulatory Motifs Finding with Maximum Density Subgraphs. *Bioinformatics*, 22(14): e150–e157.

Goldberg, A. V. 1984. Finding a Maximum Density Subgraph. Technical report.

Hager, W. W.; Phan, D. T.; and Zhu, J. 2016. Projection Algorithms for Nonconvex Minimization with Application to Sparse Principal Component Analysis. *Journal of Global Optimization*, 65: 657–676.

Harb, E.; Quanrud, K.; and Chekuri, C. 2022. Faster and Scalable Algorithms for Densest Subgraph and Decomposition. *Advances in Neural Information Processing Systems*, 35: 26966–26979.

Harb, E.; Quanrud, K.; and Chekuri, C. 2023. Convergence to Lexicographically Optimal Base in a (Contra)Polymatroid and Applications to Densest Subgraph and Tree Packing. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274, 56:1–56:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Hooi, B.; Song, H. A.; Beutel, A.; Shah, N.; Shin, K.; and Faloutsos, C. 2016. Fraudar: Bounding Graph Fraud in the Face of Camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 895–904. ACM.

Jaggi, M. 2013. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *Proceedings of the 30th International Conference on Machine Learning*, 427–435. PMLR.

Jain, S.; and Seshadhri, C. 2020. The Power of Pivoting for Exact Clique Counting. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 268–276. ACM.

Ji, Y.; Zhang, Z.; Tang, X.; Shen, J.; Zhang, X.; and Yang, G. 2022. Detecting Cash-out Users via Dense Subgraphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 687–697. ACM.

Jursa, A. 2016. Fast Algorithm for Finding Maximum Clique in Scale-Free Networks. In *Proceedings of the 16th*

ITAT Conference Information Technologies - Applications and Theory, 212–217.

Kawase, Y.; and Miyauchi, A. 2018. The Densest Subgraph Problem with a Convex/Concave Size Function. *Algorithmica*, 80: 3461–3480.

Khot, S. 2006. Ruling Out PTAS for Graph Min-Bisection, Dense k-Subgraph, and Bipartite Clique. *SIAM Journal on Computing*, 36(4): 1025–1071.

Khuller, S.; and Saha, B. 2009. On Finding Dense Subgraphs. In *International Colloquium on Automata, Languages, and Programming*, 597–608. Springer.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Konar, A.; and Sidiropoulos, N. D. 2021. Exploring the Subgraph Density-Size Trade-off via the Lovaśz Extension. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 743–751. ACM.

Lacoste-Julien, S. 2016. Convergence Rate of Frank-Wolfe for Non-Convex Objectives. *arXiv preprint arXiv:1607.00345*.

Lanciano, T.; Miyauchi, A.; Fazzone, A.; and Bonchi, F. 2024. A survey on the densest subgraph problem and its variants. *ACM Computing Surveys*, 56(8): 1–40.

Leskovec, J.; and Krevl, A. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

Li, X.; Liu, S.; Li, Z.; Han, X.; Shi, C.; Hooi, B.; Huang, H.; and Cheng, X. 2020. Flowscope: Spotting Money Laundering Based on Graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 4731–4738.

Liu, Y.; Liu, J.; and Ma, W.-K. 2024. Cardinality-Constrained Binary Quadratic Optimization via Extreme Point Pursuit, with Application to the Densest K-Subgraph Problem. In *2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 9631–9635.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Lu, Q.; Sidiropoulos, N. D.; and Konar, A. 2024. On Densest k-Subgraph Mining and Diagonal Loading. *arXiv preprint arXiv:2410.07388*.

Luo, W.; Ma, C.; Fang, Y.; and Lakshman, L. V. 2023. A Survey of Densest Subgraph Discovery on Large Graphs. *arXiv preprint arXiv:2306.07927*.

Manurangsi, P. 2017. Almost-Polynomial Ratio ETH-Hardness of Approximating Densest k-Subgraph. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, 954–961. ACM.

Motzkin, T. S.; and Straus, E. G. 1965. Maxima for Graphs and a New Proof of a Theorem of Turán. *Canadian Journal of Mathematics*, 17: 533–540.

Nguyen, T. D.; and Ene, A. 2024. Multiplicative Weights Update, Area Convexity and Random Coordinate Descent for Densest Subgraph Problems. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, 37683–37706. PMLR.

Papailiopoulos, D.; Mitliagkas, I.; Dimakis, A.; and Caramanis, C. 2014. Finding Dense Subgraphs via Low-Rank Bilinear Optimization. In *Proceedings of the 31st International Conference on Machine Learning*, 1890–1898. PMLR.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Rockafellar, R. T. 1970. *Convex Analysis*. Princeton, NJ: Princeton University Press.

Saha, B.; Hoch, A.; Khuller, S.; Raschid, L.; and Zhang, X.-N. 2010. Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs. In *14th International Conference on Research in Computational Molecular Biology (RECOMB 2010)*, 456–472. Springer.

Seidman, S. B. 1983. Network Structure and Minimum Degree. *Social Networks*, 5(3): 269–287.

Shin, K.; Eliassi-Rad, T.; and Faloutsos, C. 2016. Corescope: Graph Mining Using k-Core Analysis — Patterns, Anomalies and Algorithms. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 469–478. IEEE.

Sotirov, R. 2020. On Solving the Densest k-Subgraph Problem on Large Graphs. *Optimization Methods and Software*, 35(6): 1160–1178.

Tsourakakis, C.; Bonchi, F.; Gionis, A.; Gullo, F.; and Tsiarli, M. 2013. Denser than the Densest Subgraph: Extracting Optimal Quasi-Cliques with Quality Guarantees. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 104–112. ACM.

Veldt, N.; Benson, A. R.; and Kleinberg, J. 2021. The Generalized Mean Densest Subgraph Problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1604–1614. ACM.

Yuan, X.-T.; and Zhang, T. 2013. Truncated Power Method for Sparse Eigenvalue Problems. *Journal of Machine Learning Research*, 14(28): 899–925.