# DUAL ALGORITHMS FOR THE DENSEST SUBGRAPH PROBLEM

A Dissertation
Presented to
The Academic Faculty

By

Saurabh Sawlani

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Algorithms, Combinatorics and Optimization

Georgia Institute of Technology

March 2020

# DUAL ALGORITHMS FOR THE DENSEST SUBGRAPH PROBLEM

The only moment of possible happiness is the present.

*Arsene Wenger*

## ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor Richard Peng for his constant encouragement and guidance throughout my PhD. His genuine concern for me as a young researcher, and as a person, played a pivotal role in ensuring a smooth stay for me at Georgia Tech. Richard's enthusiasm and aptitude for research is something I continue to look up to.

I am extremely grateful to Richard Peng, Dana Randall, Eric Vigoda, Xingxing Yu, and Charalampos Tsourakakis for agreeing to serve on my thesis committee, with special thanks to Charalampos Tsourakakis for being the reader for my dissertation. I would like to thank Robin Thomas and Prasad Tetali for leading the ACO program and providing support and advice at any time I needed it.

I would like to thank all my collaborators throughout my PhD: Digvijay Boob, Timothy Chu, Laxman Dhulipala, Yihe Dong, David Durfee, Matthew Fahrbach, Yu Gao, Gorav Jindal, Pavel Kolev, Janardhan Kulkarni, Kevin Lai, Gary Miller, Richard Peng, Ilya Razenshteyn, Sushant Sachdeva, Xiaorui Sun, Charalampos Tsourakakis, Di Wang, Junxing Wang and Shen Chen Xu.

I am thankful to my friends and colleagues from the Theory Lab, my intramural teammates, my roommates, and everyone else who made my stay in Georgia Tech worthwhile. I would like to express my special gratitude to Nanditha Rajamani, for her constant love and support.

Lastly, I would like to thank my parents for their unconditional love and encouragement every step of the way.

## TABLE OF CONTENTS

# LIST OF TABLES

## LIST OF FIGURES

# SUMMARY

Dense subgraph discovery is an important primitive for many real-world applications such as community detection, link spam detection, distance query indexing, and computational biology. In this thesis, our focus is on solving the densest subgraph problem - finding a subgraph with the highest average degree in a graph.

The densest subgraph problem is solvable in polynomial time using maximum flows [42]. Since maximum flow computations are expensive despite the theoretical progress achieved over the recent years, Charikar's greedy peeling algorithm is frequently used in practice [26]. This algorithm is simple, runs in linear time, and provides a $1/2$-approximation for the problem. Bahmani *et al.* [11] show that the dual of the problem is a mixed packing and covering linear program[1], which they solve using the multiplicative weight update framework to derive a near-linear time algorithm to find the densest subgraph.

First, we give a faster width-dependent algorithm to solve mixed packing and covering LPs, a class of problems that is fundamental to combinatorial optimization in computer science and operations research. Our algorithm finds a $1 + \varepsilon$ approximate solution in time $O(Nw/\varepsilon)$, where $N$ is number of nonzero entries in the constraint matrix and $w$ is the width (maximum number of nonzeros in any constraint). This run-time is better than Nesterov's smoothing algorithm which requires $O(N\sqrt{n}w/\varepsilon)$ where $n$ is the dimension of the problem. Our work utilizes the framework of area convexity introduced by Sherman [85] to obtain the best dependence on $\varepsilon$ while breaking the infamous $\ell_\infty$ barrier to eliminate the factor of $\sqrt{n}$. The current best width-independent algorithm for this problem runs in time $O(N/\varepsilon^2)$ [100] and hence has worse running time dependence on $\varepsilon$. As a special case of our result, we report a $(1-\varepsilon)$ approximation algorithm for the densest subgraph problem which runs in time $O(m \deg_{\max}/\varepsilon)$, where $m$ is the number of edges in the graph and

[1]A mixed packing and covering linear program is a linear program in which all coefficients, variables and constraints are non-negative.

$\deg_{\max}$ is the maximum graph degree.

Secondly, we devise an iterative algorithm for the densest subgraph problem which naturally generalizes Charikar's greedy algorithm. Our algorithm draws insights from the iterative approaches from convex optimization, and also exploits the dual interpretation of the densest subgraph problem. We have empirical evidence that our algorithm is much more robust against the structural heterogeneities in real-world datasets, and converges to the optimal subgraph density even when the simple greedy algorithm fails. On the other hand, in instances where Charikar's algorithm performs well, our algorithm is able to quickly verify its optimality. Furthermore, we demonstrate that our method is significantly faster than the maximum flow based exact optimal algorithm. We conduct experiments on datasets from broad domains, and our algorithm achieves $\sim 145\times$ speedup on average to find subgraphs whose density is at least 90% of the optimal value.

Lastly, we design the first fully-dynamic algorithm which maintains a $(1-\varepsilon)$ approximate densest subgraph in worst-case poly$(\log n, \varepsilon^{-1})$ per update. We approach the densest subgraph problem by framing its dual as a graph orientation problem, which we solve using an augmenting path-like adjustment technique [61]. Our result improves upon the previous best approximation factor of $(1/4-\varepsilon)$ for fully dynamic densest subgraph [21]. We also extend our techniques to solving the problem on vertex-weighted graphs and directed graphs with similar runtimes.

# CHAPTER 1
# INTRODUCTION

Finding dense components in graphs is an important primitive for a variety real-world applications involving large networks. Applications of dense subgraph discovery include

- community detection in social networks [64, 77, 63, 30, 27],
- unsupervised detection of stories from micro-blogging streams in real time [8],
- indexing graphs for efficient distance query computation [28, 56, 5],
- anomaly detection in financial networks and social networks [41],
- link spam detection [40], and
- DNA motif detection and computational biology [54, 80, 79]

among others.

Due to its practical relevance, many notions of graph density have been studied in literature. The prototypical dense subgraph is a clique. However, the maximum clique problem is not only NP-hard, but also strongly inapproximable [49]. The notion of optimal quasi-cliques has been developed to detect subgraphs that are not necessarily fully interconnected but very dense [95]. However, finding optimal quasi-cliques is also NP-hard [59, 93]. A popular and scalable approach to finding dense components is based on $k$-cores [84, 33]. Recently, $k$-cores have also been used to detect anomalies in large-scale networks [39, 86]. [66, 90, 96] contain several other applications of dense subgraphs and related problems.

In this thesis, we study one of the most widely used formulations of dense subgraph discovery - the *densest subgraph problem*, which is formulated as follows: given an undirected graph $G(V, E)$ we want to find a set of nodes $S \subseteq V$ that maximizes the *degree density* $\frac{E(S)}{|S|}$, where $E(S)$ is the number of edges in the graph contained within $S$.

In addition to its practical applicability, the densest subgraph problem also has great theoretical relevance due to its close connection to fundamental graph problems such as *network flow* and *bipartite matching*[1]. While near-linear time algorithms exist for finding matchings in graphs [72, 37, 31], the same cannot be said for flows on directed graphs [69]. In this sense, the problem acts as an indicative middle ground, since it is both a specific instance of a flow problem [42, 11], as well as a generalization of bipartite $b$-matchings. Interestingly, the densest subgraph problem does allow near-linear time algorithms [11].

## 1.1 Preliminaries

In this section, we give a brief overview of some concepts and definitions which will be used throughout the thesis.

### 1.1.1 Subgraph Density

We represent any undirected graph $G$ as $G = \langle V, E\rangle$, where $V$ is the set of vertices in $G$, $E$ is the set of edges in $G$. For any subset of vertices $S \subseteq V$, we use $E(S)$ to denote the subset of all edges within $S$, and $G(S) = \langle S, E(S)\rangle$ to denote the graph induced by $S$.

**Definition 1.1** (Degree density)**.** The *degree density* $\rho_G(S)$ of a subgraph $G(S)$ in $G$ is defined as

$$\rho_G(S) = \frac{E(S)}{|S|}.$$

Note that $\rho_G(S)$ is simply twice the average degree of $G(S)$ - hence the nomenclature.

**Definition 1.2** (Maximum Subgraph Density)**.** The *maximum subgraph density* of $G$, $\rho^*_G$, is simply the maximum among all subgraph densities, i.e.,

$$\rho^*_G \overset{\text{def}}{=} \max_{S \subseteq V} \rho_G(S).$$

[1]We describe this connection explicitly in Section 1.2.

Equivalently, the subgraph which realizes this maximum density is called the *densest subgraph*. So, our formal problem statement is as follows:

**The Densest Subgraph Problem**

Given an undirected unweighted graph $G = \langle V, E \rangle$, find a subset of vertices $S^* \subseteq V$ such that

$$S^* = \operatorname*{argmax}_{S \subseteq V} \rho_G(S).$$

## 1.1.2 Linear Programming Duality

Linear programming is a technique to optimize a linear objective given a family of linear constraints. Formally, a linear program (LP) is of the following general form:

$$\begin{aligned} \text{maximize} \quad & \sum c^T x \\ \text{subject to} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned}$$

Every linear program (we call this the *primal* problem) can be converted to a *dual* linear program, which provides an upper bound to the objective value of the primal. Considering the above LP as the primal, the following constitutes its corresponding dual LP.

$$\begin{aligned} \text{minimize} \quad & \sum b^T y \\ \text{subject to} \quad & A^T y \geq c, \\ & y \geq 0. \end{aligned}$$

Here $A \in \mathbb{R}^{n \times m}$ is the constraint matrix. We refer to $x \in \mathbb{R}^m$ as the primal variables and $y \in \mathbb{R}^n$ as the corresponding dual variables.

We note a few important properties of LP duality which will prove useful in later discussion in the thesis. First, we state relationships between any primal solution and any dual solution.

**Fact 1.3** (Weak Duality). *Suppose $x'$ is any feasible primal solution, and $y'$ is any feasible dual solution. Then,*

$$c^T x' \leq b^T y'.$$

**Fact 1.4** (Strong Duality). *Suppose $x^*$ is an optimal primal solution, and $y^*$ is an optimal dual solution. Then,*

$$c^T x^* = b^T y^*.$$

The above remark states that finding the optimal objective value to the dual also gives the optimal to the primal and vice versa. But given an optimal dual solution $y^*$, we want to derive the optimal primal solution $x^*$ too. For this, we can use the complementary slackness theorem. Let $A_{i,:}$ denote the $i$th row and let $A_{:,j}$ denote the $j$th column of $A$.

**Fact 1.5** (Complementary Slackness). *Suppose $x$ and $y$ are feasible primal and dual solutions respectively. Then, $x$ and $y$ are both optimal solutions of their respective LPs if and only if*

- *for all $i \in [n]$,*

$$y_i(b_i - A_{i,:}^T x) = 0,$$

- *for all $j \in [m]$,*

$$x_j(A_{:,j}^T y - c_i) = 0.$$

## 1.2 Linear Programming Formulation of Densest Subgraph

The following is a well-known LP formulation of the densest subgraph problem, introduced by Charikar [26]. Associate each vertex $v$ with a variable $x_v \in \{0, 1\}$, where $x_v = 1$ signifies $v$ being included in $S$. Similarly, for each edge, let $y_e \in \{0, 1\}$ denote whether or not it is in $E(S)$. Relaxing the variables to be real numbers, we get the following LP, which we denote by $\textsc{Primal}(G)$, whose optimal is known to be $\rho^*_G$.

PRIMAL$(G)$

$$\begin{aligned}
\text{maximize} \quad & \sum_{e\in E} y_e \\
\text{subject to} \quad & y_e \le x_u, x_v, \qquad \forall e = uv \in E \\
& \sum_{v\in V} x_v \le 1, \\
& y_e \ge 0, x_v \ge 0, \quad \forall e \in E, \forall v \in V
\end{aligned}$$

The construction above gives some intuition behind this formulation. Charikar [26] showed that the LP exactly models the densest subgraph problem.

**Lemma 1.6** (LP formulation for Densest Subgraph [26])**.** *Given a graph $G = \langle V, E\rangle$, let $\rho^*_G$ denote the maximum subgraph density. Then, the optimum of* PRIMAL$(G)$ *is exactly $\rho^*_G$.*

Let $f_e(u)$ be the dual variable associated with the first $2m$ constraints of the form $y_e \le x_u$ in PRIMAL$(G)$, and let $D$ be associated with the last constraint. We get the following LP, which we denote by DUAL$(G)$.

DUAL$(G)$

$$\begin{aligned}
\text{minimize} \quad & D \\
\text{subject to} \quad & f_e(u) + f_e(v) \ge 1, \qquad \forall e = uv \in E \\
& \sum_{e\ni v} f_e(v) \le D, \qquad \forall v \in V \\
& f_e(u) \ge 0, f_e(v) \ge 0, \quad \forall e = uv \in E
\end{aligned}$$

This LP can be visualized as follows. Each edge $e = uv$ has a load of $1$, which it wants to assign to its end points: $f_e(u)$ and $f_e(v)$ such that the total load on each vertex is at most $D$. The objective is to find the minimum $D$ for which such a load assignment is feasible.

Now, consider the case where we fix the parameter $D$. Then, the LP has only non-negative constraints and hence is an instance of a *mixed packing and covering problem.*

Hence, the densest subgraph problem can be solved by finding the smallest $D$ such that the dual is feasible. This can be done by binary searching over all $O(|V|^3)$ possible values of $D$.

Note also, that this formulation resembles a bipartite graph between edges and vertices. Then, the problem is similar to a bipartite $b$-matching problem [18], where the demands on one side are at most $D$, and the other side are at least $1$.

From strong duality, we know that the optimal objective values of both linear programs are equal, i.e., exactly $\rho_G^*$. Let $\rho_G$ be the objective of any feasible solution to $\textsc{Primal}(G)$. Similarly, let $\hat{\rho}_G$ be the objective of any feasible solution to $\textsc{Dual}(G)$. Then, by optimality of $\rho_G^*$ and weak duality,

$$\rho_G \leq \rho_G^* \leq \hat{\rho}_G.$$

## 1.3 Background

Goldberg [42] gave the first polynomial-time algorithm to solve the densest subgraph problem by reducing it to $O(\log n)$ instances of maximum flow. This was subsequently improved to use only $O(1)$ instances, using parametric max-flow [38]. Charikar [26] gave an exact linear programming formulation of the problem, while at the same time giving a simple greedy algorithm which gives a $1/2$-approximate densest subgraph (first studied in [10]). Despite the approximation factor, this algorithm is popular in practice [28] due to its simplicity, its efficacy on real-world graphs, and due to the fact that it runs in linear time and space.

Obtaining fast algorithms for approximation factors better than $1/2$, however, has proved to be a harder task. One approach towards this is to sparsify the graph in a way that maintains subgraph densities [71, 73] within a factor of $1-\epsilon$, and run the exact algorithm on the resulting sparsifier. However, this algorithm still incurs a term of $n^{1.5}$ in the running time, causing it to be super-linear for sparse graphs.

A second approach is via numerical methods to solve the dual LP $\textsc{Dual}(G)$. Solving $\textsc{Dual}(G)$ can be shown to reduce to solving $O(\log |V|)$ mixed packing covering LPs approximately, as we showed in Section 1.2. Bahmani *et al.* [11] gave a $O(m \log n \cdot \epsilon^{-2})$ algorithm by bounding the width of the dual LP for this problem, and using the multiplicative weights update framework [78, 9] to find an $(1-\epsilon)$-approximate solution.

Further background details are left to future chapters, where we briefly go over results related to the contributions in each particular chapter.

## 1.4 Alternate Formulations and Related Work

The densest subgraph problem has also been studied in weighted graphs, as well as directed graphs. When the edge weights are non-negative, both the maximum flow algorithm and Charikar's greedy algorithm maintain their theoretical guarantees. In the presence of negative weights, the problem in general becomes NP-hard [97]. For directed graphs Charikar [26] provided a linear programming approach which requires the computation of $n^2$ linear programs and a $1/2$-approximation algorithm which runs in $O(n^3 + n^2 m)$ time. Khuller and Saha have provided more efficient implementations of the exact and approximation algorithms for the undirected and directed versions of the problem [60]. Furthermore, Tsourakakis et al. recently extended the problem to $k$-clique, and $(p, q)$-biclique densest subgraph problems [94, 73]. These extensions can be used for finding large near-cliques in general graphs and bipartite graphs respectively. Tatti and Gionis [91] introduced a novel graph decomposition known as *locally-dense*, that imposes certain insightful constraints on the k-core decomposition. Further, efficient algorithms to find locally-dense subgraphs were developed by Danisch et al. [29].

Notice that in our formulation of the problem, there are no restrictions on the size of the output. When restrictions on the size of $S$ are imposed the problem becomes NP-hard. The densest-$k$-subgraph problem asks for find the subgraph $S$ with maximum degree density among all possible sets $S$ such that $|S| = k$. The state-of-the art algorithm is due to Bhaskara et al. [16], and provides a $O(n^{1/4+\epsilon})$ approximation in $O(n^{1/\epsilon})$ time. A long standing question is closing the gap between this upper bound and the lower bound. Other versions where $|S| \geq k, |S| \leq k$ have also been considered in the literature see [7].

## 1.5 Organization of Contributions

We can organize our contributions in this thesis into the following three parts.

- In Chapter 2, we develop a new width-dependent algorithm for mixed packing and covering problems, thereby giving the fastest algorithm for the densest subgraph problem on graphs with bounded degree [24].
- In Chapter 3, we adapt Charikar's greedy $1/2$-approximation algorithm to give a fast and simple iterative algorithm for the densest subgraph problem. This is the first algorithm which is able to inherit the speed of Charikar's greedy algorithm, but is still able to arrive at exact solutions to the problem on real-world graphs [23].
- In Chapter 4, we give the first fully dynamic $(1 - \varepsilon)$ approximation algorithm for the densest subgraph problem [83].

# CHAPTER 2
# A FASTER WIDTH-DEPENDENT ALGORITHM FOR MIXED PACKING AND COVERING LPS

In Section 1.2, we showed that the dual of the densest subgraph problem, i.e., the load balancing problem, can be reduced to solving $O(\log n)$ instances of a *mixed packing and covering linear program.* In this chapter, we study algorithms to solve this broad class of linear programs.

## 2.1 Mixed Packing and Covering LPs

Mixed packing and covering linear programs (LPs) are a natural class of LPs where coefficients, variables, and constraints are non-negative. They model a wide range of important problems in combinatorial optimization and operations research. In general, they model any problem which contains a limited set of available resources (packing constraints) and a set of demands to fulfill (covering constraints).

Two special cases of the problem have been widely studied in literature: pure *packing*, formulated as

$$\max_{x \in \mathbb{R}_+^n} \{b^T x \mid Px \leq p\},$$

and pure *covering*, formulated as

$$\min_{x \in \mathbb{R}_+^n} \{b^T x \mid Cx \geq c\},$$

where $P, p, C, c, b$ are all non-negative. These are known to model fundamental problems such as maximum bipartite graph matching, minimum set cover, etc. [68]. Algorithms to solve packing and covering LPs have also been applied to great effect in designing flow control systems [13], scheduling problems [78], zero-sum matrix games [74] and in mechanism design [102]. In this chapter, we study the mixed packing and covering (MPC) problem, formulated as checking the feasibility of the set:

$$\{x \in \mathbb{R}_+^n \mid Px \leq p, Cx \geq c\},$$

where $P, C, p, c$ are non-negative. We say that $x$ is an $\varepsilon$-approximate solution to MPC if it belongs to the relaxed set

$$\{x \in \mathbb{R}_+^n \mid Px \leq (1+\varepsilon)p, Cx \geq (1-\varepsilon)c\}.$$

MPC is a generalization of pure packing and pure covering, hence it is applicable to a wider range of problems such as multi-commodity flow on graphs [99, 85], non-negative linear systems and X-ray tomography [99].

General LP solving techniques such as the interior point method can approximate solutions to MPC in as few as $O(\log(1/\varepsilon))$ iterations - however, they incur a large per-iteration cost. In contrast, iterative approximation algorithms based on first-order optimization methods require poly$(1/\varepsilon)$ iterations, but the iterations are fast and in most cases are conducive to efficient parallelization. This property is of utmost importance in the context of ever-growing datasets and the availability of powerful parallel computers, resulting in much faster algorithms in relatively low-precision regimes.

## 2.2 Algorithms for the Mixed Packing and Covering problem

### 2.2.1 Previous work

In literature, algorithms for the MPC problem can be grouped into two broad categories: *width-dependent* and *width-independent*. Here, *width* is an intrinsic property of a linear program which typically depends on the dimensions and the largest entry of the constraint matrix, and is an indication of the range of values any constraint can take. In the context of the MPC problem, we define $w_P$ and $w_C$ as the maximum number of non-zeros in any constraint in $P$ and $C$ respectively, and subsequently define the width of the LP as $w \stackrel{\mathrm{def}}{=} \max(w_P, w_C)$. Additionally, $N$ denotes the total number of non-zeros in the constraint matrices $P$ and $C$.

One of the first approaches used to solve LPs was Langrangian-relaxation: replacing hard constraints with loss functions which enforce the same constraints indirectly. Using this approach, Plotkin, Schmoys and Tardos [78], and Grigoriadis and Khachiyan [45] obtained width-dependent polynomial-time approximation algorithms for MPC. Luby and Nisan [68] gave the first width-dependent parallelizable algorithm for pure packing and pure covering, which ran in $\widetilde{O}(\varepsilon^{-4})$ parallel time, and $\widetilde{O}(N\varepsilon^{-4})$ total work.[1] Here, *parallel time* (sometimes termed as *depth*) refers to the longest chain of dependent operations, and *work* refers to the total number of operations in the algorithm.

Young [99] extended this technique to give the first width-independent parallel algorithm for MPC in $\widetilde{O}(\varepsilon^{-4})$ parallel time, and $\widetilde{O}(mw_{\text{COL}}\varepsilon^{-2})$ total work[2]. Young [100] later improved his algorithm to run using total work $O(N\varepsilon^{-2})$. Mahoney *et al.* [70] later gave an algorithm with a faster parallel run-time of $\widetilde{O}(\varepsilon^{-3})$.

The other most prominent approach in literature towards solving an LP is by converting it into a smooth function [74], and then applying general first-order optimization techniques [74, 76]. Although the dependence on $\varepsilon$ from using first-order techniques is much improved, it usually comes at the cost of sub-optimal dependence on the input size and width. For the MPC problem, Nesterov's accelerated method [76], as well as Bienstock and Iyengar's adaptation [22] of Nesterov's smoothing [74], give rise to algorithms with runtime linearly depending on $\varepsilon^{-1}$, but with far from optimal dependence on input size and width. For pure packing and pure covering problems, however, Allen-Zhu and Orrechia [6] were the first to incorporate Nesterov-like acceleration while still being able to obtain

[1] $\widetilde{O}$ hides factors polylogarithmic in the size of the input.

[2] Here, $m$ is the total number of constraints, and $w_{\text{COL}}$ (column-width) is the maximum number of constraints that any variable appears in.

Table 2.1: Comparison of runtimes of $\varepsilon$-approximation algorithms for the mixed packing covering problem. Recall that $m$ denotes the total number of constraints and $d_{\text{COL}}$ denotes the column-width.

| | **Parallel Runtime** | **Total Work** | **Comments** |
|---|---|---|---|
| Young [99] | $\widetilde{O}(\varepsilon^{-4})$ | $\widetilde{O}(md\varepsilon^{-2})$ | $d$ is column-width |
| Bienstock and Iyengar [22] | | $\widetilde{O}(n^{2.5}w_P^{1.5}w\varepsilon^{-1})$ | width-dependent |
| Nesterov [76] | $\widetilde{O}(w\sqrt{n}\varepsilon^{-1})$ | $\widetilde{O}(w \cdot N\sqrt{n}\varepsilon^{-1})$ | width-dependent |
| Young [100] | $\widetilde{O}(\varepsilon^{-4})$ | $\widetilde{O}(N\varepsilon^{-2})$ | |
| Mahoney *et al.* [70] | $\widetilde{O}(\varepsilon^{-3})$ | $\widetilde{O}(N\varepsilon^{-3})$ | |
| Our result | $\widetilde{O}(w\varepsilon^{-1})$ | $\widetilde{O}(wN\varepsilon^{-1})$ | width-dependent |

near-linear width-independent runtimes, giving a $\widetilde{O}(N\varepsilon^{-1})$ time algorithm for the packing problem. For the covering problem, they gave a $\widetilde{O}(N\varepsilon^{-1.5})$ time algorithm, which was then improved to $\widetilde{O}(N\varepsilon^{-1})$ in [98]. Importantly, however, the above algorithms do not generalize to MPC.

### 2.2.2 Our contribution

We give the best parallel width-dependent algorithm for MPC, while only incurring a linear dependence on $\varepsilon^{-1}$ in the parallel runtime and total work. Additionally, the total work has near-linear dependence on the input-size. Formally, we state the main result of this chapter as the following theorem.

**Theorem 2.1.** *There exists a parallel $\varepsilon$-approximation algorithm for the mixed packing covering problem, which runs in $\widetilde{O}(w\varepsilon^{-1})$ parallel time, while performing $\widetilde{O}(wN\varepsilon^{-1})$ total work, where $N$ is the total number of non-zeros in the constraint matrices, and $w$ is the width of the given LP.*

Table 2.1 compares the running time of our algorithm to previous works solving this problem.

Sacrificing width independence for faster convergence with respect to precision proves to be a valuable trade-off for several combinatorial optimization problems which naturally have low width. Prominent examples of such problems which are not pure packing or covering problems include *multicommodity flow* and *densest subgraph*, where the width is bounded by the degree of a vertex. In a large number of real-world graphs, the maximum vertex degree is usually small, hence our algorithm proves to be much faster when we want high-precision solutions.

## 2.3 Notation and Definitions

### 2.3.1 Notation

For any integer $q$, we represent using $\|\cdot\|_q$ the $q$-norm of any vector. We represent the infinity-norm as $\|\cdot\|_\infty$. We denote the infinity-norm ball (sometimes called the $\ell_\infty$ ball) as the set

$$\mathcal{B}_\infty^n(r) \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n : \|x\|_\infty \leq r\}.$$

The nonnegative part of this ball is denoted as

$$\mathcal{B}_{+,\infty}^n(r) = \{x \in \mathbb{R}^n : x \geq \mathbf{0}_n, \|x\|_\infty \leq r\}.$$

For radius $r = 1$, we drop the radius specification and use the short notation $\mathcal{B}_\infty^n$ and $\mathcal{B}_{+,\infty}^n$. We denote the extended simplex of dimension $k$ as

$$\Delta_k^+ \stackrel{\text{def}}{=} \{x \in \mathbb{R}^k : \sum_{i=1}^k x_i \leq 1\}.$$

For any $y \geq \mathbf{0}_k$, $\mathrm{proj}_{\Delta_k^+}(y) = y/\|y\|_1$ if $\|y\|_1 \geq 1$.

The function $\exp$ is applied to a vector element wise. The division of two vectors of the same dimension is also performed element wise.

For any matrix $A$, we use $\mathrm{nnz}(A)$ to denote the number of nonzero entries in it. We use $A_{i,:}$ and $A_{:,j}$ to refer to the $i$th row and $j$th column of $A$ respectively. We use notation $A_{ij}$

(or $A_{i,j}$ alternatively) to denote an element in the $i$-th row and $j$-th column of matrix $A$. $\|A\|_\infty$ denotes the operator norm $\|A\|_{\infty\to\infty} \overset{\text{def}}{=} \sup_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty}$.

### 2.3.2 Definitions

We first define the notions of convexity and smoothness that we will use in our analysis. Let $X \subseteq \mathbb{R}^n$ be a convex set.

**Definition 2.2** (Convexity)**.** A function $f : X \mapsto \mathbb{R}$ is said to be convex if for any $x, y \in X$ and any $\lambda \in [0, 1]$,

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(x).$$

**Definition 2.3** (Strong convexity)**.** A function $f : X \mapsto \mathbb{R}$ is said to be $\alpha$-strongly convex if for any $x, y \in X$,

$$f(x) - f(y) \leq \nabla f(x)^T(x-y) + \frac{\alpha}{2}||x-y||^2.$$

**Definition 2.4** (Smoothness)**.** A continuously differentiable function $f : X \mapsto \mathbb{R}$ is said to be $\beta$-smooth if for any $x, y \in X$,

$$||\nabla f(x) - \nabla f(y)|| \leq \beta ||x-y||.$$

We formally define an $\varepsilon$-approximate solution to the mixed packing-covering (MPC) problem as follows.

**Definition 2.5** ($\varepsilon$-approximate MPC solution)**.** We say that $x$ is an $\varepsilon$-approximate solution to the mixed packing-covering problem if $x$ satisfies $x \in \mathcal{B}^n_{+,\infty}$, $Px \leq (1+\varepsilon)\mathbf{1}_p$ and $Cx \geq (1-\varepsilon)\mathbf{1}_c$.

Here, $\mathbf{1}_k$ denotes a vectors of 1's of dimension $k$ for any non-negative integer $k$.

**Definition 2.6** (Saddle Point Problem)**.** The saddle point problem on two sets $x \in X$ and $y \in Y$ can be defined as follows:

$$\min_{x \in X} \max_{y \in Y} \mathcal{L}(x, y), \tag{2.1}$$

where $\mathcal{L}(x, y)$ is some bilinear form between $x$ and $y$.

**Definition 2.7** (Primal-dual Gap)**.** For a saddle point problem defined by the bilinear form $\mathcal{L}$ on sets $X$ and $Y$, we define the *primal-dual gap function* as $\sup_{(\overline{x},\overline{y}) \in X \times Y} \mathcal{L}(x, \overline{y}) - \mathcal{L}(\overline{x}, y)$.

This gap function can be used as measure of accuracy of the above saddle point solution.

**Definition 2.8** ($\varepsilon$-approximate solution to the Saddle Point Problem)**.** We say that $(x, y) \in X \times Y$ is an $\varepsilon$-optimal solution for (2.1) if

$$\sup_{(\overline{x},\overline{y}) \in X \times Y} \mathcal{L}(x, \overline{y}) - \mathcal{L}(\overline{x}, y) \leq \varepsilon.$$

## 2.4 Technical overview

The mixed packing-covering (MPC) problem is formally defined as follows.

**Mixed Packing and Covering (MPC) Problem**

Given two nonnegative matrices $P \in \mathbb{R}^{p \times n},\ C \in \mathbb{R}^{c \times n}$,

- find an $x \in \mathbb{R}^n, x \geq \mathbf{0}, \|x\|_\infty \leq 1$ such that $Px \leq \mathbf{1}_p$ and $Cx \geq \mathbf{1}_c$ if it exists,
- otherwise report infeasibility.

Note that the vector of 1's on the right hand side of the packing and covering constraints can be obtained by simply scaling each constraint appropriately. We also assume that each entry in the matrices $P$ and $C$ is at most $1$. This assumption, and subsequently the $\ell_\infty$

constraints on $x$ also cause no loss of generality[3].

We reformulate MPC as a saddle point problem, as defined in Section 2.3:

$$\lambda^* \stackrel{\text{def}}{=} \min_{x \in \mathcal{B}^n_{+,\infty}} \max_{y \in \Delta^+_c,\ z \in \Delta^+_p} L(x, y, z), \tag{2.2}$$

where

$$L(x, y, z) \stackrel{\text{def}}{=} [y^T \ z^T] \begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

The relation between the two formulations is shown in Section 2.5. For the rest of this chapter, we focus on the saddle point formulation (2.2).

Let $\eta(x) \stackrel{\text{def}}{=} \max_{y \in \Delta^+_c, z \in \Delta^+_p} L(x, y, z)$ be a piecewise linear convex function. Assuming oracle access to this "inner" maximization problem, the "outer" problem of minimizing $\eta(x)$ can be performed using first order methods like mirror descent, which are suitable when the underlying problem space is the unit $\ell_\infty$ ball. One drawback of this class of methods is that their rate of convergence, which is standard for non-accelerated first order methods on non-differentiable objectives, is $O(\varepsilon^{-2})$ to obtain an $\varepsilon$-approximate minimizer $x$ of $\eta$ - an $x$ which satisfies $\eta(x) \leq \eta^* + \varepsilon$, where $\eta^*$ is the optimal value. This means that the algorithm needs to access the inner maximization oracle $O(\varepsilon^{-2})$ times, which can become prohibitively large in the high precision regime.

Note that even though $\eta$ is a piecewise linear non-differentiable function, it is not a black-box function, but a maximization over a linear function in $x$. This structure can be exploited using Nesterov's smoothing technique [74]. In particular, $\eta(x)$ can be approximated by choosing a strongly convex[3] function $\phi : \Delta^+_p \times \Delta^+_c \to \mathbb{R}$ and considering

$$\widetilde{\eta}(x) = \max_{y \in \Delta^+_c, z \in \Delta^+_p} L(x, y, z) - \phi(y, z).$$

[3]This transformation can be achieved by adapting techniques from [98] while increasing dimension of the problem up to a logarithmic factor. Details of this fact are in Appendix A.

This strongly convex regularization yields that $\widetilde{\eta}$ is a smooth convex function. If $L$ is the constant of smoothness of $\widetilde{\eta}$ then application of any of the accelerated gradient methods in literature will converge in $O(\sqrt{L\varepsilon^{-1}})$ iterations. Moreover, it can also be shown that in order to construct a smooth $\varepsilon$-approximation $\widetilde{\eta}$ of $\eta$, the Lipschitz smoothness constant $L$ can be chosen to be of the order $O(\varepsilon^{-1})$, which in turn implies an overall convergence rate of $O(\varepsilon^{-1})$. In particular, Nesterov's smoothing achieves an oracle complexity of $O((\|P\|_\infty + \|C\|_\infty)D_x \max\{D_y, D_z\}\varepsilon^{-1})$, where where $D_x$, $D_y$ and $D_z$ respectively denote the sizes of the ranges of strongly convex regularizers for each of the three sets $X$, $Y$ and $Z$. $D_y$ and $D_z$ can be made of the order of $\log p$ and $\log c$, respectively. However, $D_x$ can be problematic since $x$ belongs to an $\ell_\infty$ ball. We expand on this in Section 2.4.1.

An alternative approach to the MPC problem is via Nesterov's dual extrapolation algorithm[75]. Although the algorithm's time complexity is the same as that of the algorithm described in the previous paragraph, is a different algorithm in that it directly addresses the saddle point formulation (2.2) rather than viewing the problem as optimizing a non-smooth function $\eta$. The final convergence for the dual extrapolation algorithm is given in terms of the *primal-dual gap* function of the saddle point problem (2.2). Moreover, as opposed to smoothing techniques which only regularize the dual, this algorithm regularizes both primal and dual parts *(joint regularization)*, hence is a different scheme altogether.

Note that for both schemes mentioned above, the maximization oracle itself has an analytical expression which involves matrix-vector multiplication. Hence each call to the oracle incurs a sequential run-time of $\mathrm{nnz}(P)+\mathrm{nnz}(C)$. Then, the overall complexity for both schemes is of order of $O((\mathrm{nnz}(P) + \mathrm{nnz}(C))(\|P\|_\infty + \|C\|_\infty)D_x \max\{D_y, D_z\}\varepsilon^{-1})$.

### 2.4.1 The $\ell_\infty$ barrier

Note that the first method, i.e., Nesterov's smoothing technique has known lower bounds due to Guzman *et al.* [48] (see Corollary 1 in their paper). According to their result, the framework of Nesterov's smoothing has a known limitation since it only regularizes the dual variables. As opposed to this, Nesterov's dual extrapolation regularizes both primal and dual variables, and has potential to skip the lower bounds mentioned in [48]. However, the complexity result of this method involves a $D_x$ term, which denotes the range of a convex function over the domain of $x$. Through the following lemma, we derive a lower bound for this range in case of $\ell_\infty$ balls.

**Lemma 2.9.** *Any strongly convex function has a range of at least $\Omega(\sqrt{n})$ on any $\ell_\infty$ ball.*

*Proof.* Consider an arbitrary strongly convex function $d$. Assume WLOG that $d(0) = 0$ (otherwise, we can shift it accordingly). We will show that $\max_{x \in \mathcal{B}_\infty^n(r)} d(x) \geq \frac{nr^2}{2}$ by induction on $n$ for set $\mathcal{B}_\infty^n(r)$. This suffices because $\mathcal{B}_{+,\infty}^n(1)$ is isomorphic to $\mathcal{B}_\infty^n(\frac{1}{2})$. The claim holds for $n = 1$ by the definition of strong convexity. Now, suppose it is true for $n-1$. Then there exists $\overline{x} \in \mathcal{B}_\infty^{n-1}(r)$ such that $d(\overline{x}) \geq \frac{(n-1)r^2}{2}$. Moving $r$ units in the last coordinate from $\overline{x}$ in the direction of nonnegative slope, suppose we reach $\widehat{x} \in \mathcal{B}_\infty^n(r)$. Then, due to strong convexity of $d$, we have $d(\widehat{x}) \geq d(\overline{x}) + \frac{1}{2}\|\widehat{x} - \overline{x}\|_\infty^2 \geq \frac{(n-1)r^2}{2} + \frac{r^2}{2} = \frac{nr^2}{2}$. $\square$

Since $D_x = \Omega(\sqrt{n})$ for each member function of this wide class, there is no hope of eliminating this $\sqrt{n}$ factor using techniques involving explicit use of strong convexity. So, the goal now is to find a joint regularization function with a small range over $\ell_\infty$ balls, but still act as good enough regularizers to enable accelerated convergence of the descent algorithm.

### 2.4.2 Area Convexity

In pursuit of breaking this $\ell_\infty$ barrier, we draw inspiration from the notion of *area convexity* introduced by Sherman [85]. Area convexity is a weaker notion than strong convexity, however, it is still strong enough to ensure that accelerated first order methods still go through when using area convex regularizers. Since this is a weaker notion than strong convexity, we can construct area convex functions which have range of $O(n^{o(1)})$ on the $\ell_\infty$

ball.

First, we define area convexity, and then go on to mention its relevance to the saddle point problem (2.2).

Area convexity is a notion defined in context of a matrix $A \in \mathbb{R}^{a \times b}$ and a convex set $K \subseteq \mathbb{R}^{a+b}$. Let $M_A \overset{\text{def}}{=} \begin{bmatrix} \mathbf{0}_{b\times b} & -A^T \\ A & \mathbf{0}_{a\times a} \end{bmatrix}$.

**Definition 2.10** (Area Convexity [85])**.** A function $\phi$ is *area convex* with respect to a matrix $A$ on a convex set $K$ if and only if for any $t, u, v \in K$, $\phi$ satisfies

$$\phi\left(\frac{t+u+v}{3}\right) \leq \frac{1}{3}\left(\phi(t)+\phi(u)+\phi(v)\right) - \frac{1}{3\sqrt{3}}(v-u)^T M_A (u-t).$$

To understand the definition above, let us first look at the notion of strong convexity. $\phi$ is said to be strongly convex if for any two points $t, u$, $\frac{1}{2}(\phi(t)+\phi(u))$ exceeds $\phi(\frac{1}{2}(t+u))$ by an amount proportional to $\|t-u\|_2^2$. Definition 2.10 generalizes this notion in context of matrix $A$ for any three points $x, y, z$. $\phi$ is area-convex on set $K$ if for any three points $t, u, v \in K$, we have $\frac{1}{3}(\phi(t)+\phi(u)+\phi(v))$ exceeds $\phi(\frac{1}{3}(t+u+v))$ by an amount proportional to the area of the triangle defined by the convex hull of $t, u$ and $v$.

Consider the case that points $t, u, v$ are collinear. For this case, the area term (i.e., the term involving $M_A$) in Definition 2.10 is 0 since matrix $M_A$ is antisymmetric. In this sense, area convexity is even weaker than strict convexity. Moreover, the notion of area is parameterized by matrix $A$.

To see a specific example of this notion of area, consider $A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ and $t, u, v \in \mathbb{R}^2$. Then, for all possible permutations of $t, u, v$, the area term takes a value equal to $\pm(t_1(u_2-v_2)+u_1(v_2-t_2)+v_1(t_2-u_2))$. Since the condition holds irrespective of the permutation so we must have that

$$\phi(\tfrac{t+u+v}{3}) \leq \tfrac{1}{3}\big(\phi(t)+\phi(u)+\phi(v)\big) - \tfrac{1}{3\sqrt{3}}|t_1(u_2-v_2)+u_1(v_2-t_2)+v_1(t_2-u_2)|.$$

But note that area of triangle formed by points $t, u, v$ is equal to $\frac{1}{2}|t_1(u_2-v_2)+u_1(v_2-t_2)+v_1(t_2-u_2)|$. Hence the area term is just a high dimensional matrix based generalization of the area of a triangle.

Coming back to the saddle point problem (2.2), we need to pick a suitable area convex function $\phi$ on the set $\mathcal{B}^n_{+,\infty} \times \Delta^+_p \times \Delta^+_c$. Since $\phi$ is defined on the joint space, it has the property of joint regularization vis a vis (2.2). However, we need an additional parameter: a suitable matrix $M_A$. The choice of this matrix is related to the bilinear form of the *primal-dual gap function* of (2.2). We delve into the technical details of this in Section 2.5, however, we state that the matrix is composed of $P, C$ and some additional constants.

The algorithm we state exactly follows Nesterov's dual extrapolation method described earlier. One notable difference is that in [75], they consider joint regularization by a strongly convex function which does not depend on the problem matrices $P$ and $C$, but only on the constraint set $\mathcal{B}^n_{+,\infty} \times \Delta^+_p \times \Delta^+_c$. Our area convex regularizer, on the other hand, is tailor made for the particular problem matrices $P$ and $C$, as well as the constraint set.

## 2.5 Area Convexity for Mixed Packing Covering LPs

In this section, we present our technical results and algorithm for the MPC problem, with the end goal of proving Theorem 2.1. First, we relate an $(1+\varepsilon)$-approximate solution to the saddle point problem to an $\varepsilon$-approximate solution to MPC. Next, we present some theoretical background towards the goal of choosing and analyzing an appropriate area-convex regularizer in the context of the saddle point formulation, where the key requirement of the area convex function is to obtain a provable and efficient convergence result. Finally, we explicitly show an area convex function which is generated using a simple "gadget" function. We show that this area convex function satisfies all key requirements and hence achieves the desired accelerated rate of convergence. This section closely follows [85], in which the author chooses an area convex function specific to the undirected multicommodity flow problem.

### 2.5.1 Saddle Point Formulation for MPC

Consider the saddle point formulation in (2.2) for MPC.

Given a feasible primal-dual feasible solution pair $(x, y, z)$ and $(\overline{x}, \overline{y}, \overline{z})$ for (2.2), we denote $w = (x, u, y, z)$ and $\overline{w} = (\overline{x}, \overline{u}, \overline{y}, \overline{z})$ where $u, \overline{u} \in \mathbb{R}$. Then, we define a function $Q : \mathbb{R}^{n+1+p+c} \times \mathbb{R}^{n+1+p+c} \to \mathbb{R}$ as

$$Q(w, \overline{w}) \overset{\text{def}}{=} [\overline{y}^T \ \overline{z}^T] \begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} - [y^T \ z^T] \begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix} \begin{bmatrix} \overline{x} \\ \overline{u} \end{bmatrix}.$$

Note that if $u = \overline{u} = 1$, then

$$\sup_{\overline{w} \in \mathcal{W}} Q(w, \overline{w}) = \sup_{\overline{x} \in \mathcal{B}^n_{+,\infty}, \overline{y} \in \Delta^+_p, \overline{z} \in \Delta^+_c} L(x, \overline{y}, \overline{z}) - L(\overline{x}, y, z)$$

is precisely the primal-dual gap function defined in Section 2.3. Notice that if $(x^*, y^*, z^*)$ is a saddle point of (2.2), then we have

$$L(x^*, y, z) \leq L(x^*, y^*, z^*) \leq L(x, y^*, z^*)$$

for all $x \in \mathcal{B}^n_{+,\infty}, y \in \Delta^+_p, z \in \Delta^+_c$. From above equation, it is clear that $Q(w, w^*) \geq 0$ for all $w \in \mathcal{W}$ where $\mathcal{W} \overset{\text{def}}{=} \mathcal{B}^n_{+,\infty} \times \{1\} \times \Delta^+_p \times \Delta^+_c$ and $w^* = (x^*, 1, y^*, z^*) \in \mathcal{W}$. Moreover, $Q(w^*, w^*) = 0$. This motivates the following accuracy measure of the candidate approximate solution $w$.

**Definition 2.11.** We say that $w \in \mathcal{W}$ is an $\varepsilon$-optimal solution of (2.2) iff

$$\sup_{\overline{w} \in \mathcal{W}} Q(w, \overline{w}) \leq \varepsilon.$$

**Remark 2.12.** *Recall the definition of $M_A$ for a matrix $A$ in Section 2.3. We can rewrite $Q(w, \overline{w}) = \overline{w}^T J w$ where $J = M_H$ and*

$$H = \begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix} \quad \Rightarrow \quad J := \begin{bmatrix} \mathbf{0}_{n\times n} & \mathbf{0}_{n\times 1} & -P^T & C^T \\ \mathbf{0}_{1\times n} & 0 & \mathbf{1}_p^T & -\mathbf{1}_c^T \\ P & -\mathbf{1}_p & \mathbf{0}_{p\times p} & \mathbf{0}_{p\times c} \\ -C & \mathbf{1}_c & \mathbf{0}_{c\times p} & \mathbf{0}_{c\times c} \end{bmatrix}.$$

*Thus, the gap function in Definition 2.11 can be written in the bilinear form $\sup_{\overline{w}\in\mathcal{W}} \overline{w}^T J w$.*

Lemma 2.13 relates the $\varepsilon$-optimal solution of (2.2) to the $\varepsilon$-approximate solution to MPC.

**Lemma 2.13.** *Let $(x, y, z)$ satisfy $\sup_{(\overline{x},\overline{y},\overline{z})\in\mathcal{B}^n_{+,\infty}\times\Delta_p^+\times\Delta_c^+} L(x, \overline{y}, \overline{z}) - L(\overline{x}, y, z) \le \varepsilon$. Then either*

*1. $x$ is an $\varepsilon$-approximate solution of MPC, or*

*2. $y, z$ satisfy $y^T(P\overline{x} - \mathbf{1}_p) + z^T(-C\overline{x} + \mathbf{1}_c) > 0$ for all $\overline{x} \in \mathcal{B}^n_{+,\infty}$.*

*Proof.* Suppose we are given $(x, y, z)$ such that

$$\sup_{(\overline{x},\overline{y},\overline{z})\in\mathcal{B}^n_{+,\infty}\times\Delta_p^+\times\Delta_c^+} L(x, \overline{y}, \overline{z}) - L(\overline{x}, y, z) \le \varepsilon.$$

If there exists $\widetilde{x}$ which is feasible for MPC, then choosing $\overline{x} = \widetilde{x}$ gives $L(\widetilde{x}, y, z) \le 0$. Hence we have that

$$\sup_{(\overline{y},\overline{z})\in\Delta_p^+\times\Delta_c^+} L(x, \overline{y}, \overline{z}) \le \varepsilon.$$

By the optimality over extended simplices $\Delta_p^+, \Delta_c^+$, we get

$$\|[Px - \mathbf{1}_p]_+\|_\infty + \|[-Cx + \mathbf{1}_c]_+\|_\infty \le \varepsilon.$$

So, if there exists a feasible solution for MPC then $x$ is $\varepsilon$-approximate solution of MPC.

On the other hand, suppose $x$ is not an $\varepsilon$-approximate solution. Then

$$\max\{\|[Px - \mathbf{1}_p]_+\|_\infty, \|[-Cx + \mathbf{1}_c]_+\|_\infty\} > \varepsilon$$

$$\Rightarrow \sup_{(\overline{y},\overline{z}) \in \Delta_p^+ \times \Delta_c^+} L(x, \overline{y}, \overline{z}) = \|[Px - \mathbf{1}_p]_+\|_\infty + \|[-Cx + \mathbf{1}_c]_+\|_\infty > \varepsilon$$

Let $(\widehat{y}, \widehat{z}) \in \Delta_p^+ \times \Delta_c^+$ such that $L(x, \widehat{y}, \widehat{z}) > \varepsilon$ then we have

$$\sup_{\overline{x} \in \mathcal{B}^n_{+,\infty}} L(x, \widehat{y}, \widehat{z}) - L(\overline{x}, y, z) \leq \varepsilon$$

$$\Rightarrow L(x, \widehat{y}, \widehat{z}) - \inf_{\overline{x} \in \mathcal{B}^n_{+,\infty}} L(\overline{x}, y, z) \leq \varepsilon$$

$$\Rightarrow \inf_{\overline{x} \in \mathcal{B}^n_{+,\infty}} L(\overline{x}, y, z) > 0$$

Hence, if $x$ is not $\varepsilon$-approximate solution of MPC then $(y, z)$ satisfy $y^T(P\overline{x} - \mathbf{1}_p) + z^T(-C\overline{x} + \mathbf{1}_c) > 0$ for all $\overline{x} \in \mathcal{B}^n_{+,\infty}(1)$ implying that MPC is infeasible. $\square$

This lemma states that in order to find an $\varepsilon$-approximate solution of MPC, it suffices to find an $\varepsilon$-optimal solution to (2.2). Henceforth, we will focus on $\varepsilon$-optimality of the saddle point formulation (2.2).

### 2.5.2 Area Convexity with Saddle Point Framework

Here we state some useful lemmas which help in determining whether a differentiable function is area convex. We start with the following remark which follows from the definition of area convexity (Definition 2.10).

**Remark 2.14.** *If $\phi$ is area convex with respect to $A$ on a convex set $K$, and $\overline{K} \subseteq K$ is a convex set, then $\phi$ is area convex with respect to $A$ on $\overline{K}$.*

For a symmetric matrix $A$ and an antisymmetric matrix $B$, we define an operator $\succeq_{\text{i}}$ as

$$A \succeq_{\text{i}} B \Leftrightarrow \begin{bmatrix} A & -B \\ B & A \end{bmatrix} \text{ is positive semi-definite.}$$

In order to handle the operator $\succeq_{\text{i}}$, we state some basic but important properties of this operator, which will come in handy in later proofs.

**Lemma 2.15.** *For symmetric matrices $A$ and $C$ and antisymmetric matrices $B$ and $D$,*

1. *If $A \succeq_{\text{i}} B$ then $A \succeq_{\text{i}} (-B)$.*
2. *If $A \succeq_{\text{i}} B$ and $\lambda \geq 0$ then $\lambda A \succeq_{\text{i}} \lambda B$.*
3. *If $A \succeq_{\text{i}} B$ and $C \succeq_{\text{i}} D$ then $A + C \succeq_{\text{i}} (B + D)$.*

*Proof.* 1.

$$\begin{aligned} A \succeq_{\text{i}} B \Leftrightarrow & \begin{bmatrix} A & -B \\ B & A \end{bmatrix} \succeq 0 \\ & \Leftrightarrow x^T Ax + y^T Ay + y^T Bx - x^T By \geq 0, \quad \forall\, x, y \\ & \Leftrightarrow x^T Ax + y^T Ay - y^T Bx + x^T By \geq 0, \quad \forall\, x, y \\ & \Leftrightarrow \begin{bmatrix} A & B \\ -B & A \end{bmatrix} \succeq 0 \Leftrightarrow A \succeq_{\text{i}} (-B) \end{aligned}$$

Here, the third equivalence follows after replacing $y$ by $-y$.

2.

$$A \succeq_{\text{i}} B \Leftrightarrow \begin{bmatrix} A & -B \\ B & A \end{bmatrix} \succeq 0 \Rightarrow \begin{bmatrix} \lambda A & -\lambda B \\ \lambda B & \lambda A \end{bmatrix} \succeq 0 \Leftrightarrow \lambda A \succeq \lambda B$$

3. $A \succeq_{\mathrm{i}} B$ implies $\begin{bmatrix} A & -B \\ B & A \end{bmatrix} \succeq 0$. Similarly $C \succeq_{\mathrm{i}} D$ implies $\begin{bmatrix} C & -D \\ D & C \end{bmatrix} \succeq 0$. Hence

$$\begin{bmatrix} A+C & -(B+D) \\ (B+D) & (A+C) \end{bmatrix} \succeq 0.$$

So we obtain $A + C \succeq_{\mathrm{i}} (B + D)$. □

The following two lemmas from [85] provide the key characterization of area convexity.

**Lemma 2.16** (Lemma X in [85]). *Let $A \in \mathbb{R}^{2\times 2}$ be a symmetric matrix. Then,*

$$A \succeq_{\mathrm{i}} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \Leftrightarrow A \succeq 0 \textit{ and } \det(A) \geq 1.$$

*Proof.* Let $A = \begin{bmatrix} a & b \\ b & d \end{bmatrix}$, $B = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ and $T = \begin{bmatrix} A & -B \\ B & A \end{bmatrix}$. From the definition of the operator, $A \succeq_{\mathrm{i}} B$ if and only if $T \succeq 0$.

First, we note that $T \succeq 0$ implies $A \succeq 0$. Secondly, $T \succeq 0$ also implies that all its principal minors are nonnegative. This means that $a, d \geq 0$, and $d(\det(A) - 1) \geq 0$ (the principal minor obtained by removing the 3rd row and column). This implies that $\det(A) \geq 1$.

So $T \succeq 0$ implies $A$ must be invertible. Then, using the property for Schur complements, we obtain that $T \succeq 0 \Leftrightarrow A + BA^{-1}B \succeq 0$. Now, $A^{-1} = \dfrac{1}{ad - b^2} \begin{bmatrix} d & -b \\ -b & a \end{bmatrix}$, and hence $A + BA^{-1}B = A(1 - \frac{1}{\det(A)})$. This implies that $T \succeq 0 \Leftrightarrow A \succeq 0$ and $\det(A) \geq 1$. □

For any set $K$, we represent its interior, relative interior and closure as

$\mathrm{int}(K), \mathrm{relint}(K)$ and $\mathrm{cl}(K)$, respectively.

**Lemma 2.17** (Theorem 1.6 in [85]). *Let $\phi$ be twice differentiable on the interior of convex set $K$, i.e., $\mathrm{int}(K)$.*

1. *If $\phi$ is area convex with respect to $A$ on $\mathrm{int}(K)$, then $d^2\phi(x) \succeq_{\mathrm{i}} M_A$ for all $x \in \mathrm{int}(K)$.*
2. *If $d^2\phi(x) \succeq_{\mathrm{i}} M_A$ for all $x \in \mathrm{int}(K)$, then $\phi$ is area convex with respect to $\frac{1}{3}A$ on $\mathrm{int}(K)$. Moreover, if $\phi$ is continuous on $\mathrm{cl}(K)$, then $\phi$ is area convex with respect to $\frac{1}{3}A$ on $\mathrm{cl}(K)$.*

Having laid a basic foundation for area convexity, we now focus on its relevance to solving the saddle point problem (2.2). Considering Remark 2.12, we can write the gap function criterion of optimality in terms of bilinear form of the matrix $J$. Suppose we have a function $\phi$ which is area convex with respect to $H$ on set $\mathcal{W}$. Then, consider the following *jointly-regularized* version of the bilinear form:

$$\widetilde{\eta}(w) := \sup_{\overline{w} \in \mathcal{W}} \overline{w}^T J w - \phi(\overline{w}). \tag{2.3}$$

Similar to Nesterov's dual extrapolation, one can attain $O(\varepsilon^{-1})$ convergence of accelerated gradient descent for function $\widetilde{\eta}(w)$ in (2.3) over variable $w$. In order to obtain gradients of $\widetilde{\eta}(w)$, we need access to $\mathrm{argmax}_{\overline{w} \in \mathcal{W}} \overline{w}^T J w - \phi(\overline{w})$. However, it may not be possible to find an exact maximizer in all cases. Again, one can get around this difficulty by instead using an approximate optimization oracle of the problem in (2.3).

**Definition 2.18.** A $\delta$-optimal solution oracle (OSO) for $\phi : \mathcal{W} \to \mathbb{R}$ takes input $a$ and outputs $w \in \mathcal{W}$ such that

$$a^T w - \phi(w) \geq \sup_{\overline{w} \in \mathcal{W}} a^T \overline{w} - \phi(\overline{w}) - \delta.$$

```
Initialize $w_0 \leftarrow (\mathbf{0}_n, 1, \mathbf{0}_{p+c})$;
for $t = 0, \ldots, T$ do
    $w_{t+1} \leftarrow w_t + \Phi(Jw_t + 2J\Phi(Jw_t))$;
```

**Algorithm 1:** Area Convex Mixed Packing Covering (AC-MPC)

Given $\Phi$ as a $\delta$-OSO for a function $\phi$, consider the procedure described in Algorithm 1. For Algorithm 1, Sherman [85] shows the following:

**Lemma 2.19** (Theorem 1.3 from [85])**.** *Let $\phi : \mathcal{W} \to [-\rho, 0]$. Suppose $\phi$ is area convex with respect to $2\sqrt{3}H$ on $\mathcal{W}$. Then for $J = M_H$ and for all t $\geq 1$ we have $w_t/t \in \mathcal{W}$ and,*

$$\sup_{\overline{w} \in \mathcal{W}} \overline{w} J \cdot \frac{w_t}{t} \leq \delta + \frac{\rho}{t}.$$

*In particular, in $\rho\varepsilon^{-1}$ iterations, Algorithm 1 obtains a $(\delta + \varepsilon)$-solution to the saddle point problem* (2.2)*.*

The analysis of this lemma closely follows the analysis of Nesterov's dual extrapolation.

Note that each iteration consists of $O(1)$ matrix-vector multiplications, $O(1)$ vector additions, and $O(1)$ calls to the approximate oracle. Since the former two are parallelizable to $O(\log n)$ depth, the same remains to be shown for the oracle computation to complete the proof of the run-time in Theorem 2.1.

Recall from the discussion in Section 2.4 that the critical bottleneck of Nesterov's method is that the diameter of the $\ell_\infty$ ball is $\Omega(\sqrt{n})$, which is achieved even in the Euclidean $\ell_2$ norm. This makes $\rho$ in Lemma 2.19 to also be $\Omega(\sqrt{n})$, which can be a major bottleneck for high dimensional LPs, which are commonplace among real-world applications.

Although, on the face of it, area convexity applied to the saddle point formulation (2.2) has a similar framework to Nesterov's dual extrapolation, the challenge is to construct a $\phi$ for which we can overcome the above bottleneck. Particularly, there are three key challenges to tackle:

1. We need to show that existence of a function $\phi$ that is area convex with respect to $H$ on $\mathcal{W}$.
2. $\phi : \mathcal{W} \to [-\rho, 0]$ should be such that $\rho$ is not too large.
3. There should exist an efficient $\delta$-OSO for $\phi$.

In the next subsection, we focus on these three aspects in order to complete our analysis.

### 2.5.3 Choosing an area convex function

First, we consider a simple 2-D gadget function and prove a "nice" property of this gadget. Using this gadget, we construct a function which can be shown to be area convex using the aforementioned property of the gadget.

Let $\gamma_\beta : \mathbb{R}^2_+ \to \mathbb{R}$ be a function parameterized by $\beta$ defined as

$$\gamma_\beta(a, b) = ba \log a + \beta b \log b.$$

**Lemma 2.20.** *Suppose $\beta \geq 2$. Then $d^2\gamma_\beta(a, b) \succeq \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ for all $a \in (0, 1]$ and $b > 0$.*

*Proof.* We use the equivalent characterization proved in Lemma 2.16. We need to show that $d^2\gamma_\beta(a, b) \succeq 0$ and $\det(d^2\gamma_\beta(a, b)) \geq 1$ for all $a \in (0, 1]$ and $b > 0$. First of all, note that $d^2\gamma_\beta$ is well-defined on this domain. In particular, we can write

$$d^2\gamma_\beta(a, b) = \begin{bmatrix} \dfrac{\beta}{b} & 1 + \log a \\ 1 + \log a & \dfrac{b}{a} \end{bmatrix}.$$

Note that a $2 \times 2$ matrix is PSD if and only if its diagonal entries and determinant are nonnegative. Clearly diagonal entries of $d^2\gamma_\beta(a,b)$ are nonnegative for the given values of $\beta, a$ and $b$. Hence, in order to prove the lemma, it suffices to show that $\det(d^2\gamma_\beta(a,b)) \geq 1$.

$\det(d^2\gamma_\beta(a,b)) = \frac{\beta}{a} - (1+\log a)^2$ is only a function of $a$ for any fixed value of $\beta \geq 2$. Moreover, it can be shown that $\det(d^2\gamma_\beta)$ is a decreasing function of $a$ on set $(0,1]$. Clearly, the minimum occurs at $a = 1$. However, $\det(d^2\gamma_\beta(1,b)) = \beta - 1 \geq 1$ for all $b > 0$. Hence we have that $\det(d^2\gamma_\beta(a,b)) \geq 1$ for all $a \in (0,1], b > 0$ and $\beta \geq 2$.

Finally to see the claim that $\det(d^2\gamma_\beta)$ is a decreasing function of $a \in (0,1]$ for any $\beta \geq 2$, consider

$$\begin{aligned}\frac{d}{da}\big(\det(d^2\gamma_\beta(a,b))\big) &= -\frac{\beta}{a^2} - \frac{2(1+\log a)}{a}\\ &\leq -\frac{2(1+a(1+\log a))}{a^2} < 0\end{aligned}$$

where the last inequality follows from the observation that $1 + a + a\log a > 0$ for all $a \in (0,1]$. $\square$

Now, using the function $\gamma_\beta$, we construct a function $\phi$ and use the sufficiency criterion provided in Lemma 2.17 to show that $\phi$ is area convex with respect to $J$ on $\mathcal{W}$. Note that our set of interest $\mathcal{W}$ is not full-dimensional, whereas Lemma (2.17) is only stated for $\mathrm{int}$ and not for $\mathrm{relint}$. To get around this difficulty, we consider a larger set $\overline{\mathcal{W}} \supset \mathcal{W}$ such that $\overline{\mathcal{W}}$ is full dimensional and $\phi$ is area convex on $\overline{\mathcal{W}}$. Then we use Remark 2.14 to obtain the final result, i.e., area convexity of $\phi$.

**Theorem 2.21.** *Let $w = (x,u,y,z)$ and define*

$$\phi(w) \stackrel{\text{def}}{=} \sum_{i=1}^{p}\sum_{j=1}^{n} P_{ij}\gamma_{p_i}(x_j,y_i) + \sum_{i=1}^{p}\gamma_2(u,y_i) + \sum_{i=1}^{c}\sum_{j=1}^{n} C_{ij}\gamma_{c_i}(x_j,z_i) + \sum_{i=1}^{c}\gamma_2(u,z_i),$$

*where $p_i = 2\frac{\|P\|_\infty}{\|P_{i,:}\|_1}$ and $c_i = 2\frac{\|C\|_\infty}{\|C_{i,:}\|_1}$. Then $\phi$ is area convex with respect to* $\frac{1}{3}\begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix}$

*on set* $\overline{\mathcal{W}} \overset{\text{def}}{=} \mathcal{B}^{n+1}_{+,\infty} \times \Delta_p^+ \times \Delta_c^+$. *In particular, it also implies* $6\sqrt{3}\phi$ *is area convex with respect to* $2\sqrt{3}\begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix}$ *on set* $\mathcal{W}$.

*Proof.* Note that $\gamma_{c_i}, \gamma_{p_i}$ are twice differentiable in $\operatorname{int}(\overline{\mathcal{W}})$. So, from Part 2 of Lemma 2.17, it is sufficient to prove that $d^2\phi(w) \succeq_{\mathrm{i}} J$ for all $w \in \operatorname{int}(\overline{\mathcal{W}})$.

By definition, we have $\gamma_{c_i} \geq 2$ for all $i \in [c]$ and $\gamma_{p_i} \geq 2$ for all $i \in [p]$. Moreover $x_j \in (0,1)$ and $y_i > 0, z_i > 0$ for any $w = (x,u,y,z) \in \operatorname{int}(\mathcal{W})$. Then by Lemma 2.20 and Proposition 2.15, we have

$$\begin{aligned} d^2\phi(w) &= \sum_{i=1}^{p}\sum_{j=1}^{n} P_{ij} d^2\gamma_{p_i}(x_j, y_i) + \sum_{i=1}^{p} d^2\gamma_2(u, y_i) + \sum_{i=1}^{c}\sum_{j=1}^{n} C_{ij} d^2\gamma_{c_i}(x_j, y_i) + \sum_{i=1}^{c} d^2\gamma_2(u, z_i) \\ &\succeq_{\mathrm{i}} \Big( \sum_{i=1}^{p}\sum_{j=1}^{n} -P_{ij} e_j \otimes e_{n+1+i} + \sum_{i=1}^{p} e_{n+1} \otimes e_{n+1+i} \\ &\qquad + \sum_{i=1}^{c}\sum_{j=1}^{n} C_{ij} e_j \otimes e_{n+p+i} + \sum_{i=1}^{c} (-1) e_{n+1} \otimes e_{n+1+p+i} \Big), \end{aligned} \tag{2.4}$$

where $e_k \otimes e_l = e_k e_l^T - e_l e_k^T$. Here we arrive at $P_{ij} d^2\gamma_{p_i}(x_j, y_i) \succeq_{\mathrm{i}} -P_{ij} e_j \otimes e_{n+1+i}$ using Lemma 2.20 and Parts 1-2 of Proposition 2.15; and $C_{ij} d^2\gamma_{c_i}(x_j, y_i) \succeq_{\mathrm{i}} C_{ij} e_j \otimes e_{n+1+p+i}$ using Lemma 2.20 and Part 2 of Proposition 2.15. Similar arguments can be made about terms inside the other two summations. Finally we use Part 3 of Proposition 2.15 to obtain (2.4). Note that the matrix in the last summation is in fact $J$.

Since $d^2\phi \succeq_{\mathrm{i}} J$, using Part 2 of Proposition 2.15, we have $d^2 6\sqrt{3}\phi \succeq_{\mathrm{i}} 6\sqrt{3}J$. Then by Part 2 of Lemma 2.17, we obtain that $6\sqrt{3}\phi$ is area convex with respect to $2\sqrt{3}\begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix}$ on set $\overline{\mathcal{W}}$.

Note that the set of interest $\mathcal{W} \subset \overline{\mathcal{W}}$. Moreover, $\mathcal{W}$ is a convex subset. By Remark 2.14, one can see that $6\sqrt{3}\phi$ is area convex with respect to $2\sqrt{3}\begin{bmatrix} P & -\mathbf{1}_p \\ -C & \mathbf{1}_c \end{bmatrix}$ on set $\mathcal{W}$. □

Theorem 2.21 addresses the first part of the key three challenges. Next, Lemma 2.22

shows an upper bound on the range of $\phi$.

**Lemma 2.22.** *Function* $\phi : \mathcal{W} \to [-\rho, 0]$ *then* $\rho = O(\|P\|_\infty \log p + \|C\|_\infty \log c)$.

*Proof.* Note that $\gamma_\beta(a,b) \le 0$ for any $a \in [0,1], b \in [0,1], \beta \ge 0$. Since $P_{ij} \ge 0, C_{kj} \ge 0$ for all possible values of $i, j, k$ hence we clearly have $\phi(w) \le 0$ for all $w \in \mathcal{W}$. Now we prove that lower bound is not too small.

We have

$$\begin{aligned}
\sum_{i=1}^{p}\sum_{j=1}^{n} P_{ij}\gamma_{p_i}(x_j, y_i) &= \sum_{i=1}^{p}\sum_{j=1}^{n} P_{ij}(y_i x_j \log x_j + p_i y_i \log y_i) \\
&\ge -\sum_{i=1}^{p}\sum_{j=1}^{n} P_{ij} y_i \frac{1}{e} + \sum_{i=1}^{p} p_i y_i \log y_i \sum_{j=1}^{p} P_{ij} \\
&= -\sum_{i=1}^{p}\sum_{j=1}^{n} P_{ij} y_i \frac{1}{e} + \sum_{i=1}^{p} 2\|P\|_\infty y_i \log y_i \\
&\ge -\sum_{i=1}^{p} \frac{\|P\|_\infty}{e} y_i + \sum_{i=1}^{p} 2\|P\|_\infty y_i \log y_i \\
&\ge -\frac{\|P\|_\infty}{e} - 2\|P\|_\infty \log p.
\end{aligned}$$

Note that $w \in \mathcal{W}$ implies $u = 1$. So

$$\sum_{i=1}^{p} \gamma_2(u, y_i) = \sum_{i=1}^{p} 2y_i \log(y_i) \ge -2\log p.$$

Similarly, we have

$$\sum_{i=1}^{c}\sum_{j=1}^{n} C_{ij}\gamma_{c_i}(x_j, z_i) \ge -\frac{\|C\|_\infty}{e} - 2\|C\|_\infty \log c.$$
$$\sum_{i=1}^{c} \gamma_2(u, z_i) \ge -2\log c.$$

Taking sum of all four terms, we conclude the proof. $\square$

Finally, we need an efficient $\delta$-OSO. Consider the alternating minimization algorithm pictured in Algorithm 2.

```
**Input:** $a \in \mathbb{R}^{n+1}, a^1 \in \mathbb{R}^p, a^2 \in \mathbb{R}^c, \delta > 0$
Initialize $(x^0, u^0) \in \mathcal{B}^n_{+,\infty} \times \{1\}$ arbitrarily;
**for** $k = 1, \ldots, K$ **do**
    $(y^k, z^k) \leftarrow \operatorname{argmax}_{y \in \Delta_c^+,\ z \in \Delta_p^+} \; y^T a^1 + z^T a^2 - \phi(x^{k-1}, u^{k-1}, y, z)$;
    $(x^k, u^k) \leftarrow \operatorname{argmax}_{(x,u) \in \mathcal{B}^n_{+,\infty} \times \{1\}} \; [x^T\ u]a - \phi(x, u, y^k, z^k)$;
```

**Algorithm 2:** $\delta$-OSO for $\phi$

Beck [14] shows the following convergence result.

**Lemma 2.23** (Theorem X from [14])**.** *For $\delta > 0$, Algorithm 2 is a $\delta$-OSO for $\phi$ which converges in $O(\log \delta^{-1})$ iterations.*

We show that for our chosen $\phi$, we can perform the two argmax computations in each iteration of Algorithm 2 analytically in time $O(\mathrm{nnz}(P) + \mathrm{nnz}(C))$, and hence we obtain a $\delta$-OSO which takes $O((\mathrm{nnz}(P) + \mathrm{nnz}(C)) \log \delta^{-1})$ total work. Parallelizing matrix-vector multiplications eliminates the dependence on $\mathrm{nnz}(P)$ and $\mathrm{nnz}(C)$, at the cost of another $\log(N)$ term.

**Lemma 2.24.** *Each* argmax *in Algorithm 2 can be computed as follows:*

$x^k = \min\left\{\exp\left(\frac{a}{P^T y^k + C^T z^k} - 1\right), \mathbf{1}_n\right\}$ *for all $j \in [n]$.*

$y^k = \mathrm{proj}_{\Delta_p^+}\left(\exp\left\{\frac{1}{2(\|P\|_\infty + 1)}(a^1 - Px^{k-1} \log x^{k-1})\right\}\right)$

$z^k = \mathrm{proj}_{\Delta_c^+}\left(\exp\left\{\frac{1}{2(\|C\|_\infty + 1)}(a^2 - Cx^{k-1} \log x^{k-1})\right\}\right)$

*In particular, we can compute $x^k, y^k, z^k$ in $O(\mathrm{nnz}(P) + \mathrm{nnz}(C))$ work and $O(\log N)$ parallel time.*

*Proof.* Note that maximization with respect to $u$ is trivial since $u = 1$ is a fixed variable. We first look at maximization with respect to $x \in \mathcal{B}^n_{+,\infty}$. Writing the first order necessary condition of Lagrange multipliers, we have

$$a_j - \sum_{i=1}^{p} P_{ij} \frac{\partial}{\partial t}\gamma_{p_i}(t,v)\Big|_{(t,v)=(x_j,y_i)} - \sum_{i=1}^{c} C_{ij} \frac{\partial}{\partial t}\gamma_{c_i}(t,v)\Big|_{(t,v)=(x_j,z_i)} - \lambda_j = 0$$
$$\Rightarrow a_j - \{\sum_{i=1}^{p} P_{ij}y_i + \sum_{i=1}^{c} C_{ij}z_i\}(1+\log x_j) - \lambda_j = 0.$$

Here $\lambda_j$ is the Lagrange multiplier corresponding to the case that $x_j = 1$. By complimentary slackness, we have $\lambda_j > 0$ iff $x_j = 1$.

This implies $x_j = \min\left\{ \exp\left( \frac{a_j}{\sum\limits_{i=1}^{p} P_{ij}y_i + \sum\limits_{i=1}^{c} C_{ij}z_i} - 1 \right), 1 \right\}$ for all $j \in [n]$.

Now we consider maximization with respect to $y, z$. Note that there are no cross-terms of $y_i$ and $z_i$, i.e., $\frac{\partial \gamma_{p_i}}{\partial y_i}$ is independent of $z$ variable and vice-versa. So we can optimize them separately. From first order necessary condition of Lagrange multipliers for $y$, we have

$$a_i^1 - \sum_{j=1}^{n} P_{ij} \frac{\partial}{\partial v}\gamma_{p_i}(t,v)\Big|_{(t,v)=(x_j,y_i)} - \frac{\partial}{\partial v}\gamma_2(t,v)\Big|_{(t,v)=(u,y_i)} - \lambda = 0$$
$$\Rightarrow a_i^1 - \sum_{j=1}^{n} P_{ij}(x_j \log x_j + p_i(1+\log y_i)) - u\log u|_{u=1} - 2(1+\log y_i) - \lambda = 0$$
$$\Rightarrow a_i^1 - \sum_{j=1}^{n} P_{ij}x_j \log x_j - 2(\|P\|_\infty + 1)(1+\log y_i) - \lambda = 0$$

where last relation follows due to definition of $p_i$ and $\lambda$ is Lagrange multiplier corresponding to the constraint $\sum\limits_{i=1}^{p} y_i \le 1$. By comple mentary slackness, we have $\lambda > 0$ iff $\sum\limits_{i=1}^{p} y_i = 1$.

Eliminating $\lambda$ from above equations, we obtain

$$y = \text{proj}_{\Delta_p^+}\left( \exp\left\{ \frac{1}{2(\|P\|_\infty + 1)}(a^1 - Px\log x) \right\} \right).$$

Similarly, we obtain

$$z = \text{proj}_{\Delta_c^+}\left( \exp\left\{ \frac{1}{2(\|C\|_\infty + 1)}(a^2 - Cx\log x) \right\} \right).$$

It is clear from the analytical expressions that for each iteration of Algorithm 2, we need $O(\text{nnz}(P) + \text{nnz}(C))$ time. Hence, the total runtime of Algorithm 2 is $O((\text{nnz}(P) + \text{nnz}(C)) \log \delta^{-1})$. $\square$

Theorem 2.21 and Lemmas 2.22-2.24 combined together complete the proof of Theorem 2.1.

## 2.6 Application to the Densest Subgraph problem

We conclude the chapter by circling back to our original problem - the densest subgraph problem. In this section, we apply the result in Theorem 2.1 to the dual LP described in 1.2. This can be viewed as using the algorithm described by Bahmani *et al.* [11], but instead of using the multiplicative weights update technique to solve the dual, we use our algorithm instead.

Parameterizing the dual LP with respect to $D$, we get the following feasibility problem.

**MPC formulation of Densest Subgraph**

Given a graph $G = \langle V, E \rangle$, and a candidate threshold value $D$, does there exist a $f$ such that:

$$\begin{aligned} f_e(u)+f_e(v) &\geq 1, && \forall e = uv \in E \\ \sum_{e \ni v} f_e(v) &\leq D, && \forall v \in V \\ f_e(u) \geq 0, f_e(v) &\geq 0, && \forall e = uv \in E \end{aligned}$$

It is easy to see that the above formulation is a Mixed Packing and Covering (MPC) problem, and hence one can directly apply Theorem 2.1 to solve it. The solution to the densest subgraph problem (i.e., the maximum subgraph density) is simply the smallest value of $D$ for which the LP is feasible. Since $D$ can take at most $O(|V||E|) \leq O(|V|^3)$ values in total, the densest subgraph problem can be reduced to solving $O(\log |V|)$ instances of MPC, where the number of nonzeros $N$ in the matrix is $O(|E|)$ and the width $w$ is simply the maximum degree in $G$. This gives the following corollary.

**Corollary 2.25.** *Given a graph $G = \langle V, E \rangle$ with maximum degree $\Delta$, we can find a $(1+\varepsilon)$-approximation to the maximum subgraph density of $G$, $\rho_G^*$, in parallel time $\widetilde{O}(\Delta\varepsilon^{-1})$ and total work $\widetilde{O}(|E|\Delta\varepsilon^{-1})$.*

Note that we have only a dual solution, which does not directly give a primal solution, i.e., a subgraph with density at least $(1-\varepsilon)$ of the optimum. We describe a way to make this translation in Chapter 4 (Section 4.2.2).

The previous fastest algorithms for densest subgraph do not depend on $\Delta$ - however, their dependence on $\varepsilon^{-1}$ is quadratic. Corollary 2.25 gives the fastest algorithm for this problem in the high precision regime ($\varepsilon < 1/\Delta$), since its dependence on $\varepsilon^{-1}$ is only linear.

# CHAPTER 3
# FAST COMBINATORIAL ALGORITHM FOR DENSEST SUBGRAPH

In Chapter 2, we designed an efficient numerical descent-based algorithm for mixed packing and covering linear programs, and thereby for the densest subgraph problem. In this chapter, we explore combinatorial algorithms to find dense subgraphs.

## 3.1 Combinatorial Algorithms for Densest Subgraph

Finding the exact solution to the densest subgraph problem can be reduced to solving a few instances of a maximum flow problem [42, 38]. Maximum flow computations, despite theoretical advancements, require a running time super-linear in the size of the input, and hence are not scalable to sizes of many current real-world networks. Although we have shown the existence of several numerical near-linear time algorithms for the problem, these are still far from being optimal algorithms in practice, especially when we want to run our algorithms on a single machine. Due to this, Charikar's greedy peeling algorithm is frequently used in practice [26] despite its approximation factor being far from optimal. This algorithm iteratively peels the lowest degree node from the graph, thus producing a sequence of subsets of nodes, of which it outputs the densest one. This simple, linear time and linear space algorithm provides a $1/2$-approximation for the densest subgraph problem (DSP).

Our work in this chapter was originally motivated by a natural question: How can we quickly assess whether the output of Charikar's algorithm on a given graph instance is closer to optimality or to the worst case $1/2$-approximation guarantee? However, we ended up answering the following intriguing question that we state as the next problem:

**Problem 3.1.** Can we design an algorithm that performs

- as well as Charikar's greedy algorithm in terms of efficiency, and
- as well as the maximum flow-based exact algorithm in terms of output quality?

As a solution to the above problem, we make the following contributions in this chapter.

- We design a novel algorithm GREEDY++ for the densest subgraph problem that combines the best of two different worlds, the accuracy of the exact maximum flow based algorithm [42, 38], and the efficiency of Charikar's greedy peeling algorithm [26].
- It is worth outlining that Charikar's greedy algorithm typically performs better on real-world graphs than the worse case $1/2$-approximation; on a variety of datasets we have tried, the worst case approximation was 0.8. Nonetheless, the only way to verify how close the output is to optimality relies on computing the exact solution using maximum flow. Our proposed method GREEDY++ can be used to assess the accuracy of Charikar's algorithm in practice. Specifically, we find empirically that for all graph instances where GREEDY++ after a couple of iterations does not significantly improve the output density, the output of Charikar's algorithm is near-optimal.
- We implement our proposed algorithm in C++ and apply it on a variety of real-world datasets. We verify the practical value of GREEDY++. Our empirical results indicate that GREEDY++ is a valuable addition to the toolbox of dense subgraph discovery; on real-world graphs, GREEDY++ is both fast in practice, and converges to a solution with an arbitrarily small approximation factor.

## 3.2 Charikar's Greedy Algorithm

Since our algorithm GREEDY++ is an improvement over Charikar's greedy algorithm, we discuss the latter algorithm in greater detail. The algorithm removes in each iteration, the node with the smallest degree. This process creates a nested sequence of sets of nodes $V = S_n \supset S_{n-1} \supset S_{n-2} \supset \ldots \supset S_1 \supset \emptyset$. The algorithm outputs the graph $G[S_j]$ that maximizes the degree density among $j = 1, \ldots, n$. The pseudocode is shown in Algorithm 3.

```
Input: Undirected graph $G$
Output: A dense subgraph of $G$: $G_{\text{densest}}$
$G_{\text{densest}} \leftarrow G$;
$H \leftarrow G$;
while $H \neq \emptyset$ do
    Find the vertex $u \in H$ with minimum $\text{deg}_H(u)$;
    Remove $u$ and all its adjacent edges $uv$ from $H$;
    if $\rho(H) > \rho(G_{\textit{densest}})$ then
        $G_{\text{densest}} \leftarrow H$;
return $G_{\textit{densest}}$
```

**Algorithm 3:** GREEDY

## 3.3 Proposed Method

### 3.3.1 The GREEDY++ algorithm

As we discussed earlier, Charikar's peeling algorithm greedily removes the node of smallest degree from the graph, and returns the densest subgraph among the sequence of $n$ subgraphs created by this procedure. While ties may exist, and are broken arbitrarily, for the moment it is useful to think as if Charikar's greedy algorithm produces a single permutation of the nodes, that naturally defines a nested sequence of subgraphs.

**Algorithm description.** Our proposed algorithm GREEDY++ iteratively runs Charikar's peeling algorithm, while keeping some information about the past runs. This information is crucial, as it results in different permutations, that naturally yield higher quality outputs. The pseudocode for GREEDY++ is shown in Algorithm 7. It takes as input the graph $G$, and a parameter $T$ of the number of passes to be performed, and runs an iterative, weighted peeling procedure. In each round the load of each node is a function of its induced degree and the load from the previous rounds. It is worth outlining that the algorithm is easy to implement, as it is essentially $T$ instances of Charikar's algorithm. What is less obvious perhaps, is why this algorithm makes sense, and works well. We answer this question in detail in Section 3.3.2.

```
Input: Undirected graph $G$, iteration count $T$
Output: An approximately densest subgraph of $G$: $G_{\mathsf{densest}}$
$G_{\mathsf{densest}} \leftarrow G$;
Initialize the vertex load vector $\ell^{(0)} \leftarrow 0 \in \mathbb{Z}^n$;
for $i : 1 \rightarrow T$ do
    $H \leftarrow G$;
    while $H \neq \emptyset$ do
        Find the vertex $u \in H$ with minimum $\ell_u^{(i-1)} + \mathsf{deg}_H(u)$;
        $\ell_u^{(i)} \leftarrow \ell_u^{(i-1)} + \mathsf{deg}_H(u)$;
        Remove $u$ and all its adjacent edges $uv$ from $H$; if $\rho(H) > \rho(G_{densest})$
         then
            $G_{\mathsf{densest}} \leftarrow H$;
return $G_{densest}$
```

**Algorithm 4:** GREEDY++

**Example.** We provide a graph instance that clearly illustrates why GREEDY++ is a significant improvement over the classical greedy algorithm. We discuss the first two rounds of GREEDY++. Consider the following graph $G = B \bigcup \left( \cup_{i=1}^{k} H_i \right)$ where $B = K_{d,D}$ and $H_i = K_{d+2}$. Namely $G$ is a disjoint union of a complete $d \times D$ bipartite graph $B$, and of $k$ $(d+2)$-cliques $H_1, \ldots, H_k$. Consider the case where $d \ll D, k \rightarrow +\infty$. $G$ is pictured in Figure 3.1(a). The density of $G$ is

$$\frac{2dD + (d+1)(d+2)k}{2d + 2D + 2k(d+2)} \rightarrow \frac{d+1}{2}.$$

Notice that this is precisely the density of any $(d+2)$-clique. However, the density of $B$ is $\frac{dD}{d+D} \approx d$, which is in fact the optimal solution. Charikar's algorithm outputs $G$ itself, since it starts eliminating nodes of degree $d$ from $B$, and by doing this, it never sees a subgraph with higher density. This example illustrates that the $1/2$ approximation is tight. Consider now a run of GREEDY++.

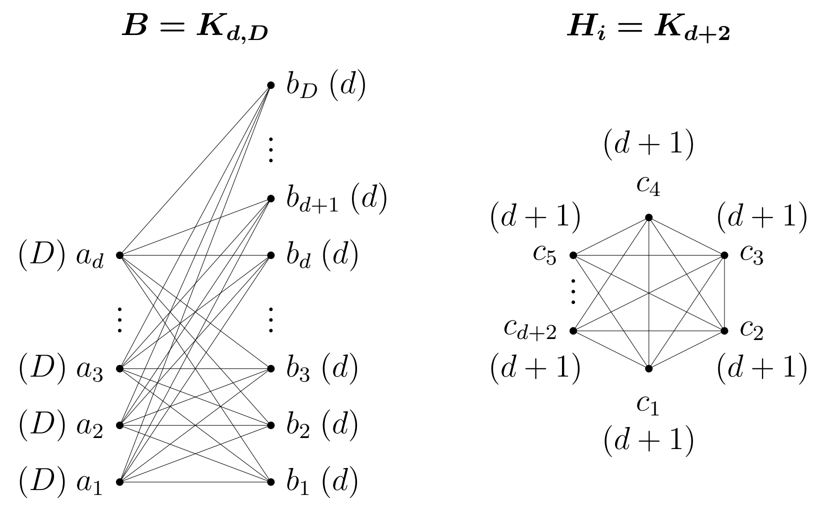
In its first iteration, it simply emulates Charikar's algorithm. The $D - d$ vertices of $B$ which were eliminated first - each have load $d$. At this stage, our input is the disjoint union of $k$ cliques and a $d \times d$ bipartite graph. Of the remaining $2d$ vertices in $B$, one vertex is charged with load $d$, two vertices each with loads $(d-1), (d-2), \ldots, 1$, and one vertex with load $0$. On the other hand, vertices in $H_i$ are charged with loads $d+1, d, \ldots, 0$. Figure 3.1(b) shows the cumulative degrees of vertices in $G$ after one iteration of GREEDY++.

Without any loss of generality let us assume the vertex from $B$ that got charged $0$ originally had degree $d$. This vertex in the second iteration will get deleted first, and the vertex whose sum of load and degree is $d + 1$ will get deleted second. But after these two, all the cliques get peeled away by the algorithm. This leaves us with a $d \times D - 2$ bipartite graph as the output after the second iteration, whose density is almost optimal.

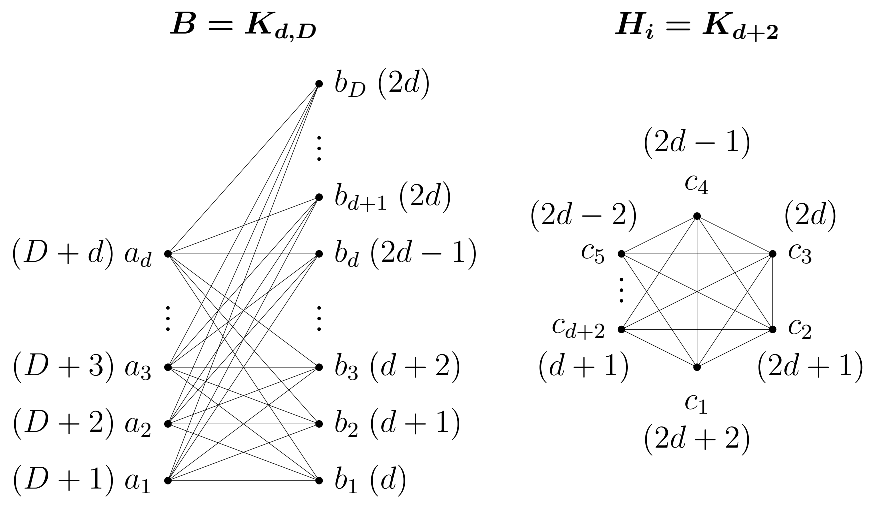
**Theoretical guarantees.** Before we prove our theoretical properties for our proposed algorithm GREEDY++, it is worth outlining that experiments indicate that the performance of GREEDY++ is significantly better than the worst-case analysis we perform. Furthermore, we conjecture that our guarantees are not tight from a theoretical perspective; an interesting open question is to extend our analysis in Section 3.3.2 for GREEDY++ to prove that it provides asymptotically an optimal solution for the DSP. We conjecture that our algorithm is a $(1+1/\sqrt{T})$-approximation algorithm for the DSP. Our fist lemma states that GREEDY++ is a 2-approximation algorithm for the DSP.

**Lemma 3.2.** *Let $G_{\textsf{densest}}$ the output of* GREEDY++. *Then, $\rho(G_{\textsf{densest}}) \geq \rho^*_G/2$, where $\rho^*_G$ denotes the optimum value of the problem.*

*Proof.* Notice that the first iteration is identical to Charikar's $1/2$-approximation algorithm, and $G_{\textsf{densest}}$ is at least as dense as the output of the first iteration. $\square$



(a) Initial degrees of $G$



(b) Cumulative degrees (degree + load) of $G$ after one iteration

Figure 3.1: Illustration of two iterations of GREEDY++ on $G$. The output after one iteration is $\boldsymbol{G}$ itself (density $\boldsymbol{\approx (d+1)/2}$), whereas the output after the second iteration is $\boldsymbol{B \setminus \{b_1, b_2\}}$ (density $\boldsymbol{\approx d}$).

The next lemma provides bounds the quality of the dual solution, i.e., at each iteration the average load (average over the algorithm's iterations) assigned to any vertex is at most $2\rho^*_G$.

**Lemma 3.3.** *The following invariant holds for* GREEDY++*: for any vertex $v$ and iteration $i$, $\ell_v^{(i)} \leq 2i \cdot \rho^*_G$.*

*Proof.* Consider the point at which vertex $v$ is chosen in iteration $i$. Denote the graph at that instant to be $G_v^{(i)} = \left\langle V_v^{(i)}, E_v^{(i)} \right\rangle$.

First, let $i = 1$. The proof for this base case goes through identically as in [26].

$$\begin{aligned} \ell_v^{(1)} = \deg_{G_v^{(1)}}(v) &\leq \frac{1}{|V_v^{(i)}|} \sum_{u \in V_v^{(i)}} \deg_{G_v^{(1)}}(u) \\ &= \frac{2|E_v^{(1)}|}{|V_v^{(1)}|} \\ &= 2 \cdot \rho_{G_v^{(1)}} \leq 2 \cdot \rho_G^*. \end{aligned}$$

Now, assume that the statement is true for some iteration index $i - 1$. For any vertex $u$ at that point, the cumulative degree is $\ell_u^{(i-1)} + \deg_{G_v^{(i)}}(u)$. Since $v$ has the minimum cumulative degree at that point,

$$\begin{aligned} \ell_v^{(i)} = \ell_v^{(i-1)} + \deg_{G_v^{(i)}}(v) &\leq \frac{1}{|V_v^{(i)}|} \sum_{u \in V_v^{(i)}} \left( \ell_u^{(i-1)} + \deg_{G_v^{(i)}}(u) \right) \\ &\leq 2(i-1)\rho_G^* + \frac{1}{|V_v^{(i)}|} \sum_{u \in V_v^{(i)}} \deg_{G_v^{(i)}}(u) \\ &\leq 2i \cdot \rho_G^*. \end{aligned}$$

□

**Running time.** Finally, we bound the runtime of the algorithm as follows. The next lemma states that our algorithm can be implemented to run in $O((n + m) \cdot \min(\log n, T))$.

**Lemma 3.4.** *Each iteration of the above algorithm runs in time* $O((n+m) \cdot \min(\log n, T))$.

*Proof.* The deletion operation, along with assigning edges to a vertex and updating degrees takes $O(m)$ time since every edge is assigned once. Finding the minimum degree vertex can be implemented in two ways:

1. Since degrees in our algorithm can go from $0$ to $2Tm$, we can create lists for each separate integer degree value. Now we need to scan each list from $\deg = 1$ to

$\deg = 2Tm$. However, after deleting a vertex of degree $d$, we only need to scan from $d-1$ onwards. So the total time taken is $O(2Tm+n) = O(mT)$.

2. We can maintain a priority queue, which needs a total of $O(m)$ update operations, each taking $O(\log n)$ time. □

Note that in the case of weighted graphs, we cannot maintain lists for each possible degree, and hence, it is necessary to use a priority queue.

### 3.3.2 Why does GREEDY++ work well?

Explaining the intuition behind GREEDY++ requires an understanding of the load balancing interpretation of Charikar's LP for the DSP [26], and the multiplicative weights update (MWU) framework by Plotkin, Shmoys and Tardos [78] used for packing/covering LPs. In the context of the DSP, the MWU framework was first used by Bahmani, Goel, and Munagala [11]. We include a self-contained exposition of the required concepts from [11, 26] in this section, that has a natural flow and concludes with our algorithmic contributions. Intuitively, the additional passes that GREEDY++ performs, improve the load balancing.

**Charikar's LP and the load balancing interpretation.** The following is a well-known LP formulation of the densest subgraph problem, introduced in [26], which we denote by PRIMAL$(G)$. Binary variables $x_u$ and $y_e$ indicate the inclusion of a vertex $u$/edge $e$ in an optimal densest subgraph. Relaxing the variables to real numbers and suitably scaling down the $x_u$ values, we get the following LP, whose optimal objective value is known to be $\rho^*_G$.

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{e\in E} y_e & \\
\text{subject to} & y_e \le x_u, & \forall e = uv \in E \\
& y_e \le x_v, & \forall e = uv \in E \\
& \displaystyle\sum_{v\in V} x_v \le 1, & \\
& y_e \ge 0, & \forall e \in E \\
& x_v \ge 0, & \forall v \in V
\end{array}$$

We then construct the dual LP for the above problem. Let $f_e(u)$ be the dual variable associated with the first $2m$ constraints of the form $y_e \le x_u$, and let $D$ be associated with the last constraint. We get the following LP, which we denote by $\textsc{Dual}(G)$, and whose optimum is also $\rho^*_G$.

$$\begin{array}{lrl}
\text{minimize} & D & \\
\text{subject to} & f_e(u)+f_e(v) \ge 1, & \forall e = uv \in E \\
& \ell_v \overset{\text{def}}{=} \displaystyle\sum_{e\ni v} f_e(v) \le D, & \forall v \in V \\
& f_e(u) \ge 0, & \forall e = uv \in E \\
& f_e(v) \ge 0, & \forall e = uv \in E
\end{array}$$

This LP can be visualized as follows. Each edge $e = uv$ has a load of $1$, which it wants to send to its end points: $f_e(u)$ and $f_e(v)$ such that the total load of any vertex $v$, $\ell_v$, is at most $D$. The objective is to find the minimum $D$ for which such a load assignment is feasible.

For a fixed $D$, the above dual problem can be framed as a flow problem on a bipartite graph as follows: Let the left side $L$ represent $V$ and the right side $R$ represent $E$. Add a super-source $s$ and edges from $s$ to all vertices in $L$ with capacity $D$. Add edges from $v \in V$ to $e \in E$ if $e$ is incident on $v$ in $G$. All vertices in $R$ have demands of $1$ unit. Although Goldberg's initial reduction [42] involved a different flow network, this graph can also be used to use maximum flow and use that to find the exact optimum to our problem. From strong duality, we know that the optimal objective values of both linear programs are equal, i.e., exactly $\rho^*_G$. Let $\rho_G$ be the objective of any feasible solution to $\textsc{Primal}(G)$. Similarly, let $\hat{\rho}_G$ be the objective of any feasible solution to $\textsc{Dual}(G)$. Then, by optimality of $\rho^*_G$ and weak duality, we obtain the optimality result $\rho_G \leq \rho^*_G \leq \hat{\rho}_G$.

**Bahmani et al. [11]** use the following covering LP formulation: decide the feasibility of constraints $f_e(u) + f_e(v) \geq 1$ for each edge $e = uv \in E$ subject to the polyhedral constraints:

$$
\begin{aligned}
\sum_{e \ni v} f_e(v) &\leq D, && \forall v \in V \\
f_e(u) &\geq 0, && \forall e = uv \in E \\
f_e(v) &\geq 0, && \forall e = uv \in E
\end{aligned}
$$

The width of this linear program is the maximum value of $f_e(u) + f_e(v)$ provided that $f_e(u), f_e(v)$ satisfy the constraints of the program. Bahmani et al. in order to provably bound the *width* of the above LP, they introduce another set of simple constraints as follows:

$$
\begin{aligned}
\sum_{e \ni v} f_e(v) &\leq D, && \forall v \in V \\
q \geq f_e(u) &\geq 0, && \forall e = uv \in E \\
q \geq f_e(v) &\geq 0, && \forall e = uv \in E
\end{aligned}
$$

where $q \geq 1$ is a small constant. So, for a particular value of $D$, they verify the approximate feasibility of the covering problem using the MWU framework. However, this necessitates running a binary search over all possible values of $D$ and finding the lowest value of $D$ for which the LP is feasible. Since the precision for $D$ can be as low as $\epsilon$, this binary search is inefficient in practice. Furthermore, due to the added $\ell_\infty$ constraint to bound the width, extracting the primal solution (i.e. an approximately densest subgraph) from the dual is no longer straightforward, and the additional rounding step to overcome this incurs additional loss in the approximation factor.

In order to overcome these practical issues, we propose an alternate MWU formulation which sacrifices the width bounds but escapes the binary search phase over $D$. Eliminating the artificial width bound makes it straightforward to extract a primal solution. Moreover, our experiments on real world graphs suggest that width is not a bottleneck for the running time of the MWU algorithm. Even more importantly, our alternate formulation naturally yields GREEDY++ as we explain in the following.

**Our MWU formulation.** We can denote the LP $\textsc{Dual}(G)$ succinctly as follows:

$$\begin{array}{ll} \text{minimize} & D \\ \text{subject to} & \mathbf{B}\mathbf{f} \leq D\mathbf{1} \\ & \mathbf{f} \in \mathcal{P} \end{array}$$

where $\mathbf{f}$ is the vector representation of the all $f_e(v)$ variables, $\mathbf{B} \in \mathbb{R}^{n \times 2m}$ is the matrix denoting the left hand side of all constraints of the form $\sum_{e \ni v} f_e(v) \leq D$. $\mathbf{1}$ denotes the vector of 1's and $\mathcal{P}$ is a polyhedral constraint set defined as follows:

$$\begin{aligned} f_e(u) + f_e(v) &\geq 1 \qquad && \forall e = uv \in E \\ f_e(u) &\geq 0 && \forall e \in E,\ \forall v \in e. \end{aligned}$$

Note that for any $\mathbf{f} \in \mathcal{P}$, we have that the minimum $D$ satisfying $B\mathbf{f} \leq D\mathbf{1}$ is equal to $\|B\mathbf{f}\|_\infty$. This follows due to the non-negativity of $B\mathbf{f}$ for any $\mathbf{f} \in \mathcal{P}$. Now a simple observation shows that for any non-negative vector $\mathbf{y}$, we can write

$$\|\mathbf{y}\|_\infty = \max_{\mathbf{x} \in \Delta_n^+} \mathbf{x}^T \mathbf{y}$$

where $\Delta_n^+ := \{\mathbf{x} \geq \mathbf{0} : \mathbf{1}^T\mathbf{x} \leq 1\}$. Hence, we can now write $\textsc{Dual}(G)$ as:

$$\begin{aligned}\min_{\mathbf{f}\in\mathcal{P}} \|\mathbf{Bf}\|_\infty &= \min_{\mathbf{f}\in\mathcal{P}} \max_{\mathbf{x}\in\Delta_n^+} \mathbf{x}^T\mathbf{Bf} \\ &= \max_{\mathbf{x}\in\Delta_n^+} \min_{\mathbf{f}\in\mathcal{P}} \mathbf{x}^T\mathbf{Bf}. \end{aligned} \tag{3.1}$$

Here the last equality follows due to strong duality of the convex optimization.

The "inner" minimization part of (3.1) can be performed easily. In particular, we need an oracle which, given a vector $\mathbf{x}$, solves

$$C(\mathbf{x}) = \min_{\mathbf{f}\in\mathcal{P}} \sum_{e=uv} x_u f_e(u) + x_v f_e(v).$$

**Lemma 3.5.** *Given a vector* $\mathbf{x}$*,* $C(\mathbf{x})$ *can be computed in* $O(m)$ *time.*

*Proof.* For each edge $e = uv$, simply check which of $x_u$ and $x_v$ is smaller. WLOG, assume it is $x_u$. Then, set $f_e(u) = 1$ and $f_e(v) = 0$. $\square$

We denote the optimal $\mathbf{f}$ for a given $\mathbf{x}$ as $\mathbf{f}(\mathbf{x})$. Now, using the above oracle, we can apply the MWU algorithm to the "outer" problem of (3.1), i.e., $\max_{\mathbf{x}\in\Delta_n^+} C(\mathbf{x})$. Additionally, to apply the MWU framework, we need to estimate the width of this linear program. The width for (3.1) can be bounded by largest degree, $d_{\max}$ of the graph $G$. Indeed, we see in Lemma 3.5 that $\mathbf{f}(\mathbf{x})$ is a $0/1$ vector. In that case, $\|B\mathbf{f}(x)\|_\infty \leq d_{\max}$.

We conclude our analysis of this alternative dual formulation of the DSP with the following theorem.

**Theorem 3.6.** *Our alternative dual formulation admits a MWU algorithm that outputs an* $\mathbf{f} \in \mathcal{P}$ *such that* $\|\mathbf{Bf}\|_\infty \leq (1+\epsilon)\rho_G^*$.

For the sake of completeness, we detail the MWU algorithm and the proof of Theorem 3.6 in Appendix B.

Let us now view Charikar's peeling algorithm in the context of this dual problem. In a sense, the greedy peeling algorithm resembles one "inner" iteration of the MWU algorithm, where whenever a vertex is removed, its edges assign their load to it. Keeping this in mind, we designed GREEDY++ to add "outer" iterations to the peeling algorithm, thus improving the approximation factor arbitrarily with increase in iteration count. By weighting vertices using their load from previous iterations, GREEDY++ implicitly performs a form of load balancing on the graph, thus arriving at a better dual solution.

## 3.4 Experiments

### 3.4.1 Experimental setup

The experiments were performed on a single machine, with an Intel(R) Core(TM) i7-2600 CPU at 3.40GHz (4 cores), 8MB cache size, and 8GB of main memory. We find densest subgraphs on the samples using binary search and maximum flow computations. The flow computations were done using C++ implementations of the push-relabel algorithm [43], HiPR[1]. We have implemented our algorithm GREEDY++ and Charikar's greedy algorithm in C++. Our implementations are efficient and our code is available publicly[2].

We use a variety of datasets obtained from the Stanford's SNAP database [67], ASU's Social Computing Data Repository [101], BioGRID [89] and from the Koblenz Network Collection [65], that are shown in table Table 3.1. A majority of the datasets are from SNAP, and hence we mark only the rest with their sources. Multiple edges, self-loops are removed, and directionality is ignored for directed graphs. The first cluster of datasets are unweighted graphs. The largest unweighted graph is the *web-trackers* graph with roughly 141M edges, while the smallest unweighted graph has roughly 25K edges. For weighted graphs, we use a set of Twitter graphs that were crawled during the first week of February 2018 [88]. Finally, we use a set of signed networks (*slashdot*, *epinions*). We remind the

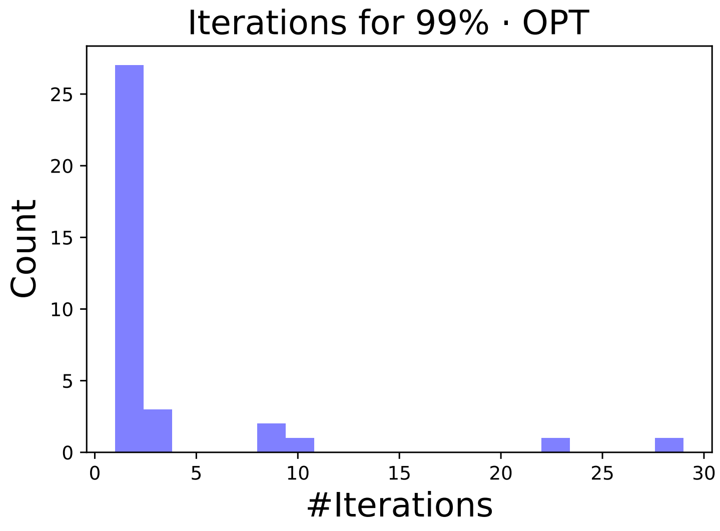
[1]HiPR is available at `http://www.avglab.com/andrew/soft/hipr.tar`

[2]Our code for GREEDY++ and the exact algorithm is available at the anonymous link `https://www.dropbox.com/s/jzouo9fjoytyqg3/code-greedy%2B%2B.zip?dl=0`

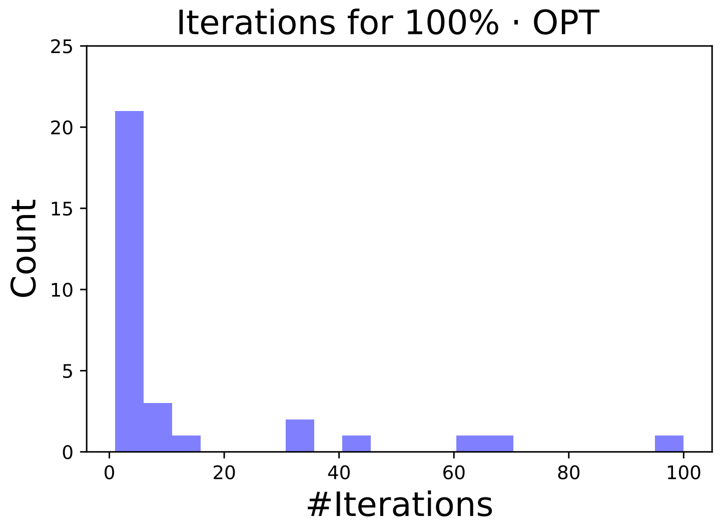
| Name | $n$ | $m$ |
|---|---|---|
| web-trackers [65] | 40 421 974 | 140 613 762 |
| orkut [65] | 3 072 441 | 117 184 899 |
| livejournal-affiliations [65] | 10 690 276 | 112 307 385 |
| wiki-topcats | 1 791 489 | 25 447 873 |
| cit-Patents | 3 774 768 | 16 518 948 |
| actor-collaborations [65] | 382 219 | 15 038 083 |
| ego-gplus | 107 614 | 12 238 285 |
| dblp-author | 5 425 963 | 8 649 016 |
| web-BerkStan | 685 230 | 6 649 470 |
| flickr [101] | 80 513 | 5 899 882 |
| wiki-Talk | 2 394 385 | 4 659 565 |
| web-Google | 875 713 | 4 322 051 |
| com-youtube | 1 134 890 | 2 987 624 |
| roadNet-CA | 1 965 206 | 2 766 607 |
| web-Stanford | 281 903 | 1 992 636 |
| roadNet-TX | 1 379 917 | 1 921 660 |
| roadNet-PA | 1 088 092 | 1 54 898 |
| Ego-twitter | 81 306 | 1 342 296 |
| com-dblp | 317 080 | 1 049 866 |
| com-Amazon | 334 863 | 925 872 |
| soc-slashdot0902 | 82 168 | 504 230 |
| soc-slashdot0811 | 77 360 | 469 180 |
| soc-Epinions | 75 879 | 405 740 |
| blogcatalog [101] | 10,312 | 333 983 |
| email-Enron | 36 692 | 183 831 |
| ego-facebook | 4 039 | 88 234 |
| ppi [89] | 3 890 | 37 845 |
| twitter-retweet [88] | 316 662 | 1 122 070 |
| twitter-favorite [88] | 226 516 | 1 210 041 |
| twitter-mention [88] | 571 157 | 1 895 094 |
| twitter-reply [88] | 196 697 | 296 194 |
| soc-sign-slashdot081106 | 77 350 | 468 554 |
| soc-sign-slashdot090216 | 81 867 | 497 672 |
| soc-sign-slashdot090221 | 82 140 | 500 481 |
| soc-sign-epinions | 131 828 | 711 210 |

Table 3.1: Datasets used in our experiments.

reader that while the DSP is NP-hard on signed graphs, Charikar's algorithm does provide certain theoretical guarantees, see Theorem 2 in [97].



(a)



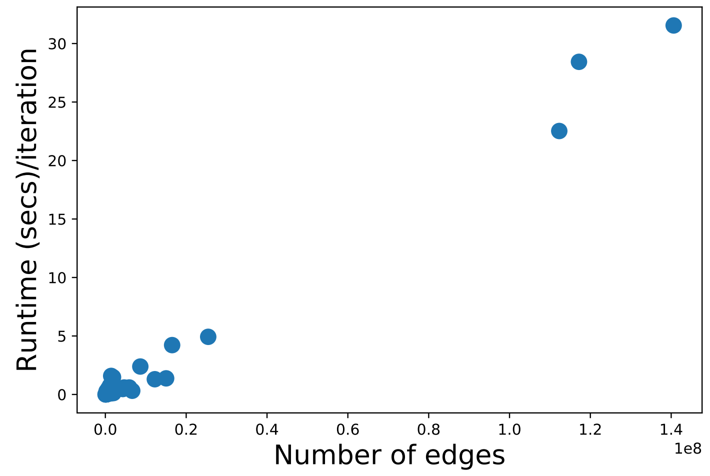
(b)

Figure 3.2: **Number of iterations for** GREEDY++. Histograms of number of iterations to reach (a) 99% of the optimum degree density, (b) the optimum degree density.

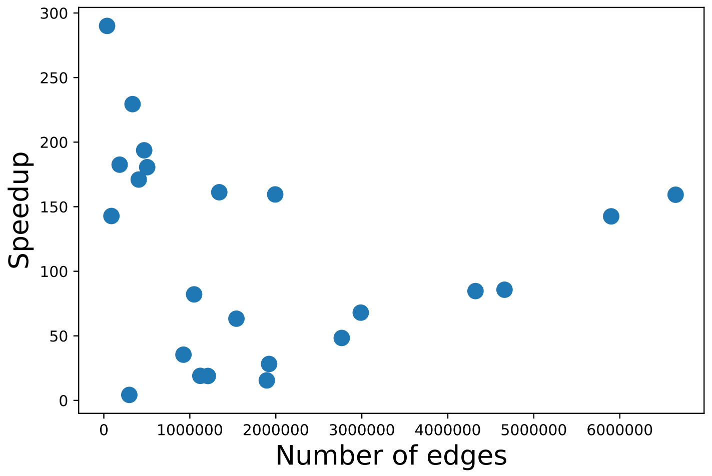
## 3.4.2 Experimental results

Before we delve in detail into our experimental findings, we summarize our key findings here:

- Our algorithm GREEDY++ when given enough number of iterations *always* finds the optimal value, and the densest subgraph. This agrees with our conjecture that running $T$ iterations of GREEDY++ gives a $1 + O(1/\sqrt{T})$ approximation to the DSP.



(a)



(b)

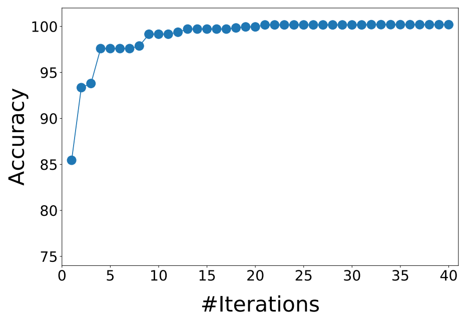
Figure 3.3: **Scalability.** (a) Running time in seconds of each iteration of GREEDY++ versus the number of edges. (b) Speedup achieved by GREEDY++ over the *exact max-flow based algorithm*, plotted vs. number of edges in the graph. Specifically, the $y$-axis is the ratio of the run time of the exact max flow algorithm divided by the run time of GREEDY++ that finds 90% of the optimal solution.

- Experimentally, Charikar's greedy algorithm always achieves at least 80% accuracy, and occasionally finds the optimum.

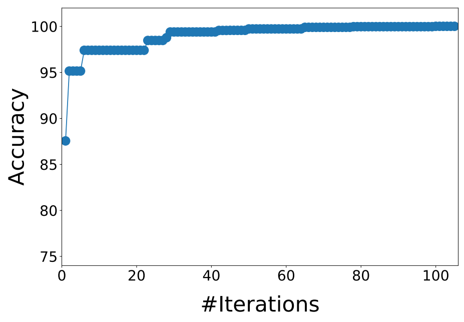
- For graphs on which the performance of Charikar's greedy algorithm is optimal, the first couple of iterations of GREEDY++ suffice to deduce convergence safely, and thus act in practice as a certificate of optimality. This is the first method to the best of our knowledge that can be used to infer quickly the actual approximation of Charikar's algorithm on a given graph instance.

- When Charikar's algorithm does not yield an optimal solution, then GREEDY++ within few iterations is able to increase the accuracy to 99% of the optimum density, and by adding a few more iterations is able to find the optimal density and extract and optimal output.
- When we are able to run the exact algorithm (for graphs with more than 8M edges, the maximum flow code crashes) on our machine, the average speedup that our algorithm provides to reach *the optimum* is $144.6\times$ on average, with a standard deviation equal to 57.4. The smallest speedup observed was $67.9\times$, and the largest speedup $290\times$. Additionally, we remark that the exact algorithm is only able to find solutions up to an accuracy of $10^{-3}$ on most graphs.
- The speedup typically increases as the size of the graph increases. In fact, the maximum flow exact algorithm cannot complete on the largest graphs we use.
- The maximum number of iterations needed to reach 90% of the optimum is at most 3, i.e., by running two more passes compared to Charikar's algorithm, we are able to boost the accuracy by 10%.
- The same remarks hold for both weighted and unweighted graphs.

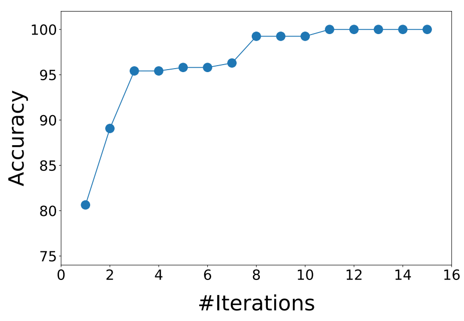
**Number of iterations.** We first measure how many iterations we need to reach 99% of the optimum, or even the optimum. Figures 3.2(a), (b) answer these questions respectively. We observe the impressive performance of Charikar's greedy algorithm; for the majority of the graph instances we observe that it finds a near-optimal densest subgraph. Nonetheless, even for those graph instances –as we have emphasized earlier– our algorithm GREEDY++ acts as a certificate of optimality. Namely, we observe that the objective remains the same



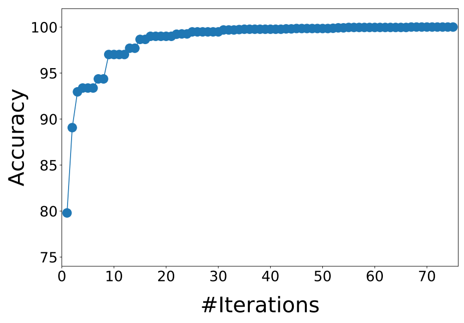
(a)



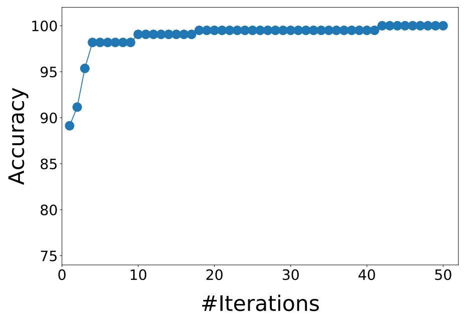
(b)



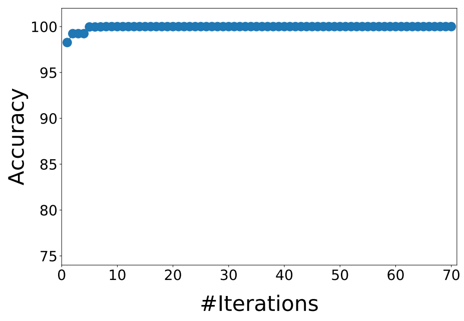
(c)



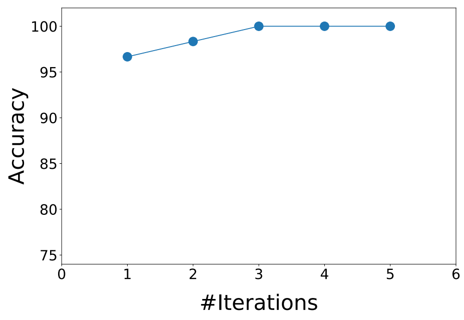
(d)



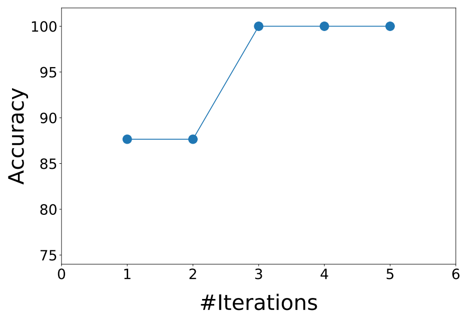
(e)



(f)



(g)



(h)

Figure 3.4: Convergence to optimum as a function of the number of iterations of GREEDY++. (a) roadNet-CA, (b) roadNet-PA, (c) roadNet-TX, (d) com-Amazon, (e) dblp-author, (f) ego-twitter, (g) twitter-favorite, (h) twitter-reply. Here, the *accuracy* is given by $\frac{\rho(H_i)}{\rho^*_G}$, where $H_i$ is the output of GREEDY++ after $i$ iterations.

after a couple of iterations if and only if the algorithm has reached the optimum. For the rest of the graphs where Charikar's greedy algorithm outputs an approximation greater than 80% but less than 99%, we observe the following: for five datasets it takes at most 3 iterations, for one graph it takes nine iterations, and then there exist three graphs for which GREEDY++ requires 10, 22, and 29 iterations respectively. If we insist on finding the optimum densest subgraph, we observe that the maximum number of iterations can go up to 100. On average, GREEDY++ requires 12.69 iterations to reach the optimum densest subgraph.

**Scalability.** Our experiments verify the intuitive facts that (i) each iteration of the greedy algorithm runs fast, and (ii) the exact algorithm that uses maximum flows is comparatively slow. We constrain ourselves on the set of data for which we were able to run the exact algorithm. Figure 3.3(a) shows the time that each iteration of the GREEDY++ takes on average (runtimes are well concentrated around the average) over the iterations performed to reach the optimal densest subgraph. Figure 3.3(b) shows the speedup achieved by our algorithm when we condition on obtaining *at least* 90% (notice that frequently the actual accuracy is greater than 95%) of the optimal solution versus the exact max-flow based algorithm. Specifically, we plot the ratio of the running times of the exact algorithm by the time of GREEDY++ versus the number of edges. Notice that for small graphs, the speedups are very large, then they drop, and they exhibit an increasing trend as the graph size grows. For the largest graphs in our collection, the exact algorithm is infeasible to run on our machine.

**Convergence.** Figure 3.4 illustrates the convergence of GREEDY++ for various datasets. Specifically, each figure plots the accuracy of GREEDY++ after $T$ iterations versus $T$. The accuracy is measured as the ratio of the degree density achieved by GREEDY++ by the optimal degree density. Figures 3.4(a),(b),(c),(d),(e),(f),(g),(h) correspond to the convergence behavior of roadNet-CA, roadNet-PA, roadNet-TX, com-Amazon, dblp-author, ego-twitter, twitter-favorite, twitter-reply respectively. These plots illustrate various interesting

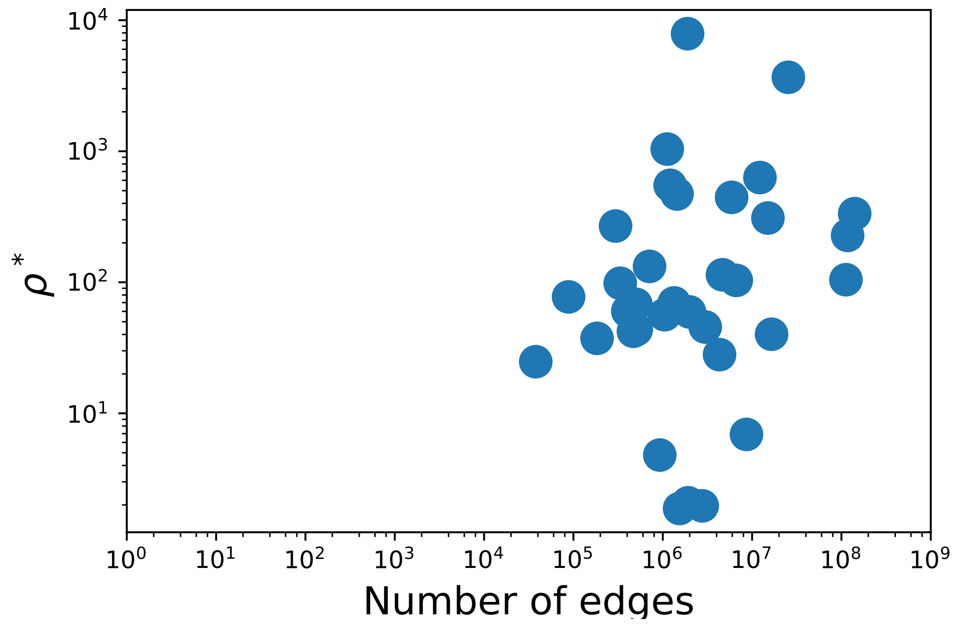


Figure 3.5: Log-log plot of optimal degree density $\rho^*$ versus the number of edges in the graph.

properties of GREEDY++ in practice. Observe Figure 3.4(e). Notice how GREEDY++ keeps outputting the same subgraph for few consecutive iterations, but then suddenly around the 10th iteration it "jumps" and finds an even denser subgraph. Recall that on average over our collection of datasets for which we can run the exact algorithm (i.e., datasets with less than 8M edges), GREEDY++ requires roughly 12 iterations to reach the optimum densest subgraph. For this reason we suggest running GREEDY++ for that many iterations in practice. Furthermore, we typically observe an improvement over the first pass, with the exception of the weighted graph twitter-reply, where the "jump" happens at the end of the third iteration.

**Anomaly detection.** It is worth outlining that GREEDY++ provides a way to compute the densest subgraph in graphs where the maximum flow approach does not scale. For example, for graphs with more than 8 million edges, the exact method does not run on our machine. By running GREEDY++ for enough iterations we can compute a near-optimal or the optimal solution. This allows us to compute a proxy of $\rho^*$ for the largest graphs, like orkut and trackers. We examined to what extent there exists a pattern between the size of the graph and the optimal density. In contrast to the power law relationship between the $k$-cores and the graph size claimed in [86], we do not observe a similar power law when we plot $\rho^*$ (the exact optimal value or the proxy value found by GREEDY++ after 100 iterations for the largest graphs) versus the number of edges in the graph. This is shown in Figure 3.5. Part of the reason why we do not observe such a law are anomalies in graphs. For instance, we observe that small graphs may contain extremely dense subgraphs, thus resulting in significant outliers.

## 3.5 Conclusion

In this chapter we provide a powerful algorithm for the *densest subgraph problem*, a popular and important objective for discovering dense components in graphs. The main practical value of our GREEDY++ algorithm is two-fold. First, by running few more iterations of Charikar's greedy algorithm we obtain (near-)optimal results that can be obtained using only maximum flows. Second, GREEDY++ can be used to answer for first time the question "Is the approximation of Charikar's algorithm on this graph instance closer to $1/2$ or to 1?" without computing the optimal density using maximum flows. Empirically, we have verified that GREEDY++ combines the best of "two worlds" on real data, i.e., the efficiency of the greedy peeling algorithm, and the accuracy of the exact maximum flow algorithm. We believe that GREEDY++ is a valuable addition to the algorithmic toolbox for dense subgraph discovery that combines the best of two worlds, i.e., the accuracy of maximum flows, and the time and space efficiency of Charikar's greedy algorithm.

An intriguing open question which remains from our work is a theoretical convergence proof for GREEDY++. In addition to the empirical evidence towards convergence, we believe that our algorithm mimics a linear programming solution given by the multiplicative-weight-update framework [9, 78]. Hence, we state this open problem as the following conjecture:

**Conjecture 3.7.** GREEDY++ is a $1 + O(1/\sqrt{T})$ approximation algorithm for the DSP, where $T$ is the number of iterations it performs.

# CHAPTER 4
# NEAR-OPTIMAL FULLY DYNAMIC DENSEST SUBGRAPH

In Chapters 2 and 3, we studied the densest subgraph problem as a static problem, in which we are required to compute a $(1-\varepsilon)$ approximate solution on a single unchanged graph. Motivated by the fact that most real-world networks undergo frequent changes, in this chapter we study the densest subgraph problem in the fully dynamic model of computation, wherein we are required to maintain a so.

## 4.1 Dynamic Graph Data Structures

A majority of real-world networks are very large in size, and a significant fraction of them are known to change rather rapidly [81]. This has necessitated the study of efficient dynamic graph algorithms - algorithms which use the existing solution to quickly find an updated solution for the new graph. Due to the size of these graphs, it is imperative that each update be processed in sub-linear time.

Formally, a graph data structure is said to be *fully dynamic* if it can support the following operations on a graph:

- insert an edge,
- delete an edge,
- query the desired optimum value or solution.

We call it a $(1-\varepsilon)$-approximate fully dynamic algorithm if we can query a $(1-\varepsilon)$ approximation to the optimum.

Data structures which efficiently maintain solutions to combinatorial optimization problems have shot into prominence over the last few decades [87, 35]. Many fundamental graph problems such as graph connectivity [52, 53, 58], maximal and maximum matchings [47, 17, 15, 19, 20], maximum flows and minimum cuts [55, 92, 44] have been shown to have efficient dynamic algorithms which only require sub-linear runtime per update. On the other hand, lower bounds exist for the update times for a number of these problems [4, 51, 1, 2, 3]. [50] contains a comprehensive survey of many graph problems and their state-of-the-art dynamic algorithms.

The state-of-the-art data structure for maintaining $(1+\varepsilon)$-approximate maximum matchings takes $O(\sqrt{m}\varepsilon^{-2})$ time per update [47]. Bhattacharya *et al.* [18] maintain a constant factor approximation to the $b$-matching problem in $O(\log^3 n)$ time. For flow-problems, algorithms which maintain a constant factor approximation in sublinear update time have proved to be elusive.

## 4.1.1 Related work

In terms of dynamic and streaming algorithms for the densest subgraph problem, the first result is by Bahmani *et al.* [12], where they modified Charikar's greedy algorithm to give a $(1/2-\varepsilon)$-approximation using $O(\log_{1+\varepsilon} n)$ passes over the input. Das Sarma *et al.* [82] adapted this idea to maintain a $(1/2-\varepsilon)$ approximate densest subgraph efficiently in the distributed CONGEST model. Using the same techniques as in the static case, Bahmani *et al.* [11] obtained a $(1-\varepsilon)$-approximation algorithm that requires $O(\log n\varepsilon^{-2})$ passes over the input.

Subsequently, Bhattacharya *et al.* [21] developed a more nuanced data structure to enable a 1-pass streaming algorithm which finds a $(1/2-\varepsilon)$ approximation. They also gave the first dynamic algorithm for DSP - a fully dynamic $(1/4-\varepsilon)$ approximation algorithm using amortized time $O(\text{poly}(\log n, \varepsilon^{-1}))$ per update. Around the same time, Epasto *et al.* [32] gave a fully dynamic $(1/2-\varepsilon)$-approximation algorithm for DSP in amortized time $O(\log^2 n\varepsilon^{-2})$ per update, with the caveat that edge deletions can only be random.

Kannan and Vinay [57] defined a notion of density on directed graphs, and subsequently gave a $O(\log n)$ approximation algorithm for the problem. Charikar [26] gave a polynomial-time algorithm for directed DSP by reducing the problem to solving $O(n^2)$ LPs. On the other hand, Khuller and Saha [60] used parametrized maximum flow to derive a polynomial-time algorithm. In the same paper, they gave a linear time 2-approximation algorithm for the problem.

An alternate approach towards a dynamic algorithm for the densest subgraph problem is to adapt the multiplicative weights update framework [9] used to solve the densest subgraph problem in [11] to allow for edge updates. This technique works in the incremental (only edge insertions) regime for bipartite matchings [46], and can similarly be adapted to work in the purely decremental case for the densest subgraph problem to give an $O(\log^3 n\varepsilon^{-3})$ amortized runtime per update.

### 4.1.2 Our results

As in previous chapters, we use the dual of the densest subgraph problem to gain insight on the optimality conditions, as in [26, 11]. Specifically, we translate it into a problem of assigning edge loads to incident vertices so as to minimize the maximum load across vertices. Viewed another way, we want to orient edges in a directed graph so as to minimize the maximum in-degree of the graph. This view gives a local condition for near-optimality of the algorithm, which we then leverage to design a data structure to handle updates efficiently. As the primary result in this chapter, we give the first fully dynamic $(1-\varepsilon)$-approximation algorithm for DSP which runs in $O(\text{poly}(\log n, \varepsilon^{-1}))$ worst-case time per update:

**Theorem 4.1.** *Given a graph $G$ with $n$ vertices, there exists a deterministic fully dynamic $(1-\varepsilon)$-approximation algorithm for the densest subgraph problem using $O(1)$ worst-case time per query and $O(\log^4 n \cdot \varepsilon^{-6})$ worst-case time per edge insertion or deletion.*

*Moreover, at any point, the algorithm can output the corresponding approximate densest subgraph in time $O(\beta + \log n)$, where $\beta$ is the number of vertices in the output.*

Charikar [26] gave a reduction from the densest subgraph problem on directed graphs to solving a number of instances of an LP. We visualize this LP as DSP on a vertex-weighted graph. We show that our approach on unweighted graphs extends naturally to those with vertex weights, thereby also giving a fully dynamic $(1-\varepsilon)$-approximation algorithm for directed DSP which runs in $O(\text{poly}(\log n, \varepsilon^{-1}))$ worst-case time per update:

**Theorem 4.2.** *Given a directed graph $G$ with $n$ vertices, there exists a deterministic fully dynamic $(1-\varepsilon)$-approximation algorithm for the densest subgraph problem on $G$ using $O(\log n \cdot \varepsilon^{-1})$ worst-case query time and worst-case update times of $O(\log^5 n \cdot \varepsilon^{-7})$ per edge insertion or deletion.*

*Moreover, at any point, the algorithm can output the corresponding approximate densest subgraph in time $O(\beta + \log n)$, where $\beta$ is the number of vertices in the output.*

## 4.2 Dynamic Algorithm for Densest Subgraph on Undirected Unweighted Graphs

In this section, we describe the main result of this chapter: a deterministic fully-dynamic algorithm which maintains a $(1-\varepsilon)$-approximation to the densest subgraph problem in $\text{poly}(\log n, \varepsilon^{-1})$ worst-case time per update.

### 4.2.1 Intuition and overview

At a high level, our approach is to view the densest subgraph problem via its dual problem, i.e., "assigning" each edge fractionally to its endpoints (as we discuss in Chapter 1). We view this as a load distribution problem, where each vertex is assigned some load from its incident edges. Then, the objective of the problem is simply to find an assignment such that the maximum vertex load is minimized. It is easy to verify that an optimal load assignment in the dual problem is achieved when no edge is able to reassign its load such that the maximum load among its two endpoints gets reduced. In other words, local optimality implies global optimality.

In fact, this property holds even for approximately optimal solutions. We show in Section 4.2.2 that any solution $\boldsymbol{f}$ which satisfies an $\eta$-additive approximation to local optimality guarantees an approximate global optimal solution with a multiplicative error of at most $1-O(\sqrt{\eta \log n/\hat{\rho}_G})$, where $\hat{\rho}_G$ denotes the maximum vertex load in $\boldsymbol{f}$. Here, an $\eta$-additive approximation implies that for any edge, the maximum among its endpoint loads can only be reduced by at most $\eta$ by reassigning the edge. So, given an estimate of $\hat{\rho}_G$ and a desired approximation factor $\varepsilon$, we can deduce the required slack parameter $\eta$, which we will alternatively denote as a function $\eta(\hat{\rho}_G, \varepsilon)$.

To do away with fractional edge assignments, in Section 4.2.3 we scale up the graph by duplicating each edge an appropriate number of times. When $\eta$ is an integer, one can always achieve an $\eta$-additive approximation to local optimality by assigning each edge completely to one of its endpoints. We visualize such a load assignment via a directed graph, by orienting each edge towards the vertex to which it is assigned. Now, the load on every vertex $v$ is simply its in-degree $\boldsymbol{d}_{\mathsf{in}}(v)$. Then, an $\eta$-approximate local optimal solution is achieved by orienting each edge such that there is no edge $\overrightarrow{uv}$ with $\boldsymbol{d}_{\mathsf{in}}(v) - \boldsymbol{d}_{\mathsf{in}}(u) > \eta$, because otherwise, we can flip the edge to achieve a better local solution. Let us call this a *locally $\eta$-stable oriented graph.*

This leaves the following challenges in extending this idea to a fully dynamic algorithm:

1. How can we maintain a *locally $\eta$-stable oriented graph* under insertion/deletion operations efficiently?

2. How do we maintain an accurate estimate of $\eta$ while the graph (and particularly $\hat{\rho}_G$) undergoes changes?

In Sections 4.2.4 and 4.2.5, we solve the first issue using a technique similar to that used by Kopelowitz *et al.* [61] for the graph orientation problem. When an edge is inserted or deleted, it causes a vertex to change its in-degree, which might cause an incident edge to break the invariant for local $\eta$-stability. If we flip the edge to fix this instability, it might cause further instabilities. To avoid this cascading of unstable edges, we first identify a maximal chain of "tight" edges - edges that are close to breaking the local stability constraint, and flip all edges in such a chain. This way, we only increment the degree of the last vertex in the chain. Since the chain was maximal, this increment maintains the stability condition. By defining a "tight" edge appropriately, and applying the same argument to the deletion operation, we show that each update incurs at most $O(\hat{\rho}_G/\eta)$ flips. This chain of tight edges closely relates to the concept of augmenting paths in network flows [34] and matchings [72, 31], which seems fitting, considering our intuition that densest subgraph relates closely to these problems.

In Section 4.2.6, we solve the second issue - by simply running the algorithm for $O(\log n)$ values of $\eta$, and using the appropriate version of the algorithm to query the solution.

### 4.2.2 Sufficiency of local approximation

From strong duality, we know that the optimal solution to $\text{DUAL}(G)$ gives the exact maximum subgraph density of $G$, $\rho^*_G$. Let us interpret the variables of $\text{DUAL}(G)$ as follows:

- Every edge $e = uv$ assigns itself fractionally to one of its two endpoints. $f_e(u)$ and $f_e(v)$ denote these fractional loads.
- $\sum_{e \ni v} f_e(v)$ is the total load assigned to $v$. We denote this using $\ell_v$.
- The objective is simply $\max_{v \in V} \ell_v$.

If there is any edge $e = uv$ such that $f_e(u) > 0$ and $\ell_u > \ell_v$. Then $e$ can transfer an infinitesimal amount of load from $u$ to $v$ while not increasing the objective. Hence, there always exists an optimal solution where for any edge $e = uv$, $f_e(u) > 0 \implies \ell_u \leq \ell_v$. Using this intuition, we write the approximate version of $\text{DUAL}(G)$ by providing a slack of $\eta$ to the above condition. We call this relaxed LP as $\text{DUAL}(G, \eta)$.

$$\begin{aligned}
&\textsc{Dual}(G,\eta)\\
\ell_v &= \sum_{e \ni v} f_e(v) && \forall u \in V\\
f_e(u) + f_e(v) &= 1, && \forall e = uv \in E\\
f_e(u), f_e(v) &\geq 0, && \forall e = uv \in E\\
\ell_u &\leq \ell_v + \eta, && \forall e = uv \in E, f_e(u) > 0
\end{aligned}$$

Theorem 4.3 states that this local condition is, in fact, also sufficient to achieve global near-optimality. Specifically, it shows that $\textsc{Dual}(G,\eta)$ provides a $\frac{1}{(1-\varepsilon)}$-approximation to $\rho^*_G$, where $\eta$ is a parameter depending on $\varepsilon$ described later. Kopelowitz *et al.* [61] use an identical argument to show the sufficiency of local optimality for the graph orientation problem.

**Theorem 4.3.** *Given an undirected graph $G$ with $n$ vertices, let $\hat{\boldsymbol{f}}, \hat{\boldsymbol{\ell}}$ denote any feasible solution to* $\textsc{Dual}(G,\eta)$*, and let $\hat{\rho}_G \overset{\text{def}}{=} \max_{v \in V} \hat{\ell}_v$. Then,*

$$\left(1 - 3\sqrt{\frac{\eta \log n}{\hat{\rho}_G}}\right) \cdot \hat{\rho}_G \leq \rho^*_G \leq \hat{\rho}_G.$$

*Proof.* Any feasible solution of $\textsc{Dual}(G,\eta)$ is also a feasible solution of $\textsc{Dual}(G)$, and so we have $\rho^*_G \leq \hat{\rho}_G$.

Denote by $T_i$ the set of vertices with load at least $\hat{\rho}_G - \eta i$, i.e., $T_i \overset{\text{def}}{=} \left\{ v \in V \mid \hat{\ell}_v \geq \hat{\rho}_G - \eta i \right\}$. Let $0 < r < 1$ be some adjustable parameter we will fix later. We define $k$ to be the maximal integer such that for any $1 \leq i \leq k$, $|T_i| \geq |T_{i-1}|(1+r)$. Note that such a maximal integer $k$ always exists because there are finite number of vertices in $G$ and the size of $T_i$ grows exponentially. By the maximality of $k$, $|T_{k+1}| < |T_k|(1+r)$. In order to bound the density of this set $T_{k+1}$, we compute the total load on all vertices in

$T_k$. For any $u \in T_k$, the load on $u$ is given by

$$\hat{\ell}_u = \sum_{uv \in E} \hat{f}_{uv}(u).$$

However, we know that $f_{uv}(u) > 0 \implies \hat{\ell}_v \geq \hat{\ell}_u - \eta$, and hence we only need to count for $v \in T_{k+1}$. Summing over all vertices in $T_{k+1}$, we get

$$\sum_{u \in T_k} \hat{\ell}_u = \sum_{u \in T_k, v \in T_{k+1}} \hat{f}_{uv}(u) \leq \sum_{u \in T_{k+1}, v \in T_{k+1}} \hat{f}_{uv}(u) = |E(T_{k+1})|.$$

Consider the density of set $T_{k+1}$,

$$\rho_G(T_{k+1}) = \frac{|E(T_{k+1})|}{|T_{k+1}|} \geq \frac{\sum_{u \in T_k} \hat{\ell}_u}{|T_{k+1}|} \geq \frac{|T_k| \cdot (\hat{\rho}_G - \eta k)}{|T_{k+1}|},$$

where the last inequality follows from the definition of $T_k$.

Using the fact that $|T_k|/|T_{k+1}| > 1/(1+r) \geq 1 - r$,

$$\rho_G(T_{k+1}) \geq (1-r)(\hat{\rho}_G - \eta k) \geq \hat{\rho}_G (1-r) \left(1 - \frac{2\eta \log n}{r \cdot \hat{\rho}_G}\right),$$

where the last inequality comes from the fact that $n \geq |T_k| \geq (1+r)^k$, which implies that $k \leq \log_{1+r} n \leq 2 \log n / r$.

Now, we can set our parameter $r$ to maximize the term on the RHS. By symmetry, the maximum is achieved when both terms in the product are equal and hence we set

$$r = \sqrt{\frac{2\eta \log n}{\hat{\rho}_G}}.$$

This gives

$$\rho_G(T_{k+1}) \geq \hat{\rho}_G \cdot \left(1 - \sqrt{\frac{2\eta \log n}{\hat{\rho}_G}}\right)^2 \geq \hat{\rho}_G \cdot \left(1 - 2\sqrt{\frac{2\eta \log n}{\hat{\rho}_G}}\right) \geq \hat{\rho}_G \cdot \left(1 - 3\sqrt{\frac{\eta \log n}{\hat{\rho}_G}}\right).$$

Lastly, since $\rho_G(T_{k+1})$ can be at most the maximum subgraph density $\rho^*_G$, the theorem follows. $\square$

The set $T_{k+1}$, in the above proof, is actually a subgraph of $G$ with density at least $\rho^*_G(1 - 3\sqrt{\eta \log n/\hat{\rho}_G})$. However, we need the exact value of $\hat{\rho}_G$ to find this set. As we will see in later sections, we will only have access to an estimate $\rho^{\mathsf{est}}$ of the form: $\rho^{\mathsf{est}} \leq \hat{\rho}_G \leq 2\rho^{\mathsf{est}}$. So, if we instead set

$$r = \sqrt{\frac{2\eta \log n}{\rho^{\mathsf{est}}}}, \tag{4.1}$$

we get

$$\rho_G(T_{k+1}) \geq \hat{\rho}_G \cdot \left(1 - \sqrt{\frac{2\eta \log n}{\hat{\rho}_G}}\right)\left(1 - 2\sqrt{\frac{\eta \log n}{\hat{\rho}_G}}\right) \geq \hat{\rho}_G \cdot \left(1 - 4\sqrt{\frac{\eta \log n}{\hat{\rho}_G}}\right).$$

Using the fact that $\hat{\rho}_G \geq \rho^*_G, \rho^{\mathsf{est}}$ gives us the following corollary.

**Corollary 4.4.**

$$\rho_G(T_{k+1}) \geq \rho^*_G \cdot \left(1 - 4\sqrt{\frac{\eta \log n}{\rho^{\mathsf{est}}}}\right),$$

*where $T_{k+1}$ is as defined in the proof of Theorem 4.3, using the value of $r$ as defined in* (4.1).

We can now set $\eta$ corresponding to the desired error $\varepsilon$ and the estimate $\rho^{\mathsf{est}}$.

### 4.2.3 Equivalence to the graph orientation problem

To obtain a $1 - \varepsilon$ approximation, we need to set $\eta = \dfrac{\varepsilon^2 \rho^{\mathsf{est}}}{16 \log n}$. For simpler analysis and to avoid working with fractional loads, we duplicate each edge $\alpha \overset{\text{def}}{=} \dfrac{64 \log n}{\varepsilon^2}$ times. By doing this, we ensure that $\rho^{\mathsf{est}} \geq \hat{\rho}_G/2 \geq \rho^*_G/2 \geq \alpha/4$, and thus, $\eta \geq 1$. This means we can do away with fractional assignments of edges and so each edge $u, v$ is now assigned to either $u$ or $v$. We can now frame the question as follows:

Given an undirected graph $G$ and an integer $\eta$, we want to assign directions to edges in such a way that for any edge $\overrightarrow{uv}$,

$$\boldsymbol{d}_{\mathsf{in}}(v) \leq \boldsymbol{d}_{\mathsf{in}}(u) + \eta.$$

The above graph orientation problem, i.e., dynamically orienting edges of a graph to minimize the maximum in-degree, is well studied [25, 62, 61]. Kopelowitz *et al.* give an efficient dynamic algorithm for the problem, where the update time depends on the arboricity[1] of the graph with worst-case time bounds. Our technique for inserting and deleting edges mimics the algorithm by Kopelowitz *et al.* [61]. However, for our problem, the slack parameter $\eta$ grows linearly with the maximum vertex load. Hence, we can exploit this additional power to arrive at worst-case times independent of any measure of actual density in the graph. Additionally, to bound the cost of a vertex informing its updated degree to its neighbors, we use a lazy round-robin informing technique, in which not all neighbors are always informed of the latest updates. We expand on these details in the rest of the section.

### 4.2.4 Data structure for edge flipping in directed graphs

At the lowest level, we want to build a data structure that maintains a directed graph undergoing changes. Ideally, we want each vertex to know its neighbors' labels, so that we can quickly find any edge violating or exactly satisfying the approximation condition. We refer to the latter as a *tight* edge. However, this property is expensive because each vertex could possibly have too many neighbors to inform. Specifically, each vertex could have up to $\hat{\rho}_G$ in-neighbors and as many as $n-1$ out-neighbors.

We deal with this issue in the following way. Since a vertex can have $\Omega(n)$ out-neighbors, it does not inform its changes to its out-neighbors, but only its in-neighbors.

[1]Arboricity is an alternate measure of density defined as $\alpha_G(V) = |E(V)|/(|V|-1)$, and is within $O(1)$ of our density measure.

So, any vertex remembers the labels of its out-neighbors. Hence, it is easy to find a tight outgoing edge; however, to find a tight incoming edge, we need to query the labels of all its in-neighbors. Hence, both the update subroutines and finding a tight incoming edge - use as many as $\hat{\rho}_G$ operations.

However, $\hat{\rho}_G$ can also get prohibitively large when the graph sees many insertions, and can reach $\Omega(n)$ (e.g. in a clique). To tackle this, we relax the requirement for tightness of an edge: we say that an edge $\overrightarrow{uv}$ is tight if $\boldsymbol{d}_{\mathsf{in}}(v) \geq \boldsymbol{d}_{\mathsf{in}}(u) + \eta/2$. Now, finding a tight edge becomes less strict - importantly it now suffices to update one's in-neighbors (or query one's in-neighbors) once every $\eta/4$ iterations. So, in each update, a vertex $v$ only informs $4\boldsymbol{d}_{\mathsf{in}}(v)/\eta$ of its neighbors in round-robin fashion. This reduces the number of operations to $O(\alpha)$ per update, as desired.

**Lemma 4.5.** *There exists a data structure* LAZYDIRECTEDLABELS$(G, \eta)$ *which can maintain a directed graph $G(V, E)$, appended with vertex labels $d : V \mapsto \mathbb{Z}^+$ while undergoing the following operations:*

- `add` $(\overrightarrow{uv})$*: add an edge into $G$,*
- `remove` $(\overrightarrow{uv})$*: remove an edge from $G$,*
- `increment`$(u)$*: increment $d(u)$ by $1$,*
- `decrement`$(u)$*: decrement $d(u)$ by $1$,*
- `flip` $(\overrightarrow{uv})$*: flip the direction of an edge in $G$,*
- `tight_in_nbr`$(u)$*: find an in-neighbor $v$ with $d(v) \leq d(u) - \eta/2$, and*
- `tight_out_nbr`$(u)$*: find an out-neighbor $v$ with $d(v) \geq d(u) + \eta/2$.*
- `label`$(u)$*: output $d(u)$.*
- `max_label`$()$*: output $\max_{v \in V} d(v)$.*
- `maximal_label_set`$(r)$*: Output all elements with labels $\geq$ `max_label()` $- \eta \cdot i$, where $i$ is the smallest integer such that $|$labels $\geq \eta \cdot (i+1)| < (1+r)|$labels $\geq \eta \cdot i|$.*

We maintain the following global data structure:

- LABELS: Balanced binary search tree with all labels. We store the max element separately.

Each vertex $u$ maintains the following data structures:

- $d(u)$: $u$'s label, initialized to $0$.
- $\text{INNBRS}_u$: List of $u$'s in-neighbors, initialized to $\emptyset$.
- $\text{OUTNBRS}_u$: Max-priority queue of $u$'s out-neighbors indexed using $d_u$, initialized to $\emptyset$.

```
Operation add($\overrightarrow{uv}$)
    Add $u$ to $\text{INNBRS}_v$
    Add $v$ to $\text{OUTNBRS}_u$ with key $d_u(v) \leftarrow d(v)$

Operation remove($\overrightarrow{uv}$)
    Remove $u$ from $\text{INNBRS}_v$
    Remove $v$ from $\text{OUTNBRS}_u$

Operation flip($\overrightarrow{uv}$)
    remove($\overrightarrow{uv}$)
    add($\overrightarrow{vu}$)

Operation increment($u$)
    $d(u) \leftarrow d(u) + 1$
    Update $d(u)$ in LABELS
    for $v \in$ the next $\frac{4d_{in}(u)}{\eta}$ $\text{INNBRS}_u$ do
        $d_v(u) \leftarrow d(u)$ in $\text{OUTNBRS}_v$

Operation decrement($u$)
    $d(u) \leftarrow d(u) - 1$
    $d(u)$ in LABELS
    for $v \in$ the next $\frac{4d_{in}(u)}{\eta}$ $\text{INNBRS}_u$ do
        $d_v(u) \leftarrow d(u)$ in $\text{OUTNBRS}_v$

Operation tight_in_nbr($u$)
    for $v \in$ the next $\frac{4d_{in}(u)}{\eta}$ $\text{INNBRS}_u$ do
        if $d(v) \leq d(u) - \eta/2$ then
            return $v$
    return null

Operation tight_out_nbr($u$)
    $t \leftarrow \text{OUTNBRS}[u]$.max
    if $d_u(v) \geq d(u) + \eta/2$ then
        return $v$
    else return null

Operation label()
    return $d(u)$

Operation max_label()
    return LABELS.max

Operation maximal_label_set($r$)
    $m \leftarrow$ max_label()
    do
        $A \leftarrow$ all elements $\geq m - \eta$ in LABELS
        $B \leftarrow$ all elements $\geq m - 2\eta$ in LABELS
        $m \leftarrow m - \eta$
    while $|B|/|A| \geq 1 + r$
    return $B$
```

**Algorithm 5:** LAZYDIRECTEDLABELS$(G, \eta)$: A data structure to maintain a directed graph with vertex labels. $V$ and $\eta$ are known.

*Moreover, the operations* add, remove *and* flip *can be processed in* $O(\log n)$ *time;* tight_in_nbr, increment *and* decrement *can be processed in* $O(\alpha)$ *time; and* tight_out_nbr *and* max_label *can be processed in* $O(1)$ *time.* maximal_label_set *can be processed in time in the order of the output size.*

*The pseudocode for this data structure is in Algorithm 5.*

*Proof.* The correctness of the data structure follows from the description in Algorithm 5. The operation add involves inserting an element into a list and a priority queue - giving a worst-case runtime of $O(\log n)$. The runtimes for remove and flip follow similarly. The operations increment and decrement involve 1 update to a balanced BST and $O(\alpha)$ priority-queue updates, giving a worst-case runtime of $O(\alpha \log n)$ per call. tight_in_nbr queries $O(\alpha)$ neighbors, resulting in a worst-case runtime of $O(\alpha)$ per call. tight_out_nbr, label and max_label simply check an element pointer, resulting in a $O(1)$ runtime. Lastly, maximal_label_set traverses a balanced BST, until it exceeds the desired threshold. The time taken is $O(\beta + \log n)$ where $\beta$ is the number of elements read, which is also the size of the output. $\square$

### 4.2.5 Fully dynamic algorithm for a given density estimate

Here, we assume that an estimate of $\rho^*_G$ (equivalently, an estimate of $\hat{\rho}_G$) is known. We denote this estimate as $\rho^{\text{est}}$, where $\rho^{\text{est}} \leq \hat{\rho}_G \leq 2\rho^{\text{est}}$. Using this, we can compute the appropriate $\eta(\rho^{\text{est}}, \varepsilon) \overset{\text{def}}{=} 2\rho^{\text{est}}/\alpha$. Recall that $\alpha$ was defined as $\alpha \overset{\text{def}}{=} 64 \log n \cdot \varepsilon^{-2}$. From Section 4.2.4, we have an efficient data structure to maintain a directed graph, which we will use to maintain a locally $\eta$-stable orientation. This in turn gives a fully dynamic algorithm which processes updates efficiently, as we explain below.

We first define a *tight edge* in a locally stable oriented graph:

**Definition 4.6.** An edge $\overrightarrow{uv}$ is said to be *tight* if $\boldsymbol{d}_{\text{in}}(v) \geq \boldsymbol{d}_{\text{in}}(u) + \eta/2$.

Now, consider inserting an edge $\overrightarrow{xy}$ into a locally $\eta$-stable oriented graph. Since $y$'s in-degree increases, it could potentially have an in-neighbor $z$ such that $\boldsymbol{d}_{\text{in}}(z) < \boldsymbol{d}_{\text{in}}(y) - \eta$.

Note that for this to happen, $\overrightarrow{zy}$ was necessarily a tight edge. To "fix" this break in stability, we flip the edge $yz$; however, this causes $w$'s in-degree to increase, which we now possibly need to fix. Before explaining how we circumvent this issue, let us define a *maximal tight chain*.

**Definition 4.7.** A maximal tight chain *from* a vertex $v$ is a path of tight edges $\overrightarrow{uv_1}, \overrightarrow{v_1v_2}, \ldots, \overrightarrow{v_kw}$, such that $w$ has no tight outgoing edges.

A maximal tight chain *to* a vertex $v$ is a path of tight edges $\overrightarrow{wv_1}, \overrightarrow{v_1v_2}, \ldots, \overrightarrow{v_kv}$, such that $w$ has no tight incoming edges.

Now, instead of fixing the "unstable" edge caused by the increase in $y$'s in-degree right away, we instead find a maximal tight chain *to* $y$ and flip all the edges in the chain. This way, the in-degrees of all vertices in the chain except the start remain the same. Due to the maximality of the chain, the start of the chain has no incoming tight edges, and hence increasing its in-degree by $1$ will not break local stability. The same argument holds when we delete $\overrightarrow{xy}$, except we find a maximal tight chain *from* $y$.

The approximate density is nothing but the highest load in the graph. For querying the actual subgraph itself, we use the observation from Section 4.2.2, where the required subgraph can be found by: (i) finding sets of vertices with load at most $\eta \cdot i$ less than the maximum ($T_i$), and (ii) returning the first $T_{i+1}$ such that $|T_{i+1}|/|T_i| < 1 + r$, where $r$ is an appropriate function of $\eta$.

**Lemma 4.8.** *There exists a data structure* THRESHOLD$(G, \eta)$ *which can maintain an undirected graph* $G(V, E)$ *while undergoing the following operations:*

- `insert`$(u, v)$*: insert an edge into* $G$*,*
- `delete`$(u, v)$*: delete an edge from* $G$*, and report the vertex with decreased load*
- `query_load`$(u)$*: output the current load of* $u$*.*
- `query_density`$()$*: output a value* $\rho_{\text{out}}$ *such that* $(1 - \varepsilon)\rho^*_G \leq \rho_{\text{out}} \leq \rho^*_G$*.*

- query_subgraph()*: output a subgraph with density at least* $(1-\varepsilon)\rho^*_G$*.*

*Moreover, the operation* insert *takes* $O(\alpha^2)$ *time,* delete *takes* $O(\alpha \log n)$ *time,* query *takes* $O(1)$ *time, and* query_subgraph *takes* $O(\beta + \log n)$ *time, where* $\beta$ *is the size of the output.*

*The pseudocode for this data structure is in Algorithm 6.*

```
• Initialize data structure $\mathcal{L} \leftarrow$ LazyDirectedLabels$(G, \eta)$ with:
  $G = (V, \emptyset)$, $\alpha \leftarrow 64 \log n \cdot \varepsilon^{-2}$, $\eta \leftarrow 2\rho^{\mathsf{est}}/\alpha$

Operation insert ($(u, v)$)
    if $d(u) \geq d(v)$ then
        $\mathcal{L}$.add ($\overrightarrow{uv}$)
        $w \leftarrow v$
    else
        $\mathcal{L}$.add ($\overrightarrow{vu}$)
        $w \leftarrow u$
    while $\mathcal{L}$.tight_in_nbr($w$) $\neq$ null do
        $w' \leftarrow \mathcal{L}$.tight_in_nbr($w$)
        $\mathcal{L}$.flip $\left(\overrightarrow{w'w}\right)$
        $w \leftarrow w'$
    $\mathcal{L}$.increment($w$)

Operation delete ($(u, v)$)
    if $u \in$ InNbrs$_v$ then
        $\mathcal{L}$.remove ($\overrightarrow{uv}$)
        $w \leftarrow v$
    else
        $\mathcal{L}$.remove ($\overrightarrow{vu}$)
        $w \leftarrow u$
    while $\mathcal{L}$.tight_out_nbr($w$) $\neq$ null do
        $w' \leftarrow \mathcal{L}$.tight_out_nbr($w$)
        $\mathcal{L}$.flip $\left(\overrightarrow{ww'}\right)$
        $w \leftarrow w'$
    $\mathcal{L}$.decrement($w$)
    return($w$)

Operation query_subgraph ()
    $r \leftarrow \sqrt{2\eta \log n/\rho^{\mathsf{est}}}$
    return $\mathcal{L}$.maximal_label_set($r$);

Operation query_density ()
    return $\mathcal{L}$.max_label $\times (1-\varepsilon)$;

Operation query_load ($u$)
    return $\mathcal{L}$.label($u$);
```

**Algorithm 6:** Threshold$(G, \rho^{\mathsf{est}}, \varepsilon)$: Update routines on $G$ when an estimate to its maximum load is known. Additionally $V$, $n = |V|$, and $\varepsilon$ are known.

Let us denote by $d_u(v)$, the apparent label of $v$ as seen by $u$. This concept is needed because when the label of a vertex changes, it doesn't relay this change to all its in-neighbors immediately. However, we can claim the following:

**Lemma 4.9.** *The local gap constraint is always maintained, i.e., for any edge $\overrightarrow{uv}$, $d(v) \leq d(u) + \eta$.*

*Proof.* There are two ways that this invariant could become unsatisfied: via a decrement to $u$ or an increment to $v$.

Recall that $v$ informs each in-neighbor its label once every $\eta/4$ updates, hence $|d_u(v) - d(v)|$ cannot be larger than $\eta/4$. $u$ only decrements if it cannot find a tight out-neighbor, which means that $d_u(v) < d(u) + \eta/2$. Hence, at any instant that $d(u)$ is decremented, $d(v) \leq d(u) + 3\eta/4$.

On the other hand, $d(v)$ is only incremented if $v$ cannot find a tight in-neighbor. Consider the last time that $d(u)$ is decremented before this instant. At this point, $d(v) \leq d(u) + 3\eta/4$. After this, there can only be less than $\eta/4$ increments of $d(v)$ before it queries $d(u)$ and flips. Hence, $d(v) \leq d(u) + \eta$. □

Using Lemma 4.9 and Corollary 4.4, we get the following corollary, which shows the correctness of Lemma 4.8.

**Corollary 4.10.** *Let $\rho_{\mathsf{out}} = (1 - \varepsilon) \max_{v \in V} d(v)$. Then, $(1 - \varepsilon)\rho^*_G \leq \rho_{\mathsf{out}} \leq \rho^*_G$.*

*Proof of Lemma 4.8.* Corollary 4.10 gives the correctness proof. It remains to show the time bounds. Note that in both `insert` and `delete` operations, the maximum chain of tight edges can only be of length at most $2\hat{\rho}_G/\eta = O(\alpha)$. The `insert` operation calls `add` and `increment` once, `flip` and `tight_in_nbr` $O(\alpha)$ times. From Lemma 4.5, this results in a worst-case runtime of $O(\alpha^2)$ per insertion. The `delete` operation calls `remove` and `decrement` once, `flip` and `tight_out_nbr` $O(\alpha)$ times. From Lemma 4.5, this results in a worst-case runtime of $O(\alpha \cdot \log n)$ per deletion. `query_density` only needs one `max_label` call which is $O(1)$ worst-case. `query_load` also needs one `label` call which is $O(1)$ worst-case. Lastly, `query_subgraph`'s runtime follows from Lemma 4.5. □

## 4.2.6 Overall algorithm

Now, we have a sufficient basis to show our main theorem, which we restate:

**Theorem 4.1.** *Given a graph $G$ with $n$ vertices, there exists a deterministic fully dynamic $(1-\varepsilon)$-approximation algorithm for the densest subgraph problem using $O(1)$ worst-case time per query and $O(\log^4 n \cdot \varepsilon^{-6})$ worst-case time per edge insertion or deletion.*

*Moreover, at any point, the algorithm can output the corresponding approximate densest subgraph in time $O(\beta + \log n)$, where $\beta$ is the number of vertices in the output.*

From Section 4.2.5, we now have an efficient fully dynamic data structure $\textsc{Threshold}(G, \rho^{\mathsf{est}}, \varepsilon)$ to maintain a $1-\varepsilon$ approximation to the maximum subgraph density, provided the optimum remains within a constant factor of some estimate $\rho^{\mathsf{est}}$. Particularly, $\textsc{Threshold}(G, \rho^{\mathsf{est}}, \varepsilon)$ requires $\hat{\rho}_G/\eta$ to be small to work efficiently. On the other hand, too small an $\eta$ results in a bad approximation factor.

To ensure that we always work with the right estimate $\rho^{\mathsf{est}}$, we will construct $\log_2 n$ copies of $\textsc{Threshold}$, one copy for each possible $\rho^{\mathsf{est}}$, or equivalently each possible value of $\eta$. In the $i$th copy of the data structure, we set $\rho_i^{\mathsf{est}} \leftarrow 2^{i-2}\alpha$, and so $\eta_i \leftarrow 2^{i-1}$. Let us call this $i$th copy of the data structure as $\mathcal{T}_i \leftarrow \textsc{Threshold}(G, \rho_i^{\mathsf{est}}, \varepsilon)$. We also define $\eta_0 = 0$ for the sake of the empty graph.

We say that $\mathcal{T}_i$ is *accurate* if $\rho_i^{\mathsf{est}} \leq \hat{\rho}_G$, or equivalently $\eta_i \leq 2\hat{\rho}_G/\alpha$. Note that we will never use a copy that is not accurate to deduce the approximate solution. On the other hand, we say that $\mathcal{T}_i$ is *affordable* if the maximum possible chain length is less than $2\alpha$, i.e., $\eta_i > \hat{\rho}_G/\alpha$. On copies that are not affordable, if there are any additions which can cause the maximum load *in that copy* to increase, we hold these off until a later time.

Lastly, note that for any value of $\hat{\rho}_G$, there is exactly one copy which is both accurate and affordable. We call this the *active* copy. The solution is extracted at any point from this copy. Suppose the index of the current active copy is $i$. Then, after an insertion, this can be either $i$ or $i+1$. We first test this by querying the maximum density in $\mathcal{T}_{i+1}$, and

```
• for $i \leftarrow 1$ to $\log_2 n$ do
    $\alpha \leftarrow 64 \log n \cdot \varepsilon^{-2}$; $\rho_i^{\mathsf{est}} \leftarrow 2^{i-2}\alpha$
    Initialize $\mathcal{T}_i \leftarrow \textsc{Threshold}(G, \rho_i^{\mathsf{est}}, \varepsilon)$
    Initialize a sorted list of edges $\mathtt{pending}_i \leftarrow \emptyset$ using two balanced BSTs (one
      sorted using the first vertex of the edge, and another using the second)
    Set $\mathtt{active} \leftarrow 0$

Operation query()
    return $\mathcal{T}_{\mathtt{active}}.\mathtt{query}()$

Operation query_subgraph()
    return $\mathcal{T}_{\mathtt{active}}.\mathtt{query\_subgraph}()$

Operation insert($(u, v)$)
    for $k \leftarrow 1$ to $\alpha$ do                          // duplicating $(u,v)$ $\alpha$ times
        for $i \leftarrow \log_2 n$ to $\mathtt{active}+1$ do $\mathcal{T}_i.\mathtt{insert}((u,v))$    // affordable
          copies
        $\rho \leftarrow \mathcal{T}_{\mathtt{active}+1}.\mathtt{query}()$
        if $\rho \geq 2\rho^{est}_{\mathtt{active}}$ then $\mathtt{active} \leftarrow \mathtt{active}+1$
        else $\mathcal{T}_{\mathtt{active}}.\mathtt{insert}((u,v))$
        for $i \leftarrow \mathtt{active}-1$ to $1$ do                 // unaffordable copies
            $\ell_u \leftarrow \mathcal{T}_i.\mathtt{query\_load}(u)$; $\ell_v \leftarrow \mathcal{T}_i.\mathtt{query\_load}(v)$
            if both $\ell_u, \ell_v \geq 2\rho^{est}$ then add $(u,v)$ to $\mathtt{pending}_i$
            else $\mathcal{T}_i.\mathtt{insert}((u,v))$          // edge is still insertable

Operation delete($(u, v)$)
    for $k \leftarrow 1$ to $\alpha$ do                          // duplicating $(u,v)$ $\alpha$ times
        for $i \leftarrow \log_2 n$ to $\mathtt{active}+1$ do $\mathcal{T}_i.\mathtt{delete}((u,v))$    // affordable
          copies
        $\rho \leftarrow \mathcal{T}_{\mathtt{active}}.\mathtt{query}()$
        if $\rho < \rho^{est}$ then
            $\mathtt{active} \leftarrow \mathtt{active}-1$
            $\mathcal{T}_{\mathtt{active}}.\mathtt{delete}((u,v))$
        for $i \leftarrow \mathtt{active}-1$ to $1$ do                 // unaffordable copies
            if $(u,v) \in \mathtt{pending}_i$ then remove one copy of $(u,v)$ from $\mathtt{pending}_i$
            else
                $w \leftarrow \mathcal{T}_i.\mathtt{delete}((u,v))$     // $w$'s load was decremented
                if $(w,w') \in \mathtt{pending}_i$ for any $w'$ then
                    $\mathcal{T}_i.\mathtt{insert}((w,w'))$
                    Remove $(w,w')$ from $\mathtt{pending}_i$
```

**Algorithm 7:** Main update algorithm. $V$, $n = |V|$, and $\varepsilon$ are known quantities.

accordingly update the active index. Similarly, after a deletion, this can be $i$ or $i-1$. For insertions which are not affordable, we store the edges in a `pending` list. Consider an insertion $(u, v)$ which is not affordable in $\mathcal{T}_i$. This means that the loads on both $u$ and $v$ are at the limit $(\eta_i \alpha)$. We save $(u, v)$ in the pending list. For $\mathcal{T}_i$ to become affordable, one of $u$'s or $v$'s load must decrease. At this point, we insert $(u, v)$. The pseudocode for the overall algorithm is in Algorithm 7.

Notice, importantly, that insertions are made into $\mathcal{T}_i$ only when it is affordable. However, we always allow deletions because these are either deletions from the pending edges or from the graph currently stored in $\mathcal{T}_i$, which is still affordable.

*Proof of Theorem 4.1.* To show the correctness of Algorithm 7, we need to prove that at all times, $\hat{\rho}_G/2 \leq \rho^{\text{est}}_{\texttt{active}} < \hat{\rho}_G$. We know that this is true at the start of the algorithm. Assume this property is true at some instant before an update. When an edge is inserted, the first inequality might break. So, we test this after every addition and increment `active` accordingly. The argument follows similarly for deletions. However, we also need to make sure that when some $\mathcal{T}_i$ is queried, there are no edges remaining in $\texttt{pending}_i$, otherwise the queried density could possibly be incorrect. Consider an edge $(u, v)$ inserted into $\texttt{pending}_i$ at some point during the algorithm. For $\mathcal{T}_i$ to be queried, it must be active, which means that at some point, the load of either $u$ or $v$ decreased, causing $(u, v)$ to be inserted. Even when there are multiple such edges adjacent to the same high-load vertex, we are assured to see at least that many decrements at that vertex.

From Lemma 4.8, it follows that a query takes $O(1)$ worst-case time, and finding the subgraph takes $O(\beta + \log n)$ time, where $\beta$ is the size of the output subgraph. Each insert or delete operation is first duplicated $\alpha$ times. Secondly, the updates are made individually in $\log_2 n$ copies of the data structure.

First, note that any insert or delete operation in `pending` can be processed in $O(\log n)$ time. This is also true for searching using a single end point of an edge owing to the manner in which `pending` is defined.

When an edge is added, it makes two load queries and then possibly inserts in $\mathcal{T}_i$. From Lemma 4.8, this gives a worst-case runtime of $O(\alpha^3 \log n)$ time per insertion.

As for deleting an edge, it sometimes also requires an insertion into $\mathcal{T}_i$. Again, plugging in runtimes from Lemma 4.8, we obtain a worst-case runtime of $O(\alpha^3 \log n)$ time per deletion. □

## 4.3 Vertex-weighted Densest Subgraph

In this section, we extend the ideas from Section 4.2 to extend to graphs with vertex weights. As we will see in Section 4.4, this extension is crucial in arriving at efficient dynamic algorithms for DSP on directed graphs.

Let us first formally define the concept of density in vertex-weighted graphs. Given a graph $G = \langle V, E, w \rangle$, where $w : V \mapsto \mathbb{Q}^{\geq 1}$, the density of a subgraph induced by a vertex subset $S \subseteq V$ is

$$\rho_G(S) \overset{\text{def}}{=} \frac{|E(S)|}{\sum_{v \in S} \omega(v)}.$$

For ease of notation we denote $\omega(S) \overset{\text{def}}{=} \sum_{v \in S} \omega(v)$. Constructing the approximate dual like in Sections 1.2 and 4.2, we get the same conditions except the *load* on a vertex $v$ is now defined as

$$\ell_v = \frac{1}{\omega(v)} \sum_{e \ni v} f_e(v).$$

Let $\omega_{\min}$ and $\omega_{\max}$ denote the smallest and largest vertex weight in $G$. We multiply all the weights by $1/\omega_{\min}$ and later divide the answer by the same amount. This ensures that all weights are at least 1, and the maximum weight is now given by $W \overset{\text{def}}{=} \omega_{\max}/\omega_{\min}$.

We first show that local approximations also suffice for vertex-weighted DSP. We reuse the notation used in Section 4.2 for the exact and approximate dual LP – $\textsc{Dual}(G)$ and $\textsc{Dual}(G, \eta)$, but with vertex weights included.

**Theorem 4.11.** *Given an undirected vertex-weighted graph $G$ with $n$ vertices, with maximum vertex weight $W$, let $\hat{f}, \hat{\ell}$ denote any feasible solution to $\textsc{Dual}(G, \eta)$, and let*

$\hat{\rho}_G \overset{\text{def}}{=} \max_{v \in V} \hat{\ell}_v$. *Then,*

$$\left(1 - 3\sqrt{\frac{\eta \log(nW)}{\hat{\rho}_G}}\right) \cdot \hat{\rho}_G \leq \rho^*_G \leq \hat{\rho}_G.$$

*Proof.* The proof follows the proof of Theorem 4.3 almost identically.

Any feasible solution of $\textsc{Dual}(G, \eta)$ is also a feasible solution of $\textsc{Dual}(G)$, and so we have $\rho^*_G \leq \hat{\rho}_G$.

Denote by $T_i$ the set of vertices with load at least $\hat{\rho}_G - \eta i$, i.e.,

$$T_i \overset{\text{def}}{=} \left\{ v \in V \mid \hat{\ell}_v \geq \hat{\rho}_G - \eta i \right\}.$$

Let $0 < \alpha < 1$ be some adjustable parameter we will fix later. We define $k$ to be the maximal integer such that for any $1 \leq i \leq k$,

$$\omega(T_i) \geq \omega(T_{i-1}) \cdot (1 + \alpha).$$

Note that such a maximal integer $k$ always exists because there are finite number of vertices in $G$ and the size of $T_i$ grows exponentially. By the maximality of $k$,

$$\omega(T_{k+1}) < \omega(T_k) \cdot (1 + \alpha).$$

In order to bound the density of this set $T_{k+1}$, we compute the total load on all vertices in $T_k$. For any $u \in T_k$, the load on $u$ is given by

$$\hat{\ell}_u = \frac{1}{\omega(u)} \sum_{uv \in E} \hat{f}_{uv}(u).$$

However, we know that

$$f_{uv}(u) > 0 \implies \hat{\ell}_v \geq \hat{\ell}_u - \eta$$

and hence we only need to count for $v \in T_{k+1}$. Summing over all vertices in $T_{k+1}$, we get

$$\sum_{u \in T_k} \omega(u) \hat{\ell}_u = \sum_{u \in T_k, v \in T_{k+1}} \hat{f}_{uv}(u) \leq \sum_{u \in T_{k+1}, v \in T_{k+1}} \hat{f}_{uv}(u) = |E(T_{k+1})|.$$

Consider the density of set $T_{k+1}$,

$$\rho(T_{k+1}) = \frac{|E(T_{k+1})|}{\omega(T_{k+1})} \geq \frac{\sum_{u \in T_k} \hat{\ell}_u}{\omega(T_{k+1})} \geq \frac{\omega(T_k) \cdot (\hat{\rho}_G - \eta k)}{\omega(T_{k+1})},$$

where the last inequality follows from the definition of $T_k$.

Since $\rho(T_{k+1})$ can be at most the maximum subgraph density $\rho^*_G$, and using the fact that $\omega(T_k)/\omega(T_{k+1}) > 1/(1+\alpha) \geq 1 - \alpha$,

$$\rho^*_G \geq (1-\alpha)(\hat{\rho}_G - \eta k) \geq \hat{\rho}_G (1-\alpha) \left(1 - \frac{2\eta \log(nW)}{\alpha \cdot \hat{\rho}_G}\right),$$

where the last inequality comes from the fact that $nW \geq \omega(T_k) \geq (1+\alpha)^k$, which implies that $k \leq \log_{1+\alpha}(nW) \leq 2\log(nW)/\alpha$.

Now, we can set our parameter $\alpha$ to maximize the term on the RHS. By symmetry, the maximum is achieved when both terms in the product are equal and hence we set

$$\alpha = \sqrt{\frac{2\eta \log(nW)}{\hat{\rho}_G}}.$$

This gives

$$\rho^*_G \geq \hat{\rho}_G \cdot \left(1 - \sqrt{\frac{2\eta \log(nW)}{\hat{\rho}_G}}\right)^2 \geq \hat{\rho}_G \cdot \left(1 - 2\sqrt{\frac{2\eta \log(nW)}{\hat{\rho}_G}}\right) \geq \hat{\rho}_G \cdot \left(1 - 3\sqrt{\frac{\eta \log(nW)}{\hat{\rho}_G}}\right).$$

□

Once again, scaling the graph up by a factor of $\alpha \stackrel{\text{def}}{=} \frac{64 \log(nW)}{\varepsilon^2}$, we can frame the question as the following graph orientation problem:

Given an undirected graph $G$ with vertex-weights $w : V \mapsto \mathbb{Q}^+$ and a slack parameter $\eta$, we want to assign directions to edges in such a way that for any edge $u \to v$,

$$\frac{\boldsymbol{d}_{\mathsf{in}}(v)}{\omega(v)} \leq \frac{\boldsymbol{d}_{\mathsf{in}}(u)}{\omega(u)} + \eta.$$

To adapt the data structure from Algorithm 5, we only need to make the following change:

- `increment`$(u)$ and `decrement`$(u)$ no longer increment/decrement by $1$ but by $1/\omega(u)$.
- Each entry in the LABELS data structure is additionally appended with vertex weights - because instead of computing $|A|$ and $|B|$, we need to compute $\omega(A)$ and $\omega(B)$ in `maximal_label_set`.
- Since we assumed that $\omega(v) \geq 1$ for all $v \in V$, we do not have to adjust the conditions for tight edges.

Once we are provided with an estimate of $\hat{\rho}_G$, we can use the data structure from Algorithm 6 without any changes. Similar to Section 4.2.6, we now need to guess a value for $\hat{\rho}_G$. Notice that the range of values can now be $O(nW)$. Hence, using $O(\log(nW))$ values, we can apply Algorithm 7 to also solve the vertex-weighted version of DSP.

This gives us the following result.

**Theorem 4.12.** *Given a vertex-weighted graph $G$ with $n$ vertices, and vertex-weights in the range $\omega_{\min}$ and $\omega_{\max}$, there exists a deterministic fully dynamic $(1-\varepsilon)$-approximation algorithm for the densest subgraph problem on $G$ using $O(1)$ worst-case query time and worst-case update times of $O\left(\log^4\left(n \cdot \frac{\omega_{\max}}{\omega_{\min}}\right) \cdot \varepsilon^{-6}\right)$ per edge insertion or deletion.*

*Moreover, at any point, the algorithm can output the corresponding approximate densest subgraph in time $O(\beta + \log n)$, where $\beta$ is the number of vertices in the output.*

## 4.4 Directed Densest Subgraph

The directed version of the densest subgraph problem was introduced by Kannan and Vinay [57]. In a directed graph $G = \langle V, E \rangle$, for a pair of sets $S, T \subseteq V$, we denote using $E(S,T)$ the set of directed edges going from a vertex in $S$ to a vertex in $T$. The density of a pair of sets $S, T \subseteq V$ is defined as:

$$\rho_G(S,T) \overset{\text{def}}{=} \frac{|E(S,T)|}{\sqrt{|S||T|}}.$$

The maximum subgraph density of $G$ is then defined as:

$$\rho_G^* \overset{\text{def}}{=} \max_{S,T \subseteq V} \rho_G(S,T).$$

Note that we use the same notation for density for undirected and directed graphs, as the distinction is clear from the graph in the subscript.

Charikar [26] reduced directed DSP to $O(n^2)$ instances of solving an LP, and also observed that only $O(\log n/\varepsilon)$ suffice to extract a $(1-\varepsilon)$ approximation. Khuller and Saha [60] used the same reduction, but further simplified the algorithm to $O(1)$ instances of a parametrized maximum flow problem.

In this section, we recount this reduction, but by visualizing the problem reduced to as a densest subgraph problem on vertex-weighted graphs, as defined in Section 4.3.

### 4.4.1 Reduction from Directed DSP to Vertex-weighted Undirected DSP

Given a directed graph $G = \langle V, E \rangle$ and a parameter $t > 0$, we construct a vertex-weighted undirected graph

$$G_t = \langle V_t, E_t, \omega_t \rangle$$

where,

- $V_t \overset{\text{def}}{=} V_t^{(L)} \cup V_t^{(R)}$, in which $V_t^{(L)}$ and $V_t^{(R)}$ are both clones of the original vertex set $V$;

- $E_t \overset{\text{def}}{=} \left\{(u,v) \mid u \in V_t^{(L)}, v \in V_t^{(R)}, (u,v) \in E\right\}$ projects each original directed edge $(u,v) \in E$ into an undirected edge between $V_t^{(L)}$ and $V_t^{(R)}$, and
- $\omega_t(u) \overset{\text{def}}{=} \begin{cases} 1/2t & u \in V_t^{(L)} \\ t/2 & u \in V_t^{(R)} \end{cases}$

To understand the intuition behind this reduction, consider a pair of sets $S, T \subseteq V$. Consider the set $S^{(L)}$ corresponding to $S$ in $V_t^{(L)}$, and the set $T^{(R)}$ corresponding to $T$ in $V_t^{(R)}$. $\rho_G(S,T) = \frac{|E(S,T)|}{\sqrt{|S||T|}}$, whereas $\rho_{G_t}(S^{(L)} \cup T^{(R)}) = \frac{2|E(S,T)|}{(1/t)|S|+t|T|}$. Picking $t$ carefully lets us relate the two notions, leveraging the AM-GM inequality as indicated by the two denominators. Lemmas 4.13 and 4.14 show this relation in detail.

**Lemma 4.13.** *For any directed graph $G = \langle V, E\rangle$, let $G_t$ be defined as above. Then for any choice of parameter t,*

$$\rho_G^* \geq \rho_{G_t}^*.$$

*Proof.* Let $S^{(L)} \cup T^{(R)}$ denote the densest (vertex-weighted) subgraph in $G_t$, where $S^{(L)} \in V_t^{(L)}$ and $T^{(R)} \in V_t^{(R)}$. Let $S$ and $T$ denote the corresponding vertex sets in $V$. Then we have

$$\begin{aligned} |E_t(S^{(L)} \cup T^{(R)})| &= \rho_{G_t}^* \cdot (|S^{(L)}|/t + t|T^{(R)}|)/2 \\ &\geq \rho_{G_t}^* \sqrt{|S^{(L)}| \cdot |T^{(R)}|}, \end{aligned}$$

where the inequality follows from the AM-GM property. Using the facts $|E_t(S^{(L)} \cup T^{(R)})| = E(S,T)$, $|S^{(L)}| = |S|$, and $|T^{(R)}| = |T|$, we get that

$$\frac{E(S,T)}{\sqrt{|S| \cdot |T|}} \geq \rho_{G_t}^*.$$

Lastly, since the density of the pair of sets $S, T$ in the directed graph $G$ is at most $\rho_G^*$, we get that $\rho_G^* \geq \rho_{G_t}^*$. □

So, $G_t$ provides a ready lower bound for computing maximum subgraph density, for any $t$. The next lemma shows that a careful choice of $t$ can give equality between the two optimums.

**Lemma 4.14.** *For any directed graph $G = \langle V, E \rangle$ and a pair of subsets $S, T$ that provides the maximum subset density, i.e., $\rho^*_G = \rho_G(S, T)$, we have*

$$\rho^*_G = \rho^*_{G_t},$$

*where $t = \sqrt{\frac{|S|}{|T|}}$.*

*Proof.* Now, consider the sets $S^{(L)} \in V_t^{(L)}$ and $T^{(R)} \in V_t^{(R)}$ corresponding to $S$ and $T$ respectively. The density of set $S \cup T$ can be at most $\rho^*_{G_t}$:

$$\rho^*_{G_t} \geq \frac{2|E(S,T)|}{|S|/t + |T| \cdot t}.$$

Substituting $t$ with $|S|/|T|$,

$$\rho^*_{G_t} \geq \frac{2|E(S,T)|}{|S|\sqrt{\frac{|T|}{|S|}} + |T|\sqrt{\frac{|S|}{|T|}}} = \frac{|E(S,T)|}{\sqrt{|S| \cdot |T|}} = \hat{\rho}^*_G.$$

Combining this with the bound from Lemma 4.13 gives that $\rho^*_G = \rho^*_{G_t}$. □

Note, however, that this does not directly give an algorithm for directed densest subgraph, since we do not know the optimum value of $|S|/|T|$. Since both $|S|$ and $|T|$ are integers between 0 and $n$, there can be at most $O(n^2)$ distinct values of $|S|/|T|$. So, to find the exact solution, we can simply find $\rho^*_{G_t}$ for all possible $t$ values, and report the maximum.

This connection was first observed by Charikar [26], where he reduced the directed densest subgraph problem to solving $O(n^2)$ linear programs. However, our construction helps view these LPs as DSP on vertex-weighted graphs, for which there are far more optimized algorithms than solving generic LPs, in both static and dynamic paradigms. Charikar [26] also observed that a $1+\varepsilon$ approximate solution could be obtained by only checking $O(\log n/\varepsilon)$ values of $t$. As one would expect, to obtain an approximate solution for the directed DSP, it is not necessary to obtain an exact solution to the undirected vertex-weighted DSP. As we show in Lemma 4.15, we only require $O(\log n/\varepsilon)$ computations of a $1+\varepsilon/2$ approximation to the densest subgraph problem.

**Lemma 4.15.** *For any directed graph $G = \langle V, E\rangle$ and a pair of subsets $S, T$ that provides the maximum subset density, i.e., $\rho_G(S,T) = \rho^*_G$, we have*

$$\rho^*_{G_t} \geq (1-\varepsilon)\rho^*_G,$$

*where* $\sqrt{\frac{|S|}{|T|}} \cdot (1-\varepsilon) \leq t \leq \sqrt{\frac{|S|}{|T|}} \cdot \frac{1}{(1-\varepsilon)}$.

*Proof.* Consider the vertices $S^{(L)} \in V_t^{(L)}$ and $T^{(R)} \in V_t^{(R)}$ corresponding to $S$ and $T$ respectively. The density of set $S^{(L)} \cup T^{(R)}$ can be at most $\rho^*_{G_t}$:

$$\rho^*_{G_t} \geq \frac{2|E(S,T)|}{|S|/t + |T| \cdot t}.$$

Substituting the bounds for $t$,

$$\rho^*_{G_t} \geq \frac{2(1-\varepsilon)|E(S,T)|}{|S|\sqrt{\frac{|T|}{|S|}} + |T|\sqrt{\frac{|S|}{|T|}}} = (1-\varepsilon)\rho_G(S,T) = (1-\varepsilon)\rho^*_G. \qquad \square$$

### 4.4.2 Implications of the reduction

The above reduction implies that finding a $(1-\varepsilon)$-approximate solution to directed DSP can be reduced to $O(\log n \cdot \varepsilon^{-1})$ instances of $(1-\varepsilon/2)$-approximate vertex-weighted undirected DSP.

**Theorem 4.16.** *Given a directed graph $G$, with $m$ edges and $n$ vertices, and a $T(m,n,\varepsilon)$ time algorithm for $(1-\varepsilon)$-approximate vertex-weighted undirected densest subgraph, then* *there exists an* $(1-\varepsilon)$*-approximate algorithm for finding the densest subgraph in G in time* $T(m, 2n, \varepsilon/2) \cdot O(\log n/\varepsilon)$.

*Proof.* For each value of $t$ in

$$\left[\frac{1}{\sqrt{n}}, \frac{1}{(1-\varepsilon/2)\sqrt{n}}, \frac{1}{(1-\varepsilon/2)^2\sqrt{n}}, \ldots, \sqrt{n}\right],$$

we find an approximate value $\rho$ such that $\rho \geq (1-\varepsilon/2)\rho^*_{G_t}$, and output the maximum such value. Using $\varepsilon/2$ as the error parameter in Lemma 4.15, we get that $\rho \geq (1-\varepsilon)\rho^*_G$.

The number of values of $t$ is $\log_{1/(1-\varepsilon/2)} n = O(\log n/\varepsilon)$. □

The current fastest algorithms for $(1-\varepsilon)$-approximate static densest subgraph [11, 24] rely on approximately solving $\textsc{Dual}(G)$, which is a positive linear program, and subsequently extracting a primal solution. Both these parts of the algorithm extend naturally to vertex-weighted graphs. Substituting these runtimes in for $T(m, n, \varepsilon)$, we get the following corollary.

**Corollary 4.17.** *Let G be a directed graph with* $m$ *edges and* $n$ *vertices, and let* $\Delta$ *be the maximum value among all its in-degrees and out-degrees. Then, there exists an algorithm to find a* $(1-\varepsilon)$*-approximate densest subgraph in G in time* $\widetilde{O}(m\varepsilon^{-2} \cdot \min(\Delta, \varepsilon^{-1}))$.

Here, $\widetilde{O}$ hides polylogarithmic factors in $n$.

The same reduction also applies to fully dynamic algorithm for directed DSP.

**Theorem 4.18.** *Suppose there exists a fully dynamic* $(1-\varepsilon)$*-approximation algorithm for undirected vertex-weighted DSP on an* $n$*-vertex graph with update time* $U(n, \varepsilon)$ *and query time* $Q(n, \varepsilon)$*. Then, there exists a deterministic fully dynamic* $(1-\varepsilon)$*-approximation algorithm for directed DSP on an* $n$*-vertex graph using* $U(2n, \varepsilon/2) \cdot O(\log n/\varepsilon)$ *query time and* $Q(2n, \varepsilon/2) \cdot O(\log n/\varepsilon)$ *query time.*

Substituting the runtimes from Theorem 4.12 in Section 4.3, we get our result for dynamic DSP on directed graphs.

**Theorem 4.2.** *Given a directed graph $G$ with $n$ vertices, there exists a deterministic fully dynamic $(1 - \varepsilon)$-approximation algorithm for the densest subgraph problem on $G$ using $O(\log n \cdot \varepsilon^{-1})$ worst-case query time and worst-case update times of $O(\log^5 n \cdot \varepsilon^{-7})$ per edge insertion or deletion.*

*Moreover, at any point, the algorithm can output the corresponding approximate densest subgraph in time $O(\beta + \log n)$, where $\beta$ is the number of vertices in the output.*

# Appendices

# APPENDIX A
# PROOF OF WIDTH REDUCTION FOR THE MPC PROBLEM

In Section 2.4, we made the assumption that all entries in the constraint matrix can be assumed to be bounded by $1$, with only a $O(\log n)$ extra factor in running time. This assumption follows from the results in [98]. We outline this proof in this section for completeness.

For the purpose of this proof, we introduce notation $[k] := \{1, \ldots, k\}$.

Suppose we are given an instance of mixed packing covering of the form

$$Px \leq \mathbf{1}_p, Cx \geq \mathbf{1}_c, x \geq \mathbf{0}_n. \tag{A.1}$$

**Case 1:** For each column $P_{:,i}$ associated with variable $x_i$, let $P_{j_i,i} \overset{\text{def}}{=} \max_{j\in[p]} P_{ji} > 0$. Then we consider the following updates to MPC in order to reduce diameter.

Suppose, without loss of generality, $C_{1,i} = \max_{j\in[c]} C_{ji}$ and $C_{ci} = \min_{j\in[c]} C_{ji}$. If $C_{1i} \leq P_{j_i,i}$ then we can update $\overline{P}_{:,i} = \frac{1}{P_{j_i,i}} P_{:,i}$, $\overline{C}_{:,i} = \frac{1}{P_{j_i,i}} C_{:,i}$ and $\overline{x}_i = P_{j_i,i} x_i$. Then we observe that each element in $\overline{P}_{:,i}, \overline{C}_{:,i}$ is at most 1. Moreover, due to the packing constraint $\overline{P}_{j_i,:}\overline{x} \leq 1$, we note that for any feasible $\overline{x}$, $\overline{P}_{j_i,i}\overline{x}_i \leq 1$. Finally, since $\overline{P}_{j_i,i} = 1$, we have that $\overline{x}_i \leq 1$ lies in the support of constraint set. So we replaced the $i$-th column and corresponding $i$-th variable of the system by an equivalent system.

Similarly, if $C_{c,i} \geq P_{j_i,i}$ then consider $x^{sol}$ defined as

$$x_k^{sol} := \begin{cases} \frac{1}{P_{j_i,i}} & \text{if } k = i \\ 0 & \text{otherwise.} \end{cases}$$

Then $x^{sol}$ is already a feasible solution of MPC. So we may assume that $C_{ci} < P_{j_i,i} < C_{1i}$.

In this case, define $r_i = \dfrac{C_{1i}}{P_{j_i,i}}$ and $n_i = \lceil \log r_i \rceil$. We make $n_i$ copies of the column $C{:}, i$ and denote by the tuple $(i, l)$ the columns of a new matrix $\widehat{C}_{:,(i,l)}$ where $l \in [n_i]$. Similarly, we add $n_i$ copies of variable $x_i$, denoted as $\widehat{x}_{(i,l)}$. We make similar changes to $P_{:,i}$. Note that this system is equivalent to earlier system in the sense that any solution $\widehat{x}_{(i,l)}, l \in [n_i]$ can be converted into a solution of the earlier system since $x_i = \sum_{l \in [n_i]} \widehat{x}_{(i,l)}$. However, this allows us to reduce the elements of $\widehat{C}$ along with certain box constraints on $\widehat{x}_i$, which was our original goal. For each $j \in [c], l \in [n_i]$, redefine

$$\widehat{C}_{j,(i,l)} = \min\{C_{ji}, 2^l P_{j_i,i}\}$$

and for variable $\widehat{x}_{(i,l)}$, add the constraint

$$\widehat{x}_{i,l} \leq \frac{2}{2^l P_{j_i,i}}. \tag{A.2}$$

**Claim A.1.** *MPC* (A.1) *and the new system defined by matrices $\widehat{C}, \widehat{P}$ and variable $\widehat{x}$ are equivalent.*

*Proof.* For this proof, let us focus on $i$-th column and $i$-th variable.

For any feasible solution $\widehat{x}$, consider $x_i = \sum_{l \in [n_i]} \widehat{x}_{i,l}$. This $x_i$ does not violate any covering constraint since $\widehat{C}_{j,(i,l)} \leq C_{ji}$. The packing constraints also follow because we have not made any changes to the elements corresponding to the packing constraints $\widehat{P}_{j,(i,l)}$.

For the other direction, the key fact to note is that any feasible $x$ satisfies $x_i \leq \dfrac{1}{P_{j_i,i}}$ due to packing constraint $P_{j_i,:}x \leq 1$. Let $l_i$ be the largest index such that

$$x_i \leq \frac{2}{2^{l_i} P_{j_i,i}},$$

and then let

$$\widehat{x}_{(i,l)} = \begin{cases} x_i & \text{if } l = l_i \\ 0 & \text{otherwise.} \end{cases}$$

By construction, $\widehat{x}_{(i,l)}$ satisfies the constraint in (A.2) for all $l \in [n_i]$. Moreover, for constraint $j$, we must have $\widehat{C}_{j,:}\widehat{x} \geq 1$. Note that if $\widehat{C}_{j,(i,l_i)} = C_{ji}$ then there is nothing to prove. So we assume that $C_{ji} > \widehat{C}_{j,(i,l_i)} = 2^{l_i} P_{j_i,i}$. Then we must have that $l_i < n_i$ in this case, by definition of $n_i$. This then gives $\widehat{x}_{(i,l_i)} = x_i \geq \frac{1}{2^{l_i} P_{j_i,i}}$ by our choice of $l_i$ being the largest possible. Then we know that $\widehat{C}_{j,(i,l_i)} = 2^{l_i} P_{j_i,i}$, and hence the $j$-th covering constraint is satisfied.

Packing constraints are satisfied trivially since there is no change in elements of $\widehat{P}_{:,(i,l)}$ for all $l \in [n_i]$. Hence the claim follows. $\square$

Finally the proof follows by change of variables as $\overline{x}_{(i,l)} = 2^{l-1} P_{j_i,i}$ and $\overline{C}_{:,(i,l)} = \frac{1}{2^{l-1} P_{j_i,i}} \widehat{C}_{:,(i,l)}$. Further, note that all elements of $\overline{P}_{:,(i,l)}$ are at most 1 for all $l \in [n_i]$, and all elements of $\overline{C}_{:,(i,l)}$ are at most 2 for all $l \in [n_i]$ and $\overline{x}_{i,l} \leq 1$ for all $l \in [n_i]$.

**Case 2:** Suppose $P_{j_i,i} = 0.$. This implies that in variable $x_i$, this is a purely covering problem. So we can increase $x_i$ to satisfy the $j$th covering constraint such that $C_{ji} > 0$ independent of the packing constraints and problem reduces to smaller packing covering problem in remaining variables and covering constraints $j$ such that $C_{ji} = 0$. For this smaller packing covering problem, we can apply the method in Case 1 again.

# APPENDIX B
# MULTIPLICATIVE WEIGHTS UPDATE ALGORITHM

In this section, we give an algorithm to solve the zero-sum game $\max_{\mathbf{x}\in\Delta_n}\min_{\mathbf{f}\in\mathcal{P}}\mathbf{x}^T\mathbf{B}\mathbf{f}$, which corresponds to solving the dual of the densest subgraph problem, as described in Section 3.3.2. Given that we have an oracle access to $\min_{\mathbf{f}\in\mathcal{P}}\mathbf{x}^T\mathbf{B}\mathbf{f}$, we can use the multiplicative weights update framework to get an $\varepsilon$-approximation of the game [36].

The pseudocode for the MWU algorithm is shown in Algorithm 8.

```
Input: Matrix $\mathbf{B}$, approximation factor $\varepsilon$
Output: An approximate solution to the zero-sum game.
Initialize the weight vector as $w_i^{(1)} \leftarrow 1$ for all $i \in [n]$;
Initialize $\eta \leftarrow \varepsilon/2\deg_{\max}$;
for $t : 1 \to T$ do
    $x_i^{(t)} \leftarrow w_i^{(t)}/\|\mathbf{w}^{(t)}\|_1$ for all $i \in [n]$;
    Find $\mathbf{f}(\mathbf{x}^{(t)})$ using Oracle($\mathbf{x}^{(t)}$);
    Set $C(\mathbf{x}^{(t)}) \leftarrow (\mathbf{x}^{(t)})^T\mathbf{B}\mathbf{f}(\mathbf{x}^{(t)})$;
    Let $\mathbf{b}_i^T\mathbf{f}(\mathbf{x}^{(t)})$ be the $i$-th element in $\mathbf{B}\mathbf{x}^{(t)}$;
    Update the weights as $w_i^{(t+1)} \leftarrow w_i^{(t)}(1+\eta\mathbf{b}_i^T\mathbf{f}(\mathbf{x}^{(t)}))$
return $\frac{1}{T}\sum_{t\in[T]} C(\mathbf{x}^{(t)})$
```

**Algorithm 8:** Multiplicative Weight Update Algorithm

To prove the convergence of Algorithm 8, we use the following theorem from [9]. We modify it slightly to accommodate for the fact that the width of the DSP, $||Bf(x)||_\infty$, can be at most $\deg_{\max}$. In other words, the oracle can assign at most $\deg_{\max}$ edges to any particular vertex.

**Lemma B.1** (Theorem 3.1 from [9])**.** *Given an error parameter $\varepsilon$, there is an algorithm which solves the zero-sum game up to an additive factor of $\varepsilon$ using $O(W\log n/\varepsilon^2)$ calls to*

ORACLE, *with an additional processing time of* $O(n)$ *per call, where* $W$ *is the width of the problem.*

Using the fact that our ORACLE runs in $O(m)$ time (from Lemma 3.5), and using $W = \deg_{\max}$, we get the following corollary.

**Corollary B.2.** *The Multiplicative Weight Update algorithm (Algorithm 8) outputs a* $(1+\varepsilon)$ *approximate solution to the densest subgraph problem in time* $O(m \deg_{\max} \log n/\varepsilon^2)$.

## REFERENCES

[1] A. Abboud and S. Dahlgaard, “Popular conjectures as a barrier for dynamic planar graph algorithms,” in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, 2016, pp. 477–486.

[2] A. Abboud, L. Georgiadis, G. F. Italiano, R. Krauthgamer, N. Parotsidis, O. Trabelsi, P. Uznanski, and D. Wolleb-Graf, “Faster algorithms for all-pairs bounded min-cuts,” in *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, C. Baier, I. Chatzigiannakis, P. Flocchini, and S. Leonardi, Eds., ser. LIPIcs, vol. 132, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 7:1–7:15.

[3] A. Abboud, R. Krauthgamer, and O. Trabelsi, “New algorithms and lower bounds for all-pairs max-flow in undirected graphs,” in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, S. Chawla, Ed., SIAM, 2020, pp. 48–61.

[4] A. Abboud and V. V. Williams, “Popular conjectures imply strong lower bounds for dynamic problems,” in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, 2014, pp. 434–443.

[5] T. Akiba, Y. Iwata, and Y. Yoshida, “Fast exact shortest-path distance queries on large networks by pruned landmark labeling,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’13, New York, New York, USA: ACM, 2013, pp. 349–360, ISBN: 978-1-4503-2037-5.

[6] Z. Allen-Zhu and L. Orecchia, “Nearly linear-time packing and covering LP solvers - achieving width-independence and -convergence,” *Math. Program.*, vol. 175, no. 1-2, pp. 307–353, 2019.

[7] R. Andersen and K. Chellapilla, “Finding dense subgraphs with size bounds,” in *International Workshop on Algorithms and Models for the Web-Graph*, Springer, 2009, pp. 25–37.

[8] A. Angel, N. Koudas, N. Sarkas, D. Srivastava, M. Svendsen, and S. Tirthapura, “Dense subgraph maintenance under streaming edge weight updates for real-time story identification,” *VLDB J.*, vol. 23, no. 2, pp. 175–199, 2014.

[9] S. Arora, E. Hazan, and S. Kale, “The multiplicative weights update method: A meta-algorithm and applications,” *Theory of Computing*, vol. 8, no. 1, pp. 121–164, 2012.

[10] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama, “Greedily finding a dense subgraph,” in *Algorithm Theory — SWAT’96*, R. Karlsson and A. Lingas, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 136–148.

[11] B. Bahmani, A. Goel, and K. Munagala, “Efficient primal-dual graph algorithms for mapreduce,” in *Algorithms and Models for the Web Graph - 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings*, 2014, pp. 59–78.

[12] B. Bahmani, R. Kumar, and S. Vassilvitskii, “Densest subgraph in streaming and mapreduce,” *Proc. VLDB Endow.*, vol. 5, no. 5, pp. 454–465, Jan. 2012.

[13] Y. Bartal, J. W. Byers, and D. Raz, “Fast, distributed approximation algorithms for positive linear programming with applications to flow control,” *SIAM J. Comput.*, vol. 33, no. 6, pp. 1261–1279, 2004.

[14] A. Beck, “On the convergence of alternating minimization for convex programming with applications to iteratively reweighted least squares and decomposition schemes,” *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 185–209, 2015.

[15] A. Bernstein and C. Stein, “Faster fully dynamic matchings with small approximation ratios,” in *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA ’16, Arlington, Virginia: Society for Industrial and Applied Mathematics, 2016, pp. 692–711, ISBN: 978-1-611974-33-1.

[16] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, “Detecting high log-densities: An o (n $1/4$) approximation for densest k-subgraph,” in *Proceedings of the forty-second ACM symposium on Theory of computing*, ACM, 2010, pp. 201–210.

[17] S. Bhattacharya, M. Henzinger, and G. F. Italiano, “Deterministic fully dynamic data structures for vertex cover and matching,” *SIAM J. Comput.*, vol. 47, no. 3, pp. 859–887, 2018.

[18] S. Bhattacharya, M. Henzinger, and G. F. Italiano, “Dynamic algorithms via the primal-dual method,” *Inf. Comput.*, vol. 261, no. Part, pp. 219–239, 2018.

[19] S. Bhattacharya, M. Henzinger, and D. Nanongkai, “New deterministic approximation algorithms for fully dynamic matching,” in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, 2016, pp. 398–411.

[20] S. Bhattacharya, M. Henzinger, and D. Nanongkai, “Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time,” in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, 2017, pp. 470–489.

[21] S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. Tsourakakis, “Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams,” in *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC ’15, Portland, Oregon, USA: ACM, 2015, pp. 173–182, ISBN: 978-1-4503-3536-2.

[22] D. Bienstock and G. Iyengar, “Approximating fractional packings and coverings in o(1/epsilon) iterations,” *SIAM J. Comput.*, vol. 35, no. 4, pp. 825–854, 2006.

[23] D. Boob, Y. Gao, R. Peng, S. Sawlani, C. E. Tsourakakis, D. Wang, and J. Wang, “Flowless: Extracting densest subgraphs without flow computations,” *CoRR*, vol. abs/1910.07087, 2019. arXiv: 1910.07087.

[24] D. Boob, S. Sawlani, and D. Wang, “Faster width-dependent algorithm for mixed packing and covering lps,” in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 15 253–15 262.

[25] G. S. Brodal and R. Fagerberg, “Dynamic representation of sparse graphs,” in *Algorithms and Data Structures, 6th International Workshop, WADS ’99, Vancouver, British Columbia, Canada, August 11-14, 1999, Proceedings*, 1999, pp. 342–351.

[26] M. Charikar, “Greedy approximation algorithms for finding dense components in a graph,” in *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, ser. APPROX ’00, Berlin, Heidelberg, 2000, pp. 84–95, ISBN: 3-540-67996-0.

[27] J. Chen and Y. Saad, “Dense subgraph extraction with application to community detection,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 7, pp. 1216–1230, 2012.

[28] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, “Reachability and distance queries via 2-hop labels,” *SIAM J. Comput.*, vol. 32, no. 5, pp. 1338–1355, May 2003.

[29] M. Danisch, T. H. Chan, and M. Sozio, "Large scale density-friendly graph decomposition via convex programming," in *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, 2017, pp. 233–242.

[30] Y. Dourisboure, F. Geraci, and M. Pellegrini, "Extraction and classification of dense communities in the web," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07, Banff, Alberta, Canada: ACM, 2007, pp. 461–470, ISBN: 978-1-59593-654-7.

[31] R. Duan and S. Pettie, "Linear-time approximation for maximum weight matching," *J. ACM*, vol. 61, no. 1, 1:1–1:23, 2014.

[32] A. Epasto, S. Lattanzi, and M. Sozio, "Efficient densest subgraph computation in evolving graphs," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15, Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 300–310, ISBN: 978-1-4503-3469-3.

[33] H. Esfandiari, S. Lattanzi, and V. Mirrokni, "Parallel and streaming algorithms for k-core decomposition," *arXiv preprint arXiv:1808.02546*, 2018.

[34] D. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 2010, ISBN: 0691146675, 9780691146676.

[35] G. N. Frederickson, "Data structures for on-line updating of minimum spanning trees, with applications," *SIAM J. Comput.*, vol. 14, no. 4, pp. 781–798, 1985.

[36] Y. Freund and R. E. Schapire, "Game theory, on-line prediction and boosting," in *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, ser. COLT '96, Desenzano del Garda, Italy: ACM, 1996, pp. 325–332, ISBN: 0-89791-811-8.

[37] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for general graph-matching problems," *J. ACM*, vol. 38, no. 4, pp. 815–853, 1991.

[38] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan, "A fast parametric maximum flow algorithm and applications," *SIAM J. Comput.*, vol. 18, no. 1, pp. 30–55, Feb. 1989.

[39] C. Giatsidis, F. Malliaros, D. Thilikos, and M. Vazirgiannis, "Corecluster: A degeneracy based graph clustering framework," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[40] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *Proceedings of the 31st International Conference on Very*

*Large Data Bases*, ser. VLDB '05, Trondheim, Norway: VLDB Endowment, 2005, pp. 721–732, ISBN: 1-59593-154-6.

[41] A. Gionis and C. E. Tsourakakis, "Dense subgraph discovery: Kdd 2015 tutorial," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2015, pp. 2313–2314.

[42] A. V. Goldberg, "Finding a maximum density subgraph," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-84-171, 1984.

[43] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM (JACM)*, vol. 35, no. 4, pp. 921–940, 1988.

[44] G. Goranci, M. Henzinger, and T. Saranurak, "Fast incremental algorithms via local sparsifiers," unpublished manuscript, 2018.

[45] M. D. Grigoriadis and L. G. Khachiyan, "Approximate minimum-cost multicommodity flows in Õ(epsilon$^{-2}$knm) time," *Math. Program.*, vol. 75, pp. 477–482, 1996.

[46] M. Gupta, "Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time," in *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, 2014, pp. 227–239.

[47] M. Gupta and R. Peng, "Fully dynamic (1+ e)-approximate matchings," in *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, 2013, pp. 548–557.

[48] C. Guzmán and A. Nemirovski, "On lower complexity bounds for large-scale smooth convex optimization," *Journal of Complexity*, vol. 31, no. 1, pp. 1 –14, 2015.

[49] J. Hastad, "Clique is hard to approximate within $n^{1-\epsilon}$," *Acta Mathematica*, vol. 182, no. 1, 1999.

[50] M. Henzinger, "The state of the art in dynamic graph algorithms," in *SOFSEM 2018: Theory and Practice of Computer Science*, A. M. Tjoa, L. Bellatreche, S. Biffl, J. van Leeuwen, and J. Wiedermann, Eds., Cham: Springer International Publishing, 2018, pp. 40–44, ISBN: 978-3-319-73117-9.

[51] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *Proceedings of the Forty-seventh Annual ACM Symposium on*

*Theory of Computing*, ser. STOC '15, Portland, Oregon, USA: ACM, 2015, pp. 21–30, ISBN: 978-1-4503-3536-2.

[52] M. R. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," *J. ACM*, vol. 46, no. 4, pp. 502–516, 1999.

[53] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *J. ACM*, vol. 48, no. 4, pp. 723–760, 2001.

[54] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou, "Mining coherent dense subgraphs across massive biological networks for functional discovery," *Bioinformatics*, vol. 21, no. 1, pp. 213–221, Jan. 2005.

[55] G. F. Italiano and P. Sankowski, "Improved minimum cuts and maximum flows in undirected planar graphs," *CoRR*, vol. abs/1011.2843, 2010. arXiv: `1011.2843`.

[56] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, "3hopp: A high-compression indexing scheme for reachability query," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09, Providence, Rhode Island, USA: ACM, 2009, pp. 813–826, ISBN: 978-1-60558-551-2.

[57] R. Kannan and V. V., "Analyzing the structure of large graphs," unpublished manuscript, 1999.

[58] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in polylogarithmic worst case time," in *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, 2013, pp. 1131–1142.

[59] Y. Kawase and A. Miyauchi, "The densest subgraph problem with a convex/concave size function," *Algorithmica*, vol. 80, no. 12, pp. 3461–3480, 2018.

[60] S. Khuller and B. Saha, "On finding dense subgraphs," in *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I*, ser. ICALP '09, Rhodes, Greece: Springer-Verlag, 2009, pp. 597–608, ISBN: 978-3-642-02926-4.

[61] T. Kopelowitz, R. Krauthgamer, E. Porat, and S. Solomon, "Orienting fully dynamic graphs with worst-case time bounds," in *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, 2014, pp. 532–543.

[62] L. Kowalik, "Adjacency queries in dynamic sparse graphs," *Inf. Process. Lett.*, vol. 102, no. 5, pp. 191–195, 2007.

[63] R. Kumar, J. Novak, and A. Tomkins, “Structure and evolution of online social networks,” in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’06, Philadelphia, PA, USA: ACM, 2006, pp. 611–617, ISBN: 1-59593-339-5.

[64] R. Kumar, P. Raghavan, S. Rajagopalan, S. Rajagopalan, A. Tomkins, A. Tomkins, and A. Tomkins, “Trawling the web for emerging cyber-communities,” *Comput. Netw.*, vol. 31, no. 11-16, pp. 1481–1493, May 1999.

[65] J. Kunegis, “Konect: The koblenz network collection,” in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW ’13 Companion, Rio de Janeiro, Brazil: ACM, 2013, pp. 1343–1350, ISBN: 978-1-4503-2038-2.

[66] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, “A survey of algorithms for dense subgraph discovery,” in *Managing and Mining Graph Data*. Boston, MA: Springer US, 2010, pp. 303–336, ISBN: 978-1-4419-6045-0.

[67] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, Jun. 2014.

[68] M. Luby and N. Nisan, “A parallel approximation algorithm for positive linear programming,” in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, 1993, pp. 448–457.

[69] A. Madry, “From graphs to matrices, and back: New techniques for graph algorithms,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2011.

[70] M. W. Mahoney, S. Rao, D. Wang, and P. Zhang, “Approximating the solution to mixed packing and covering lps in parallel o(epsilon^{-3}) time,” in *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, 2016, 52:1–52:14.

[71] A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu, “Densest subgraph in dynamic graph streams,” in *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, 2015, pp. 472–482.

[72] S. Micali and V. V. Vazirani, “An o(sqrt($\Gamma a30c$v$\Gamma a30c$) $\Gamma a30c$e$\Gamma a30c$) algorithm for finding maximum matching in general graphs,” in *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, 1980, pp. 17–27.

[73] M. Mitzenmacher, J. Pachocki, R. Peng, C. Tsourakakis, and S. C. Xu, “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*, ser. KDD '15, Sydney, NSW, Australia: ACM, 2015, pp. 815–824, ISBN: 978-1-4503-3664-2.

[74] Y. Nesterov, "Smooth minimization of non-smooth functions," *Math. Program.*, vol. 103, no. 1, pp. 127–152, 2005.

[75] Y. Nesterov, "Dual extrapolation and its applications to solving variational inequalities and related problems," *Math. Program.*, vol. 109, no. 2-3, pp. 319–344, 2007.

[76] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.

[77] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006. eprint: `https://www.pnas.org/content/103/23/8577.full.pdf`.

[78] S. A. Plotkin, D. B. Shmoys, and É. Tardos, "Fast approximation algorithms for fractional packing and covering problems," *Mathematics of Operations Research*, vol. 20, no. 2, pp. 257–301, 1995.

[79] J. Ren, J. Wang, M. Li, and L. Wang, "Identifying protein complexes based on density and modularity in protein-protein interaction network," *BMC Systems Biology*, vol. 7, no. 4, S12, 2013.

[80] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang, "Dense subgraphs with restrictions and applications to gene annotation graphs," in *Research in Computational Molecular Biology*, B. Berger, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 456–472, ISBN: 978-3-642-12683-3.

[81] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *Proc. VLDB Endow.*, vol. 11, no. 4, pp. 420–431, Dec. 2017.

[82] A. D. Sarma, A. Lall, D. Nanongkai, and A. Trehan, "Dense subgraphs on dynamic networks," in *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, 2012, pp. 151–165.

[83] S. Sawlani and J. Wang, "Near-optimal fully dynamic densest subgraph," unpublished manuscript, 2019.

[84] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269 –287, 1983.

[85] J. Sherman, “Area-convexity, $l_\infty$ regularization, and undirected multicommodity flow,” in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, 2017, pp. 452–460.

[86] K. Shin, T. Eliassi-Rad, and C. Faloutsos, “Corescope: Graph mining using k-core analysis: Patterns, anomalies and algorithms,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, IEEE, 2016, pp. 469–478.

[87] D. D. Sleator and R. E. Tarjan, “A data structure for dynamic trees,” *J. Comput. Syst. Sci.*, vol. 26, no. 3, pp. 362–391, Jun. 1983.

[88] K. Sotiropoulos, J. W. Byers, P. Pratikakis, and C. E. Tsourakakis, “Twittermancer: Predicting interactions on twitter accurately,” *arXiv preprint arXiv:1904.11119*, 2019.

[89] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers, “Biogrid: A general repository for interaction datasets,” *Nucleic acids research*, vol. 34, pp. D535–9, Jan. 2006.

[90] L. Tang and H. Liu, “Graph mining applications to social network analysis,” in *Managing and Mining Graph Data*. Boston, MA: Springer US, 2010, pp. 487–513, ISBN: 978-1-4419-6045-0.

[91] N. Tatti and A. Gionis, “Density-friendly graph decomposition,” in *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, 2015, pp. 1089–1099.

[92] M. Thorup, “Fully-dynamic min-cut,” *Combinatorica*, vol. 27, no. 1, pp. 91–127, 2007.

[93] C. Tsourakakis, “Streaming graph partitioning in the planted partition model,” in *Proceedings of the 2015 ACM on Conference on Online Social Networks*, ACM, 2015, pp. 27–35.

[94] C. Tsourakakis, “The k-clique densest subgraph problem,” in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW ’15, Florence, Italy: International World Wide Web Conferences Steering Committee, 2015, pp. 1122–1132, ISBN: 978-1-4503-3469-3.

[95] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli, “Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees,” in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2013, pp. 104–112.

[96] C. E. Tsourakakis, “A novel approach to finding near-cliques: The triangle-densest subgraph problem,” *CoRR*, vol. abs/1405.1477, 2014. arXiv: `1405.1477`.

[97] C. E. Tsourakakis, T. Chen, N. Kakimura, and J. Pachocki, “Novel dense subgraph discovery primitives: Risk aversion and exclusion queries,” *arXiv preprint arXiv:1904.08178*, 2019.

[98] D. Wang, S. Rao, and M. W. Mahoney, “Unified acceleration method for packing and covering problems via diameter reduction,” in *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, 2016, 50:1–50:13.

[99] N. E. Young, “Sequential and parallel algorithms for mixed packing and covering,” in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, 2001, pp. 538–546.

[100] N. E. Young, “Nearly linear-time approximation schemes for mixed packing/covering and facility-location linear programs,” *CoRR*, vol. abs/1407.3015, 2014. arXiv: `1407.3015`.

[101] R. Zafarani and H. Liu, *Social computing data repository at ASU*, 2009.

[102] E. Zurel and N. Nisan, “An efficient approximate allocation algorithm for combinatorial auctions,” in *Proceedings 3rd ACM Conference on Electronic Commerce (EC-2001), Tampa, Florida, USA, October 14-17, 2001*, 2001, pp. 125–136.