Faster and Scalable Algorithms for Densest Subgraph and Decomposition

Anonymous Author(s)
Affiliation
Address
email

Abstract

We study the densest subgraph problem (DSG) and the densest subgraph local decomposition problem (DSG-LD) in undirected graphs. We also consider supermodular generalizations of these problems. For large scale graphs simple iterative algorithms perform much better in practice than theoretically fast algorithms based on network-flow or LP solvers. Boob *et al.* [1] recently gave a fast iterative algorithm called GREEDY++ for DSG. It was shown in [2] that it converges to a $(1-\epsilon)$ relative approximation to the optimum density in $O(\frac{1}{\epsilon^2}\frac{\Delta(G)}{\lambda^*})$ iterations where $\Delta(G)$ is the maximum degree and λ^* is the optimum density. Danisch *et al.* [3] gave an iterative algorithm based on the Frank-Wolfe algorithm for DSG-LD that takes $O(\frac{m\Delta(G)}{\epsilon^2})$ iterations to converge to an ϵ-additive approximate local decomposition vector $\hat{b}$, where m is number of edges in the graph.

In this paper we give a new iterative algorithm for both problems that takes at most $O(\frac{\sqrt{m\Delta(G)}}{\epsilon})$ iterations to converge to an ϵ-additive approximate local decomposition vector; each iteration can be implemented in $O(m)$ time. We describe a fractional peeling technique which has strong empirical performance as well as theoretical guarantees. The algorithm is scalable and simple, and can be applied to graphs with hundreds of millions of edges. We test our algorithm on real and synthetic data sets and show that it provides a significant benefit over previous algorithms. The algorithm and analysis extends to hypergraphs.

1. Introduction

The densest subgraph problem (DSG) is a classical problem in combinatorial optimization and has many real world applications in data mining, network analysis, and machine learning. The input for DSG is an undirected graph $G = (V, E)$ with $m = |E|$ and $n = |V|$. The goal is to return a subset $S \subseteq V$ that maximizes $\frac{|E(S)|}{|S|}$ where $E(S) = \{\{u, v\} \in E : u, v \in S\}$ is the set of edges with both end points in S. DSG has a variety of applications in which dense subgraphs reveal important information about the underlying network such as communities. One can view it as a subroutine in an unsupervised clustering procedure. It is a canonical problem in the broad area of dense subgraph discovery which has seen many developments and applications in the past two decades. We point the reader's attention to a (non-exhaustive) list of recent, and some not so recent, important work and the pointers therein [4, 5, 6, 7, 1, 3, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. Constrained versions of DSG such as the densest k-subgraph problem DKSG, densest at most k-subgraph, and densest at least k-subgraph DALKSG are also well-studied and many of these are NP-Hard. See [24, 25, 26, 10, 6, 27] for some positive and negative results on approximation algorithms.

One of the key advantages of DSG is its polynomial-time solvability. The first polynomial-time algorithms were due to Goldberg [4] and Picard and Queuranne [28]. The decision version of DSG

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

is the following: given G and a rational number λ, is the density in G at least λ? The algorithms in [4, 28] construct an auxiliary directed flow-network H such that the maximum flow in H allows one to answer the decision version. Binary search over λ leads to the final algorithm. Maximum flow is a powerful algorithmic subroutine, however, it is not a practical or scalable algorithm for modern graph data sets with millions and even billions of vertices and edges. For example, authors from [1] noted that the Goldberg's maximum flow algorithm failed on many large scale graphs even though they used a highly optimized maximum flow library. Charikar [5] described a linear programming formulation for DSG that gives an exact solution and has $O(|E|)$ variables and constraints. LP solvers are also unsuitable for large data sets due to memory limitations among others. Charikar [5] described a very simple $\frac{1}{2}$-approximation algorithm for DSG known as the GREEDY or PEELING algorithm. The algorithm creates an ordering of the vertices as follows. The first vertex v_1 is the one with the smallest degree in G (ties broken arbitrarily). It selects v_2 to be the smallest degree vertex in $G - v_1$. Letting G^i be the graph after removing $v_1, v_2, \ldots, v_{i-1}$ (with $G^0 = G$), the algorithm returns the graph among $G^0, \ldots, G^n$ with the highest density. The algorithm can be implemented in $O(m)$ time. The ordering created by the algorithm is the same as the one to compute a k-core decomposition of a graph — this is a well-studied graph decomposition procedure with several applications [29] . The simplicity and the efficiency of the Greedy algorithm, and its approximation guarantees, has led to its adoption for a number of other density measures.

Despite Greedy's advantages, its worst-case approximation guarantee is only $\frac{1}{2}$, and is worse for other density measures. The goal is to develop algorithms that obtain a $(1-\varepsilon)$ relative approximation for a given parameter $\varepsilon \in (0, 1)$ while also being scalable to large graphs. One approach to obtain such algorithms is via the dual of Charikar's LP relaxation. It is a mixed packing and covering LP for a given guess of the optimal value. Such LPs can be approximately solved via iterative methods such as the multiplicative weight updates (MWU) or other methods based on convex optimization. Bahmani, Goel and Munagala [30] applied this methodology to obtain an algorithm that yields a $(1-\varepsilon)$-approximation in $\tilde{O}(m/\varepsilon^2)$-time. Boob, Sawlani and Wang [31] described an algorithm that yields a $(1-\varepsilon)$-approximation in $\tilde{O}(m\Delta(G)/\varepsilon)$-time, where $\Delta(G)$ is the maximum degree in G. More recently Chekuri, Quanrud and Torres [2] obtained a $(1-\varepsilon)$-approximation in $\tilde{O}(m/\varepsilon)$-time via approximate flow techniques. Some of these nice theoretical developments have not yet led to practically useful algorithms for large scale graphs. In a different direction, Boob *et al.* [1] described a fast iterative algorithm called GREEDY++ which builds on the Greedy algorithm and insights from the LP relaxation. It does extremely well in experiments, and the authors conjectured that it yields a $(1-\epsilon)$ relative approximation in $O(\frac{1}{\epsilon^2})$ iterations. (Each iteration takes $O(m)$ time.) Chekuri *et al.* [2] proved that GREEDY++ converges to a $(1-\epsilon)$-approximation in $O(\frac{\Delta(G)}{\lambda^* \epsilon^2})$ iterations where λ^* is the optimum density. This gives evidence of the theoretical soundness of the algorithm.

Our goal in this paper is to develop new and scalable algorithms that outperform GREEDY++ while having strong theoretical guarantees. In addition, we are interested in an algorithm that finds an approximate *dense subgraph decomposition* which gives information on the density structure of the graph and is different from another such decomposition, namely the k-core decomposition. Further, as shown in previous work [32, 33, 2], a dense subgraph decomposition allows one to compactly represent approximate solutions to densest subgraphs of different sizes; in particular the representation yields a 2-approximation for densest at least k-subgraph for any given k [32, 2].

Dense subgraph decomposition. One can show that every graph $G = (V, E)$ admits a certain structured nested decomposition based on decreasing density. In particular, the vertex set V can partitioned into $S_1, S_2, \ldots, S_k$ such that S_1 is the unique maximal densest subgraph in G, and S_i is the unique maximal densest set with respect to $S_1 \cup \ldots \cup S_{i-1}$ (a formal definition is deferred to Section 3). For each vertex v we let λ_v denote the density of the set containing v. The existence of such a decomposition follows from the supermodularity properties of the function $f(S) = |E(S)|$, and is implicitly known from past work on submodular functions (in particular, the classical result of Fujishige [34]). For graphs this decomposition was rediscovered by Tatti and Gionis [35, 12] under the name of locally-dense decomposition. Tatti showed that one can compute the exact decomposition via n maximum flow computations. Danisch *et al.* [3] showed that computing the λ_v values can be cast as solving a quadratic program that extends the LP of Charikar, and applied the Frank-Wolfe algorithm. They showed that their algorithm needs $O(\frac{|E|\Delta(G)}{\epsilon^2})$ iterations (each taking $O(|E|)$ time) to converge to an ε-approximate vector. Their work shows that the Frank-Wolfe algorithm leads to a good approximation to the densest subgraph in a relatively small number of

iterations based on experiments. However, they do not describe a systematic way to extract a dense subgraph decomposition from the approximate vector with provable guarantees.

Densest Supermodular Set. Efficient solvability of DSG can also be seen via a connection to a more general problem called the densest supermodular set problem, which we refer to as DSS. A real-valued set function $f : 2^V \to \mathbb{R}_+$ is said to be supermodular iff $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq V$. The goal in DSS is to return $S \subseteq V$ that maximizes $\frac{f(S)}{|S|}$. For any graph $G = (V, E)$, the function $f : 2^V \to \mathbb{R}_+$ defined as $f(S) = |E(S)|$ for each $S \subseteq V$, is known to be supermodular. Hence DSS generalizes DSG. Several algorithms and structural features for DSG are easier to understand via supermodularity, as shown in recent work [2].

Notation: For an undirected graph $G = (V, E)$ and $S_1, S_2 \subseteq V$, we use $E(S_1, S_2)$ for the edge set $\{\{u, v\} \in E : u \in S_1, v \in S_2\}$. For a vector $b \in \mathbb{R}^V$, and a subset $S \subseteq V$, we let $b(S)$ denote $\sum_{i \in S} b_i$. We let $ord(E)$ be the set of $2|E|$ edges that are ordered edges uv and vu for each unordered edge $\{u, v\} \in E$. Finally, m and n denote $|E|$ and $|V|$ respectively.

2. Technical Contributions

We summarize the main contributions of this paper.

A fast and scalable algorithm based on projections: We describe an iterative algorithm for DSG and dense subgraph decomposition based on solving a quadratic objective with linear constraints that is derived from the dual of Charikar's LP. Unlike previous work [36] that relied on the Frank-Wolfe method, we use a projection-based approach. The algorithm is extremely simple and highly parallelizable. We show that it converges to an ε-additive approximation in $O(\sqrt{m\Delta(G)}/\varepsilon)$ iterations which is significantly better than the bound for the Frank-Wolfe method. The algorithm scales to extremely large graphs and outperforms existing algorithms in both running time and the approximation quality on almost all data sets. The algorithm and analysis generalizes to hypergraphs.

Fractional Peeling: We introduce the idea of fractional peeling to round a fractional solution to the underlying LP/QP relaxation and show its effectiveness in theory and practice. In experiments it significantly outperforms ordering based rounding algorithms considered previously. We use it to obtain a provably approximate dense subgraph decomposition from a fractional solution.

Connections to DSS: We explicitly connect the result of Fujishige [34] on the existence of a lexicographically optimal base in a polymatroid with the problem of computing a densest decomposition of a supermodular function. We show that the a vertex load vector is feasible for the dual of Charikar's LP for DSG iff it is feasible for the base contrapolymatroid associated with the supermodular function $f(S) = |E(S)|$ induced by the graph. These connections clarify past results such as the one in [36, 2] and allow us to compare Frank-Wolfe versus projection based methods.

Experimental evaluation: We compare five algorithms for DSG, namely GREEDY++, FRANK-WOLFE, MWU, FISTA, and FISTA-PARALLEL a parallel version of FISTA. We test the algorithms on real world and synthetic data sets and show the effectiveness of FISTA when combined with fractional peeling.

To make the paper self contained we provide all proofs in the appendix, even for results implicitly contained in past literature. We focus on DSG (unweighted), however, the ideas generalize to hypergraphs and weighted graphs, and we leave a more in-depth exploration for future work.

3. Densest Subgraph Decomposition and Supermodularity

Let $G = (V, E)$ be an undirected graph. Each graph admits a unique nested decomposition of decreasing densities. This property is more transparently seen via supermodularity. Let $f : 2^V \to \mathbb{R}_+$ be a non-negative supermodular set function[1]. For example, fix $f(S) = |E(S)|$ which is supermodular. Supermodularity implies that there is a unique inclusion-wise *maximal* densest set S_1. It is convenient to describe the decomposition in an algorithmic fashion. The algorithm calculates the maximal set $S_1 \subseteq V_0 = V$ that maximizes $\frac{|E(S)|}{|S|}$ in G with density $\lambda_1 = \lambda^* = \frac{|E(S_1)|}{|S_1|}$. For DSS this

[1]We restrict attention to non-negative supermodular set functions that satisfy $f(\emptyset) = 0$ and this automatically also implies that they are monotone, that is, $f(A) \leq f(B)$ for $A \subset B$

corresponds to finding the unique maximal set S_1 that achieves the maximum density $\max_S \frac{f(S)}{|S|}$. In iteration i, letting $U_{i-1} = \cup_{1 \le j < i} S_j$, it calculates the maximal set $S_i \subseteq V_i = V - U_{i-1}$ that maximizes $(|E(S \cup U_{i-1}| - |E(U_{i-1})|)/|S|$. The algorithm is described formally in Algorithm 1.

Algorithm 1 Dense graph decomposition (left) and dense supermodular set decomposition (right)

$U_0 \leftarrow \emptyset, V_0 \leftarrow V, k \leftarrow 0$	$U_0 \leftarrow \emptyset, V_0 \leftarrow V, k \leftarrow 0$
while $V_k \neq \emptyset$ **do**	**while** $V_k \neq \emptyset$ **do**
$k \leftarrow k+1$	$k \leftarrow k+1$
$S_k \leftarrow \underset{\substack{S \subseteq V_{k-1} \\ S \text{ maximal}}}{\arg\max} \frac{\lvert E(S)\rvert + \lvert E(S, U_{k-1})\rvert}{\lvert S\rvert}$	$S_k \leftarrow \underset{\substack{S \subseteq V_{k-1} \\ S \text{ maximal}}}{\arg\max} \frac{f(S \cup U_{k-1}) - f(U_{k-1})}{\lvert S\rvert}$
$U_k \leftarrow U_{k-1} \cup S_k, V_k \leftarrow V_{k-1} - S_k$	$U_k \leftarrow U_{k-1} \cup S_k, V_k \leftarrow V_{k-1} - S_k$
return $S_1, ..., S_k$	**return** $S_1, ..., S_k$

The algorithm outputs a partition of V into $S_1, ..., S_k$, where k is the *dense decomposition depth* of G (or any supermodular function f). With each set S_i, we associate a density defined as $\lambda_i = \frac{|E(S_i)|+|E(S_i,U_{i-1})|}{|S_i|}$ (or in the case of f, $\lambda_i = \frac{f(S_i \cup U_{i-1}) - f(U_{i-1})}{|S_i|}$). In addition, for any $u \in S_i$, we say the density of u, $\lambda_u = \lambda_i$. Refer to Lemma 6.2 (appendix) for some basic properties of the dense decomposition. Specifically, the densities monotonically decrease, that is, $\lambda_1 > \lambda_2 > ... > \lambda_k$.

4. LP and QP for DSG and Decomposition

Charikar's [5] exact LP relaxation for DSG 4.1, and its dual 4.2 (in a slightly modified form), are given below. The primal has a variable y_u for each vertex $u \in V$ which indicates whether u is chosen in the densest set. For each edge $e = \{u, v\} \in E$ there is a variable z_e to indicate whether it is chosen. An edge $\{u, v\}$ can be chosen only if both u and v are chosen which explains the constraints linking z_e to y_u, y_v. The LP normalizes $\sum_u y_u$ to 1 for linearity, and maximizes chosen edges.

$$\begin{aligned} \max \sum_{e \in E} z_e & \qquad (4.1) \\ \text{s.t. } z_e &\le y_u \qquad \forall e = \{u,v\} \in E \\ z_e &\le y_v \qquad \forall e = \{u,v\} \in E \\ \sum_{u \in V} y_u &\le 1 \\ z_e, y_u &\ge 0 \end{aligned}$$

$$\begin{aligned} \min \max_{u \in V} b_u & \qquad (4.2) \\ \text{s.t. } \sum_{v \in \delta(u)} x_{uv} &= b_u \qquad \forall u \in G \\ x_{uv} + x_{vu} &= 1 \qquad \forall \{u,v\} \in E \\ x_{uv}, x_{vu}, b_u &\ge 0 \end{aligned}$$

The dual can be viewed as orienting each edge $\{u, v\}$ fractionally towards u and v. The orientation induces loads on the vertices and the goal is to find an orientation that minimizes the maximum load over vertices. It is not hard to see that LP 4.2 is actually the same as the dual of 4.1; the variables b_u can be replaced by a single variable b. The optimum value of LP 4.1 and LP 4.2 is the density $\lambda^* = \lambda_1$. However, we show that the b_u values have additional information.

Theorem 4.1. *Let $S_1, ..., S_k$ and $\lambda_1, ..., \lambda_k$ be the densest subgraph decomposition of a graph G, and for any $u \in S_i$, let $\lambda_u = \lambda_i$. There is an optimal solution (x^*, b^*) to 4.2 such that $b^*_u = \lambda_u$.*

We give two proofs in the appendix, one a direct proof using Rado's theorem [37], and a second by relating 4.2 to another relaxation and using a known result of Fujishige that is captured in Theorem 4.3. We say a vector $a \in \mathbb{R}^n$ is *lexicographically* smaller than vector $b \in \mathbb{R}^n$ if the sorted vector a (in descending order) is lexicographically smaller than the sorted vector b. Theorem 4.1 suggests that there exists a lexicographically least optimal solution to LP 4.2 where each vertex load b_u is precisely λ_u. To obtain the lexicographic solution, it suffices to introduce some strict convexity into the objective. Let $P(x, b)$ denote the polyhedron defined by the constraints in 4.2. Consider the quadratic program 4.3:

$$\min \sum_{u \in V} b_u^2 \text{ such that } (x, b) \in P(x, b) \qquad (4.3)$$

The theorem below was shown by Danisch *et al.* [3] but as we will show later, this is a special case of Fujishige's result [34] from 1980 via Theorem 4.3. See Appendix.

Theorem 4.2. *There is a unique optimum solution b^* to 4.3 and for each $u \in V(G), b^*_u = \lambda_u$.*

4.1. LP and QP for DSS and densest decomposition

We will now generalize Theorem 4.1 and Theorem 4.2 for the DSS problem. We start by recapping the notion of a contrapolymatroid (see [38]) which is the relevant notion for supermodular functions (as polymatroids are for submodular functions). For a normalized non-negative supermodular function $f : 2^V \to \mathbb{R}_+$, the contrapolymatroid with it is the following polyhedron

$$P_f = \{x \in \mathbb{R}^V \mid x \geq 0,\, x(S) \geq f(S) \text{ for all } S \subseteq V\} \tag{4.4}$$

A vector $x \in P_f$ is a base if $x(V) = f(V)$. The base contrapolymatroid is defined as:

$$B_f = \{x \in \mathbb{R}^V \mid x \geq 0,\, x(S) \geq f(S) \text{ for all } S \subseteq V,\, x(V) = f(V)\} \tag{4.5}$$

Now consider problem 4.6 where we are given a monotone non-negative supermodular function $f : 2^V \to \mathbb{R}_+$ and want to find the lexicographically minimal solution b^* for Problem 4.6. We will show that we can do this by instead solving Problem 4.7:

$$\begin{array}{ll} \text{minimize} & \max_{u \in V} b_u \\ \text{subject to} & b \in B_f \end{array} \tag{4.6}$$

$$\begin{array}{ll} \text{minimize} & \sum_{u \in V} b_u^2 \\ \text{subject to} & b \in B_f \end{array} \tag{4.7}$$

Theorem 4.3. *Let $S_1, ..., S_k$ and $\lambda_1, ..., \lambda_k$ be the densest supermodular set decomposition of f, and for any $u \in S_i$, let $\lambda_u = \lambda_i$. Then the following must hold*

1. *The solution b where $b_u = \lambda_u$ is feasible in the base polytope (i.e $b \in B_f$)*
2. *The lexicographically minimal solution b^* for Problem 4.6 satisfies $b_u = \lambda_u$*
3. *The optimal solution of 4.7, b^* is unique, and for each $u \in V, b^*_u = \lambda_u$*

We give a self-contained proof in the appendix and note that the theorem is implied by (essentially equivalent to) Fujishige's result [34] on the existence of a lexicographically optimal base of a polymatroid with respect to a weight vector. LP 4.2 and LP 4.6 can be related via the following theorem.

Theorem 4.4. *Consider a graph $G = (V, E)$ and the associated supermodular function $f : 2^V \to \mathbb{R}_+$ where $f(S) = |E(S)|$. A vector $b \in B_f$ if and only if there is an $x \in \mathbb{R}^{ord(E)}, x \geq 0$ such that the pair (x, b) satisfy the constraints of the LP 4.2.*

See appendix for a proof. This implies that Theorem 4.2 is a corollary of Theorem 4.3.

5. Solving the Quadratic Program using proximal projections, and rounding

In this section we show how to approximately solve 4.3. We let $f(x) = \sum_{u \in V} \left(\sum_{v \in \delta(u)} x_{uv} \right)^2$. Note that this is simply the objective function rewritten in terms of x. Similarly, let $h(x)$ be an indicator function where $h(x) = 0$ if $x_{uv} \geq 0$ and $x_{uv} + x_{vu} = 1, \forall (u, v) \in E$ and $+\infty$ otherwise. Then Problem 4.3 can be rewritten as minimizing the unconstrained objective $f(x) + h(x)$ for $x \in \mathbb{R}^{2m}$. We will use a proximal gradient method to solve the problem. For that, we need two lemmas.

Lemma 5.1. *The Lipschitz constant of ∇f is at most $2\Delta(G)$ where $\Delta(G)$ is the max degree of G.*

Lemma 5.2. *Let $x \in \mathbb{R}^{2m}$. Define the* proximal mapping *$prox_h(x)$ as the point $p \in \mathbb{R}^{2m}$ that minimizes $\|p - x\|^2$ such that $h(p) = 0$. Then we have that for $u < v$, $prox_h(x)_{uv} = \frac{x_{uv} - x_{vu} + 1}{2}$ if $|x_{uv} - x_{vu}| \leq 1$, $prox_h(x)_{uv} = 1$ if $x_{uv} - x_{vu} > 1$, and $prox_h(x)_{uv} = 0$ if $x_{uv} - x_{vu} < -1$. Additionally, $prox_h(x)_{vu} = 1 - prox_h(x)_{uv}$.*

We present the algorithm now. We are interested in the unconstrained optimization problem of minimizing $f(x) + h(x)$ where f is convex, and h has an easy to compute proximal mapping. This

type of problem can be solved using proximal gradient methods. From a high level, the (basic) algorithm is described Algorithm 2. At any iteration t it has a guess for the minimizer $x^{(t)}$. It then calculates the gradient of f and moves slightly against it. However, since this might make the new guess infeasible, it uses the proximal mapping to project the new guess to a feasible solution.

Algorithm 2 Basic Proximal Gradient Method (left) and accelerated FISTA (right)

Input: f and h with $\text{prox}_h(x)$, learning rate α and iterations T. Initialize $x^{(0)}$ with $h(x^{(0)}) = 0$

for $t \in [1, T]$ **do**
  $x^{(t)} = \text{prox}_h(x^{(t-1)} - \alpha \nabla f(x^{(t-1)}))$
return $x^{(T)}$

$y^{(0)} = x^{(0)}$
for $t \in [1, T]$ **do**
  $x^{(t)} = \text{prox}_h(y^{(t-1)} - \alpha \nabla f(y^{(k-1)}))$
  $y^{(t)} = x^{(t)} + \frac{t-1}{t+2}(x^{(t)} - x^{(t-1)})$
return $x^{(T)}$

While the basic proximal gradient method works, we will use an even faster (both theoretically and practically) version known as the accelerated proximal gradient method which incorporates Nesterov-like momentum terms [39] in the projection step. It has other names in the literature such as proximal gradient method with extrapolation and FISTA [40]. The algorithm is outlined in Algorithm 2. We have the following known result on the FISTA algorithm.

Lemma 5.3. *[40]. Let x^* be the minimizer of f. Suppose that the learning rate satisfies $\alpha \leq \frac{1}{L(f)}$ where $L(f)$ is the Lipschitz constant of ∇f. Then after k iterations, $f(x^{(k)}) - f(x^*) \leq \frac{2\left\|x^{(0)} - x^*\right\|^2}{\alpha k^2}$.*

In our case, we bounded the Lipschitz constant to $2\Delta(G)$ and an easy upper bound on $\left\|x^{(0)} - x^*\right\|^2$ is $2m = 2|E(G)|$. Combining this with a learning rate of $\frac{1}{2\Delta(G)}$, we get the following result

Lemma 5.4. *If FISTA is applied to the objective $f(x) + h(x)$ as previously defined, then in the kth iteration, we have $f(x^{(k)}) - f(x^*) \leq \frac{8\Delta(G)m}{k^2}$.*

The final FISTA algorithm for our problem is shown in Algorithm 3 in the Appendix with the correct gradient computation, projective mappings, and iteration updates. Lastly, to get a good additive approximation on each λ_u, we have the following result

Theorem 5.5. *Let $b_u^* = \lambda_u$. After $t = O(\frac{\sqrt{\Delta(G)m}}{\epsilon})$ iterations of FISTA, we must have $\left\|b^{(t)} - b^*\right\| \leq \epsilon$. This trivially implies $\left|b_u^{(t)} - \lambda_u\right| \leq \epsilon$ for all $u \in V$*

5.1. Fractional Peeling.

Given an approximate solution (b, x) to Problem 4.3, how do we round it to obtain a good densest decomposition? The natural approach is to sort the vertices in non-increasing values of b and take suffixes. This is used in [36]. This is an exact rounding algorithm when b is an *optimum* solution, however one can construct approximate solutions for which this rounding is not ideal. We describe a peeling algorithm inspired by GREEDY++ [1] that takes advantage of the auxiliary information provided by the vector x. First, set $b' = b$ and $G^{(0)} = G$. In iteration, $t \geq 1$, peel the vertex u with minimum current load b'_u. Then, for each $v \in \delta_{G^{t-1}}(u)$, set $b'_v \leftarrow b'_v - x_{vu}$ (i.e subtract the fractional value of x_{vu} from v's load). Update $G^t = G^{t-1} - u$. Repeat this process to obtain $G^{(0)}, ..., G^{(n)}$. Finally, return a graph $G^{(i)}$ with maximum density. The algorithm can be implemented in $O(m + n \log n)$ using a Fibonacci heap. We will refer to this process as *fractional peeling*. We show both theoretically and experimentally that fractional peeling leads to better algorithms.

5.2. $\epsilon-$dense local decomposition

We show that fractional peeling can be used on an approximate solution to 4.3 to obtain an approximate dense decomposition with a theoretical guarantee. This is in contrast to previous work in [36]. We define a strong notion of approximate decomposition. Given a solution (b, x) to 4.3, we say $\hat{b}$ is an *ϵ-load-vector* if $\left\|\hat{b} - b^*\right\| \leq \epsilon$. Given a partition of the vertices $T_1, ..., T_r$, we say the partition is

an ϵ-*approximate dense decomposition* to $S_1, ..., S_k$ (the true dense decomposition) if

$$u \in S_i, u \in T_h \implies \frac{|E(T_h)| + |E(T_h, \cup_{j<h} T_j)|}{|T_h|} \geq \frac{|E(S_i)| + |E(S_i, \cup_{j<i} S_j)|}{|S_i|} - \epsilon$$

Intuitively, what this says is that every $u \in V$ belongs to a set T_h that has a density that is not much less than λ_u.

Theorem 5.6. *Given an ϵ load vector b and an edge vector x that induces b, we can calculate an $\epsilon\sqrt{n}$-approximate dense decomposition in $\tilde{O}(mn)$ time.*

Note that the notion of error is additive and holds for every vertex u. Although the $\sqrt{n}$-factor is large, the analysis shows that one can usually obtain a much stronger bound. Qualitatively it shows that fractional peeling leads to good dense decomposition as $\varepsilon \to 0$.

5.3. Projections vs Frank-Wolfe vs MWU for DSG and DSS

The Frank-Wolfe method is natural to apply to solve 4.7 since each iteration requires optimizing a linear objective over the base contrapolymatroid B_f; this is easy and fast via the greedy algorithm, as shown originally by Edmonds in the context of polymatroids (see [38]). Danisch *et al.* work with 4.3. However, as we observed earlier, 4.3 is a compact way to represent the associated base polyhedron, and hence the algorithm in [36] is the same as the Frank-Wolfe algorithm applied to 4.7. The Fujishige-Wolfe minimum norm point algorithm for submodular function minimization is related but is based on Wolfe's method (see [41, 42]). The fact that the optimum point in the base polytope has additional information was already pointed out in [34], and also explored in the context of size-constrained submodular function minimization by Nagano *et al.* [33]. The advantage of proximal gradient methods such as FISTA is their faster convergence rates when compared to the Frank-Wolfe method, although each iteration requires a projection oracle for B_f. Our algorithm is based on the observation that there is an $O(m)$-time projection oracle for DSG due to the alternate characterization of B_f via the edge variable LP 4.2. This same observation holds for hypergraphs and we obtain a fast algorithm for them. We outline the formal details in the appendix. The multiplicative weight update (MWU) method is another broad methodology for solving linear and convex programs and there are several variants. One variant of MWU for solving 4.2 can be interpreted as a Frank-Wolfe method with a certain convex objective and with a certain step size; each iteration of this algorithm is again a greedy optimization over B_f. We provide more details in the Appendix 6.7.

5.4. A theoretically fast algorithm for approximate load vector via min-cost flow

While Algorithm 3 (detailed in the appendix) works extremely well in practice, there is room to improve the theoretical run time. Each iteration takes $O(m)$ time and can easily be parallelized. However, in the worst-case it would need $O(m\sqrt{\Delta(G)m}/\epsilon)$ time to get an additive ϵ-approximation. Can we obtain a theoretically faster algorithm? In the Appendix we describe a reduction to computing *exact* minimum-cost flow in a directed network. Via the new breakthrough near-linear time algorithm for min-cost flows [43], we obtain an $O(\frac{m\sqrt{n}}{\epsilon})^{1+o(1)})$ time algorithm to get an ϵ-approximate load vector. We hope that this theoretical result will inspire the development of new algorithms that are provably faster in theory while also having good empirical performance.

6. Experimental Evaluation

Datasets & Implementation Details. We ran experiments on 7 real world datasets (6 from the SNAP database [44], and 1 from [45]), and one tailored synthetic dataset (used for clarifying an important difference between all algorithms) for a total of 8 datasets. The dataset information is summarized in the table below. The CLOSE-CLIQUES dataset consists of the complete bipartite graph $K_{d,D}$ for $d = 30$ and $D = 2000$, and 20 copies of the complete graph K_h where $h = 60$.

We tested 5 algorithms to approximate DSG: The FISTA based algorithm (sequential, no parallelism), FISTA-PARALLEL, GREEDY++ from [1], the FRANK-WOLFE based algorithm from [3], and the MWU algorithm respectively. See supplementary section for implementation details of the 5 algorithms. Also see appendix for details on the specific variant of MWU we tested. All algorithms were implemented in C++17 and were compiled with O3 and UNROLL-LOOPS optimizations. The

Dataset	Vertices	Edges	Source
cit-Patents	3,774,768	16,518,947	[44]
com-Amazon	334,863	925,872	[44]
orkut	3,072,441	117,185,083	[45]
roadNet-PA	1,088,092	1,541,898	[44]

Dataset	Vertices	Edges	Source
roadNet-CA	1,965,206	2,766,607	[44]
Close-Cliques	3,230	95,400	Synthetic
dblp-author	317,080	1,049,866	[44]
wiki-topcats	1,791,489	28,508,141	[44]

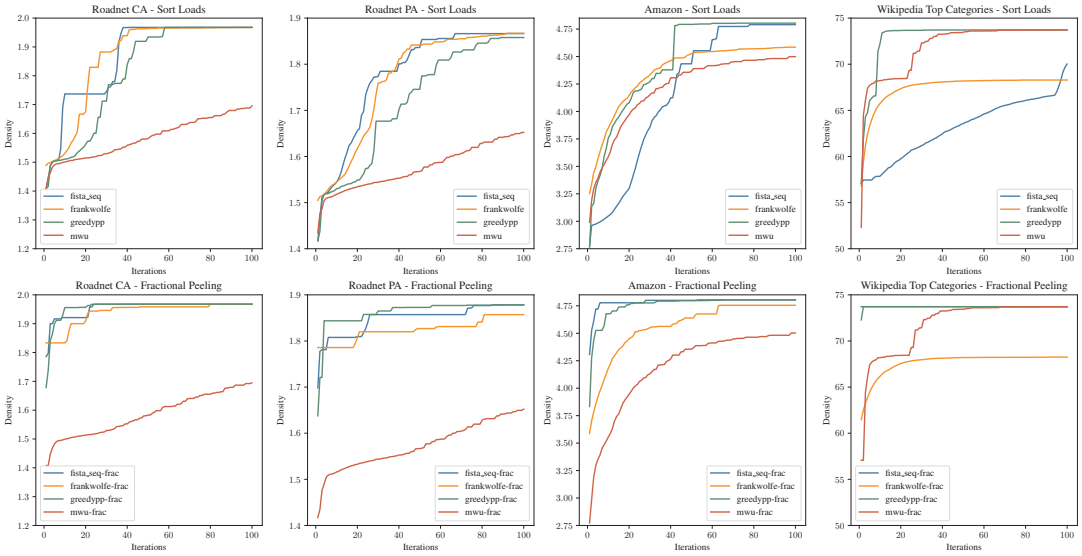

Figure 6.1: Density based on sorting loads vs fractional peeling.

implementations of Algorithms FISTA, FISTA-PARALLEL, FRANK-WOLFE, and MWU are the authors' implementations, but we used the original implementation for GREEDY++ [1] as it was extremely well optimized. We modified their implementation minimally to log basic information needed for the evaluation. FISTA-PARALLEL used Open MPI [46] for parallelism. We ran our experiments on a Slurm-based university campus cluster. For all machines, we requested 1 node and 16 cores per experiment. The nodes had 64 GB of RAM and Xeon PHI 5100 CPUs.

Densities. In the first experiment, we ran all the algorithms for a number of iterations, and monitored the maximum density reached by the algorithm until iteration t. The density in each iteration was calculated by statically sorting the vertices by the value of their load vector, and peeling in that order and returning the maximum density subgraph in that iteration. The result is shown in Figure 6.1 (Top 4 plots, we only show 4 datasets, remaining plots in appendix). Specifically, this does not use the fractional peeling technique we discussed. We also plot the maximum density reached by using fractional peeling instead (Bottom 4 plots in 6.1, we show 4 datasets, remaining plots in appendix).

When the peeling order is changed from a static peel (based on the load vector value) to a fractional peel, the results improve dramatically. See Figure 6.1 and appendix. In all algorithms (except MWU), fractional peeling leads to a dramatic speedup in terms of the number of iterations needed to get the densest subgraph on almost all datasets. Fractional peeling provides little benefit to MWU. We can observe that for all the real world datasets, GREEDY++ and FISTA are extremely competitive and reach near-optimal densities in just a few iterations. Meanwhile, the FRANK-WOLFE based algorithm lagged behind in the beginning, but steadily made progress towards the maximum density in later iterations. MWU strongly lagged behind all algorithms in several datasets. The only exception is for the CLOSE-CLIQUES dataset (see Appendix). The component in the densest subgraph has density $(30 \times 2000)/(30 + 2000) \approx 29.5566$. Meanwhile, the remaining components have density $\binom{60}{2}/60 = 29.5$. FISTA, FRANK-WOLFE, and MWU eventually found the densest subgraph in the first 100 iterations, but GREEDY++ was stuck in a local optima for 200 iterations! Note that the example $K_{d,D}$ and r copies of K_{2d} (for sufficiently large r, D) can be used to theoretically show that GREEDY++ requires $\Omega(\frac{1}{\epsilon})$ iterations to converge to a $(1 - \epsilon)$ approximation for DSG. MWU

and FRANK-WOLFE did better for this synthetic example where the densities are very close and it is worth understanding in more detail.

Wall clock time and run-time per iteration. In the second experiment we examine the wall-clock time and time per iteration. We study the maximum density reached by an algorithm after T seconds of wall-clock time (cumulative time of all iterations). Figure 6.12 (appendix) and last 4 plots of Figure 6.2 shows the result. As can be seen, FISTA-PARALLEL and FISTA (sequential) finds the maximum density on almost all datasets in the least wall-clock time (albeit sequential FISTA and FRANK-WOLFE are close runners if we only restrict to non-parallel implementations). On the other hand, MWU performed poorly in wall-clock time, often taking orders of magnitude longer than FISTA or GREEDY++ to find suboptimal dense subgraphs. Overall, the number of iterations of FISTA is the lowest, and each iteration is fast and can be parallelized, so it is the best performer.

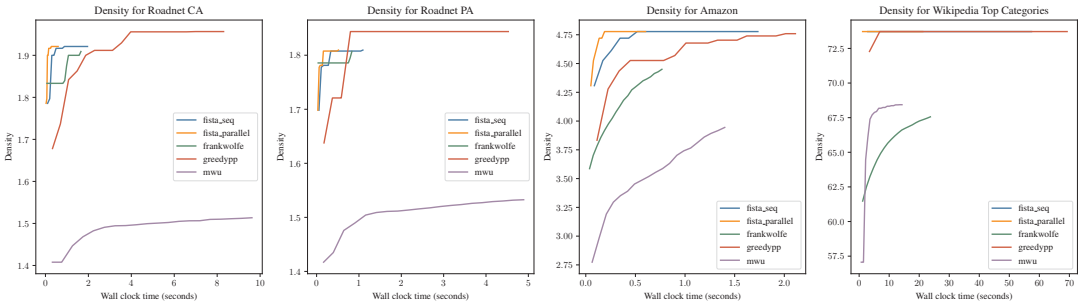

Figure 6.2: Wall clock time of algorithms.

We also considered the run time per iteration for the algorithms. Figure 6.8 (appendix) shows the result. The per iteration time depends on several factors. Although all algorithms have linear or near-linear in m running time, the specific data structure used and cache performance can have substantial impact. Overall, FRANK-WOLFE (due to its simple implementation) and FISTA-PARALLEL had the best per-iteration performance. The average speedup *per iteration* from FISTA-PARALLEL over GREEDY++ was roughly 5 fold. Note that due to the peeling nature of GREEDY++, it cannot be parallelized like FISTA. We believe that there is further room to improve the memory and cache performance of the FISTA-based algorithm by using only one variable for edge rather than two. This can further improve the overall wall-clock time of FISTA and FISTA-PARALLEL.

Convergence to optimal load vector. Recall that we are minimizing $\sum_u b_u^2$, and hence the norm $\left\|b^{(t)}\right\|$ gives a proxy for the convergence of the errors in $b^{(t)}$. In this experiment, we plotted the norm of the vector $b^{(t)}$ for each algorithm in each dataset. The result is shown in Figure 6.9 (Appendix). GREEDY++ does very well in the first few iterations, but slows down in its improvement on $\left\|b^{(t)}\right\|$ as the number of iterations increases. In comparison, FISTA eventually surpasses GREEDY++ and reduces the error at a faster rate than GREEDY++. In comparison, FRANK-WOLFE and MWU start with a substantial error in the b vector but quickly reduce it, however, their error was always worse that both GREEDY++ and FISTA in all iterations even when left to run up to 200 iterations.

Conclusion. We introduced a new iterative algorithm for the densest subgraph and densest decomposition problems. We also described a new fractional peeling technique which has strong empirical performance as well as theoretical guarantees and showed experimentally how it improved almost all existing algorithms compared to static load sorting. The new algorithm is scalable and simple, and can be applied to graphs with hundreds of millions of edges. Our experiments support the theory established on the utility of the new algorithm and fractional peeling. Our work also adds value via a detailed comparison of the practical performance of existing algorithms and the new algorithm. A few limitations remain for the paper. First, reducing the $\sqrt{n}$ dependency in Theorem 5.6 is important, even if fractional peeling shows strong experimental bounds. Finally, we need to evaluate empirically the quality of the decomposition achieved by algorithms (with and without fractional peeling), and also algorithms for approximating at least k DSG and related problems via the decomposition.

References

[1] Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. *Flowless: Extracting Densest Subgraphs Without Flow Computations*, page 573–583. Association for Computing Machinery, New York, NY, USA, 2020.

[2] Chandra Chekuri, Kent Quanrud, and Manuel R. Torres. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1531–1555.

[3] Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 233–242, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[4] A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.

[5] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In Klaus Jansen and Samir Khuller, editors, *Approximation Algorithms for Combinatorial Optimization*, pages 84–95, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[6] Samir Khuller and Barna Saha. On finding dense subgraphs. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 597–608, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[7] Chenhao Ma, Yixiang Fang, Reynold Cheng, Laks V.S. Lakshmanan, Wenjie Zhang, and Xuemin Lin. Efficient algorithms for densest subgraph discovery on large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1051–1066, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1122–1132, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[9] Bintao Sun, Maximilien Danisch, T-H. Hubert Chan, and Mauro Sozio. Kclist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proc. VLDB Endow.*, 13(10):1628–1640, jun 2020.

[10] Reid Andersen and Kumar Chellapilla. Finding dense subgraphs with size bounds. In Konstantin Avrachenkov, Debora Donato, and Nelly Litvak, editors, *Algorithms and Models for the Web-Graph*, pages 25–37, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[11] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria 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*, KDD '13, page 104–112, New York, NY, USA, 2013. Association for Computing Machinery.

[12] Nikolaj Tatti. Density-friendly graph decomposition. *ACM Trans. Knowl. Discov. Data*, 13(5), sep 2019.

[13] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proc. VLDB Endow.*, 5(5):454–465, jan 2012.

[14] Francesco Bonchi, David García-Soriano, Atsushi Miyauchi, and Charalampos E. Tsourakakis. Finding densest k-connected subgraphs. *Discrete Appl. Math.*, 305(C):34–47, dec 2022.

[15] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 300–310, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[16] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald T. Sannella, editors, *Mathematical Foundations of Computer Science 2015*, pages 472–482, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[17] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos 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*, STOC '15, page 173–182, New York, NY, USA, 2015. Association for Computing Machinery.

[18] Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. Discovering dynamic communities in interaction networks. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 678–693, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[19] Oana Denisa Balalau, Francesco Bonchi, T-H. Hubert Chan, Francesco Gullo, and Mauro Sozio. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, page 379–388, New York, NY, USA, 2015. Association for Computing Machinery.

[20] Yuko Kuroki, Atsushi Miyauchi, Junya Honda, and Masashi Sugiyama. Online dense subgraph discovery via blurred-graph feedback. In *ICML*, 2020.

[21] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proc. VLDB Endow.*, 5(6):574–585, feb 2012.

[22] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Corescope: Graph mining using k-core analysis — patterns, anomalies and algorithms. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 469–478, 2016.

[23] Xiangfeng Li, Shenghua Liu, Zifeng Li, Xiaotian Han, Chuan Shi, Bryan Hooi, He Huang, and Xueqi Cheng. Flowscope: Spotting money laundering based on graphs. In *AAAI*, 2020.

[24] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities – an O(n 1/4) approximation for densest k-subgraph.

[25] U. Feige, D. Peleg, and G. Kortsarz. The dense k -subgraph problem. *Algorithmica*, 29(3):410–421, mar 2001.

[26] Subhash Khot. Ruling out ptas for graph min-bisection, dense $k-$subgraph, and bipartite clique. *SIAM Journal on Computing*, 36(4):1025–1071, 2006.

[27] Pasin Manurangsi. Inapproximability of maximum biclique problems, minimum k-cut and densest at-least-k-subgraph from the small set expansion hypothesis. *ArXiv*, abs/1705.03581, 2018.

[28] Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982.

[29] Fragkiskos D Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, and Michalis Vazirgiannis. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal*, 29(1):61–92, 2020.

[30] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for mapreduce. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–78. Springer, 2014.

[31] Digvijay Boob, Saurabh Sawlani, and Di Wang. Faster width-dependent algorithm for mixed packing and covering lps. *Advances in Neural Information Processing Systems 32 (NIPS 2019)*, 2019.

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

[33] Kiyohito Nagano, Yoshinobu Kawahara, and Kazuyuki Aihara. Size-constrained submodular minimization through minimum norm base. In *ICML*, 2011.

[34] Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980.

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

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

[37] R. Rado. Theorems on linear combinatorial topology and general measure. *Annals of Mathematics*, 44(2):228–270, 1943.

[38] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[39] Yurii Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[40] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[41] Philip Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–149, 1976.

[42] Satoru Fujishige and Shigueo Isotani. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7(1):3–17, 2011.

[43] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time, 2022.

[44] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[45] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, page 1343–1350, New York, NY, USA, 2013. Association for Computing Machinery.

[46] Richard L. Graham, Timothy S. Woodall, and Jeffrey M. Squyres. Open mpi: A flexible high performance mpi. In Roman Wyrzykowski, Jack Dongarra, Norbert Meyer, and Jerzy Waśniewski, editors, *Parallel Processing and Applied Mathematics*, pages 228–239, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[47] Martin Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *International Conference on Machine Learning*, pages 427–435. PMLR, 2013.

[48] Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[49] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012.

[50] Weiran Wang and Miguel Á. Carreira-Perpiñán. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application, 2013.

[51] Franck Iutzeler and Laurent Condat. Distributed projection on the simplex and ℓ_1 ball via admm and gossip. *IEEE Signal Processing Letters*, 25(11):1650–1654, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In the experimental discussion, we describe the limitations of different algorithms (including ours) on the datasets
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Work is mainly theoretical with an applied flavor and doesn't involve inherent potential societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs are included in Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is in supplemental section, Data available online (not our data to include)
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A] The algorithm isn't randomized, no random seed or error plots applicable.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Code for Greedy++ was cited appropriately together with used libraries. All the sources data were also cited.
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] No personal data of people was used
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Personal data is not used
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Crowdsourcing is not used
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Crowdsourcing is not used
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Crowdsourcing is not used