Convergence to Lexicographically Optimal Base in a (Contra)Polymatroid and Applications to Densest Subgraph and Tree Packing

Elfarouk Harb ✉
University of Illinois at Urbana Champaign, IL, USA

Kent Quanrud ✉
Purdue University, West Lafayette, IN, USA

Chandra Chekuri ✉
University of Illinois at Urbana Champaign, IL, USA

Abstract

Boob et al. [7] described an iterative peeling algorithm called Greedy++ for the Densest Subgraph Problem (DSG) and conjectured that it converges to an optimum solution. Chekuri, Qaunrud and Torres [10] extended the algorithm to supermodular density problems (of which DSG is a special case) and proved that the resulting algorithm Super-Greedy++ (and hence also Greedy++) converges. In this paper we revisit the convergence proof and provide a different perspective. This is done via a connection to Fujishige's quadratic program for finding a lexicographically optimal base in a (contra) polymatroid [18], and a noisy version of the Frank-Wolfe method from convex optimization [17, 25]. This yields a simpler convergence proof, and also shows a stronger property that Super-Greedy++ converges to the optimal dense decomposition vector, answering a question raised in Harb et al. [24]. A second contribution of the paper is to understand Thorup's work on ideal tree packing and greedy tree packing [46, 47] via the Frank-Wolfe algorithm applied to find a lexicographically optimum base in the graphic matroid. This yields a simpler and transparent proof. The two results appear disparate but are unified via Fujishige's result and convex optimization.

2012 ACM Subject Classification Networks → Network algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Polymatroid, lexicographically optimum base, densest subgraph, tree packing

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.56

Related Version *Full Version*: https://arxiv.org/abs/2305.02987

Funding *Elfarouk Harb*: Supported in part by NSF grants CCF-2028861 and CCF-1910149.
Kent Quanrud: Supported in part by NSF grant CCF-2129816.
Chandra Chekuri: Supported in part by NSF grants CCF-2028861 and CCF-1910149.

1 Introduction

In this paper we consider iterative greedy algorithms for two different combinatorial optimization problems and show that the convergence of these algorithms can be understood by combining two general tools, one coming from the theory of submodular functions, and the other coming from convex optimization. This yields simpler proofs via a unified perspective, while also yielding additional properties that were previously unknown.

Densest subgraph and supermodularity. We start with the problem that motivated this work, namely, the densest subgraph problem (DSG). The input to 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 $E(S) = \{uv \in E : u, v \in S\}$ is the set of edges with both end points

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 56; pp. 56:1–56:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in S. Throughout the paper, we let $\lambda(G) = \frac{|E(G)|}{|V(G)|}$ denote the density of graph $G(V, E)$. We treat the unweighted case for simplicity; all the results generalize to edge-weighted graphs. Goldberg [22] and Picard and Queyranne [37] showed that DSG can be efficiently solved via a reduction to the *s*-*t* maximum-flow problem.

A different connection that shows polynomial-time solvability of DSG is important to this paper. Consider a real-valued set function $f : 2^V \to \mathbb{R}_+$ defined over the vertex set V, where $f(S) = |E(S)|$. This function is *supermodular*. A function f is supermodular iff $-f$ is *submodular*. A real-valued set function $f : 2^V \to \mathbb{R}$ is submodular iff $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subseteq B$. Submodular and supermodular set functions are fundamental in combinatorial optimization – see [41, 19].

Coming back to DSG, maximizing $|E(S)|/|S|$ is equivalent to finding the largest λ such that $\lambda|S| - |E(S)| \geq 0$ for all $S \subseteq V$. This corresponds to minimizing the submodular set function g where $g(S) = \lambda|S| - |E(S)|$. A classical result in combinatorial optimization is that the minimum of a submodular set function can be found in polynomial-time in the value oracle setting [41]. Thus, DSG can be solved via reduction to submodular set function minimization and binary search. The preceding connection also motivates the definition of a generalization of DSG called the densest supermodular set problem (DSS) [10]. The input is a non-negative supermodular function $f : 2^V \to \Re_+$, and the goal is to find $S \subseteq V$ that maximizes $\frac{f(S)}{|S|}$. DSS is polynomial-time solvable via submodular set function minimization. DSG, DSS and its variants have several applications in practice, and they are routinely used in graph and network analysis to find dense clusters or communities. We refer the reader to the extensive literature on this topic [32, 7, 14, 48, 43, 1, 49, 16, 34, 39, 6, 27, 2, 42, 30, 28]. DSG is also of interest in algorithms via its connection to arboricity and related notions – see [40, 13] for recent work.

Faster algorithms, Greedy and Greedy++. Although DSG is polynomial-time solvable via maxflow or submodular function minimization, the corresponding algorithms are not yet practical for the large graphs that arise in many applications; this is despite the fact that we now have very fast theoretical algorithms for maxflow and mincost flow [12]. For this reason there has been considerable interest in fast (approximation) algorithms. More than 20 years ago Charikar [9] showed that a simple "peeling" algorithm (Greedy) yields a 1/2-approximation for DSG. An ordering of the vertices as $v_{i_1}, v_{i_2}, \ldots, v_{i_n}$ is computed as follows: v_{i_1} is a vertex of minimum degree in G (ties broken arbitrarily), v_{i_2} is a minimum degree vertex in $G - v_{i_1}$ and so on[1]. After creating the ordering, the algorithm picks the best suffix, in terms of density, among the n-possible suffixes of the ordering. Charikar also developed a simple exact LP relaxation for DSG. Charikar's results have been quite influential. Greedy can be implemented in (near)-linear time and has also been adapted to other variants. The LP relaxation has also been used in several algorithms that yield a $(1 - \epsilon)$-approximate solution [5, 8], and has led to a flow-based $(1 - \epsilon)$-approximation [10]. More recently, Boob et al. [7] developed an algorithm called Greedy++ that is based on combining Greedy with ideas from multiplicative weight updates (MWU); the algorithm repeatedly applies a simple peeling algorithm with the first iteration coinciding with Greedy but later iterations depending on a weight vector that is maintained on the vertices – the formal algorithm is described in a later section. The advantage of the algorithm is its simplicity, and Boob et al. [7] showed that it has good empirical performance. Moreover

[1] This peeling order is the same as the one used to create the so-called core decomposition of a graph [33] and the Greedy algorithm itself was suggested by Asahiro et al. [4].

they conjectured that Greedy++ converges to a $(1-\epsilon)$-approximation in $O(1/\epsilon^2)$ iterations. Although their strong conjecture is yet unverified, Chekuri, Quanrud and Torres [10] proved that Greedy++ converges in $O(\frac{\Delta \log |V|}{\epsilon^2 \lambda^*(G)})$ iterations where Δ is the maximum degree of G and $\lambda^*(G)$ is the optimum density.

The convergence proof in [10] is non-trivial and relies crucially in considering DSS and supermodularity. [10] shows that Greedy and Greedy++ can be generalized to SuperGreedy and SuperGreedy++ for DSS, and that SuperGreedy++ converges to a $(1-\epsilon)$-approximation solution in $O(\alpha_f/\epsilon^2)$ iterations where α_f depends (only) on the function f.

Dense subgraph decomposition and connections. As we discussed, DSG is a special case of DSS and hence DSG inherits certain nice structural properties from supermodularity. One of these is the fact that the vertex set V of every graph $G=(V,E)$ admits a unique decomposition into $S_1, S_2, \ldots, S_k$ for some k using the following procedure: S_1 is the vertex set of the *unique maximal* densest subgraph, S_2 is the unique maximal densest subgraph after "contracting" S_1, and so on. The existence of such a unique decomposition is more transparent in the setting of DSS. The fact that there is a *unique* maximal densest set S_1 follows from supermodularity; if A and B are optimum dense sets then so is $A \cup B$. One can then consider a new supermodular function $f_{S_1} : 2^{V-S_1} \to \mathbb{R}$ defined over $V - S_1$ where $f_{S_1}(A) = f(S_1 \cup A) - f(S_1)$ for all $A \subseteq V - S_1$. The new function is also supermodular. Then S_2 is the unique maximal densest set for f_{S_1}. We iterate this process until we obtain an empty set. The decomposition also allows us to assign a density value λ_v to each $v \in V$ (which corresponds to the density of the set when v is in the maximal set). We call this the density vector associated with f. Dense decompositions follow from the theory of principal partitions of submodular functions [35, 36, 20]. In the context of graphs and DSG this was rediscovered by Tatti and Gionis who called it the locally-dense decomposition [45, 44], and gave algorithms for computing it. Subsequently, Danisch et al. [14] applied the well-known Frank-Wolfe algorithm for constrained convex optimization to a quadratic program derived from Charikar's LP relaxation for DSG. More recently, Harb et al. [24] obtained faster algorithms for computing the dense decomposition in graphs via Charikar's LP; they used a different method called FISTA for constrained convex optimization based on acceleration. Although DSS was not the main focus, [24] also made an important connection to Fujishige's result on lexicographically optimal base in polymatroids [18] which elucidated the work of Danisch et al. on DSG. We describe this next.

Lexicographical optimal base and dense decomposition. We briefly describe Fujishige's result [18] and its connection to dense decompositions. Let $f : 2^V \to \mathbb{R}_+$ be a monotone submodular set function ($f(A) \le f(B)$ if $A \subset B$) that is also normalized ($f(\emptyset) = 0$). Following Edmonds, the polymatroid associated with f, denote by P_f is the polyhedron $\{x \in \mathbb{R}^V \mid x \ge 0, x(S) \le f(S) \quad \forall S \subseteq V\}$, where $x(S) = \sum_{i \in S} x_i$. The base polyhedron associated with f, denote by B_f, is the polyhedron $P_f \cap \{x \in \mathbb{R}^V \mid x(V) = f(V)\}$ obtained by intersecting P_f with the equality constraint $x(V) = f(V)$. Each vector x in B_f is called a base. If f is a monotone normalized *supermodular* function, we consider the contrapolymatroid $P_f = \{x \in \mathbb{R}^V \mid x \ge 0, x(S) \ge f(S) \quad \forall S \subseteq V\}$ (the inequalities are reversed), and similarly B_f is the base contrapolymatroid obtained by intersecting P_f with equality constraint $x(V) = f(V)$. Fujishige proved that there exists a unique lexicographically minimal base in any polymatroid, and morover it can found by solving the quadratic program: $\min \sum_v x_v^2$ s.t $x \in B_f$. In the context of supermodular functions, one obtains a similar result;

the quadratic program $\min \sum_v x_v^2$ s.t $x \in B_f$ where B_f is contrapolymatroid associated with f has a unique solution. As observed explicity in [24], the lexicographically optimal base gives the dense decomposition vector for DSS. That is, if x^* is the optimal solution to the quadratic program then for each v, $x_v^* = \lambda_v$. In particular, as noted in [24], one can apply the well-known Frank-Wolfe algorithm to the quadratic program and it converges to the dense decomposition vector. As we will see later, each iteration corresponds to finding a maximum weight base in a contrapolymatroid which is easy to find via the greedy algorithm.

(Ideal) Tree packings in graphs and the Tutte–Nash-Williams theorem. Our discussion so far focused on DSG. Now we describe a different problem on graphs and relevant background. Our goal is to present a unified perspective on these two problems. The well-known Tutte–Nash-Williams theorem in graph theory (see [41]) establishes a min-max result for the maximum number of edge-disjoint spanning trees in a multi-graph G. Given an undirected graph $G = (V, E)$, and a partition P of the vertices, let $E(P)$ denote the set of edges crossing the partition. The strength of a partition P is defined as $\frac{|E(P)|}{|P|-1}$. Let $\mathcal{T}(G)$ denote all possible spanning trees of G. Let $\tau^*(G)$ denote the maximum number of edge-disjoint spanning trees in G. Then $\tau^*(G) = \min_P \lfloor \frac{|E(P)|}{|P|-1} \rfloor$. Further, if we define $\tau(G)$ to be the maximum *fractional* packing of spanning trees, then the floor can be removed and we have $\tau(G) = \min_P \frac{|E(P)|}{|P|-1}$. We note that the graph theoretic result is a special case of matroid base packing. Tree packings are useful for a number of applications. In particular, Karger [26] used tree packings and other ideas in his well-known near-linear randomized algorithm for computing the global minimum cut of a graph. We are mainly concerned here with Thorup's work in [46, 47] that was motivated by dynamic mincut and k-cut problems. He defined the so-called *ideal* edge loads and ideal tree packing (details in later section) by recursively decomposing the graph via Tutte–Nash-Williams partitions [46]. He also proved that a simple iterative greedy tree packing algorithm converges to the ideal loads [47]. He used the approximate ideal tree packing to obtain new deterministic algorithms for the k-cut problem, and his approach has been quite influential in a number of subsequent results [21, 11, 31, 29, 23]. Thorup obtained his tree packing result from first principles. We ask: is there a connection between ideal tree packing and DSG?

1.1 Contributions of the paper

This paper has two main contributions. The first is a new proof of the convergence of SuperGreedy++ for DSS. Our proof is based on showing that SuperGreedy++ can be viewed as a "noisy" or "approximate" variant of the Frank-Wolfe algorithm applied to the quadratic program defined by Fujishige. The advantage of the new proof is twofold. First, it shows that SuperGreedy++ not only converges to a $(1 - \epsilon)$-approximation to the densest set, but that in fact it converges to the densest decomposition vector. This was empirically observed in [24] for DSG, and was left as an open problem to resolve. The proof in [10] on convergence of SuperGreedy++ is based on the MWU method via LPs, and does not exploit Fujishige's result which is key to the stronger property that we prove here. Second, the proof connects two powerful tools directly and at a high-level: Fujishige's result on submodular functions, and a standard method for constrained convex optimization.

▶ **Theorem 1.** *Let b^* be the dense decomposition vector for a non-negative monotone supermodular set function $f : 2^V \to \mathbb{R}_+$ where $|V| = n$. Then, SuperGreedy++ converges in $O(\alpha_f/\epsilon^2)$ iterations to a vector b such that $||b - b^*||_2 \leq \epsilon$, where α_f depends only on f. For a graph with m edges and n vertices, Greedy++ converges in $O(mn^2/\epsilon^2)$ iterations for unweighted multigraphs.*

▶ Remark 2. The new convergence gives a weaker bound than the one in [10] in terms of convergence to a $(1-\epsilon)$ *relative* approximation to the maximum density. However, it gives a strong *additive* guarantee to the *entire* dense decomposition vector.

Our second contribution builds on our insights on DSG and DSS, and applies it towards understanding ideal tree packing and greed tree packing. We connect the ideal tree packing of Thorup to the dense decomposition associated with the rank function of the underlying graphic matroid (which is submodular). We then show that greedy tree packing algorithm can be viewed as the Frank-Wolfe algorithm applied to the quadratic program defined by Fujishige, and this easily yields a convergence guarantee.

▶ **Theorem 3.** *Let $G = (V, E)$ be a graph. The ideal edge load vector $\ell^* : E \to \mathbb{R}_+$ for G is given by the lexicographically minimal base in the polymatroid associated with the rank function of the graphic matroid of G. The Frank-Wolfe algorithm with step size $\frac{1}{k+1}$, when applied to the quadratic program for computing the lexicographically minimal base in the graphic matroid of G, coincides with the greedy tree packing algorithm. For unweighted graphs on m edges, the generic analysis of Frank-Wolfe method's convergence shows that greedy tree packing converges to a load vector $\ell : E \to \mathbb{R}_+$ such that $||\ell - \ell^*||_2 \leq \epsilon$ in $O(\frac{m\log(m/\epsilon)}{\epsilon^2})$ iterations. The standard step size algorithm converges in $O(\frac{m}{\epsilon^2})$ iterations.*

▶ Remark 4. Although the algorithm is the same (greedy tree packing), Thorup's analysis guarantees a strongly polynomial-bound even in the capacitated case [47]. However we obtain a stronger additive guarantee via a *generic* Frank-Wolfe analysis and our analysis has a $1/\epsilon^2$ dependence while Thorup's has a $1/\epsilon^3$ dependence. We give a more detailed comparison in Section 5.

Organization. The rest of the paper is devoted to proving the two theorems. The paper relies on tools from theory of submodular functions and an adaptation of the analysis of Frank-Wolfe. We first describe the relevant background and then prove the two results in separate sections. Due to space constraints, most of the proofs are provided in the full version.

2 Background on Frank-Wolfe algorithm and a variation

Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function. Consider the problem of $\min_{x\in\mathcal{D}} f(x)$. Frank-Wolfe [17] is a first order method and it relies on access to a linear minimization oracle, LMO, for f that can answer $\text{LMO}(w) = \arg\min_{s\in\mathcal{D}}\langle s, \nabla f(w)\rangle$ for any given $w \in \mathcal{D}$. In several applications such oracles with fast running times exist. Given $f, \mathcal{D}$ as above, the Frank-Wolfe algorithm is an iterative algorithm that converges to the minimizer $\mathbf{x}^* \in \mathcal{D}$ of f. See Algorithm 1. The algorithm starts with a guess of the minimizer $b^{(0)} \in \mathcal{D}$. In each iteration, it finds a direction $d^{(k+1)}$ to move towards by calling the linear minimization oracle on the current guess $b^{(k)}$. It then moves slightly towards that direction using a convex combination to ensure that the new point is in $\mathcal{D}$. The amount the algorithm moves towards the new direction decreases as k increases signifying the "confidence" in its current guess as the minimizer.

The original convergence analysis for the Frank-Wolfe algorithm is from [17]. Jaggi [25] gave an elegant and simpler analysis. His analysis characterizes the convergence rate in terms of the *curvature constant* C_f of the function f.

Algorithm 1 Frank-Wolfe-Original.

1: Initialize $b^{(0)} \in \mathcal{D}$
2: **for** $k \leftarrow 0$ to $T-1$ **do**
3: $\gamma \leftarrow \frac{2}{k+2}$
4: $d^{(k+1)} \leftarrow \arg\min_{s\in\mathcal{D}}(\langle s, \nabla f(b^{(k)})\rangle)$ ▷ Call oracle on $b^{(k)}$
5: $b^{(k+1)} \leftarrow (1-\gamma)b^{(k)} + \gamma d^{(k+1)}$
return $b^{(T)}$

▶ **Definition 5.** *Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function. The curvature constant C_f of f is defined as*

$$C_f = \sup_{x,s\in D, \gamma\in[0,1], y=x+\gamma(s-x)} \frac{2}{\gamma^2}(f(y) - f(x) - \langle y-x, \nabla f(x)\rangle).$$

▶ **Definition 6.** *Let $g : \mathcal{D} \to \Re$ be a differentiable function. Then g is Lipschitz with constant L if for all $x, y \in \mathcal{D}$, $\|g(\mathbf{x}) - g(\mathbf{y})\|_2 \le L\|x-y\|_2$.*

Let $\mathrm{diam}(\mathcal{D}) = \max_{x,y\in\mathcal{D}} \|x-y\|_2$ be the diameter of $\mathcal{D}$. One can show that $C_f \le L{\cdot}\mathrm{diam}(\mathcal{D})^2$ where L is the Lipschitz constant of ∇f.

▶ **Theorem 7** ([25]). *Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function with minimizer $\mathbf{b}^*$. Let $\mathbf{b}^{(k)}$ denote the guess on the k-th iteration of the Frank-Wolfe algorithm. Then $f(\mathbf{b}^{(k)}) - f(\mathbf{b}^*) \le \frac{2C_f}{k+2}$.*

Jaggi's proof technique can be used to prove the convergence rate of "noisy/approximate" variants of the Frank-Wolfe algorithm. This motivates the following definition. An ϵ-*approximate linear minimization oracle* is an oracle that for any $\mathbf{w} \in \mathcal{D}$, returns $\hat{\mathbf{s}}$ such that $\langle \hat{\mathbf{s}}, \nabla f(\mathbf{w})\rangle \le \langle \mathbf{s}^*, \nabla f(\mathbf{w})\rangle + \epsilon$, where $s^* = \mathrm{LMO}(\mathbf{w})$. While an efficient *exact* linear minimization oracle exists in some applications, in others one can only ϵ-approximate it (using numerical methods or otherwise). Jaggi's proof technique extends to show that an approximate linear minimization oracles suffices for convergence as long as the approximation quality improves with the iterations. Suppose the oracle, in iteration k, provides a $\frac{\delta C_f}{k+2}$-approximate solution where $\delta > 0$ is some fixed constant. The convergence rate will only deteriorate by a $(1+\delta)$ multiplicative factor. Qualitatively, this says that we can afford to be inaccurate in computing the Frank-Wolfe direction in early iterations, but the approximation should approach $\mathrm{LMO}(b^{(k)})$ as $k \to \infty$.

Another question of interest is the resilience of the Frank-Wolfe algorithm to changes in the learning rate $\gamma_k = \frac{2}{k+2}$. Indeed, the variants we will look at will *require* $\gamma_k = \frac{1}{k+1}$. Jaggi's proof can again be adapted to handle this case, with only an $O(\log k)$ multiplicative deterioration in the convergence rate. We state the following theorem whose proof we defer to the appendix.

▶ **Theorem 8.** *Let $\mathcal{D} \subseteq \Re^d$ be a compact convex set, and $f : \mathcal{D} \to \Re$ be a convex, differentiable function with minimizer $\mathbf{b}^*$. Suppose instead of computing $\mathbf{d}^{(k+1)}$ by calling LMO$(\mathbf{b}^{(\mathbf{k})})$ in iteration k, we call a $\frac{\delta C_f}{k+2}$-approximate linear minimization oracle, for some fixed $\delta > 0$. Also, suppose instead of using $\gamma_k = \frac{2}{k+2}$, we use $\gamma_k = \frac{1}{k+1}$ as a step size. Then $f(\mathbf{b}^{(k)}) - f(\mathbf{b}^*) \le \frac{2C_f(1+\delta)H_{k+1}}{k+1}$, where H_n is the n-th Harmonic term.*

We refer to the variant of Frank-Wolfe algorithm, as described by Theorem 8, as *noisy* Frank-Wolfe.

3 Sub and supermodular functions, and dense decompositions

We already defined submodular and supermodular set functions, polymatroids and contrapolymatroids. We restrict attention to functions satisfying $f(\emptyset) = 0$ which together with supermodularity and non-negativity implies monotonocity, that is, $f(A) \leq f(B)$ for $A \subseteq B$. An alternative definition of submodularity is via diminishing marginal values. We let $f(v \mid A) = f(A \cup \{v\}) - f(A)$ denote the marginal value of v to A. Submodularity is equivalent to $f(v \mid A) \geq f(v \mid B)$ whenever $A \subseteq B$ and $v \in V \setminus B$; the inequality is reversed for supermodular set functions. We need the following simple lemma.

▶ **Lemma 9.** *For a submodular function $f : 2^V \to \Re$, the function $g(X) = f(V) - f(V \setminus X)$ is supermodular. In particular if f is a normalized monotone submodular function then g is a normalized monotone supermodular function.*

Deletion and contraction, and non-negative summation. Sub and supermodular functions are closed under a few simple operations. Given $f : 2^V \to \mathbb{R}$, restricting it to a subset V' corresponds to deleting $V \setminus V'$. Given $A \subset V$, contracting f to A yields the function $g : 2^{V \setminus A} \to \mathbb{R}$ where $g(X) = g(X \cup A) - g(A)$. Given two functions f and g we can take their non-negative sum $af + bg$ where $a, b \geq 0$. Monotonicity and normalization is also preserved under these operations.

3.1 Dense decompositions for submodular and supermodular functions

Following the discussion in the introduction, we are interested in decompositions of supermodular and submodular functions. Dense decompositions follow from the theory of principal partitions of submodular functions that have been explored extensively. We refer the reader to Fujishige's survey [20] as well as Naraynan's work [35, 36]. The standard perspective comes from considering the minimizers of the function f_λ for a scalar λ where $f_\lambda(S) = f(S) - \lambda|S|$. As λ varies from $-\infty$ to ∞ the minimizers change only at a finite number of break points. In this paper we are interested in the notion of density, in the form of ratios, for non-negative submodular and supermodular functions. For this reason we follow the notation from recent work [44, 14, 10, 24] and state lemmas in a convenient form, and provide proofs in the appendix for the sake of completeness.

Supermodular function dense decomposition. The basic observation is the following.

▶ **Lemma 10.** *Let $f : 2^V \to \Re_+$ be a non-negative supermodular set function. There exists a* unique maximal *set $S \subseteq V$ that maximizes $\frac{f(S)}{|S|}$.*

The preceding lemma can be used in a simple fashion to derive the following corollary (this was explicitly noted in [10] for instance).

▶ **Corollary 11.** *Let $f : 2^V \to \Re_+$ be a non-negative supermodular set function. There is a unique partition $S_1, S_2, \ldots, S_h$ of V with the following property. Let $V_i = V - \cup_{j<i} S_j$ and let $A_i = \cup_{j<i} S_i$. Then, for each $i = 1$ to h, S_i is the unique maximal densest set for the function $f_{D_i} : 2^{V_i} \to \mathbb{R}_+$. Moroever, letting λ_i be the optimum density of f_{D_i}, we have $\lambda_1 > \lambda_2 \ldots > \lambda_h$.*

Based on the preceding corollary, we can associated with each $v \in V$ a value $\lambda(v)$: $\lambda(v) = \lambda_i$ where $v \in S_i$. See Figure 1 (full version) for an example of a dense decomposition of the function $f(S) = |E(S)|$.

Dense decomposition for submodular functions. We now discuss submodular functions. We consider two variants. We start with a basic observation.

▶ **Lemma 12.** *Let $f : 2^V \to \Re_+$ be a monotone non-negative submodular set function such that $f(v) > 0$ for all $v \in V$. There is a unique minimal set $S \subseteq V$ that minimizes $\frac{|V|-|S|}{f(V)-f(S)}$ for submodular function f.*

Consider the following variant of a decomposition of f. We let $S_0 = V$ and find S_1 as the unique *minimal* set $S \subseteq V$ that minimizes $\frac{|V|-|S|}{f(V)-f(S)}$. Then we "delete" $\hat{S}_1 = V \setminus S_1$, and find the minimal set $S_2 \subseteq S_1$ that minimizes $\frac{|S_1|-|S|}{f(S_1)-f(S)}$. In iteration i, we find the unique minimal set $S_i \subset S_{i-1}$ that minimizes $\frac{|S_{i-1}|-|S_i|}{f(S_{i-1})-f(S_i)}$. Notice that $S_k \subset S_{k-1} \subset ... \subset S_1 \subset V$. We say the relative density of $\hat{S}_i = S_{i-1} \setminus S_i$ is $\lambda_i = \frac{|S_{i-1}|-|S_i|}{f(S_{i-1})-f(S_i)}$. For $u \in \hat{S}_i$, we say the density of u is $\lambda_u = \lambda_i$. Hence the dense decomposition of f is $\hat{S}_1, ..., \hat{S}_k$ with densities $\lambda_1, \ldots, \lambda_k$. We refer to this decomposition as the first variant which is based on deletions.

We now describe a second dense decomposition for submodular functions. Let $f : 2^V \to \mathbb{R}_+$ be a monotone submodular function. Consider the supermodular function $g : 2^V \to \mathbb{R}_+$ where $g(X) = f(V) - f(V \setminus X)$ for all $X \subseteq V$. From Lemma 9, g is monotone supermodular. We can then apply Corollary 11 to obtain a dense decomposition of g. Let $T_1, T_2, \ldots, T_{k'}$ be the unique decomposition obtained by considering g and let $\hat{\lambda}_1, ..., \hat{\lambda}_{k'}$ be the corresponding densities. Note that this second decomposition is based on contractions.

Not too surprisingly, the two decompositions coincide, as we show in the next theorem. The main reason to consider them separately is for technical ease in applications where one or the other view is more natural.

▶ **Theorem 13.** *Let $\hat{S}_1, ..., \hat{S}_k$ be a dense decomposition (using deletion variant) of a submodular function f with densities $\lambda_i, \ldots, \lambda_k$. Let $T_1, ..., T_{k'}$ be a dense decomposition (using contraction variant) of the same function with densities $\hat{\lambda}_1, ..., \hat{\lambda}_{k'}$. We have (i) $k' = k$, (ii) $\hat{S}_1, ...\hat{S}_k$ is exactly $T_1, ..., T_k$, and (iii) $\hat{\lambda}_i = \frac{1}{\lambda_i}$ for $1 \leq i \leq k$.*

3.2 Fujishige's results on lexicographically optimal bases

Fujishige [18] gave a polyhedral view of the dense decomposition which is the central ingredient in our work. He stated his theorem for polymatroids, however, it can be easily generalized to contrapolymatroids. We restrict attention to the unweighted case for notational ease – [18] treats the weighted case.

Vectors in $\mathbb{R}^n$ can be totally ordered by sorting the coordinates in increasing order of value and considering the lexicographical ordering of the two sorted sequences of length n. In the following, for $a, b \in \mathbb{R}^n$ we use $a \prec_\uparrow b$ and $a \preceq_\uparrow b$ to refer to this order. We say that a vector x in a set D is lexicographically maximum if for all $y \in D$ we have $y \preceq_\uparrow x$.

Fujishige proved the following theorem for polymatroids.

▶ **Theorem 14** ([18]). *Let $f : 2^V \to \mathbb{R}_+$ be a monotone submodular function (a polymatroid) and let B_f be its base polytope. Then there is a unique lexicographically maximum base $b^* \in B_f$ and for each $v \in V$, $b^*_v = \lambda_v$. Moroever, b^* is the optimum solution to the quadratic program: $\min \sum_v x_v^2$ subject to $x \in B_f$.*

Another ordering is to sort the coordinates in decreasing order of value and then taking the lexicographic ordering on the two sorted sequences. We denote this ordering by $\prec_\downarrow, \preceq_\downarrow$ for strict and non-strict ordering respectively. We say that a vector x in a set D is lexicographically minimum if for all $y \in D$ we have $x \preceq_\downarrow y$. The preceding theorem can be generalized to contrapolymatroids in a straight forward fashion and this was explicitly pointed out in [24]. We paraphrase it to be similar to the preceding theorem statement.

▶ **Theorem 15.** *Let $f : 2^V \to \mathbb{R}_+$ be a monotone supermodular function (a contrapolymatroid) and let B_f be its base polytope. Then there is a unique lexicographically minimum base $b^* \in B_f$ and for each $v \in V$, $b^*_v = \lambda_v$. Moreover, b^* is the optimum solution to the quadratic program:* $\min \sum_v x_v^2$ *subject to $x \in B_f$.*

3.3 Approximating a lexicographically optimal base using Frank-Wolfe

Consider the convex quadratic program $\min \sum_{v \in V} x_v^2$ subject to $x \in B_f$ where B_f is the base polytope of f (could be submodular of supermodular). We can use the Frank-Wolfe method to approximately solve this optimization problem. The gradient of the quadratic function is $2x$ and it follows that in each iteration, we need to answer the linear minimization oracle of $\mathrm{LMO}(w) = \arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, 2\mathbf{w} \rangle$ for $\mathbf{w} \in B_f$. This is equivalent to $\arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, \mathbf{w} \rangle$, in other words optimizing a linear objective over the base polytope. Edmonds [15] showed that the simple greedy algorithm is an $O(|V| \log |V|)$ time exact algorithm (assuming $O(1)$ time oracle access to f).

▶ **Theorem 16** ([15])**.** *Fix a polymatroid $f : 2^V \to \Re_+$. Given a weight vector $\mathbf{w} \in \Re^n$, let $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ be a sort of $V = \{v_1, ..., v_n\}$ in ascending order of $\mathbf{w}_i$ values. Let $A_i = \{v_{j_1}, ..., v_{j_i}\}$ for $1 \le i \le n$ with $A_0 = \emptyset$. Define $\mathbf{s}^*_i = f(A_i) - f(A_{i-1})$. Then* $\mathbf{s}^* = \arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, \mathbf{w} \rangle$*.*

The theorem also holds for supermodular functions but by reversing the order from ascending to descending order of $\mathbf{w}$ and complimenting the set A_i.

▶ **Theorem 17** ([15])**.** *Fix a contrapolymatroid $f : 2^V \to \Re_+$. Given a weight vector $\mathbf{w} \in \Re^n$, let $v_{j_1}, v_{j_2}, \ldots, v_{j_n}$ be a sort of $V = \{v_1, ..., v_n\}$ in descending order of $\mathbf{w}_i$ values. Let $A_i = \{v_{j_i}, ..., v_{j_n}\}$ for $1 \le i \le n$ with $A_{n+1} = \emptyset$. Define $\mathbf{s}^*_i = f(A_i) - f(A_{i+1})$. Then* $\mathbf{s}^* = \arg\min_{\mathbf{s} \in B_f} \langle \mathbf{s}, \mathbf{w} \rangle$*.*

Both algorithms are dominated by the sorting step and thus takes $O(|V| \log |V|)$ time. These simple algorithms imply that the Frank-Wolfe algorithm can be used on the quadratic program to obtain an approximation to the lexicographically maximum (respectively minimum) bases of submodular (respectively supermodular) functions. The standard Frank-Wolfe algorithm would need $O(\frac{\mathrm{diam}(B_f)^2}{\epsilon^2})$ iterations to converge to a vector $\hat{b}$ satisfying $\left\|\hat{b} - b^*\right\|_2 \le \epsilon$.

4 Application 1: Convergence of Greedy++ and SuperGreedy++

We begin by describing Greedy++ [7] and its generlization SuperGreedy++ [10]. Greedy++ is built upon the peeling idea of Greedy, and applies it over several iterations. The algorithm initializes a weight/load on each $v \in V$, denoted by $w(v)$, to 0. In each iteration it creates an ordering by peeling the vertices: the next vertex to be chosen is $\arg\min_{v \in V(G')}(w(v) + \deg_{G'}(v))$ where G' is the current graph (after removing the previously peeled vertices). At the end of the iteration, $w(v)$ is increased by the degree of v when it was peeled in the current iteration. A precise description can be found below. SuperGreedy is a natural generalization of Greedy, and SuperGreedy++ generalizes Greedy++. A formal description of SuperGreedy++ is given below.

Algorithm 2 GREEDY++($G(V,E),T$) [7].

Initialize $w(u) \leftarrow 0$ for all $u \in V$
$G^* \leftarrow G$
for $k \leftarrow 0$ to $T-1$ **do**
  $G' \leftarrow G$
  while $|G'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in G'}(w(u) + deg_{G'}(u))$
    $w(u) \leftarrow w(u) + deg_{G'}(u)$
    $G' \leftarrow G' - \{u\}$
    if $\lambda(G') > \lambda(G^*)$ **then**
      $G^* \leftarrow G'$
return G^*

Algorithm 3 SUPER-GREEDY++(f,T) [10].

Initialize $w(u) \leftarrow 0$ for all $u \in V$
$S^* \leftarrow V$
for $k \leftarrow 0$ to $T-1$ **do**
  $V' \leftarrow V$
  while $|V'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in V'}(w(u)+f(V')-f(V'-u))$
    $w(u) \leftarrow w(u) + f(V') - f(V'-u)$
    $V' \leftarrow V' - u$
    if $\frac{f(V')}{|V'|} > \frac{f(S^*)}{|S^*|}$ **then**
      $S^* \leftarrow V'$
return S^*

The goal of this section is to prove Theorem 1 on the convergence of SUPERGREEDY++ and GREEDY++ to the lexicographically maximal base.

4.1 Intuition and main technical lemmas

As we saw in Section 3.3, if one applies the Frank-Wolfe algorithm to solve the qaudratic program $\min \sum_{v \in V} x_v^2$ subject to $x \in B_f$, each iteration corresponds to finding a minimum weight base of f where the weights are given by the current vector x. Finding a minimum weight base corresponds to sorting V by x. However, SUPERGREEDY++ and GREEDY++ use a more involved peeling algorithm in each iteration; the peeling is based on the weights as well as the degrees of the vertices and it is not a static ordering (the degrees change as peeling proceeds). This is the difficulty in formally analyzing these algorithms. In [10], the authors used a connection to the multiplicative weight update method via LP relaxations. Here we rely on the quadratic program and noisy Frank-Wolfe. The high-level intuition, that originates in [10], is the following. As the algorithm proceeds in iterations, the weights on the vertices accumulate; recall that the total increase in the weight in the case of DSG is $m = |E|$. The degree term, which influences the peeling, is dominant in early iterations, but its influence on the ordering of the vertices decreases eventually as the weights of the vertices get larger. It is then plausible to conjecture that the algorithm behaves like the standard Frank-Wolfe method in the limit. The main question is how to make this intuition precise. [10] relies on a connection to the MWU method while we use a connection to noisy Frank-Wolfe.

For this purpose, consider an iteration of GREEDY++ and SUPERGREEDY++. The algorithm peels based on the current weight vector and the degrees. We isolate and abstract this peeling algorithm and refer to it as Weighted-Greedy and Weighted-SuperGreedy respectively, and formally describe them with the weight vector w as a parameter.

Algorithm 4 WEIGHTED-GREEDY(G, w).

Input: $G(V, E)$ and $w(u)$ for $u \in V$
  $G' \leftarrow G$
  Initialize $\hat{d}(u) = 0$ for all $u \in V$.
  while $|G'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in G'}(w(u) + deg_{G'}(u))$
    $\hat{d}(u) \leftarrow deg_{G'}(u)$
    $G' \leftarrow G' - \{u\}$
  return $\hat{d}$

Algorithm 5 WEIGHTED-SUPERGREEDY(f, w).

Input: Supermodular $f : 2^V \to \Re_+$, $w(u)$ for $u \in V$
  $V' \leftarrow V$
  Initialize $\hat{d}(u) = 0$ for all $u \in V$.
  while $|V'| > 1$ **do**
    $u \leftarrow \arg\min_{u \in G'}(w(u) + f(V') - f(V' - u)$
    $\hat{d}(u) \leftarrow f(V') - f(V' - u)$
    $V' \leftarrow V' - u$
  return $\hat{d}$

The peeling algorithms also compute a base $\hat{d} \in B_f$. In the case of graphs and DSG, $\hat{d}(u)$ is set to the degree of the vertex u when it is peeled. One can alternatively view the base as an orientation of the edges of E. Define for each edge $uv \in G$ two weights x_{uv}, x_{vu}. We say that $\mathbf{x}$ is *valid* if $x_{uv} + x_{vu} = 1$ and $x_{uv}, x_{vu} \geq 0$ for all $\{u, v\} \in E(G)$. For $b \in \Re^{|V|}$, we say x *induces* b if $b_u = \sum_{v \in \delta(u)} x_{uv}$ for all $u \in V$. We say a vector d is an *orientation* if there is a valid x that induces it.

▶ **Lemma 18** ([24]). *For $f(S) = |E(S)|$, $b \in B_f$ if and only if b is an orientation.*

Recall that the Frank-Wolfe algorithm, for a given weight vector $w : V \to \mathbb{R}_+$, computes the minimum-weight base b with respect to w since $\langle w, b\rangle = \min_{y \in B_f}\langle w, y\rangle$. It is worth taking a moment to note that this base (or orientation due to Lemma 18) is easily computable: we orient each edge integrally (i.e $x_{vu} = 1, x_{uv} = 0$) from v to u if $w(u) \geq w(v)$, and from u to v otherwise. A simple exchange argument yields a proof of correctness and is implicit in many works [14][2]. This induces an optimal base d^*_w with respect to w. Our goal is to compare how the peeling order created by Weighted-Greedy (and Weighted-SuperGreedy) compares with the best base. The following two technical lemmas formalize the key idea. The first is tailored to DSG and the second applies to DSS.

▶ **Lemma 19.** *Let $\hat{d}$ be the output from* WEIGHTED-GREEDY(G, w) *and d^*_w be the optimal orientation with respect to w. Then $\langle w, \hat{d}\rangle \leq \langle w, d^*_w\rangle + \sum_u deg_G(u)^2$. In particular, the additive error* ***does not depend*** *on the weight vector w.*

▶ **Lemma 20.** *For a supermodular function $f : 2^V \to \Re_+$, let $\hat{d}$ be the output from* WEIGHTED-SUPERGREEDY(f, w) *(Algorithm 5) and d^*_w be the optimal vector with respect to w as described in Theorem 17. Then $\langle w, \hat{d}\rangle \leq \langle w, d^*_w\rangle + n\sum_{u \in V} f(u \mid V - u)^2$. In particular, the additive error* ***does not depend*** *on the weight vector w.*

4.2 Convergence proof for Greedy++

Why is Lemma 19 crucial? First, observe that the minimizer d^*_w of $\langle w, d\rangle$ is exactly the same minimizer as $\langle Kw, d\rangle$ for any constant $K > 0$ (and vice-versa).

▶ **Lemma 21.** *Let $\hat{d}_K$ be the output of* WEIGHTED-GREEDY(G, Kw). *Then $\langle w, \hat{d}_K\rangle \leq \langle w, d^*_w\rangle + \frac{\sum_u deg_G(u)^2}{K}$.*

[2] Since the optimal orientation is easily computable, one can replace the "peeling" iteration of GREEDY++ with the optimum base. This would result in the Frank-Wolfe based algorithm of [14].

Proof. By Lemma 19, $\sum_{u \in V} Kw(u)\hat{d}_K(u) \leq \min_{\text{orientation } d} \left(\sum_{u \in V} Kw(u)d(u)\right) + \sum_u deg_G(u)^2$. Dividing by K implies the claim. ◀

We are now ready to view Greedy++ as a noisy Frank-Wolfe algorithm. Algorithm 6 shows how Greedy++ could be interpreted.

Algorithm 6 Greedy++$(G(V, E))$.

Input: $G = (V, E)$ and $w(u)$ for $u \in V$
  Initialize $b^{(0)} \leftarrow$ Weighted-Greedy$(G, \mathbf{0})$ ▷ $b^{(0)}$ is a valid orientation
  for $k \leftarrow 0$ to $T - 1$ **do**
    $\gamma \leftarrow \frac{1}{k+1}$
    $d^{(k+1)} \leftarrow$ Weighted-Greedy$(G, (k + 1)b^{(k)})$
    $b^{(k+1)} \leftarrow (1 - \gamma)b^{(k)} + \gamma d^{(k+1)}$
  return $b^{(T)}$

The algorithm is exactly the same as the one described in Algorithm 2. Indeed, one can prove that $kb^{(k)}$ is precisely the weights that Greedy++ ends with at round k by induction. Observe that $(k+1)b^{(k+1)} = kb^{(k)} + d^{(k+1)}$ which is precisely the load as described in Algorithm 2 (via induction). We note that $\gamma \leftarrow 1/(k + 1)$ is crucial here to ensure we are taking the average. Lemma 25 in the appendix (full version) implies that each peel in Algorithm 2 is $\frac{\delta C_f}{k+2}$-approximate linear minimization oracle. Using Theorem 8, this implies that Greedy++ (as described in Algorithm 2) converges to b^* in $\tilde{O}(\frac{mn^2}{\epsilon^2})$ iterations since $\delta = O(\frac{\sum_u d_G(u)^2}{m})$ and $C_f = O(\sum_u d_G(u)^2)$. We use the probabilistic method to bound C_f in the full version.

Extension to SuperGreedy++. An essentially similar analysis works for SuperGreedy++. Instead of Lemma 19, we rely on Lemma 20. For technical reasons, the convergence analysis of SuperGreedy++ is slightly weaker than for Greedy++.

5 Application 2: Greedy Tree Packing interpreted via Frank-Wolfe

Let $G = (V, E)$ be a graph with non-negative edge capacities. The goal of this section is to view Thorup's definitions of ideal edge loads and the associated tree packing from a different perspective, and to derive an alternate convergence analysis of his greedy tree packing algorithm [46, 47]. In previous work, Chekuri, Quanrud and Xu [11] obtained a different tree packing based on an LP relaxation for k-cut, and used it in place of ideal tree packing. Despite this, there was a gap in our understanding which we address here.

We restrict our attention to unweighted multi-graphs throughout this section, and comment on the capacitated case at the end of the section. Let $G = (V, E)$ be a connected multi-graph, with n vertices and m edges. Consider the graphic matroid $\mathcal{M}_G(E, \mathcal{F})$ induced by G; E is the ground set, and $\mathcal{F}$ consists of all sub-forests of G. The bases of the matroid are precisely the spanning trees of G. Consider the rank function $r : 2^E \to \mathbb{Z}_+$ of $\mathcal{M}_G$. r is submodular, and it is well-known that for a edge subset $X \subseteq E$, $r(X) = n - \kappa(X)$ where $\kappa(X)$ is the number of connected components induced by X.

5.1 Thorup's recursive algorithm as dense decomposition

For consistency with previous notation, we use f to denote the submodular rank function r. We first describe ideal loads as defined by Thorup. Consider the Tutte–Nash-Williams partition P for G. Recall that P minimizes the ratio $\frac{|E(P)|}{|P|-1}$ among all partitions, and this ratio is $\tau(G)$. For each edge $e \in E(P)$, assign $\ell^*(e) = \frac{1}{\tau(G)}$. Remove the edges in $E(P)$ to obtain a graph G' which now consists of several disconnected components. Recursively compute ideal loads for the edges in each connected component of G' (the process stops when G has no edges).

We claim that Thorup's recursive decomposition coincides with the dense decomposition of f (the first variant). To see this, it suffices to see the first step of the dense decomposition. We find the minimal set $S_1 \subseteq E$ that minimizes $\frac{|E|-|S|}{f(E)-f(S)}$. We let $\hat{S}_1 = E \setminus S_1$ and assign the edges in $\hat{S}_1$ the density $\frac{f(E)-f(S)}{|E|-|S|}$. Then, we "delete" $\hat{S}_1$. Observe that $\hat{S}_1 = E \setminus S_1$ is just the edges crossing the partition $P(S_1)$ defined by the $\kappa(S_1)$ connected components spanned by S_1. Also, recall that $\frac{f(E)-f(S_1)}{|E|-|S_1|} = \frac{\kappa(S_1)-1}{|E \setminus S_1|} = \frac{|P(S_1)|-1}{E(P(S_1))} = \frac{1}{\tau(G)}$. Hence, the density assigned to edges in $\hat{S}_1$ is exactly $\frac{1}{\tau(G)}$ by the Tutte–Nash-Williams theorem. The next step is deleting $\hat{S}_1 = E \setminus S_1$, which, as discussed above, are the edges crossing the partition $P(S_1)$.

Via induction we prove the following lemma.

▶ **Lemma 22.** *The weights given to the edges by the dense decomposition algorithm on f coincide with ℓ^*.*

5.2 Greedy tree packing converge to ideal relative loads

Thorup considered the following greedy tree packing algorithm. For each edge define a load $\ell(e)$ which is initialized to 0. The algorithm proceeds in iterations. In iteration i the algorithm computes an MST T_i in G with respect to edge weights $w(e) = \ell(e)$. The load of each edge $e \in T_i$ is increased by 1. Thorup showed that as $k \to \infty$, the quantity $\ell(e)/k$ converges to $\ell^*(e)$ for each edge e. His proof is fairly technical. In this section, we present a different proof of this fact that uses the machinery we have built thus far.

▶ **Lemma 23.** *The vector ℓ^* is the lexicographically maximal base of the spanning tree polytope.*

Proof. We showed that Thorup's definition of ideal loads is obtained by simply running the dense decomposition on the rank function of the graphic matroid induced by G. The bases of the graphic matroid are the spanning trees of G and hence the base polytope of f is the spanning tree polytope of G. The dense decomposition of f gives us the lexicographically maximum base, and hence ℓ^* is the lexicographically maximal base in the spanning tree polytope of G. ◀

Hence, ℓ^* is the unique solution to the quadratic program: min $\sum_e \ell(e)^2$ subject to $\ell \in \mathrm{SPT}(G)$ where $\mathrm{SPT}(G)$ is the spanning tree base polytope. We can thus apply a noisy Frank-Wolfe algorithm to the quadratic program to obtain Algorithm 7.

The main observation is that this algorithm is **exactly** the same as Thorup's greedy tree packing algorithm. Indeed, observe that $(k+1)\ell^{(k+1)} \leftarrow k\ell^{(k)} + d^{(k+1)} = k\ell^{(k)} + \mathbb{1}\{e \in \mathrm{MST}(G, \ell^{(k)})\}$ where $\mathrm{MST}(G, w)$ is a minimum spanning tree of G with respect to edge weights w. Since noisy Frank-Wolfe converges, then $\ell^{(k)}$ converges to $\ell^*(e)$, and greedy tree packing converges.

Algorithm 7 FRANK-WOLFE-GREEDY-TREEPACK($G(V,E)$).

Input: $G(V,E)$
  Initialize $l^{(0)}(u) = \mathbb{1}\{e \in T\}$ for any spanning tree T.
  for $k \leftarrow 0$ to $T-1$ **do**
    $\gamma \leftarrow \frac{1}{k+1}$
    $d^{(k+1)} \leftarrow \min_{s \in \mathrm{SPT}(G)} \langle l^{(k)}, s\rangle$ ▷ This is the minimum spanning tree with respect to $l^{(k)}$
    $l^{(k+1)} \leftarrow (1-\gamma)l^{(k)} + \gamma d^{(k+1)}$
  return $b^{(T)}$

We now establish the convergence guarantee for greedy tree packing. For the spanning tree polytope of an m edge graph, the curvature constant $C_f \leq 4m$ because for $x, y \in B_f$, $2(x-y)^T(x-y) = \sum_{e \in E}(x_e - y_e)^2 \leq 4m$. Plugging this bound into Theorem 8, after $k = O(\frac{m \log(m/\epsilon)}{\epsilon^2})$ iterations, $\left\|\ell^{(k)} - \ell^*\right\|_2 \leq \epsilon$.

Suppose we run the standard Frank-Wolfe algorithm with $\gamma = 2/(k+2)$. Then, the convergence guarantee improves to $O(\frac{m}{\epsilon^2})$. Note that each iteration still corresponds to finding an MST in the graph with weights. However, the load vector is no longer a simple average of the trees taken so far.

Comparison to Thorup's bound and analysis. Thorup [47] considered ideal tree packings in capacitated graphs; let $c(e) \geq 1$ (via scaling) denote the capacity of edge e. Via [18], one sees that the optimum solution of the quadratic program $\sum_e x_e^2/c(e)$ subject to $x \in SP(G)$ is the ideal load vector ℓ^*. Greedy tree packing generalizes to the capacitated case easily; in each iteration we compute the MST with respect to weights $w(e) = \ell(e)c(e)$. Thorup proved the following.

▶ **Theorem 24** ([47]). *Let $G = (V,E)$ be capacitated graph. Greedy tree packing after $O(\frac{m \log(mn/\epsilon)}{\epsilon^3})$ iterations ouputs a load vector ℓ such that for each edge $e \in E$, $\ell(e) \leq (1+\epsilon)\ell^*(e)$.*

We observe that if all capacities are 1 (or identical) then Thorup's guarantee is that $\ell(e) - \ell^*(e) \leq O(\epsilon)$ for each edge e. For this case, via Frank-Wolfe, we obtain the much stronger guarantee that $||\ell - \ell^*||_2 \leq \epsilon$ which easily implies the per edge condition, however the per edge guarantee does not imply a guarantee on the norm. Further, in the unweighted case, our iteration complexity dependence on ϵ is $1/\epsilon^2$ while Thorup's is $1/\epsilon^3$. Thorup's guarantee works for the capacitated case in strongly polynomial number of iterations. We can adapt the Frank-Wolfe analysis to the capacitated case but it would yield a bound that depends on $C = \sum_e c(e)$ (in the unweighted case $C = m$); on the other hand the guarantee provided by Frank-Wolfe is stronger.

It may seem surprising that the same greedy tree packing algorithm yields different types of guarantees based on the type of analysis used. We do not have a completely satisfactory explanation but we point out the following. Thorup's analysis is a non-trivial refinement of the standard MWU type analysis of tree packing [38, 50, 3]. As already noted in [24], if one uses Frank-Wolfe (with $\gamma = 1/(k+1)$) with the softmax potential function that is standard in the MWU framework, then the resulting algorithm would also be greedy tree packing. Fujishige's uses a quadratic objective to guarantee that the optimum solution is the unique maximal base but in fact any increasing strongly convex function would suffice. In the context of optimizing a linear function over B_f, due to the optimality of the greedy algorithm, the only thing that determines the base is the ordering of the elements of V according to the weight vector; the weights themselves do not matter. Thus, Frank-Wolfe applied to different convex objectives can result in the same greedy tree/base packing algorithm. However, the specific objective can determine the guarantee one obtains after a number of iterations. The softmax objective is better suited for obtaining relative error guarantees while the quadratic objective is better suited for obtaining additive error guarantees. Thorup's analysis is more sophisticated due to the per edge guarantee in the capacitated setting. A unified analysis that explains both the relative and additive guarantees is desirable. We leave this is an interesting direction for future research.

References

1 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.

2 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, February 2012. doi:10.14778/2168651.2168658.

3 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.

4 Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

5 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.

6 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, pages 379–388, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2684822.2685298.

7 Digvijay Boob, Yu Gao, Richard Peng, Saurabh Sawlani, Charalampos Tsourakakis, Di Wang, and Junxing Wang. *Flowless: Extracting Densest Subgraphs Without Flow Computations*, pages 573–583. Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3366423.3380140.

8 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.

9 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.

10 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, 2022. doi:10.1137/1.9781611977073.64.

11 Chandra Chekuri, Kent Quanrud, and Chao Xu. Lp relaxation and tree packing for minimum k-cut. *SIAM Journal on Discrete Mathematics*, 34(2):1334–1353, 2020.

12 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. arXiv:2203.00671.

13 Aleksander B. G. Christiansen, Jacob Holm, Ivor van der Hoog, Eva Rotenberg, and Chris Schwiegelshohn. Adaptive out-orientations with applications, 2023. arXiv:2209.14087.

14 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, pages 233–242, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee. doi:10.1145/3038912.3052619.

15 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In R. Guy, H. Hanani, N. Sauer, and J. Schönheim, editors, *Combinatorial Structures and Their Applications (Proceedings Calgary International Conference on Combinatorial Structures and Their Applications, Calgary, Alberta, 1969; R. Guy, H. Hanani, N. Sauer, J. Schönheim, eds.)*, New York, 1970. Gordon and Breach.

16 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, pages 300–310, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. `doi:10.1145/2736277.2741638`.

17 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956. `doi:10.1002/nav.3800030109`.

18 Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980. URL: `http://www.jstor.org/stable/3689149`.

19 Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.

20 Satoru Fujishige. Theory of principal partitions revisited. *Research Trends in Combinatorial Optimization: Bonn 2008*, pages 127–162, 2009.

21 Takuro Fukunaga. Computing minimum multiway cuts in hypergraphs from hypertree packings. In *IPCO*, pages 15–28. Springer, 2010.

22 A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1984/5956.html`.

23 Anupam Gupta, David G Harris, Euiwoong Lee, and Jason Li. Optimal bounds for the k-cut problem. *ACM Journal of the ACM (JACM)*, 69(1):1–18, 2021.

24 Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. Faster and scalable algorithms for densest subgraph and decomposition. In *Advances in Neural Information Processing Systems*, volume 35, pages 26966–26979. Curran Associates, Inc., 2022. URL: `https://proceedings.neurips.cc/paper_files/paper/2022/file/ac8fbba029dadca99d6b8c3f913d3ed6-Paper-Conference.pdf`.

25 Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, number 1 in Proceedings of Machine Learning Research, pages 427–435, Atlanta, Georgia, USA, 17–19 June 2013. PMLR. URL: `https://proceedings.mlr.press/v28/jaggi13.html`.

26 David R Karger. Minimum cuts in near-linear time. *Journal of the ACM (JACM)*, 47(1):46–76, 2000.

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

28 Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants, 2023. `arXiv:2303.14467`.

29 Jason Li. Faster minimum k-cut of a simple graph. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1056–1077. IEEE, 2019.

30 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.

31 Daniel Lokshtanov, Saket Saurabh, and Vaishali Surianarayanan. A parameterized approximation scheme for min k-cut. *SIAM Journal on Computing*, 0:FOCS20–205, 2022.

32 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, pages 1051–1066, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3318464.3389697`.

33 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.

34 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.

35 H Narayanan. The principal lattice of partitions of a submodular function. *Linear Algebra and its Applications*, 144:179–216, 1991.

36 Hariharan Narayanan. *Submodular functions and electrical networks*, volume 54. Elsevier, 1997.

37 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.

38 Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. URL: `http://www.jstor.org/stable/3690406`.

39 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.

40 Saurabh Sawlani and Junxing Wang. Near-optimal fully dynamic densest subgraph. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 181–193. ACM, 2020. `doi:10.1145/3357713.3384327`.

41 Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

42 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. `doi:10.1109/ICDM.2016.0058`.

43 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, June 2020. `doi:10.14778/3401960.3401962`.

44 Nikolaj Tatti. Density-friendly graph decomposition. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(5):1–29, 2019.

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

46 Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007. Preliminary version in Proc. of ACM STOC 2001.

47 Mikkel Thorup. Minimum k-way cuts via deterministic greedy tree packing. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 159–166, 2008.

48 Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1122–1132, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. `doi:10.1145/2736277.2741098`.

49 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, pages 104–112, New York, NY, USA, 2013. Association for Computing Machinery. `doi:10.1145/2487575.2487645`.

50 N. Young. Randomized rounding without solving the linear program. In *ACM-SIAM Symposium on Discrete Algorithms*, 1995.