
Corporate Needs You to Find the Difference: Revisiting Submodular and Supermodular Ratio Optimization Problems

Anonymous Author(s)

Affiliation

Address

email

Abstract

We consider the following question: given a submodular/supermodular set function $f : 2^V \rightarrow \mathbb{R}$, how should one minimize/maximize its average value $f(S)/|S|$ over non-empty subsets $S \subseteq V$? This problem generalizes several well-known objectives, including Densest Subgraph (DSG), Densest Supermodular Set (DSS), and Submodular Function Minimization (SFM). Motivated by recent applications [40, 32], we formalize two new broad problems: the Unrestricted Sparsest Submodular Set (USSS) and Unrestricted Densest Supermodular Set (UDSS), both of which allow negative and non-monotone functions.

Using classical results, we show that DSS, SFM, USSS, UDSS, and MNP are all equivalent under strongly polynomial-time reductions. This equivalence enables algorithmic cross-over: methods designed for one problem can be repurposed to solve others efficiently. In particular, we use the perspective of the minimum norm point in the base polyhedron of a sub/supermodular function, which, via Fujishige’s results, yields the dense decomposition as a byproduct. Through this perspective, we show that a recent converging heuristic for DSS, SUPERGREEDY++ [16, 30], and Wolfe’s minimum norm point algorithm are both universal solvers for all of these problems.

On the theoretical front, we explain the observation made in recent work [40, 32] that SUPERGREEDY++ appears to work well even in settings beyond DSS. Surprisingly, we also show that this simple algorithm can be used for Submodular Function Minimization, including acting as a practical minimum s - t cut algorithm.

On the empirical front, we explore the utility of several algorithms for recent problems. We conduct over 400 experiments across seven problem types and large-scale synthetic and real-world datasets (up to ≈ 100 million edges). Our results reveal that methods historically considered inefficient, such as convex-programming methods, flow-based solvers, and Fujishige-Wolfe’s algorithm, outperform state-of-the-art task-specific baselines by orders of magnitude on concrete problems like HNSN [40]. These findings challenge prevailing assumptions and demonstrate that with the proper framing, general optimization algorithms can be both scalable and state-of-the-art for supermodular and submodular ratio problems.

1 Introduction and Background

Submodular and supermodular functions play a fundamental role in combinatorial optimization and, thanks to their generality, capture a wide variety of highly relevant problems. For a finite ground set V , the real-valued set function $f : 2^V \rightarrow \mathbb{R}$ is submodular iff $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all $A, B \subseteq V$. A set function f is supermodular iff $-f$ is submodular; f is normalized if $f(\emptyset) = 0$ and monotone if $f(A) \leq f(B)$ whenever $A \subseteq B$. When working with these functions,

we typically assume they are available through a value oracle that returns $f(S)$ given a set $S \subseteq V$. We make this assumption throughout the paper. A problem of central interest is the following:

Problem 1. (*Submodular Function Minimization, SFM*). Let $f : 2^V \rightarrow \mathbb{R}$ be a submodular function given via a value oracle. Compute $\min_{\emptyset \neq S \subseteq V} f(S)$.

A classical result in combinatorial optimization is that SFM can be solved in strongly polynomial-time [28]. There have been many theoretical developments since then [35, 45, 48, 33, 11, 37, 3].

In this paper, we are interested in *ratio* problems involving submodular and supermodular functions, which have been very interesting in various applications. We start with a concrete and canonical problem of this form.

Problem 2. (*Densest Subgraph, DSG*). Given an undirected graph $G = (V, E)$ find a subset $S \subseteq V$ that maximizes the density $|E(S)|/|S|$ where $E(S) = \{(u, v) \in E : u, v \in S\}$.

DSG is a classical problem with wide-ranging applications in data mining, network analysis, and machine learning. Dense subgraphs often reveal crucial structural properties of networks and thus DSG has been a very active area of recent research; see, for example, [13, 42, 8, 17, 52, 51, 1, 53, 21, 43, 47, 6, 38, 2, 50, 41, 39, 44, 12, 7]. A key feature of DSG is its polynomial-time solvability. There are several algorithms to solve it exactly: (i) via network flow [27, 46], (ii) via reduction to SFM (folklore), and (iii) via an LP relaxation [14]. Despite these exact algorithms, there has been considerable interest in fast approximation algorithms and heuristics to scale to the large networks that arise in practice. Hence, the Greedy peeling algorithm that yields a $1/2$ -approximation [14] has been popular in practice. Furthermore, there are several theoretical algorithms that obtain a $(1 - \varepsilon)$ -approximation in near-linear time for any fixed ε via different techniques [4, 9, 16]. An important recent algorithm, Greedy++, is an iterative version of Greedy that was proposed in [8]. It is simple, combinatorial, and has strong empirical performance. Moreover, it was conjectured to converge to a $(1 - \varepsilon)$ -approximation in $O(1/\varepsilon^2)$ -iterations, each of which takes (near) linear time like Greedy. [16] proved that Greedy++ converges to a $(1 - \varepsilon)$ -approximate solution in $O(\Delta(G) \log |V|/\varepsilon^2)$ iterations where $\Delta(G)$ is the maximum degree. Crucial to their proof was a perspective based on supermodularity. In particular, they considered the following general ratio problem.

Problem 3. (*Densest Supermodular Set, DSS*). Let $f : 2^V \rightarrow \mathbb{R}_{\geq 0}$ be a normalized, monotone supermodular function. Compute $\max_{\emptyset \neq S \subseteq V} f(S)/|S|$.

DSG is a special case of DSS; for all graphs $G = (V, E)$, the function $f : 2^V \rightarrow \mathbb{R}$ where $f(S) = |E(S)|$ is monotone supermodular. DSS can be solved exactly via SFM. [16] defined SuperGreedy and SuperGreedy++ for DSS and showed that SuperGreedy++ converges to a $(1 - \varepsilon)$ -approximate optimum solution in $O(\alpha_f \log |V|/\varepsilon^2)$ iterations where $\alpha_f = \max_v (f(V) - f(V - v))$. This is a useful result since several non-trivial problems in dense subgraph discovery can be modeled as a special case of DSS including hypergraph density problems, p -mean density [54] and others — for details, we refer the reader to the recent survey [39]. Several other converging iterative algorithms for DSG have recently been developed via convex optimization methods such as Frank-Wolfe [17], FISTA [29], and accelerated coordinate descent [44]. In another direction, flow-based exact algorithms have been revisited [32, 31]. Many of these algorithms are based on structural properties of DSG inherited from supermodularity, which we describe in more detail following the discussion of the motivation for this work.

Motivation for this work: Our initial motivation originates from two recent ratio problems. [40] studied the following problem.

Problem 4. (*Heavy Nodes in a Small Neighborhood, HNSN*). Given a bipartite graph $G(L, R, E)$ and a weight function $w : R \rightarrow \mathbb{R}_{\geq 0}$, find a set $S \subseteq R$ of nodes such that $\sum_{v \in R} w(v)/|N(S)|$ is maximized where $N(S)$ is the set of neighbors of S . Equivalently the goal is to minimize $|N(S)|/(\sum_{v \in R} w(v))$.

In the preceding problem, the function $f : 2^V \rightarrow \mathbb{R}$ defined as $f(S) = |N(S)|$ is a monotone submodular function; it is the coverage function of the set system induced by the bipartite graph G (which can also be viewed as a hypergraph). This leads us to consider the following ratio problem.

Problem 5. (*Unrestricted Sparsest Submodular Set, USSS*). Given a normalized submodular function $f : 2^V \rightarrow \mathbb{R}$, find $\min_{\emptyset \neq S \subseteq V} f(S)/|S|$.

89 A second motivating problem is the following, which is studied in [32].

90 **Problem 6.** (*Anchored Densest Subgraph, ADS*). Given a graph $G = (V, E)$ and a set $R \subseteq V$,
 91 find a vertex set $S \subseteq V$ maximizing $(2|E(S)| - \sum_{v \in S \cap \bar{R}} \deg_G(v)) / |S|$.

92 The numerator in the preceding problem is supermodular but is no longer non-negative or monotone.
 93 This motivates us to further define an unrestricted version of DSS.

94 **Problem 7.** (*Unrestricted Densest Supermodular Set, UDSS*). Given a normalized supermodular
 95 function $f : 2^V \rightarrow \mathbb{R}$, find $\max_{\emptyset \neq S \subseteq V} f(S) / |S|$.

96 [40] showed that the HNSN problem can be reformulated as a special case of DSS, allowing iterative
 97 peeling algorithms (SUPERGREEDY++) to converge to the optimal solution. In [32], this perspective
 98 is extended by applying SUPERGREEDY++ to the unrestricted version of DSS (UDSS), noting that
 99 non-negativity and monotonicity can be enforced if one first shifts the function by a large modular
 100 term, i.e., adding $C|S|$ for some sufficiently large constant C . This ensures non-negativity and
 101 monotonicity while retaining the exact optimal solution (since the density of all sets simply shifts by
 102 a constant C). However, the shift renders the relative approximation guarantee in [16] inapplicable
 103 in a direct way; indeed, [32] state the following: “we hypothesize that iterative peeling remains an
 104 effective practical heuristic”.

105 More generally, while USSS and UDSS are equivalent in the exact optimization sense (via nega-
 106 tion), approximation guarantees do not easily carry over, especially when relying on relative error
 107 measures. This highlights a key open question: can we formally prove that SUPERGREEDY++ con-
 108 verges for all the above problem classes under appropriate approximation guarantees? Addressing
 109 this question, as we will in this paper, would provide a unified theoretical foundation for its empirical
 110 success observed across diverse submodular and supermodular ratio problems.

111 **Min-norm point and the Fujishige-Wolfe algorithm.** Another motivation for this work comes
 112 from some essential properties of the min-norm point problem relevant to submodular optimization.

113 **Problem 8.** (*Minimum Norm Point, MNP*). Let $f : 2^V \rightarrow \mathbb{R}$ be a normalized submodular or super-
 114 modular function, and let $B(f)$ be the corresponding base polytope. Find $\arg \min_{x \in B(f)} \|x\|_2^2$.

115 It is known that SFM can be reduced to MNP [24, 49]. Further, [23, 24] showed that the optimum
 116 value yields a lexicographically optimal and unique base, which reveals the entire dense decom-
 117 position of f . Recent work in the context of DSG has exploited MNP via convex optimization
 118 techniques [17, 29, 44] to obtain fast iterative algorithms for DSG and the dense decomposition
 119 version of DSG. Wolfe defined the min-norm point problem in the more general context of convex
 120 optimization and developed an iterative algorithm which is well-known [56] — we note that his algo-
 121 rithm is tailored to the quadratic norm objective and is quite different from the more general convex
 122 optimization algorithms. Fujishige tailored the algorithm to submodular polytopes, yielding one of
 123 the fastest practical SFM solvers [26, 10]. Despite its well-known empirical performance for SFM,
 124 the Fujishige-Wolfe algorithm has not been empirically evaluated for any recent ratio problems.

125 **Revisiting flow-based exact algorithms.** Flow-based approaches to DSG were deemed impracti-
 126 cal for large graphs because they required an expensive binary search procedure. Recent indepen-
 127 dent work by Huang *et al.* [32] and Hochbaum [31] introduced an iterative *density-improvement*
 128 framework that overcomes this limitation. Letting $f(S) = |E(S)|$, and starting with $S_0 = V$
 129 and $\lambda_0 = f(S_0) / |S_0|$, the method repeatedly computes $S_{k+1} = \arg \max_{S \subseteq S_k} \{f(S) - \lambda_k |S|\}$
 130 via max-flow, updating $\lambda_{k+1} = f(S_{k+1}) / |S_{k+1}|$ until convergence ($S_{k+1} = S_k$). This approach,
 131 rooted in Dinkelbach’s classical method from 1967 [18, 32], guarantees optimality. Although this
 132 process may appear worse than binary search in the worst case, potentially requiring up to $|V| + 1$
 133 flow computations, empirical evidence shows that the number of iterations is typically minimal and
 134 often far fewer than those required by binary search. These insights have renewed interest in flow-
 135 based algorithms for solving DSG. One can reduce HNSN to a max-flow problem. This raises the
 136 question of the performance of flow-based algorithms for HNSN.

137 **Summary of Motivation.** We are motivated by the observation that several closely related sub-
 138 modular and supermodular ratio problems are treated disparately across the literature, often with
 139 distinct algorithms and analyses, despite underlying connections. We are interested in whether a
 140 unified perspective can bridge these gaps. For instance, while the simple combinatorial algorithm

141 SUPERGREEDY++ has provable guarantees for DSS, its empirical success in broader settings lacks
 142 a corresponding unified theoretical analysis.

143 Similarly, the Fujishige-Wolfe algorithm, despite its established effectiveness for submodular func-
 144 tion minimization, has surprisingly not been evaluated for ratio problems in USSS and UDSS.
 145 Moreover, with recent progress in flow-based exact algorithms for densest subgraph problems, it
 146 is natural to ask how these methods perform on related submodular ratio problems like HNSN. In
 147 essence, our goal is to investigate whether algorithmic successes in one domain can be transferred
 148 to others, leveraging tools that have been overlooked to improve empirical performance and deepen
 149 theoretical understanding across these problem classes.

150 **Our Contributions.** We formalize connections between five problems via reductions: Mini-
 151 mum Norm Point (MNP), Submodular Function Minimization (SFM), Densest Supermodular Set
 152 (DSS), Unrestricted Sparsest Submodular Set (USSS), and Unrestricted Densest Supermodular Set
 153 (UDSS) by proving that they are **equivalent**. All the reductions run in either $O(n)$ or $O(n \log n)$
 154 calls and are highly efficient.

155 While these reductions are rooted in classical concepts, several of these problems—such as USSS
 156 and UDSS—have not been formally defined in the literature. For instance, although DSS can
 157 be exactly solved via SFM, its formal definition in [16] was instrumental in advancing special-
 158 ized algorithms like SuperGreedy and SuperGreedy++. By establishing these reductions, we enable
 159 **algorithmic cross-over**, where methods developed for one problem can be effectively applied to
 160 others. This is demonstrated in our experiments: Wolfe’s Minimum Norm Point algorithm (MNP),
 161 originally for MNP, achieves up to $595\times$ speedups over prior state-of-the-art baselines on HNSN
 162 (a special case of USSS). Similarly, SUPERGREEDY++, designed for DSS, delivers scalable and
 163 competitive performance on the minimum s - t cut problem (a special case of SFM). Interestingly, it
 164 is surprising that SUPERGREEDY++ serves as an effective algorithm for minimum s - t cut, despite
 165 being extremely simple, requiring no data structures, and working in the dual (cut) space. To the
 166 best of our knowledge, all existing algorithms for minimum s - t cut in the literature operate in the
 167 primal space via max-flow formulations.

168 Additionally, we demonstrate that an approximate solution to the Minimum Norm Point (MNP)
 169 problem also provides approximate solutions to all the other problems. We use the notion of *additive*
 170 approximation rather than the relative approximation notion used in the DSG and DSS context. This
 171 is necessary since we are working with unrestricted functions that may be negative. We also prove
 172 that SUPERGREEDY++, the Frank-Wolfe algorithm, and the Wolfe MNP Algorithm are, in fact, all
 173 (under the hood) solving the Minimum Norm Point problem, which explains why these methods
 174 have seen so much success across seemingly unrelated problems.

175 It is worth highlighting an interesting observation about SUPERGREEDY++. Although the algorithm
 176 was originally developed for the DSS problem— a task that itself reduces to SFM —our results
 177 show that SUPERGREEDY++ can also be *directly* interpreted as an algorithm for solving SFM. This
 178 “full-circle” insight, where an approximation algorithm for a derived problem effectively addresses
 179 a more general problem, is surprising and conceptually satisfying.

180 We summarize our contributions more concretely:

- 181 1. **Problem Equivalence.** We prove that the following problems can be reduced to one another in
 182 strongly polynomial time using efficient reductions when solved exactly: Submodular Function
 183 Minimization (SFM), Densest Supermodular Set (DSS), Minimum Norm Point (MNP), Unre-
 184 stricted Sparsest Submodular Set (USSS), and Unrestricted Densest Supermodular Set (UDSS).
- 185 2. **Approximation Transfer.** We prove that approximate solutions to MNP can be transformed into
 186 approximate solutions for all the above problems via approximation-preserving reductions.
- 187 3. **SuperGreedy++ and Wolfe MNP Algorithm as Universal Solvers.** Despite being designed for
 188 specific settings, we prove both SUPERGREEDY++ and Wolfe’s MNP algorithm are universal
 189 solvers for this broader class of problems.
- 190 4. **Efficient New Algorithms for HNSN and Minimum s - t Cut.** As a direct consequence, our
 191 results imply that the Unrestricted Sparsest Submodular Set, a primary motivation of our work,
 192 can be efficiently approximated and solved. This concretely includes, for instance, new faster
 193 algorithms for the HNSN problem that are orders of magnitude faster than all 6 state-of-the-
 194 art baselines compared in [40]. Additionally, we show that SUPERGREEDY++ is, in fact, an

efficient submodular function minimization algorithm and can thus be used to solve the classical minimum s - t cut problem, often converging to the minimum cut within a few iterations.

5. **Empirical Validation at Scale.** We conduct an extensive experimental study involving over 400 trials run across seven distinct problems, encompassing both synthetic and real datasets containing graphs with up to ≈ 100 million edges or elements for set functions. Our experiments systematically compare all major algorithmic paradigms, including flow-based methods, LP solvers, convex optimization techniques (e.g., Wolfe’s MNP algorithm, Frank-Wolfe), and combinatorial baselines (e.g., SUPERGREEDY++), for each problem class: DSG, DSS, USSS, UDSS, SFM, and MNP. Notably, many algorithms that had never been previously explored in certain problem settings (e.g., Wolfe’s MNP algorithm on HNSN or flow-based methods on USSS instances) outperform problem-specific state-of-the-art algorithms from prior work by orders of magnitude.

Our results demonstrate that methods often written off as inefficient, such as Wolfe’s algorithm and max-flow solvers, are not only competitive but frequently faster and more accurate than tailored heuristics *when applied correctly with the right lens*.

Remark on Scope and Related Work. Throughout this paper, we discuss several optimization problems such as SFM, HNSN, DSS, DSG, and MNP, each with a rich history and extensive literature. Due to space constraints, we cannot provide an exhaustive survey or explore all technical nuances and historical developments of each problem. Instead, we have focused on presenting the key connections and ideas necessary for the unified perspective proposed in this work. An expanded version of this paper will include a more comprehensive discussion, including detailed related work and contextual background. Due to space constraints, we include several proofs and experimental results in the supplementary section.

2 Preliminaries and Main Results

Recall that we already defined submodularity, supermodularity, and monotonicity in the introduction. Throughout this paper, we assume that all set functions are normalized without loss of generality

Definition 9 (Base Polymatroid). For a submodular function $f : 2^V \rightarrow \mathbb{R}$, the *base polymatroid* is defined as $B(f) = \{x \in \mathbb{R}^{|V|} : x(S) \leq f(S) \quad \forall S \subseteq V, \quad x(V) = f(V)\}$.

Definition 10 (Base Contrapolymatroid). For a supermodular function $f : 2^V \rightarrow \mathbb{R}$, the *base contrapolymatroid* is defined as $B(f) = \{x \in \mathbb{R}^{|V|} : x(S) \geq f(S) \quad \forall S \subseteq V, \quad x(V) = f(V)\}$.

For functions that are either submodular or supermodular, we refer collectively to the base polymatroid and the base contrapolymatroid as the *base polytope*.

Importance of the Minimum Norm Point in the Base Polytope. A key approach in submodular optimization connects continuous minimization over the base polytope with discrete problems. Specifically, the MINIMUM-NORM-POINT (MNP) problem minimizes $\|x\|_2^2$ over $x \in B(f)$, yielding a unique minimizer x^* . Thresholding x^* often recovers combinatorial minimizers for penalized objectives. While the following connection is classical in submodular function minimization literature, its utility for submodular or supermodular ratio problems has been underexplored. Its significance became clear to us only after recognizing how it simplifies several prior results.

Lemma 11. *Let f be a normalized submodular set function and let x^* be the optimal solution of $\min_{x \in B(f)} \|x\|_2^2$. For any $\lambda \in \mathbb{R}$, define the set $S_\lambda = \{v \in V : x_v^* \leq \lambda\}$. Then, S_λ is a minimizer of the function $f(S) - \lambda|S|$.*

Discussion. Lemma 11 has strong implications, helping establish the equivalence of several problems through efficient, strongly polynomial reductions. Specifically, it shows (together with other ideas) that all the following problems are equivalent, with very efficient strongly-polynomial time reductions. *Proof in Appendix A.2.*

Theorem 12. *The following problems are all equivalent **when solved exactly**, with efficient (strongly polynomial) near-linear time reductions between them: (1) MINIMUM NORM POINT (MNP) (2) SUBMODULAR FUNCTION MINIMIZATION (SFM) (3) DENSEST SUPERMODULAR SET (DSS) (4) UNRESTRICTED SPARSEST SUBMODULAR SET (USSS) (5) UNRESTRICTED DENSEST SUPERMODULAR SET (UDSS)*

Approximation equivalence. Although Theorem 12 shows that the aforementioned problems are efficiently reducible to one another when solved exactly, this does not imply that the reductions are approximation-preserving. However, as the following Theorem demonstrates, it suffices to focus on approximating the MINIMUM NORM POINT problem (MNP).

Theorem 13. Let $f : 2^V \rightarrow \mathbb{R}$ be a normalized submodular or supermodular function, and suppose we can compute $\hat{x} \in B(f)$ satisfying

$$\|\hat{x}\|_2^2 \leq \langle q, \hat{x} \rangle + \varepsilon^2 \quad \text{for all } q \in B(f),$$

i.e., an upper bound on the duality gap. Let $n = |V|$. Then, \hat{x} can be used to efficiently obtain approximate solutions:

1. **(Submodular Function Minimization):** A set \hat{S}_{sfm} with $f(\hat{S}_{\text{sfm}}) \leq f(S_{\text{sfm}}^*) + 2n\varepsilon$.
2. **(Unrestricted Sparsest Submodular Set):** A set \hat{S}_{sparse} with $\frac{f(\hat{S}_{\text{sparse}})}{|\hat{S}_{\text{sparse}}|} \leq \frac{f(S_{\text{sparse}}^*)}{|S_{\text{sparse}}^*|} + 2\varepsilon$.
3. **(Unrestricted Densest Supermodular Set):** A set \hat{S}_{dense} with $\frac{f(\hat{S}_{\text{dense}})}{|\hat{S}_{\text{dense}}|} \geq \frac{f(S_{\text{dense}}^*)}{|S_{\text{dense}}^*|} - 2\varepsilon$.

Notably, this theorem shows that approximating SFM, DSS, UDSS, or USSS reduces to approximating the minimum norm point in $B(f)$.

3 SUPERGREEDY++ and Wolfe’s MNP Algorithm as Universal Solvers

Let $f : 2^V \rightarrow \mathbb{R}$ be a normalized submodular or supermodular function. As indicated in Theorem 13, the main challenge in obtaining approximations for USSS, DSS, UDSS, and SFM is to compute an approximate minimum-norm-point $\hat{x} \in B(f)$. We will discuss three different methods, which we will refer to collectively as *Universal Solvers*.

Frank-Wolfe Algorithm. A classical method for solving MNP is the Frank-Wolfe algorithm, an iterative approach for minimizing a convex function $h : \mathcal{D} \rightarrow \mathbb{R}$ over a compact convex set \mathcal{D} . Each iteration computes the *linear minimization oracle* (LMO) $d^{(k)} = \arg \min_{s \in \mathcal{D}} \langle s, \nabla h(x^{(k-1)}) \rangle$, and updates $x^{(k)} = (1 - \alpha_k)x^{(k-1)} + \alpha_k d^{(k)}$ for $\alpha_k = \frac{2}{k+2}$. For MNP, we set $\mathcal{D} = B(f)$ and $h(x) = \|x\|_2^2$, reducing the LMO to $d^{(k)} = \arg \min_{d \in B(f)} \langle d, x^{(k-1)} \rangle$. This oracle is efficiently realized via Edmonds’ greedy algorithm for (super)submodular functions[20].

Fujishige-Wolfe Minimum Norm Point Algorithm. Another standard approach is the Fujishige-Wolfe algorithm [25], which computes $\hat{x} \in B(f)$ satisfying $\|\hat{x}\|_2^2 \leq \langle \hat{x}, q \rangle + \varepsilon^2$ for all $q \in B(f)$. The algorithm requires $O(nQ^2/\varepsilon^2)$ iterations, where $Q = \max_{q \in B(f)} \|q\|_2$, $n = |V|$.

Like Frank-Wolfe, the Fujishige-Wolfe MNP algorithm repeatedly calls the same linear minimization oracle over $B(f)$ but additionally requires an oracle that minimizes $\|x\|_2^2$ over the affine hull of a set S of selected extreme points of $B(f)$. This affine minimization oracle can be implemented efficiently in $O(|S|^3 + n|S|^2)$ time by computing the inverse of $B^\top B$, where B is the $n \times |S|$ matrix whose columns are the points in S .

SUPERGREEDY++. Next, we demonstrate that SUPERGREEDY++, originally designed for solving the DSS problem, can be used to approximate MNP. Consequently, it is a universal solver for approximating the problems mentioned above.

Theorem 14. Given a normalized submodular or supermodular function $f : 2^V \rightarrow \mathbb{R}$, SUPERGREEDY++ returns a vector \hat{x} satisfying $\|\hat{x}\|_2^2 \leq \langle q, \hat{x} \rangle + \varepsilon^2$ for all $q \in B(f)$, after

$$T = \tilde{O}\left(\frac{\max_{s, d \in B(f)} \|s - x\|_2^2 + n \sum_{u \in V} f(u | V - u)^2}{\varepsilon^2}\right)$$

iterations, where \tilde{O} hides polylogarithmic factors. Consequently, all approximation guarantees from Theorem 13 follow after the same number of iterations.

Proof in Appendix A.4..

4 Experiments

We evaluate our algorithms and baselines on a diverse set of datasets and problem settings, running over 400 experiments across SFM, USSS, and UDSS. Our experiments aim to answer three key questions:

- Which algorithms perform best for which problems, especially when applied beyond their original setting? Are there cases where general-purpose methods (e.g., MNP for USSS) outperform problem-specific heuristics?
- How do convex optimization methods (e.g., Frank-Wolfe, Fujishige-Wolfe MNP) compare to combinatorial (e.g., SUPERGREEDY++), LP-based, and flow-based approaches in runtime and solution quality?
- Can SUPERGREEDY++, Frank-Wolfe, and MNP serve as effective general-purpose solvers across multiple problem classes?

Our empirical study is organized *per problem class*, covering Unrestricted Sparsest Submodular Set (USSS), Submodular Function Minimization (SFM), and Unrestricted Densest Supermodular Set (USSS). We specify problems, datasets, algorithms, and evaluation metrics for each. Experiments were run in parallel on a Slurm-managed cluster (AMD EPYC 7763, 128 cores, 256GB RAM). All methods are implemented in C++20, primarily by the authors, except for specialized baselines (e.g., GREEDY++). **Due to space constraints, we only discuss two experiments here; most of our experimental results are provided in the supplementary section.**

(1) Heavy Nodes in a Small Neighborhood (HNSN). Recall the HNSN problem from the introduction. Since the function $f(S) = |N(S)|$ is submodular, HNSN is a special case of the weighted USSS problem. Ling *et al.* further reformulate the problem over the left vertex set L , defining the function $\bar{N} : 2^L \rightarrow \mathbb{R}$ by $\bar{N}(S) = \{v \in R \mid \delta(v) \subseteq S\}$ where $\delta(v)$ denotes the set of neighbors of v . Under this formulation, the problem becomes that of maximizing $w(\bar{N}(S))/|S|$ over non-empty subsets $S \subseteq L$. We adopt this viewpoint in our implementation.

Algorithms. We compare against the six baselines of Ling *et al.* [40], and introduce a novel flow-based algorithm (detailed in Appendix B). In total, we evaluate **ten algorithms**: IP (SUPERGREEDY++), FW (Frank-Wolfe), MNP (Fujishige-Wolfe Minimum Norm Point Algorithm), FLOW (our new flow-based method), CD (ContractDecompose, [40]), LP (Linear Programming, solved via Gurobi with academic license), GAR (Greedy Approximation, [40]), GR (Greedy, [40]), FGR (Fast Greedy, [40]), and GRR (GreedRatio, [5]). The first four are our implementations; the remaining six follow Ling *et al.*'s code [40].

Datasets. Table 1 summarizes the datasets used for the HNSN problem, following the benchmarks introduced by Ling *et al.* [40]. Each dataset is represented as a weighted bipartite graph with left vertices L , right vertices R , weights on the right vertices $w : R \rightarrow \mathbb{R}_{\geq 0}$, and edges E . The graphs are derived from diverse real-world domains, including recommendation systems (e.g., YooChoose, Kosarak), citation networks (ACM), social networks (NotreDame, IMDB, Digg), and financial transactions (e.g., E-commerce, Liquor). The datasets exhibit a wide range of sizes, with up to $\sim 10^6$ vertices and $\sim 2 \times 10^7$ edges.

Discussion of HNSN Results. All algorithms were given a 30-minute time limit per dataset. The full results are provided in Appendix C; we plot two example runs in Figures 1a and 1b. Across all datasets, FW and MNP consistently emerged as the fastest to converge. In nearly every case, MNP outperformed FW, achieving the best solution quality in the shortest time, with only a single dataset where FW was slightly faster. Relative to the six baselines from Ling *et al.* [40], these methods achieved speedups ranging from $5\times$ to $595\times$.

Our new flow-based algorithm and SUPERGREEDY++ (IP) typically followed, ranking third and fourth in performance across datasets. Figure 1a shows a particularly striking example where, on the YooChoose dataset, MNP achieved a $595\times$ speedup over the best previously published baseline. In this instance, all algorithms converged to the correct final density, except for GRR, which terminated at a suboptimal solution and was therefore excluded from the runtime comparison.

These results underscore a recurring insight consistently observed across our experiments: universal solvers such as MNP and FW, originally developed for broader optimization tasks, can significantly outperform specialized heuristics when applied to specific problems like HNSN, provided they are used within the proper conceptual framework that we have outlined.

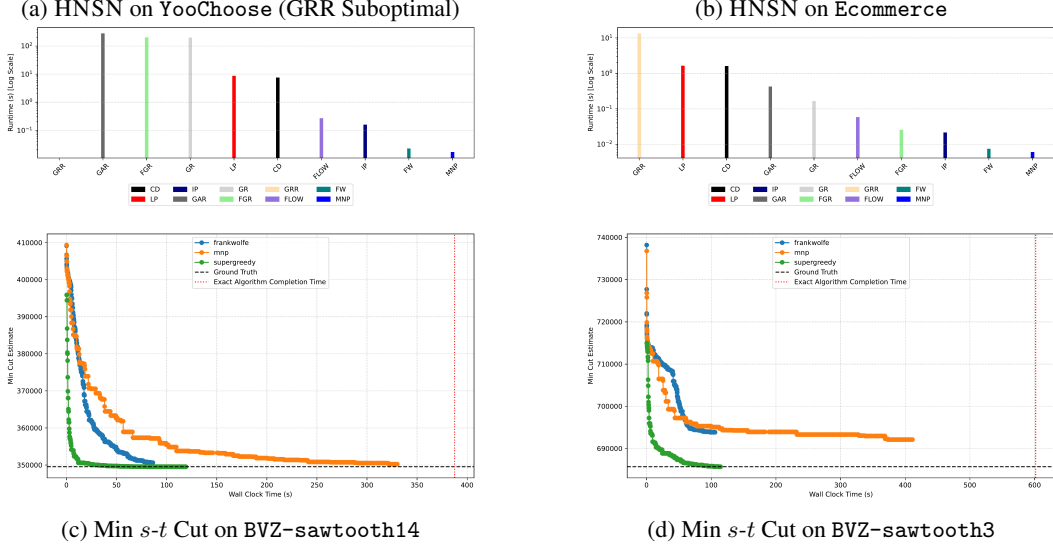


Figure 1: Performance of algorithms across selected HNSN and Minimum $s-t$ Cut instances.

(2) Minimum $s-t$ Cut. We next evaluate our algorithms on the classical *Minimum $s-t$ Cut* problem, a fundamental task in combinatorial optimization with widespread applications. Specifically, the Minimum $s-t$ Cut problem can be framed as minimizing the normalized submodular function $g(S) = |\delta(S \cup \{s\})| - |\delta(\{s\})|$ over subsets $S \subseteq V \setminus \{s, t\}$, ensuring that any feasible solution corresponds to a valid $s-t$ cut. Within this formulation, the Minimum $s-t$ Cut problem becomes an instance of *Submodular Function Minimization (SFM)*, and thus naturally fits within the unified framework established in this paper. This connection allows us to apply universal solvers, such as SUPERGREEDY++, Frank-Wolfe, and Wolfe’s Minimum Norm Point algorithm, to this classical combinatorial problem.

Algorithms. We evaluate three iterative algorithms—Frank-Wolfe (FRANKWOLFE), Fujishige-Wolfe Minimum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY)—on the Minimum $s-t$ Cut problem. We also compute the exact minimum cut value for each instance using standard combinatorial flow-based methods (Edmonds Karp algorithm), serving as ground truth. This allows us to assess both the solution quality and runtime efficiency of the optimization-based approaches relative to the exact combinatorial solution.

Datasets. Our evaluation spans two groups of datasets. First, we consider four classical benchmark instances from the first DIMACS Implementation Challenge on minimum cut [19], known for their small size but challenging cut structures. Second, to assess scalability and robustness, we include a large-scale dataset family, specifically the BVZ-SAWTOOTH instances [36], which consist of 20 distinct minimum $s-t$ cut problems. Each instance contains approximately 500,000 vertices and 800,000 edges, testing algorithmic scalability.

Discussion of Minimum $s-t$ Cut Results. Full results are provided in Appendix D; we plot two select examples in Figures 1c and 1d. Across all datasets, SUPERGREEDY++ was consistently the fastest to approximate the minimum cut, often by large margins over exact flow-based methods, MNP, and FW. In 16 of 24 instances, it found the exact min-cut rapidly. In the remaining cases, its solution was within $1.000023 \times$ the optimum after 500 iterations at most, always converging before the exact flow solver. A notable example is the BVZ-SAWTOOTH14 instance, where SUPERGREEDY++ converges orders of magnitude faster (Figure 1c). This strong performance parallels its success in dense subgraph problems, where it efficiently finds high-quality solutions but can slow down near the optimum. Combining SUPERGREEDY++ with flow-based refinement remains an interesting direction for future work. *Interestingly, while MNP and FW excel on HNSN, they both underperform here, underscoring how problem structure affects solver efficiency.*

Additional Experiments. Beyond the two highlighted experiments, we conduct a comprehensive evaluation across additional problem instances corresponding to each of the general problem classes

Table 1: Summary of HNSN Datasets. All graphs are weighted bipartite graphs.

Dataset	$ L $	$ R $	$ E $	k (Connected components)
Foodmart (FM)	1,559	4,141	18,319	1
E-commerce (EC)	3,468	14,975	174,354	12
Liquor (LI)	4,026	52,131	410,609	165
Fruithut (FR)	1,265	181,970	652,773	4
YooChoose (YC)	107,276	234,300	507,266	22,033
Kosarak (KS)	41,270	990,002	8,019,015	271
Connectious (CN)	458	394,707	1,127,525	117
Digg (DI)	12,471	872,622	22,624,727	13
NotreDame (ND)	127,823	383,640	1,470,404	3,142
IMDB (IM)	303,617	896,302	3,782,463	7,885
NBA Shot (NBA)	129	1,603	13,726	1
ACM Citation (ACM)	751,407	739,969	2,265,837	1

studied. For SFM, we include the classical Contrapolymatroid Membership problem (Appendix H), which tests whether a given vector belongs to the base polytope. For DSS, we evaluate both the Densest Subgraph problem (Appendix E) and the generalized p -mean Densest Subgraph problem [55] (Appendix F). For UDSS, we consider the Anchored Densest Subgraph problem (Appendix G). We also report experiments on the MNP problem (Appendix I). We refer readers to the supplementary section for the complete set of experiments across all problem classes.

Flow-Based Methods: Strengths and Limitations. For the classical Densest Subgraph problem, our experiments reaffirm that flow-based methods remain among the fastest approaches when combined with the density-improvement framework introduced by Veldt *et al.* [32] and Hochbaum [31]. While Hochbaum uses the PseudoFlow algorithm for this task, our experiments demonstrate that comparable performance can be achieved using more standard push-relabel max-flow solvers within the same density-improvement framework. Essentially, the primary source of speedup comes from the iterative density-improvement strategy rather than the specific choice of flow solver. This holds because for a fixed density threshold λ , finding a subset S minimizing $\lambda|S| - f(S) = \lambda|S| - |E(S)|$ naturally reduces to a standard min-cut instance. A similar phenomenon occurs in the HNSN problem: as we outline in the appendix, there exists a flow reduction that efficiently solves subproblems of the form $\lambda|S| - f(S)$ for HNSN, making flow-based methods highly effective here as well.

However, flow-based methods are not universally applicable. For more general problems, such as the generalized p -mean Densest Subgraph problem, no known linear flow network formulations can minimize objectives like $\lambda|S| - f(S)$ for arbitrary supermodular functions. In such cases, flow-based methods cannot be used, and one must resort to algorithms like SUPERGREEDY++, Frank-Wolfe, or MNP, which only require access to a value oracle. These methods thus offer broader applicability across diverse problem settings where flow reductions are unavailable.

Conclusion and Limitations. The overarching conclusion of our study is clear: SUPERGREEDY++, Frank-Wolfe, and the Fujishige-Wolfe MNP algorithms consistently achieve state-of-the-art performance across a wide range of submodular and supermodular ratio problems. Given the broad applicability of DSS, USSS, SFM, and related formulations, we advocate for these three methods to be included as essential baselines in future empirical evaluations of such problems.

A limitation of our current work is that while one of these algorithms consistently outperforms problem-specific heuristics, the identity of the best-performing method varies with the problem instance. Developing a deeper understanding of when and why each algorithm excels remains an open question. Moreover, while our results provide general additive approximation guarantees via reductions to the Minimum Norm Point problem, it remains an interesting open question to develop a direct, problem-specific analysis of algorithms like SUPERGREEDY++ in settings such as minimum s - t cut. Finally, while we discussed related work on coordinate descent methods, such as recent work by Nguyen *et al.* [44], we did not dive deep into the broader family of decomposable submodular function minimization (DSFM) techniques in detail. A systematic exploration of general DSFM methods and their applicability remains an important 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, feb 2012.
- [3] Brian Axelrod, Yang P Liu, and Aaron Sidford. Near-optimal approximate discrete and continuous submodular function minimization. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 837–853. SIAM, 2020.
- [4] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for mapreduce. In *Algorithms and Models for the Web Graph: 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings*, page 59–78, Berlin, Heidelberg, 2014. Springer-Verlag.
- [5] Wenruo Bai, Rishabh Iyer, Kai Wei, and Jeff Bilmes. Algorithms for optimizing the ratio of submodular functions. In *International Conference on Machine Learning*, pages 2751–2759. PMLR, 2016.
- [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*, page 379–388, New York, NY, USA, 2015. Association for Computing Machinery.
- [7] Yuly Billig. Eigenvalue approach to dense clusters in hypergraphs. *Journal of Graph Theory*, 2025.
- [8] 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.
- [9] 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.
- [10] Deeparnab Chakrabarty, Prateek Jain, and Pravesh Kothari. Provable submodular minimization using wolfe’s algorithm. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1, NIPS’14*, page 802–809, Cambridge, MA, USA, 2014. MIT Press.
- [11] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic submodular function minimization. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1220–1231, 2017.
- [12] Karthekeyan Chandrasekaran, Chandra Chekuri, and Shubhang Kulkarni. On deleting vertices to reduce density in graphs and supermodular functions. *arXiv preprint arXiv:2503.08828*, 2025.
- [13] Karthekeyan Chandrasekaran, Chandra Chekuri, Manuel R. Torres, and Weihao Zhu. On the Generalized Mean Densest Subgraph Problem: Complexity and Algorithms. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024)*, volume 317 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [14] 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.

- [15] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.
- [16] 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.
- [17] 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.
- [18] Werner Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.
- [19] Michal Dobrogost. Solving Maximum Flow Problems in J — cucave.net. <http://www.cucave.net/papers/jmaxflow/>. [Accessed 16-05-2025].
- [20] 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.
- [21] 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.
- [22] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.
- [23] Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, 1980.
- [24] Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.
- [25] Satoru Fujishige, Takumi Hayashi, and Shiguelo Isotani. The minimum-norm-point algorithm applied to submodular function minimization and linear programming. 2006.
- [26] Satoru Fujishige and Shiguelo Isotani. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7(1):3–17, 2011.
- [27] A. V. Goldberg. Finding a maximum density subgraph. Technical Report UCB/CSD-84-171, EECS Department, University of California, Berkeley, 1984.
- [28] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.
- [29] Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. Faster and scalable algorithms for densest subgraph and decomposition. *Advances in Neural Information Processing Systems*, 35:26966–26979, 2022.
- [30] Elfarouk Harb, Kent Quanrud, and Chandra Chekuri. Convergence to lexicographically optimal base in a (contra) polymatroid and applications to densest subgraph and tree packing. *arXiv preprint arXiv:2305.02987*, 2023.
- [31] Dorit S. Hochbaum. Flow is best, fast and scalable: The incremental parametric cut for maximum density and other ratio subgraph problems. In *The 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KDIR 2024), Proceedings*, November 2024. 17-19 November, 2024.

- [32] Yufan Huang, David F. Gleich, and Nate Veldt. Densest subhypergraph: Negative supermodular functions and strongly localized methods. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 881–892, New York, NY, USA, 2024. Association for Computing Machinery.
- [33] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.
- [34] 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 Jun 2013. PMLR.
- [35] Stefanie Jegelka, Hui Lin, and Jeff A Bilmes. On fast approximate submodular minimization. *Advances in Neural Information Processing Systems*, 24, 2011.
- [36] Patrick Møller Jensen, Niels Jeppesen, Anders Bjorholm Dahl, and Vedrana Andersen Dahl. Min-Cut/Max-Flow Problem Instances for Benchmarking. 3 2022.
- [37] Haotian Jiang. Minimizing convex functions with rational minimizers. *Journal of the ACM*, 70(1):1–27, 2022.
- [38] Yuko Kuroki, Atsushi Miyauchi, Junya Honda, and Masashi Sugiyama. Online dense subgraph discovery via blurred-graph feedback. In *ICML*, 2020.
- [39] Tommaso Lanciano, Atsushi Miyauchi, Adriano Fazzzone, and Francesco Bonchi. A survey on the densest subgraph problem and its variants, 2023.
- [40] Ling Li, Hilde Verbeek, Huiping Chen, Grigorios Loukides, Robert Gwadera, Leen Stougie, and Solon P. Pissis. Heavy nodes in a small neighborhood: Exact and peeling algorithms with applications. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–16, 2024.
- [41] 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.
- [42] 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.
- [43] 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.
- [44] Ta Duy Nguyen and Alina Ene. Multiplicative weights update, area convexity and random coordinate descent for densest subgraph problems. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.
- [45] James B Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.
- [46] 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.
- [47] 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.
- [48] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

- 554 [49] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24.
555 Springer, 2003.
- 556 [50] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Corescope: Graph mining using k-core
557 analysis — patterns, anomalies and algorithms. In *2016 IEEE 16th International Conference*
558 *on Data Mining (ICDM)*, pages 469–478, 2016.
- 559 [51] Bintao Sun, Maximilien Danisch, T-H. Hubert Chan, and Mauro Sozio. Kclist++: A sim-
560 ple algorithm for finding k-clique densest subgraphs in large graphs. *Proc. VLDB Endow.*,
561 13(10):1628–1640, jun 2020.
- 562 [52] Charalampos Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th*
563 *International Conference on World Wide Web, WWW ’15*, page 1122–1132, Republic and
564 Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Commit-
565 tee.
- 566 [53] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria
567 Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guar-
568 antees. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge*
569 *Discovery and Data Mining, KDD ’13*, page 104–112, New York, NY, USA, 2013. Associa-
570 tion for Computing Machinery.
- 571 [54] Nate Veldt, Austin R. Benson, and Jon Kleinberg. The generalized mean densest subgraph
572 problem, 2021.
- 573 [55] Nate Veldt, Austin R. Benson, and Jon Kleinberg. The generalized mean densest subgraph
574 problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery*
575 *& Data Mining, KDD ’21*, page 1604–1614, New York, NY, USA, 2021. Association for
576 Computing Machinery.
- 577 [56] Philip Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–
578 149, 1976.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: All the claims made in the abstract and introduction are proved later on in the paper.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: The limitations is discussed in the last paragraph.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All proofs are given in the appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The code for every single experiment is given in the supplementary section. All datasets are also provided.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: All data is open access. The code base gives clear instructions on how to reproduce the experiments.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: All methods are explained in details in the appendix or supplementary section.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [NA]

Justification: All algorithms are deterministic, so they produce the same output on every run (so there are no error bars).

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The system specifications and resources were given in the paper.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines?>

Answer: [Yes]

768 Justification: Yes, we followed it as far as we are aware.
769 Guidelines:
770 • The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
771 • If the authors answer No, they should explain the special circumstances that require a
772 deviation from the Code of Ethics.
773 • The authors should make sure to preserve anonymity (e.g., if there is a special consid-
774 eration due to laws or regulations in their jurisdiction).

775 10. Broader impacts

776 Question: Does the paper discuss both potential positive societal impacts and negative
777 societal impacts of the work performed?

778 Answer: [NA]

779 Justification: This paper presents work whose goal is to advance theoretical machine learn-
780 ing and applied algorithms. The work is mostly theoretical, so we do not believe there are
781 many potential societal consequences of our work (unless used very nefariously) which we
782 feel must be specifically highlighted here.

783 Guidelines:

- 784 • The answer NA means that there is no societal impact of the work performed.
785 • If the authors answer NA or No, they should explain why their work has no societal
786 impact or why the paper does not address societal impact.
787 • Examples of negative societal impacts include potential malicious or unintended uses
788 (e.g., disinformation, generating fake profiles, surveillance), fairness considerations
789 (e.g., deployment of technologies that could make decisions that unfairly impact spe-
790 cific groups), privacy considerations, and security considerations.
791 • The conference expects that many papers will be foundational research and not tied
792 to particular applications, let alone deployments. However, if there is a direct path to
793 any negative applications, the authors should point it out. For example, it is legitimate
794 to point out that an improvement in the quality of generative models could be used to
795 generate deepfakes for disinformation. On the other hand, it is not needed to point out
796 that a generic algorithm for optimizing neural networks could enable people to train
797 models that generate Deepfakes faster.
798 • The authors should consider possible harms that could arise when the technology is
799 being used as intended and functioning correctly, harms that could arise when the
800 technology is being used as intended but gives incorrect results, and harms following
801 from (intentional or unintentional) misuse of the technology.
802 • If there are negative societal impacts, the authors could also discuss possible mitiga-
803 tion strategies (e.g., gated release of models, providing defenses in addition to attacks,
804 mechanisms for monitoring misuse, mechanisms to monitor how a system learns from
805 feedback over time, improving the efficiency and accessibility of ML).

806 11. Safeguards

807 Question: Does the paper describe safeguards that have been put in place for responsible
808 release of data or models that have a high risk for misuse (e.g., pretrained language models,
809 image generators, or scraped datasets)?

810 Answer: [NA]

811 Justification: No such risks.

812 Guidelines:

- 813 • The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [\[Yes\]](#)

Justification: When we use other authors' implementation, code, or datasets, they are properly credited.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: All datasets and code are provided.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Not involving crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Not involving crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [NA]

Justification: No LLM usage beyond basic writing and language editing.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.

A Proofs

A.1 Proof of Lemma 11

Proof. Fix λ and $S \subseteq V$. Since $x^* \in B(f)$, we have

$$f(S) - \lambda|S| \geq \sum_{v \in S} (x_v^* - \lambda). \quad (1)$$

The right-hand side of (1) is minimized by taking $S_\lambda = \{v \in V \mid x_v^* \leq \lambda\}$, so

$$\sum_{v \in S} (x_v^* - \lambda) \geq \sum_{v \in S_\lambda} (x_v^* - \lambda) = x^*(S_\lambda) - \lambda|S_\lambda|. \quad (2)$$

We claim that $x^*(S_\lambda) = f(S_\lambda)$. Combining this with (1) and (2) gives

$$f(S_\lambda) - \lambda|S_\lambda| \leq f(S) - \lambda|S|,$$

as desired.

To prove the claim, order V so that $x_{v_1}^* \leq x_{v_2}^* \leq \dots \leq x_{v_n}^*$, and define $S_i = \{v_1, \dots, v_i\}$. For some i , we have $S_\lambda = S_i$, so it suffices to show $x^*(S_i) = f(S_i)$ for all i . This follows from [24] Lemma 7.4 (pp. 218–219, equations 7.12–7.15).

□

A.2 Proof of Theorem 12

Proof. (1) \Rightarrow (4)/(5): Let $f : 2^V \rightarrow \mathbb{R}$ be a normalized submodular or supermodular function. We focus on submodular functions for USSS; the supermodular case (UDSS) is analogous via negation.

Our goal is to compute the minimum norm point $x^* \in B(f)$ using an oracle for USSS. We iteratively build x^* by decomposing f into a sequence of sparsest sets.

Initialize $f_0 = f$. At iteration $i \geq 1$, define S_i as the minimizer of $f_i(S)/|S|$ over non-empty $S \subseteq V \setminus (S_1 \cup \dots \cup S_{i-1})$. Set $x_v = f_i(S_i)/|S_i|$ for all $v \in S_i$.

After assigning x_v for $v \in S_i$, contract S_i by defining:

$$f_{i+1}(S) = f_i(S \cup S_i) - f_i(S_i), \quad \forall S \subseteq V \setminus (S_1 \cup \dots \cup S_i).$$

Repeat until $S_1 \cup \dots \cup S_k = V$. This process requires at most n iterations since the S_i are disjoint.

Claim: The resulting vector x is the minimum norm point in $B(f)$.

Proof of Claim: Let x be the vector constructed by the above process, where for each $v \in S_i$, we set $x_v = \lambda_i = f_i(S_i)/|S_i|$.

We now prove that x is the minimum norm point in $B(f)$ by showing that x is the lexicographically maximum base of $B(f)$. Recall that the lex-maximal vector corresponds to the minimum norm point in a polymatroid.

We proceed by induction on i to show that for each i , any lex-minimal base $x^* \in B(f)$ must satisfy $x_v^* = \lambda_i$ for all $v \in S_i$.

Base case ($i = 1$): Since $x^* \in B(f)$, we have

$$x^*(S_1) \leq f(S_1) = \lambda_1|S_1|.$$

But by construction, $x_v = \lambda_1$ for all $v \in S_1$, so $x(S_1) = \lambda_1|S_1|$.

Therefore, $x^*(S_1) \leq x(S_1)$. In particular, there exists at least one $v \in S_1$ with $x_v^* \leq \lambda_1$.

Since x^* is lexicographically maximal, we must have $x_v^* = \lambda_1$ for all $v \in S_1$; otherwise, if any $x_v^* < \lambda_1$ for $v \in S_1$, lex-maximality would be violated.

Induction step: Assume for $j = 1, \dots, i$, we have $x_v^* = \lambda_j$ for all $v \in S_j$.

Consider S_{i+1} . Since $x^* \in B(f)$, we have

$$x^*(S_1 \cup \dots \cup S_{i+1}) \leq f(S_1 \cup \dots \cup S_{i+1}).$$

940 Using telescoping sums, this becomes:

$$f(S_1 \cup \dots \cup S_{i+1}) = \sum_{h=1}^{i+1} [f(S_1 \cup \dots \cup S_h) - f(S_1 \cup \dots \cup S_{h-1})] = \sum_{h=1}^{i+1} \lambda_h |S_h|.$$

941 Thus,

$$x^*(S_{i+1}) = x^*(S_1 \cup \dots \cup S_{i+1}) - x^*(S_1 \cup \dots \cup S_i) \leq \lambda_{i+1} |S_{i+1}|,$$

942 where the induction hypothesis gives $x^*(S_1 \cup \dots \cup S_i) = \sum_{h=1}^i \lambda_h |S_h|$.

943 Hence, the average value of x_v^* over $v \in S_{i+1}$ is at most λ_{i+1} . Therefore, there must exist at least
944 one $v \in S_{i+1}$ with $x_v^* \leq \lambda_{i+1}$.

945 Since x^* is lexicographically maximal, we must have $x_v^* = \lambda_{i+1}$ for all $v \in S_{i+1}$; otherwise, if any
946 $x_v^* < \lambda_{i+1}$, lex-maximality would be contradicted.

947 By induction, $x_v^* = \lambda_i$ for all $v \in S_i$ and all i . Hence, $x^* = x$. Thus, the constructed vector x is the
948 lex-maximal base of $B(f)$, which is also the minimum norm point.

949 **(4) \Leftrightarrow (5):** Immediate from duality. If f is supermodular, $-f$ is submodular. The sparsest sub-
950 modular set problem becomes densest supermodular set under negation. Hence, solving USSS is
951 equivalent to solving UDSS on $-f$.

952 **(5) \Rightarrow (3):** Given a normalized supermodular function f , define:

$$C = \max\{0, \max_{S \subseteq V} (-f(S))\},$$

953 and set $g(S) = f(S) + C|S|$. Then g is normalized, monotone, non-negative, and supermodular.
954 Furthermore,

$$\arg \max_{S \subseteq V} \frac{g(S)}{|S|} = \arg \max_{S \subseteq V} \frac{f(S)}{|S|},$$

955 as adding a constant C to all sets gains affects neither the maximizer (as it just shifts the sparsity
956 ratio). Thus, UDSS reduces to DSS.

957 **(3) \Rightarrow (2):** This reduction is classical (e.g., [18, 32, 31]). Given monotone, normalized, non-negative
958 supermodular f , we iteratively refine a maximizer of $f(S)/|S|$ using SFM. Initialize $S_0 = V$,
959 $\lambda_1 = f(S_0)/|S_0|$. At iteration i , solve:

$$S_i = \arg \min_{S \subseteq V, S \neq \emptyset} (\lambda_i |S| - f(S)),$$

960 using SFM. Update $\lambda_{i+1} = f(S_i)/|S_i|$. Iterate until $\lambda_k |S_k| - f(S_k) = 0$. Since the sequence of λ_i
961 decreases and f is normalized, the process terminates in at most n steps. The last S_{k-1} maximizes
962 $f(S)/|S|$, solving DSS via SFM.

963 **(2) \Rightarrow (1):** Given f , compute the minimum-norm point $x^* \in B(f)$. By Lemma 11, the set

$$S_0 = \{v \in V \mid x_v^* \leq 0\}$$

964 minimizes $f(S)$. Thus, SFM can be reduced to computing x^* , completing the reduction.

965 □

966 A.3 Proof of Theorem 13

967 *Proof.* **(1)** This result follows directly from the analysis in [10].

968 **(2)** We generalize the argument of [10]. Without loss of generality, reorder indices so that $\hat{x}_1 \leq$
969 $\hat{x}_2 \leq \dots \leq \hat{x}_n$. Consider the set $\hat{S} = [i] := \{1, \dots, i\}$ that minimizes $f([i])/i$ over $i = 1, \dots, n$.

970 Let λ^* denote the density of the sparsest submodular set, i.e., $\lambda^* = \min_{S \neq \emptyset} f(S)/|S|$. Define k as
971 the smallest index satisfying:

972 (C1) $\hat{x}_{k+1} > \lambda^*$,

973 (C2) $\hat{x}_{k+1} - \hat{x}_k \geq \frac{\varepsilon}{k}$.

974 From [10], we have the key inequality:

$$\sum_{i=1}^{n-1} (\hat{x}_{i+1} - \hat{x}_i)(f([i]) - \hat{x}([i])) \leq \varepsilon^2.$$

975 Let $t = |\{i \in [k] : \hat{x}_i > \lambda^*\}|$. Observe:

$$\begin{aligned} \sum_{i \in [k] : \hat{x}_i > \lambda^*} \hat{x}_i &= \lambda^* t + \sum_{i \in [k] : \hat{x}_i > \lambda^*} (\hat{x}_i - \lambda^*) \\ &\leq \lambda^* t + \sum_{i \in [k]} \frac{i \cdot \varepsilon}{k} \\ &\leq \lambda^* t + k\varepsilon, \end{aligned}$$

976 where the second inequality follows since (C2) fails for all $i < k$ with $\hat{x}_i > \lambda^*$.

977 Since $\hat{x}_{k+1} - \hat{x}_k \geq \frac{\varepsilon}{k}$, it follows that

$$f([k]) - \hat{x}([k]) \leq k\varepsilon,$$

978 and therefore,

$$\begin{aligned} \frac{f([k])}{k} &\leq \frac{\hat{x}([k]) + k\varepsilon}{k} \\ &= \frac{\sum_{i \in [k] : \hat{x}_i \leq \lambda^*} \hat{x}_i + \sum_{i \in [k] : \hat{x}_i > \lambda^*} \hat{x}_i + k\varepsilon}{k} \\ &\leq \frac{\lambda^*(k-t) + \lambda^* t + 2k\varepsilon}{k} = \lambda^* + 2\varepsilon. \end{aligned}$$

979 **(3)** The argument parallels that of (2). We now reorder indices so that $\hat{x}_1 \geq \hat{x}_2 \geq \dots \geq \hat{x}_n$, and
980 select the set $\hat{S} = [i] := \{1, \dots, i\}$ that maximizes $f([i])/i$.

981 Let $\lambda^* = \max_{S \neq \emptyset} f(S)/|S|$. Define k as the smallest index satisfying:

982 (C1) $\hat{x}_{k+1} < \lambda^*$,

983 (C2) $\hat{x}_k - \hat{x}_{k+1} \geq \frac{\varepsilon}{k}$.

984 Using the analogous inequality from [10]:

$$\sum_{i=1}^{n-1} (\hat{x}_i - \hat{x}_{i+1})(\hat{x}([i]) - f([i])) \leq \varepsilon^2.$$

985 Let $t = |\{i \in [k] : \hat{x}_i < \lambda^*\}|$. We have:

$$\begin{aligned} \sum_{i \in [k] : \hat{x}_i < \lambda^*} \hat{x}_i &= \lambda^* t + \sum_{i \in [k] : \hat{x}_i < \lambda^*} (\hat{x}_i - \lambda^*) \\ &\geq \lambda^* t - \sum_{i \in [k]} \frac{i \cdot \varepsilon}{k} \\ &\geq \lambda^* t - k\varepsilon, \end{aligned}$$

986 where the second inequality holds since (C2) fails for all $i < k$ with $\hat{x}_i < \lambda^*$.

987 Since $\hat{x}_{k-1} - \hat{x}_k \geq \frac{\varepsilon}{k}$, we deduce:

$$\hat{x}([k]) - f([k]) \leq k\varepsilon,$$

988 and consequently,

$$\begin{aligned} \frac{f([k])}{k} &\geq \frac{\hat{x}([k]) - k\varepsilon}{k} \\ &= \frac{\sum_{i \in [k] : \hat{x}_i \geq \lambda^*} \hat{x}_i + \sum_{i \in [k] : \hat{x}_i < \lambda^*} \hat{x}_i - k\varepsilon}{k} \\ &\geq \frac{\lambda^*(k-t) + \lambda^* t - 2k\varepsilon}{k} = \lambda^* - 2\varepsilon. \end{aligned}$$

989

□

990 A.4 Proof of Theorem 14

991 *Proof.* Our analysis follows the approach of Harb *et al.* [30], based on the Frank-Wolfe framework
992 of Jaggi [34].

993 **Jaggi’s Frank-Wolfe Framework.** The Frank-Wolfe algorithm is a classical method for minimiz-
994 ing a convex function over a convex set. While its convergence has been known since [22], Jaggi’s
995 analysis [34] offers a modern, simplified treatment that expresses convergence rates in terms of the
996 so-called *curvature constant* of the objective function.

997 **Definition 15** (Curvature constant). Let $\mathcal{D} \subseteq \mathbb{R}^d$ be a compact convex set, and $h : \mathcal{D} \rightarrow \mathbb{R}$ a convex,
998 differentiable function. The curvature constant C_h is defined as

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

999 One advantage of Jaggi’s formulation is that it seamlessly extends to *approximate* versions of Frank-
1000 Wolfe, where the linear minimization oracle (LMO) may only be computed approximately.

1001 **Definition 16** (Approximate LMO). Given $h : \mathcal{D} \rightarrow \mathbb{R}$, an ε -approximate linear minimization
1002 oracle returns $\hat{s} \in \mathcal{D}$ satisfying

$$\langle \hat{s}, \nabla h(w) \rangle \leq \langle s^*, \nabla h(w) \rangle + \varepsilon,$$

1003 where $s^* = \arg \min_{s \in \mathcal{D}} \langle s, \nabla h(w) \rangle$ is the exact LMO.

1004 Jaggi’s main result shows that if, at iteration k , we compute an approximate LMO with error at most
1005 $\frac{\delta C_h}{k+2}$, then the Frank-Wolfe algorithm converges with only a constant-factor slowdown.

1006 **Lemma 17** ([34]). Define the duality gap at $x \in \mathcal{D}$ as

$$g(x) = \max_{q \in \mathcal{D}} \langle x - q, \nabla h(x) \rangle.$$

1007 If we use a $\frac{\delta C_h}{k+2}$ -approximate LMO at iteration k , then after K iterations there exists an iterate $x^{(k)}$,
1008 with $1 \leq k \leq K$, satisfying

$$g(x^{(k)}) \leq \frac{7C_h}{K+2} (1 + \delta).$$

1009 **Applying to SUPERGREEDY++ and MNP.** Our goal is to analyze SUPERGREEDY++ for the
1010 Minimum Norm Point (MNP) problem on the base polytope of a normalized submodular or super-
1011 modular function $f : 2^V \rightarrow \mathbb{R}$. Specifically, we consider

$$h(x) = \|x\|_2^2, \quad \mathcal{D} = B(f).$$

1012 For this quadratic objective, the curvature constant admits a simple form:

Lemma 18.

$$C_h = 2 \max_{s, d \in B(f)} \|s - d\|_2^2.$$

1013 *Proof.* This follows by substituting $h(x) = \|x\|_2^2$ into the definition of C_h . The supremum is
1014 attained when x , s , and d are chosen to maximize $\|s - d\|_2^2$, yielding the claimed expression. \square

1015 Next, note that we can, without loss of generality, assume f is a supermodular function. This is
1016 because submodular and supermodular MNP problems are duals of each other under negation:

1017 **Lemma 19.** Given a normalized submodular function g , define $f = -g$. If we can solve the MNP
1018 problem for f , i.e., find $\hat{x} \in B(f)$ such that

$$\|\hat{x}\|_2^2 \leq \langle q, \hat{x} \rangle + \varepsilon^2 \quad \text{for all } q \in B(f),$$

1019 then we can obtain a corresponding solution $x' = -\hat{x} \in B(g)$ with the same guarantee:

$$\|x'\|_2^2 \leq \langle q, x' \rangle + \varepsilon^2 \quad \text{for all } q \in B(g).$$

1020 *Proof.* Given g , let $f = -g$. Since f is submodular, compute $\hat{x} \in B(f)$ such that

$$\|\hat{x}\|_2^2 \leq \langle q, \hat{x} \rangle + \varepsilon^2 \quad \text{for any } q \in B(f).$$

1021 Define $x' = -\hat{x}$. Observe that $q \in B(f)$ if and only if $-q \in B(g)$. Specifically, $q(V) = f(V)$
 1022 implies $-q(V) = -f(V) = g(V)$, and $q(S) \geq f(S)$ if and only if $-q(S) \leq -f(S) = g(S)$. Thus,
 1023 $x' \in B(g)$ and, for any $q \in B(g)$, since $-q, -x' \in B(f)$,

$$\|x'\|_2^2 = \|-x'\|_2^2 \leq \langle -q, -x' \rangle + \varepsilon^2 = \langle q, x' \rangle + \varepsilon^2.$$

1024 □

1025 **Approximate LMO in SUPERGREEDY++.** Algorithm 2 reframes SUPERGREEDY++ as a
 1026 “noisy” Frank-Wolfe algorithm. At each iteration, SUPERGREEDY++ computes a descent direction
 1027 using the PEELWEIGHTED subroutine. While this is not an exact LMO, Harb *et al.* [30] showed that
 1028 it approximates the LMO to within an error term that decays with t :

1029 **Lemma 20** ([30]). *Let s_t denote the exact LMO direction at iteration t , and \hat{d}_t be the direction*
 1030 *returned by PEELWEIGHTED. Then,*

$$\langle s_t, b^{(t-1)} \rangle \leq \langle \hat{d}_t, b^{(t-1)} \rangle + \frac{n \sum_{u \in V} f(u \mid V \setminus \{u\})^2}{t}.$$

1031 This shows that at iteration t , SUPERGREEDY++ uses a

$$\frac{n \sum_{u \in V} f(u \mid V \setminus \{u\})^2}{t}$$

1032 approximate LMO error. Normalizing by the curvature constant C_h , this corresponds to

$$\delta = \frac{n \sum_{u \in V} f(u \mid V \setminus \{u\})^2}{C_h}.$$

1033 **Convergence of SUPERGREEDY++.** Substituting this approximation quality into Jaggi’s conver-
 1034 gence lemma, we find that after T iterations, there exists an iterate $x^{(k)}$ with

$$g(x^{(k)}) \leq \frac{7C_h}{T+2} (1 + \delta) = \frac{7C_h}{T+2} \left(1 + \frac{n \sum_{u \in V} f(u \mid V \setminus \{u\})^2}{C_h} \right).$$

1035 Simplifying, this gives

$$g(x^{(k)}) \leq \frac{7}{T+2} \left(C_h + n \sum_{u \in V} f(u \mid V \setminus \{u\})^2 \right).$$

1036 To ensure $g(x^{(k)}) \leq \varepsilon^2$, it suffices to choose

$$T = O \left(\frac{C_h + n \sum_{u \in V} f(u \mid V \setminus \{u\})^2}{\varepsilon^2} \right) = O \left(\frac{\max_{s, d \in B(f)} \|s - d\|_2^2 + n \sum_{u \in V} f(u \mid V \setminus \{u\})^2}{\varepsilon^2} \right),$$

1037 The \tilde{O} polylog term comes from the fact that we use a learning rate (in SUPERGREEDY++) of
 1038 $1/(t+1)$ instead of $2/(t+2)$.

1039 □

Algorithm 1 PeelingWeighted++

```
1: function PEELWEIGHTED( $f : 2^V \rightarrow \mathbb{R}, w \in \mathbb{R}$ )
2: Initialize:  $\hat{d}(u) = 0$  for all  $u \in V$ 
3:  $S_1 \leftarrow V$ 
4: for  $j = 1$  to  $n$  do
5:    $v_j \leftarrow \arg \max_{v \in S_j} \{w(v) + f(v \mid S_j - v)\}$ 
6:    $\hat{d}(v_j) \leftarrow f(v_j \mid S_j - v_j)$ 
7:    $S_{j+1} \leftarrow S_j \setminus \{v_j\}$ 
8: end for
9: return  $\hat{d}$ 
```

Algorithm 2 SuperGreedy++ For Minimum Norm Point

```
1: function SUPERGREEDY++( $f : 2^V \rightarrow \mathbb{R}, T$ )
2: Initialize:  $x^{(0)}(u) = 0$  for all  $u \in V$ 
3: for  $t = 1$  to  $T$  do
4:    $\hat{d}_t \leftarrow \text{PEELWEIGHTED}(f, (t-1)x^{(t-1)})$ 
5:    $x^{(t)} \leftarrow \left(1 - \frac{1}{t+1}\right)x^{(t-1)} + \frac{1}{t+1}\hat{d}_t$ 
6: end for
7: return  $x^{(k)}$ , for  $0 \leq k \leq T$ , that minimizes  $\|x^{(k)}\|_2^2 - \max_{q \in B(f)} \langle q, x^{(k)} \rangle$ 
```

B A New Flow-Based Algorithm for HNSN

Recall the HNSN problem from the introduction. Since the function $f(S) = |N(S)|$ is submodular, HNSN is a special case of the weighted USSS problem. Ling *et al.* further reformulate the problem over the left vertex set L , defining the function $\bar{N} : 2^L \rightarrow \mathbb{R}$ by $\bar{N}(S) = \{v \in R \mid \delta(v) \subseteq S\}$ where $\delta(v)$ denotes the set of neighbors of v . Under this formulation, the problem becomes that of maximizing $w(\bar{N}(S))/|S|$ over non-empty subsets $S \subseteq L$. We adopt this viewpoint in our implementation.

We adopt the density improvement framework from the proof of Theorem 12. We begin by setting $S_0 = L$ and define the function $f : 2^L \rightarrow \mathbb{R}_{\geq 0}$ by $f(S) = w(\bar{N}(S))$, where $\bar{N}(S) = \{v \in R \mid \delta(v) \subseteq S\}$. We initialize the density parameter as $\lambda_0 = f(S_0)/|S_0|$, and then minimize the function $\lambda_0|S| - f(S)$ over subsets $S \subseteq L$. Let S_1 be the minimizer of this objective, and set $\lambda_1 = f(S_1)/|S_1|$. This process is repeated iteratively. We will show how each minimization step can be performed using a single min-cut (or, equivalently, max-flow) computation. Algorithm 3 summarizes this iterative process.

Algorithm 3 Density Improvement via Min-Cut

Require: Left vertex set L , right vertex set R , weight function $w : R \rightarrow \mathbb{R}_{\geq 0}$, neighborhood function $\delta : R \rightarrow 2^L$

- 1: Define $\bar{N}(S) \leftarrow \{v \in R \mid \delta(v) \subseteq S\}$
- 2: Define $f(S) \leftarrow \sum_{v \in \bar{N}(S)} w(v)$
- 3: Initialize $S_0 \leftarrow L, t \leftarrow 1$
- 4: Compute $\lambda_0 \leftarrow f(S_0)/|S_0|$
- 5: **repeat**
- 6: Define $\Phi(S) \leftarrow \lambda_{t-1} \cdot |S| - f(S)$
- 7: Find $S_t \subseteq L$ minimizing $\Phi(S)$ via a min-cut or max-flow computation
- 8: Compute $\lambda_t \leftarrow f(S_t)/|S_t|$
- 9: $t \leftarrow t + 1$
- 10: **until** $S_t = S_{t-1}$
- 11: **return** S_t

We now show how to minimize $\Phi(S) = \lambda|S| - f(S)$ for a fixed λ .

Theorem 21. *We can minimize $\Phi(S) = \lambda|S| - f(S)$ for a fixed λ using a single minimum s - t cut.*

Proof. Construct a directed graph $G = (\{s, t\} \cup L \cup R, E)$ as follows:

- Add an edge from s to each $u \in L$ with capacity λ .
- For each edge (u, v) with $u \in L, v \in R$, if $v \in \delta(u)$, add an edge $u \rightarrow v$ with capacity ∞ .
- Add an edge from each $v \in R$ to t with capacity $w(v)$.

Consider an s - t cut (A, B) , with $s \in A, t \in B$. Define the subset $S = L \cap B$, i.e., the left-hand vertices that are **cut off from** s by the cut. We now compute the total capacity of the cut.

The cut capacity consists of:

1. Edges from s to S , each with capacity λ , totaling $\lambda|S|$.
2. For each $v \in R$, the edge $v \rightarrow t$ is only cut if there exists any of its neighbors in S , i.e., $\delta(v) \cap S \neq \emptyset$, meaning $v \in \bar{N}(S)$ are not cut to t . The edge $v \rightarrow t$ contributes $w(R) - w(\bar{N}(S))$ to the cut.

Thus, the total cut capacity is:

$$\lambda|S| + w(R) - w(\bar{N}(S))$$

To minimize $\Phi(S) = \lambda|S| - f(S)$, we observe that this is equivalent (up to constants) to minimizing $\lambda|S| + w(R) - w(\bar{N}(S))$ since adding $w(R)$ from both sides doesn't change the argmin. Hence, the minimum s - t cut yields a subset that minimizes $\Phi(S)$, as claimed. \square

C HNSN Experiments Results

Problem Definition. Recall the HNSN problem from the introduction. Since the function $f(S) = |N(S)|$ is submodular, HNSN is a special case of the weighted USSS problem. Ling *et al.* further reformulate the problem over the left vertex set L , defining the function $\bar{N} : 2^L \rightarrow \mathbb{R}$ by $\bar{N}(S) = |\{v \in R \mid \delta(v) \subseteq S\}|$ where $\delta(v)$ denotes the set of neighbors of v . Under this formulation, the problem becomes that of maximizing $w(\bar{N}(S))/|S|$ over non-empty subsets $S \subseteq L$. We adopt this viewpoint in our implementation.

Algorithms. We compare against the six baselines of Ling *et al.* [40], and introduce a new flow-based algorithm (detailed in Appendix B). In total, we evaluate **ten algorithms**: IP (SUPERGREEDY++), FW (Frank-Wolfe), MNP (Fujishige-Wolfe Minimum Norm Point Algorithm), FLOW (our new flow-based method), CD (ContractDecompose, [40]), LP (Linear Programming, solved via Gurobi with academic license), GAR (Greedy Approximation, [40]), GR (Greedy, [40]), FGR (Fast Greedy, [40]), and GRR (GreedRatio, [5]). The first four are our implementations; the remaining six follow Ling *et al.*'s code [40].

Approach. For the three iterative algorithms—IP, FW, and MNP—we run each for 100 iterations and plot the following: (1) the time taken per iteration, (2) the best density found up to that point. We report only the final runtime and the density achieved at termination for the remaining algorithms. In the runtime comparison, we include only those algorithms that (1) terminated within the time limit and (2) successfully found the optimal solution.

Our implementation of IP is custom and includes an optimized method for computing marginals. It maintains a heap over a set $S \subseteq L$ of vertices to peel from and a set $S' \subseteq R$ where $\delta(v) \subseteq S$ for all $v \in S'$. Once a vertex $\ell \in S$ is peeled, all its neighbors in R are removed from the data structure S' . This allows each iteration to be implemented in $O(m \log n)$ time, where m and n are the number of edges and vertices in the bipartite graph, respectively.

Datasets. Table 1 summarizes the datasets used for the HNSN problem, following the benchmarks introduced by Ling *et al.* [40]. Each dataset is represented as a weighted bipartite graph with left vertices L , right vertices R , weights on the right vertices $w : R \rightarrow \mathbb{R}_{\geq 0}$, and edges E . The graphs are derived from diverse real-world domains, including recommendation systems (e.g., YooChoose, Kosarak), citation networks (ACM), social networks (NotreDame, IMDB, Digg), and financial transactions (e.g., E-commerce, Liquor). The datasets exhibit a wide range of sizes, with up to $\sim 10^6$ vertices and $\sim 2 \times 10^7$ edges.

Resources: In total, for all algorithms and all datasets, we request 4 CPUs per node and 35G of memory per experiment. All algorithms were given a 30-minute time limit per dataset, after which they were marked as Time Limit Exceeded and given blank values in the plots below.

Discussion of HNSN Results. See Figure 2 for all plots. For each dataset, the top plot shows the final density found by each algorithm (the higher, the better), and the bottom plot shows the time each algorithm takes on a **log-scale** (the lower, the better). We exclude algorithms with a time limit (more than 30 minutes) or if they give a suboptimal answer. Across all datasets, the Frank-Wolfe (FW) algorithm and Fujishige-Wolfe Minimum Norm Point (MNP) algorithm consistently emerged as the fastest to converge. In nearly every case, MNP outperformed FW, achieving the best solution quality in the shortest time, with only a single dataset where FW was slightly faster. Relative to the six baselines from Ling *et al.* [40], these methods achieved speedups ranging from $5\times$ to $595\times$.

Our new flow-based algorithm and SUPERGREEDY++ (IP) typically followed, ranking third and fourth in performance across datasets. Figure 1a shows a particularly striking example, where on the YooChoose dataset, MNP achieved a $595\times$ speedup over the best previously published baseline. In this instance, all algorithms converged to the correct final density, except for GRR, which terminated at a suboptimal solution and was therefore excluded from the runtime comparison.

In particular, the algorithms GRR, GR, GAR, and LP frequently encountered difficulties, either due to slow performance (time limit exceeding) or convergence to incorrect solutions. These results underscore a recurring insight consistently observed across our experiments: universal solvers such as MNP and FW, originally developed for broader optimization tasks, can significantly outperform specialized heuristics when applied to specific problems like HNSN, provided they are used within the proper conceptual framework that we have outlined.

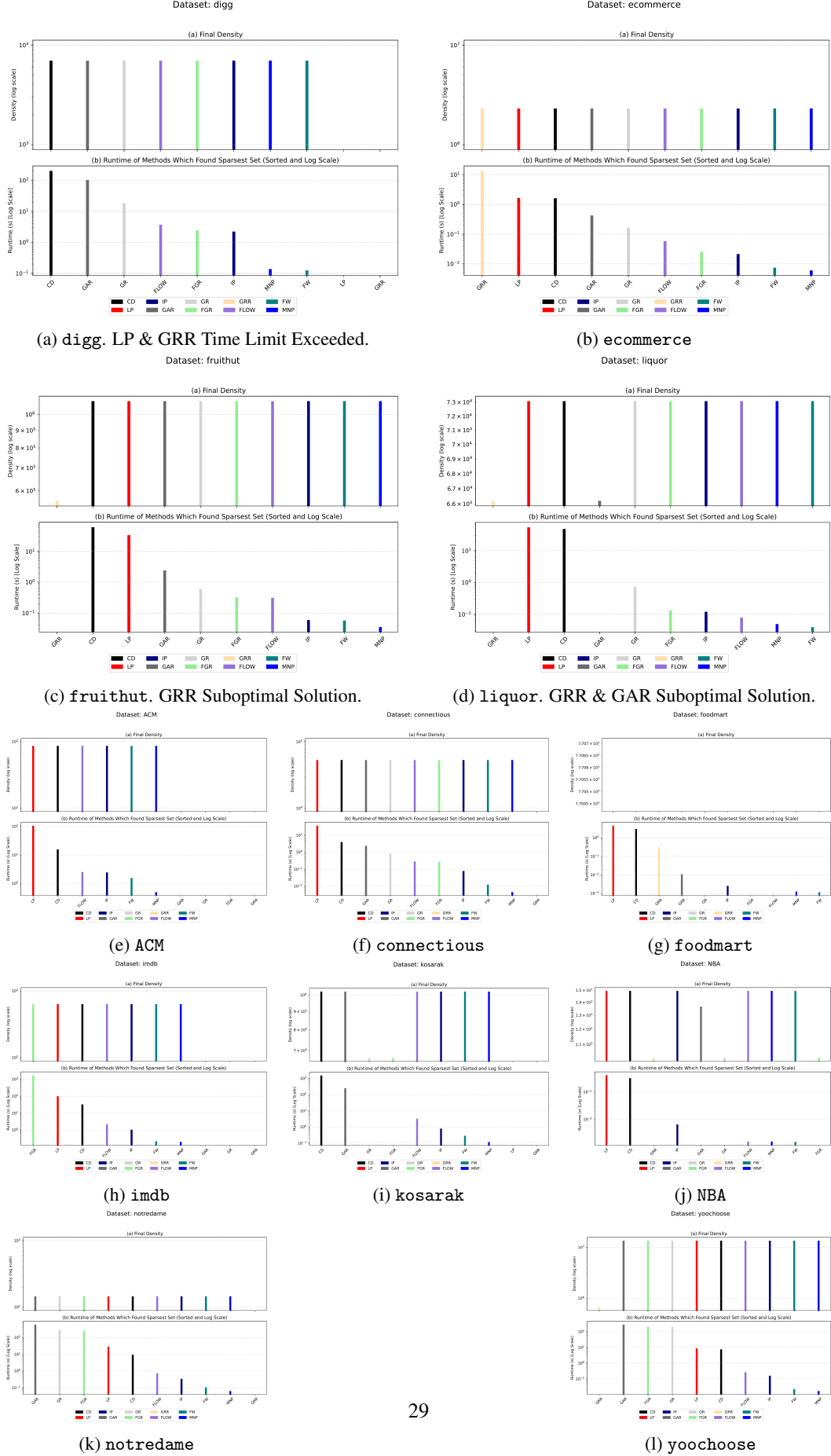


Figure 2: HNSN results across 12 datasets.

1124 D Min s - t Cut Experiments

1125 **Problem Definition.** We next evaluate our algorithms on the classical *Minimum s - t Cut* prob-
 1126 lem, a fundamental task in combinatorial optimization with widespread applications. Specifically,
 1127 the Minimum s - t Cut problem can be framed as minimizing the normalized submodular function
 1128 $g(S) = |\delta(S \cup \{s\})| - |\delta(\{s\})|$ over subsets $S \subseteq V \setminus \{s, t\}$, ensuring that any feasible solution
 1129 corresponds to a valid s - t cut. Within this formulation, the Minimum s - t Cut problem becomes
 1130 an instance of *Submodular Function Minimization (SFM)*, and thus naturally fits within the unified
 1131 framework established in this paper. This connection allows us to apply universal solvers, such as
 1132 SUPERGREEDY++, Frank-Wolfe, and Wolfe’s Minimum Norm Point algorithm, to this classical
 1133 combinatorial problem.

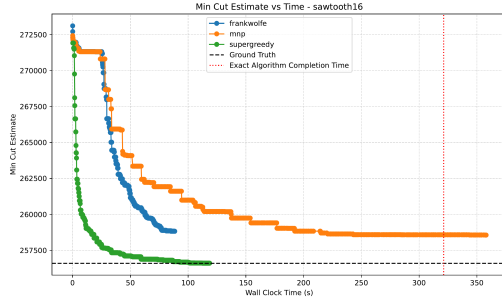
1134 **Algorithms.** We evaluate three iterative algorithms—Frank-Wolfe (FRANKWOLFE), Fujishige-
 1135 Wolfe Minimum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY)—on the Minimum
 1136 s - t Cut problem. For each instance, we also compute the exact minimum cut value using standard
 1137 combinatorial flow-based methods (*Edmonds Karp algorithm*), serving as ground truth. This allows
 1138 us to assess both the solution quality and runtime efficiency of the optimization-based approaches
 1139 relative to the exact combinatorial solution. We run SUPERGREEDY++ for 1000 iterations on each
 1140 dataset, 10,000 iterations for FRANKWOLFE, and 500 for MNP.

1141 **Marginals.** The iterative algorithms leverage the marginal $g(v|S - v) = g(S) - g(S - v)$; the
 1142 difference in the cut value when moving v from the source partition to the sink partition. It follows
 1143 that $g(v|S - v)$ is realized by $\deg^+(v) - \deg^-(v)$ where $\deg^-(v)$ is the (weighted) degree of v to
 1144 all vertices $u \in S \cup \{s\}$ and $\deg^+(v)$ is the (weighted) degree of v to all vertices $w \in V \setminus (S \cup \{s\})$.

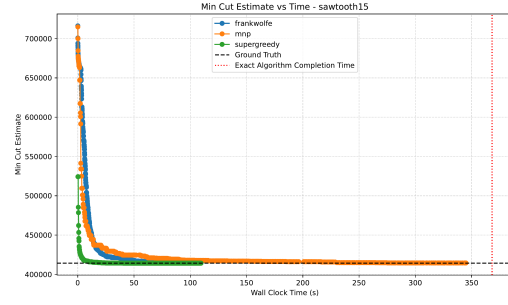
1145 **Datasets.** Our evaluation spans two groups of datasets. First, we consider four classical benchmark
 1146 instances from the first DIMACS Implementation Challenge on minimum cut [19], known for their
 1147 small size but challenging cut structures. Second, to assess scalability and robustness, we include
 1148 a large-scale dataset family, specifically the BVZ-SAWTOOTH instances [36], which consist of 20
 1149 distinct minimum s - t cut problems. Each instance contains approximately 500,000 vertices and
 1150 800,000 edges, providing a test of algorithmic scalability.

1151 **Resources:** In total, for all algorithms and all datasets, we request 4 cpus per node, and 40G of
 1152 memory per experiment. All algorithms were given a 25-minute time limit per dataset (all algorithms
 1153 finished within this time, and there was no time limit exceeded). We mark with a vertical dotted line
 1154 the time for the flow algorithm to find the correct minimum cut.

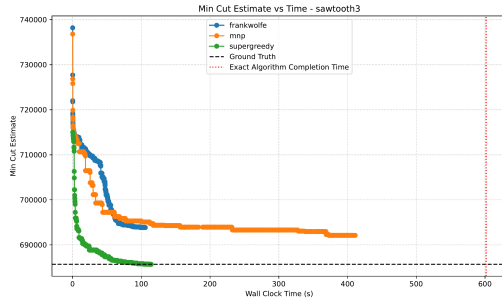
1155 **Discussion of Minimum s - t Cut Results.** Full results are provided in Figure 3 and 4. Across all
 1156 datasets, SUPERGREEDY++ was consistently the fastest to approximate the minimum cut, often by
 1157 large margins over exact flow-based methods, MNP, and FW. In 16 of 24 instances, it found the ex-
 1158 act min-cut rapidly. In the remaining cases, its solution was within $1.000023 \times$ the optimum after at
 1159 most 500 iterations, always converging before the exact flow solver. A notable example is the BVZ-
 1160 SAWTOOTH14 instance, where SUPERGREEDY++ converges orders of magnitude faster (Figure
 1161 1c). This strong performance parallels its success in dense subgraph problems, where it efficiently
 1162 finds high-quality solutions but can slow down near the optimum. Combining SUPERGREEDY++
 1163 with flow-based refinement remains an interesting direction for future work. *Interestingly, while*
 1164 *MNP and FW excel on HNSN, they both underperform here, underscoring how problem structure*
 1165 *affects solver efficiency.*



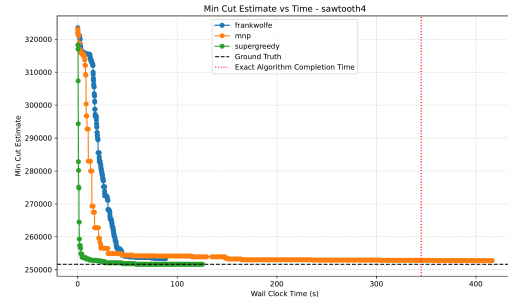
(a) sawtooth16



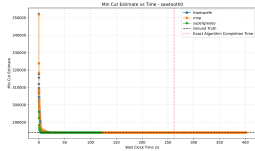
(b) sawtooth15



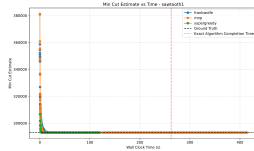
(c) sawtooth3



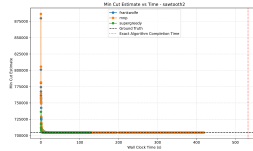
(d) sawtooth4



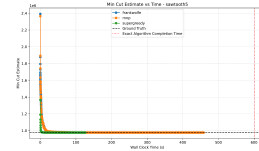
(e) sawtooth0



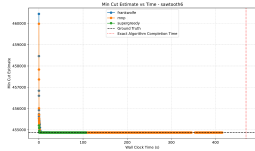
(f) sawtooth1



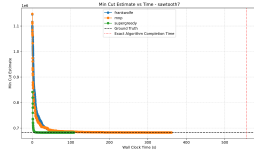
(g) sawtooth2



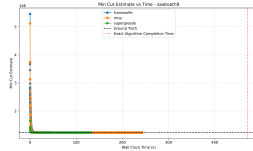
(h) sawtooth5



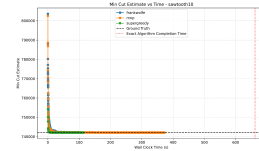
(i) sawtooth6



(j) sawtooth7



(k) sawtooth9



(l) sawtooth10

Figure 3: Min-Cut vs Wall Clock Time (Part 1).

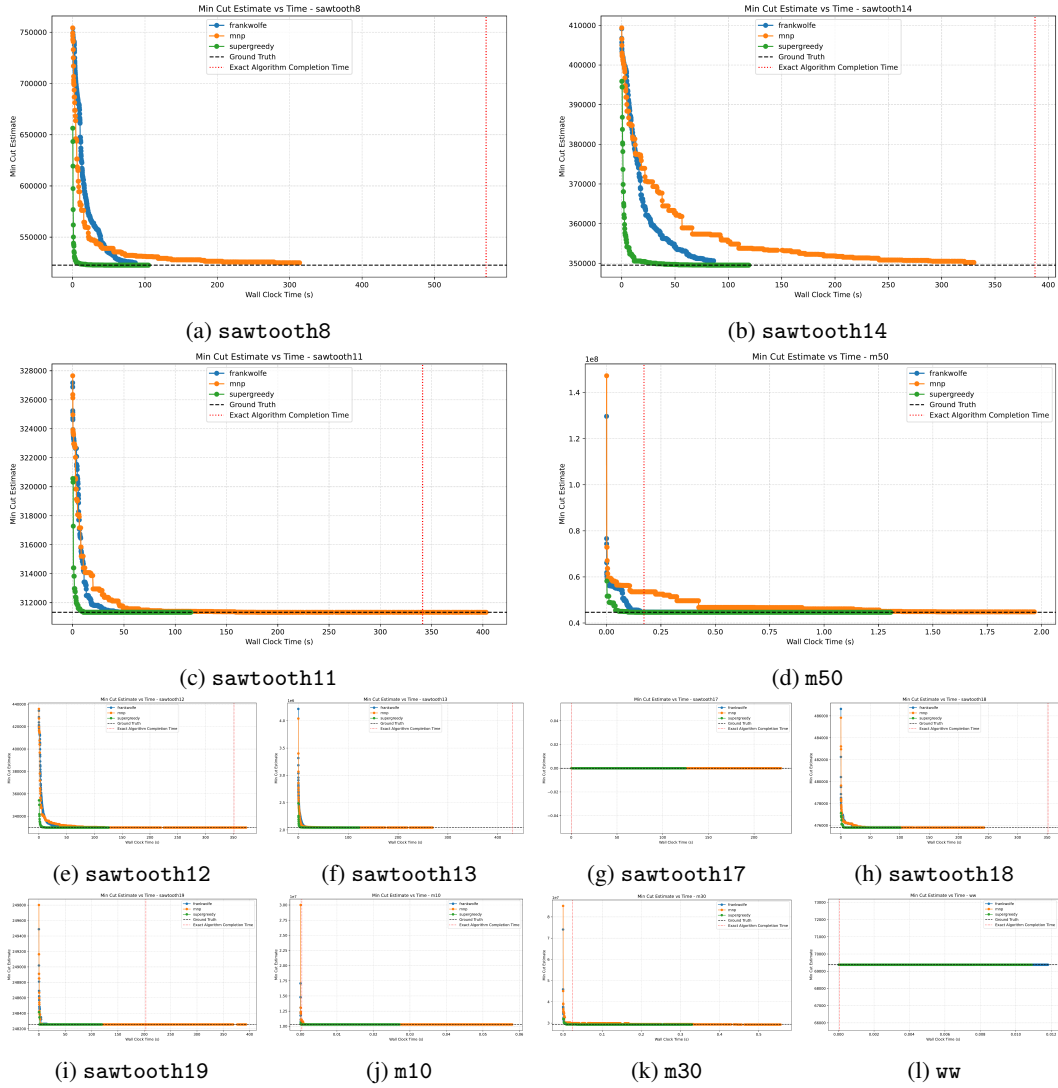


Figure 4: Min-Cut vs Wall Clock Time (Part 2).

E Densest Subgraph Problem Experiments

Problem Definition. Given a graph $G = (V, E)$, find a vertex set $\emptyset \neq S \subseteq V$ maximizing $|E(S)|/|S|$.

Algorithms. We evaluate *seven* algorithms—Frank-Wolfe (FRANKWOLFE, [17]), Fujishige-Wolfe Minimum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY, [8]), FISTA [29], RCDM [44], the incremental algorithm [31], and a new push-relabel flow-based algorithm based on the density improvement mechanism, but using push-relabel flow.

Table 2: Summary of Densest Subgraph Datasets.

Dataset	# Nodes	# Edges
close-cliques	3,230	95,400
com-amazon	334,863	925,872
com-dblp	317,080	1,049,866
com-orkut	3,072,441	117,185,083
disjoint_union_Ka	35,900	16,158,800
roadNet-PA	1,088,092	3,083,796
roadNet-CA	1,965,206	5,533,214

Datasets. Table 2 summarizes the 7 datasets we used for the densest subgraph experiments, two of which are synthetic from [29].

Resources: In total, for all algorithms and all datasets, we request 4 CPUs per node and 40G of memory per experiment. The only exception is for the dataset `com_orkut`, where we requested 100G of memory. All algorithms were given a 30-minute time limit per dataset; otherwise, they were marked as having exceeded the time limit.

Discussion of Densest Subgraph Results. Figure 5 presents the evolution of subgraph density over time for all datasets in the classic densest subgraph problem. Each plot includes the full runtime and a zoomed-in view of the first 20% execution time to highlight early algorithm behavior.

Some literature on densest subgraph algorithms has lacked consistent evaluations, with many works often reaching conflicting conclusions about which algorithms are the most efficient. We aim to contribute a more nuanced and systematic perspective on performance evaluation. We hope future work adopts similar care when assessing and comparing algorithms for the densest subgraph problem.

Memory Comparison: Flow-based algorithms consistently delivered strong performance, particularly when paired with the iterative density-improvement framework of Veldt et al. [32] and Hochbaum [31]. However, a significant practical drawback is their memory usage: in our experiments, flow-based methods often consumed 2–3× more memory than other approaches. This overhead stems from the substantial bookkeeping in constructing and maintaining the flow network.

Moreover, flow-based methods incurred noticeable delays during the initial stages due to the cost of constructing flow networks on large graphs. While Hochbaum employs the Pseudoflow algorithm, we observed comparable performance using standard Push-Relabel solvers within the same density-improvement framework, reaffirming that the primary source of speedup lies in the iterative framework itself rather than the specific max-flow implementation.

That said, flow-based methods are not universally applicable. In problems such as the generalized p -mean Densest Subgraph, no known linear flow formulations minimize objectives like $\lambda|S| - f(S)$ for arbitrary supermodular functions. In such cases, algorithms like SUPERGREEDY++, Frank-Wolfe, and the Fujishige-Wolfe Minimum Norm Point algorithm (MNP) remain the only viable choices, as they operate purely via oracle access and require no specialized structure.

Convergence to a $(1 - \varepsilon)$ approximation: For fast convergence to near-optimal solutions, convex programming-based methods were clear standouts. Across nearly all datasets, RCDM consistently reached $(1 - \varepsilon)$ -approximate dense subgraphs in just a few iterations. It was closely followed by FISTA and SUPERGREEDY++, which also showed rapid convergence and high-quality intermediate solutions.

However, it is essential to note that RCDM and FISTA rely on a projection oracle from [29], which may not be available for generalized objectives (e.g., p -mean Densest Subgraph). In contrast, SU-

1208 PERGREEDY++ maintains its effectiveness without requiring such oracles, making it more broadly
1209 applicable.

1210 On the other hand, Frank-Wolfe and Fujishige-Wolfe’s MNP Algorithm often required thousands of
1211 iterations to achieve a reasonable approximation and were generally outperformed by other meth-
1212 ods. While their theoretical underpinnings are strong, their practical convergence to high-quality
1213 solutions is often prohibitively slow.

1214 **Convergence to the optimal solution:** When the goal is **exact optimality**, flow-based meth-
1215 ods paired with the density-improvement framework outperformed all other algorithms by a wide
1216 margin. Even though convex programming methods like RCDM, FISTA, and SUPERGREEDY++
1217 rapidly approached near-optimal values, they often stalled and required hundreds of iterations to
1218 fully converge to the exact solution *on some datasets*.

1219 In contrast, the flow-based approaches often converged to the optimal solution orders of magnitude
1220 faster, especially on mid-sized and large graphs. Notably, this speed is not a consequence of the spe-
1221 cific flow algorithm used but instead of the density-improvement strategy, which efficiently reduces
1222 the problem to a sequence of min-cut instances.

1223 Thus, flow-based methods remain the most effective choice for applications where exact solutions
1224 are necessary, provided memory usage is manageable and a suitable flow reduction exists.

1225 **General Takeaways:** Our experiments highlight a few key insights:

- 1226 1. Flow-based algorithms remain among the most potent tools for the exact densest subgraph
1227 discovery, especially when integrated with iterative frameworks, but they require significant
1228 memory and do not apply to generalized objectives.
- 1229 2. Convex optimization methods like RCDM and FISTA converge rapidly to high-quality ap-
1230 proximate solutions, making them ideal for problems with projection oracles and when
1231 approximate results suffice.
- 1232 3. SUPERGREEDY++ offers a strong balance of speed, quality, and generality, consistently
1233 performing well across both real and synthetic datasets and requiring no specialized prob-
1234 lem structure.
- 1235 4. Finally, algorithm selection should be guided by the problem setting: whether exactness is
1236 required, whether a flow reduction exists and whether projection oracles are available.

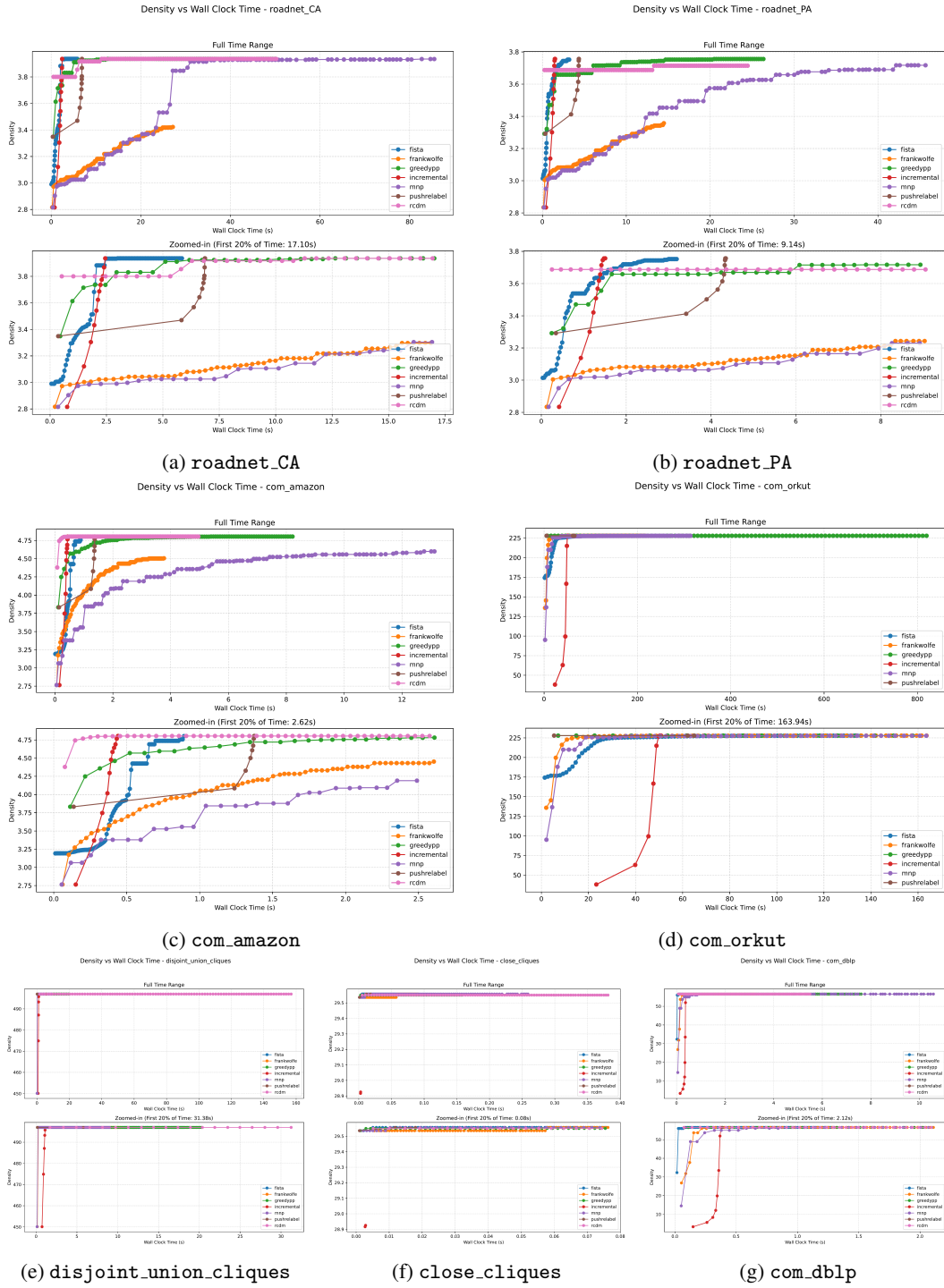


Figure 5: DSG density over time

F Generalized p -mean Densest Subgraph Experiments

Problem Definition. Given a graph $G = (V, E)$, find a vertex set $\emptyset \neq S \subseteq V$ maximizing the generalized density $\left(\frac{1}{|S|} \sum_{v \in S} \deg_S^p(v)\right)^{1/p}$.

Algorithms. We evaluate three algorithms—Frank-Wolfe (FRANKWOLFE), Fujishige-Wolfe Minimum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY). All algorithms are implemented in C++.

Marginals. We adopt the approach from [55], noting that maximizing the objective is equivalent to maximizing $f(S)/|S|$, where $f(S) = \sum_{v \in S} \deg_S^p(v)$ which is supermodular. Peeling of any vertex affects the denominator equivalently; hence, we act greedily with respect to the numerator. The marginal cost of peeling a vertex v from S is then given by

$$f(v|S - v) = f(S) - f(S - v) = \deg_S^p(v) + \sum_{u \in \delta_S(v)} [\deg_S^p(u) - (\deg_S(u) - 1)^p]$$

where $\delta_S(v)$ denotes v 's neighbours in S .

Table 3: Summary of Generalized p -mean Densest Subgraph Datasets.

Dataset	# Nodes	# Edges
close-cliques	3,230	95,400
com-amazon	334,863	925,872
com-dblp	317,080	1,049,866
roadNet-PA	1,088,092	3,083,796
roadNet-CA	1,965,206	5,533,214

Datasets. Table 3 summarizes the 5 datasets we used for the generalized p -mean Densest Subgraph experiments. Note that these are a subset of the datasets we used for DSG. We consider four values for $p \in \{1.1, 1.25, 1.5, 1.75\}$.

Resources. We request 4 CPUs per node and 40G of memory per experiment across all algorithms and datasets. Each trial is given a 30-minute time limit; none exceeded this limit.

Discussion of generalized p -mean Densest Subgraph Results. Results for all datasets are presented in Figures 6 to 10. We plot the best density found against wall-clock time, and, as in previous sections, each plot includes both the full runtime and a zoomed-in view covering the first 20% of execution time. Overall, we find that SUPERGREEDY++ generally achieves the highest density within the allotted time and tends to converge more quickly than the other algorithms.

On the `close_cliques` dataset, however, FW and MNP demonstrate stronger early-stage performance, reaching the maximum density faster than SUPERGREEDY++, though the latter eventually converges to the same value. By contrast, SUPERGREEDY++ often converges within the first few iterations on real-world instances, while the other two algorithms trail behind. Relative performance between FW and MNP varies, with each occasionally outperforming the other.

We also observe that increasing the value of p in the generalized objective improves the performance of both FW and MNP, whereas SUPERGREEDY++ remains largely unaffected. In fact, for all values of p considered, both FW and MNP show improved performance compared to the classical Densest Subgraph case ($p = 1$).

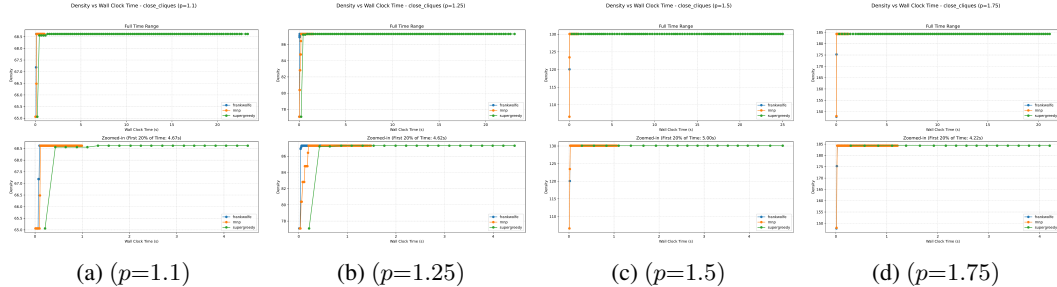


Figure 6: DSS density over time - `close_cliques`

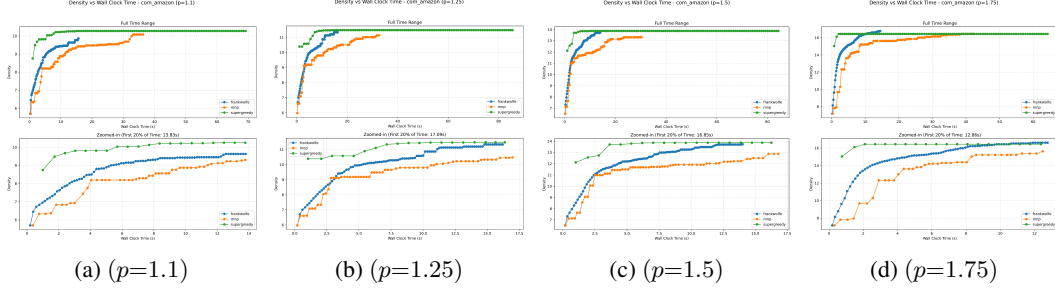


Figure 7: DSS density over time - com.amazon

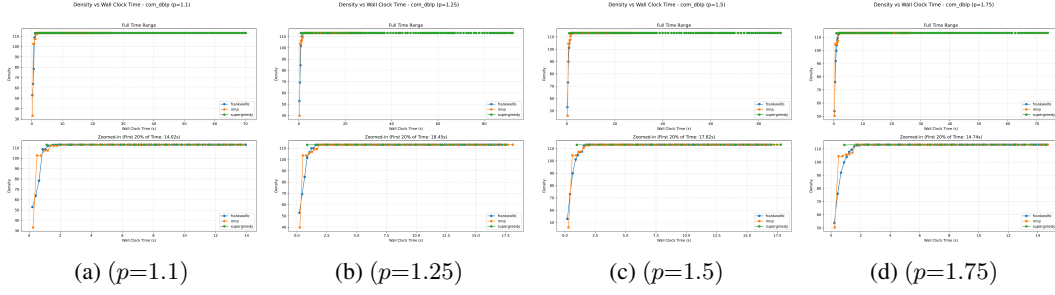


Figure 8: DSS density over time - com.db1p

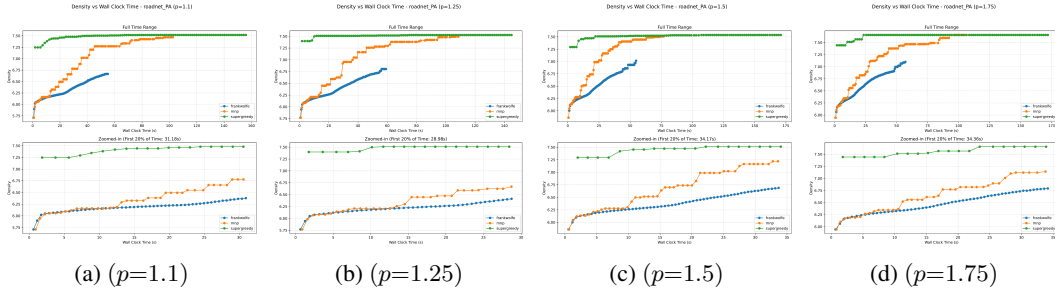


Figure 9: DSS density over time - roadnet_pa

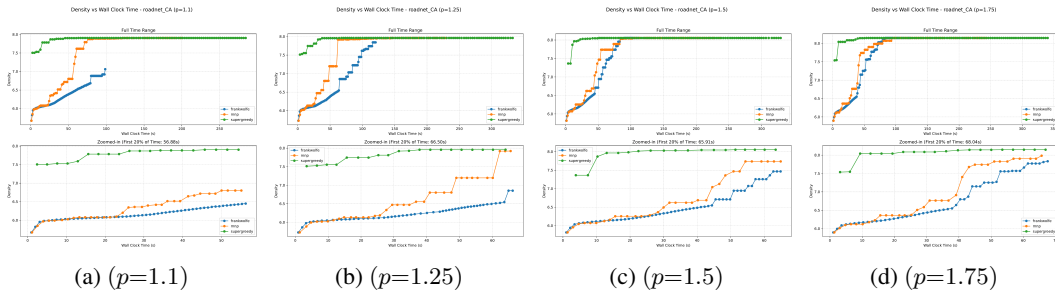


Figure 10: DSS density over time - roadnet_ca

1267 G Anchored Densest Subgraph

1268 **Problem Definition.** Given a graph $G = (V, E)$ and a set $R \subseteq V$, find a vertex set $\emptyset \neq S \subseteq V$
 1269 maximizing $(2|E(S)| - \sum_{v \in S \cap \bar{R}} \deg_G(v)) / |S|$.

1270 **Algorithms.** We evaluate four algorithms—Frank-Wolfe (FRANKWOLFE), Fujishige-Wolfe Mini-
 1271 mum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY), and the flow-based algorithm by
 1272 Huang *et al.* [32]. The authors’ code was implemented in Julia, so we re-implement all algorithms
 1273 in C++.

1274 **Marginals.** Similar to the generalized p -mean DSG, we recognize that maximizing the objective
 1275 can be achieved by acting greedy with respect to the numerator where $|E(S)|$ is supermodular, and
 1276 the degree sum is modular as it’s with respect to G . Hence, the objective is supermodular. The
 1277 marginal cost of peeling v is then

$$f(v|S - v) = 2 \deg_S(v) - \mathbb{I}_{v \notin R}[\deg_G(v)]$$

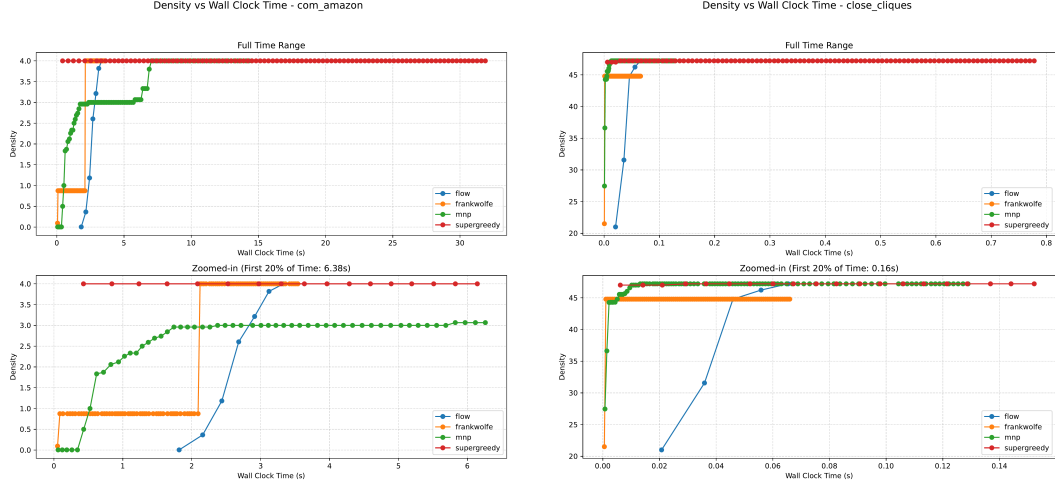
1278 **Datasets.** We reused the base graphs from our generalized p -mean Densest Subgraph Experiments,
 1279 summarized in Table 3. These are the initial graphs G given, for which anchor sets R are created
 1280 via random walks.

1281 **Generating Random R .** As in [32], we generate R with the following process: first, we sample
 1282 10 seed nodes from the set of vertices uniformly at random. Next, using their 2-neighborhood, we
 1283 sample more nodes using a random walk until the total number of nodes in R are 201, and mark that
 1284 set as R .

1285 **Resources:** In total, for all algorithms and all datasets, we request 4 CPUs per node and 40G of
 1286 memory per experiment. All algorithms were given a 10-minute time limit per dataset (all algorithms
 1287 finished within this time, and no time limit was exceeded except on one dataset for the flow-based
 1288 algorithm).

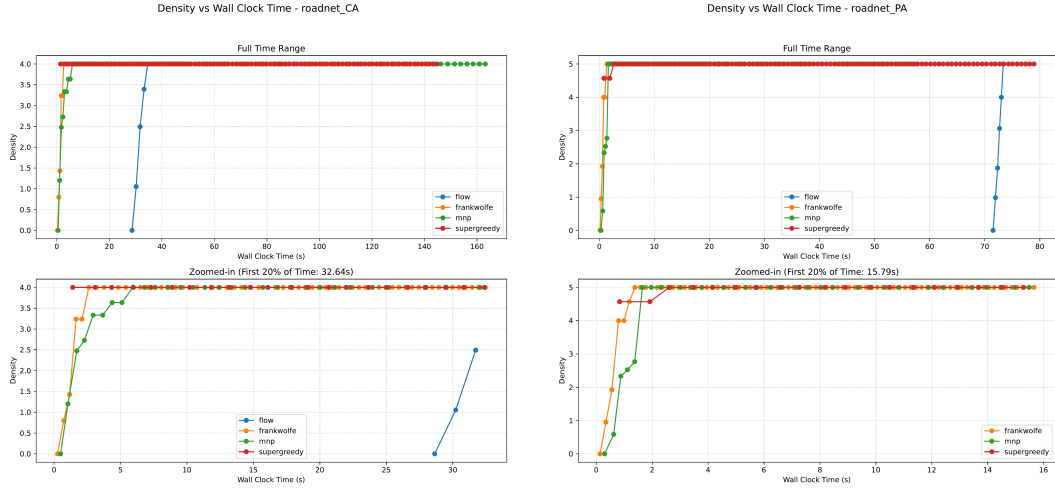
1289 **Discussion of Anchored Densest Subgraph Results.** Figure 11 summarizes the results for an-
 1290 chored densest subgraph. The top plot shows results over the entire runtime, while the bottom plot
 1291 zooms in on the first 20% to highlight early iterations. For each dataset, we plot the density of the
 1292 best-anchored subgraph found against wall-clock time.

1293 Across all experiments, SUPERGREEDY++ consistently delivered the best performance, achieving
 1294 the highest densities in the shortest time. It was followed by FW and MNP, which ranked second
 1295 and third overall, though their relative performance varied across datasets—FW often had the edge
 1296 over MNP, but not uniformly. The flow-based method, FLOW, performed significantly worse on
 1297 most datasets (with the notable exception of `com_amazon`). This underperformance may be attributed
 1298 to high variance in edge capacities within the flow network, which can cause the algorithm to stall
 1299 when attempting to push flow efficiently. In future work, it is worth studying the performance of
 1300 flow algorithms in the weighted case, where such capacity imbalances are more pronounced or may
 1301 offer new opportunities for optimization.



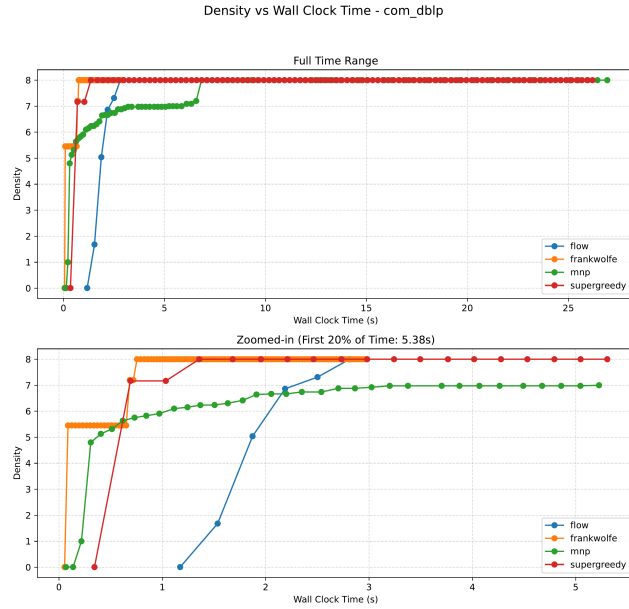
(a) com_amazon

(b) close_cliques



(c) roadnet_CA

(d) roadnet_PA



(e) com_dblp

Figure 11: Anchored density over time

H Contrapolymatroid Membership Experiments

Problem Definition (CM). Given a supermodular function $f : 2^V \rightarrow \mathbb{R}$ and arbitrary vector $y \in \mathbb{R}^{|V|}$, determine if $y \in B(f)$.

Approach. The problem can be reduced to an SFM instance by recognizing that $-f(S)$ is submodular, hence $x(S) - f(S)$ is also submodular. We minimize $y(S) - f(S)$ to answer the membership question. It is more convenient to interpret this as maximization of $f(S) - y(S)$, whose value we plot; y is then a NO instance iff $f(S) - y(S) > 0$.

For our experiments, we consider undirected graphs $G = (V, E)$ with $f(S) = |E(S)|$, $S \subseteq V$.

Generating Query Vectors. Harb et al. [29] (Theorem 4.4) showed that a vector $b \in B(f)$ if and only if there exists a vector $x \in \mathbb{R}^{2|E|}$ such that the pair (x, b) satisfies the dual of Charikar’s densest subgraph LP [15]. Such a vector b corresponds to a YES instance of the Contrapolymatroid Membership (CM) problem. Small perturbations to b can then be used to generate arbitrarily hard NO instances.

Given a graph $G = (V, E)$, we construct a feasible vector $b \in B(f)$ by orienting edges. We begin by computing the exact densest subgraph using PUSHRELABEL. For each vertex $v \in S^*$ —the vertex set of the densest subgraph—we set $b_v = \lambda^*$, where λ^* is the maximum density. Harb et al. [29] (Theorem 4.1) show that such an assignment is always realizable via some orientation of the edges in S^* . For edges not entirely contained in S^* , we assign a value of 0.5 to each endpoint if both lie outside S^* and a value of 1 to the endpoint outside S^* otherwise.

To generate non-trivial NO instances (i.e., where $x(V) = f(V)$), we perturb the vector as follows: select two vertices $u \in S^*$ and $w \notin S^*$, then set $b_u \leftarrow b_u - \varepsilon$ and $b_w \leftarrow b_w + \varepsilon$, where ε is the perturbation magnitude. This results in

$$x(S^*) = |S^*|\lambda^* - \varepsilon = |E(S^*)| - \varepsilon < |E(S^*)| = f(S^*),$$

violating the contrapolymatroid inequality for at least the set S^* .

In our experiments, we reuse the generalized p -mean DSG setup with $p = 1$, using the supermodular objective $f(S) = \sum_{v \in S} \deg_S(v)$. Note that $|E(S)| - y(S) > 0$ if and only if $\sum_{v \in S} \deg_S(v) - 2y(S) > 0$. We maximize and report the latter expression, which is always at least 2ε , with the lower bound attained when $S = S^*$.

Algorithms. We evaluate three algorithms—Frank-Wolfe (FW), Fujishige-Wolfe Minimum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY). All algorithms are implemented in C++.

Datasets. We reuse the datasets summarized in Table 3. We generate four increasingly difficult NO instances for each dataset with perturbation magnitudes $\varepsilon \in \{12, 6, 1, 1e-1\}$.

Resources. In total, for all algorithms and all datasets, we request 4 CPUs per node and 40G of memory per experiment. All algorithms were given a 30-minute time limit per dataset (all algorithms finished within this time, and no time limit was exceeded).

Discussion of Contrapolymatroid Membership Results. Figures 12-16 summarize the results. An algorithm detects a NO instance when its value *climbs* from 0 to a non-zero value. The CM results continue the observed trend: SUPERGREEDY++ performs strongly in general but is outperformed by FW and MNP on CLOSE CLIQUES, where SUPERGREEDY++ fails to identify NO instances until the perturbation is sufficiently large ($\varepsilon \geq 6$), and even then, does so more slowly. For all other instances, SUPERGREEDY++ is usually the first—and frequently the only—algorithm to identify NO instances, often outpacing FW and MNP by a wide margin. SUPERGREEDY++ also tends to find larger values, though this is irrelevant for CM. As expected, performance improves with larger ε .

This strong empirical performance of SUPERGREEDY++ is particularly surprising given that Contrapolymatroid Membership was the original motivation for studying submodular function minimization, and MNP has historically been regarded as the algorithm of choice—making it especially noteworthy that a combinatorial method like SUPERGREEDY++ outperforms it so consistently across all but one dataset.

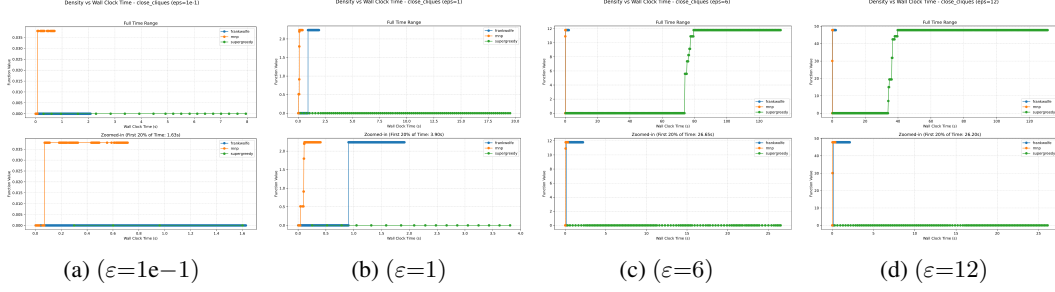


Figure 12: Contrapolymatroid Membership Function Value over time - close_cliques

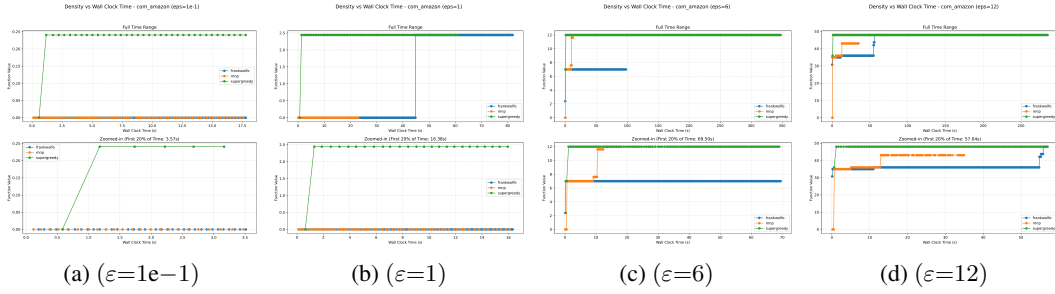


Figure 13: Contrapolymatroid Membership Function Value over time - com_amazon

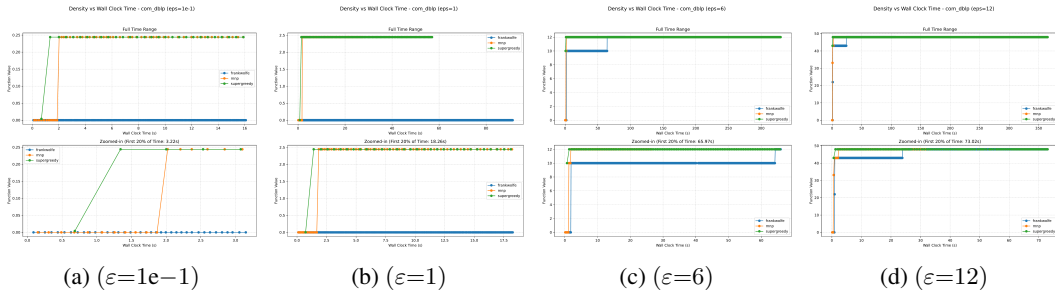


Figure 14: Contrapolymatroid Membership Function Value over time - com_dblp

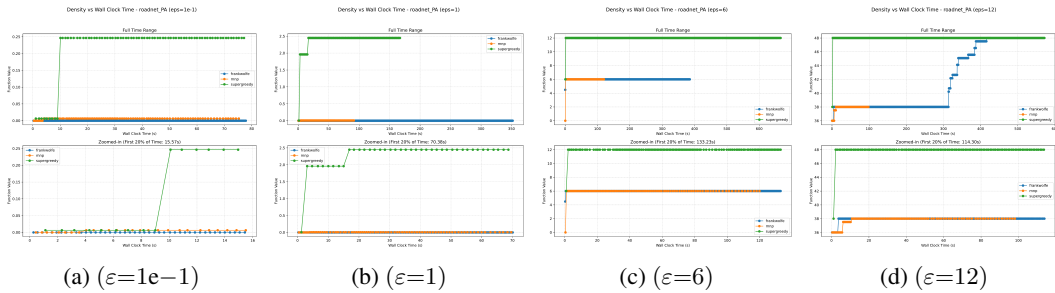


Figure 15: Contrapolymatroid Membership Function Value over time - roadnet_PA

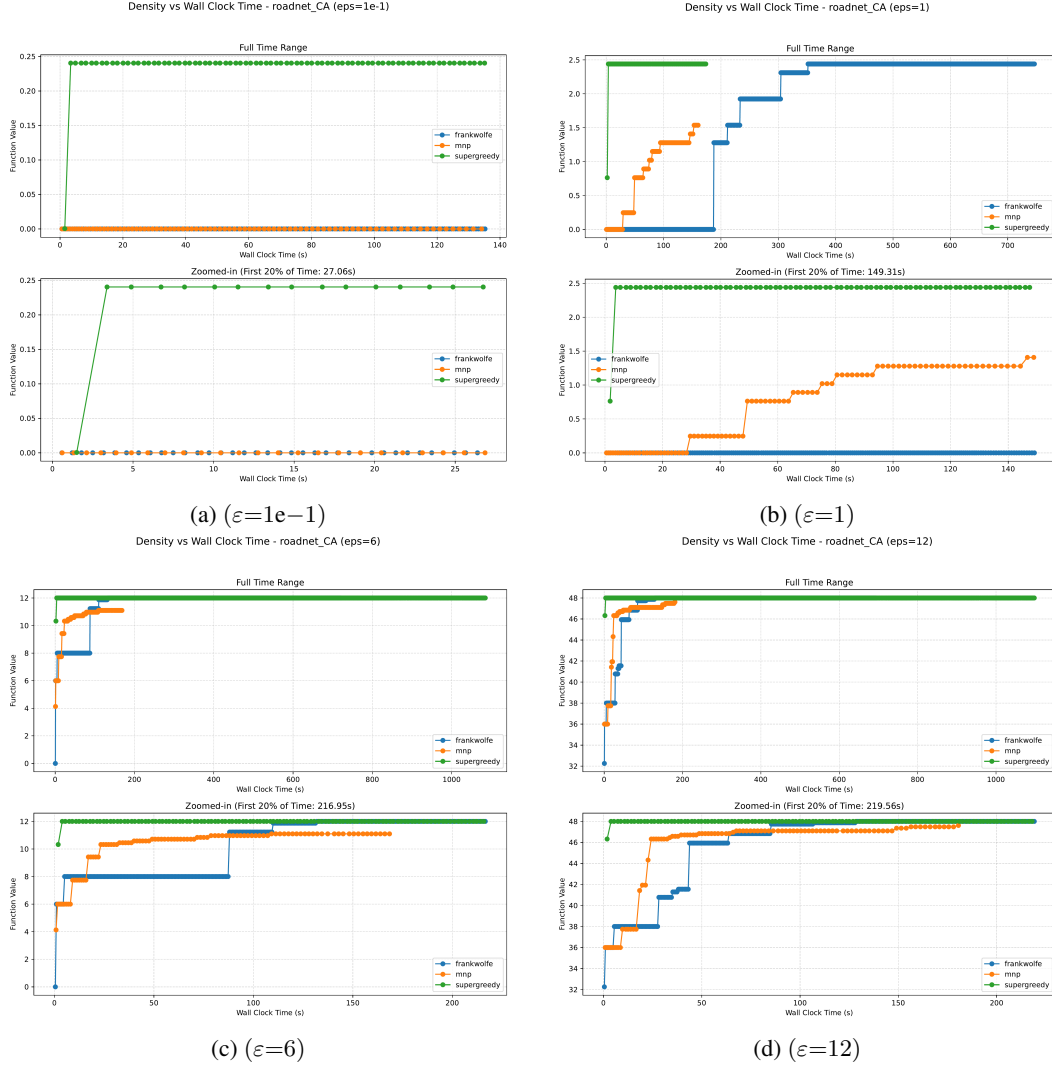


Figure 16: Contrapolymatroid Membership Function Value over time - roadnet_CA

I Minimum Norm Point Problem Experiments

Problem Definition. Let $f : 2^V \rightarrow \mathbb{R}$ be a normalized (super)submodular function, and let $B(f)$ denote its base polytope. Our objective is to find the point in $B(f)$ with the minimum Euclidean norm, i.e., $\arg \min_{x \in B(f)} \|x\|_2^2$.

Algorithms. We evaluate three algorithms: Frank-Wolfe (FRANKWOLFE), Fujishige-Wolfe Minimum Norm Point (MNP), and SUPERGREEDY++ (SUPERGREEDY). All algorithms are implemented in C++.

Datasets. We reuse all instances from prior experiments but focus on the load vector norm rather than the original objective.

Resources. Please refer to the previously cited resources for further information on each problem and its resources.

Discussion of Minimum Norm Point Results. Algorithm performance varies significantly across problems and instances. Overall, SUPERGREEDY++ performs best on DSG, consistently achieving the lowest norm point; the other algorithms behave similarly and sub-optimally. For Anchored DSG, Generalized p -mean DSG, and HNSN, FW and MNP typically outperform SUPERGREEDY++. However, all methods generally converge to similar norm values—except in generalized p -mean DSG, where SUPERGREEDY++ often settles at suboptimal norms despite excelling in the classical DSG case ($p = 1$). A comparable pattern appears in the `close_cliques` instance for Contrapolymatroid Membership, while in other CM instances, SUPERGREEDY++ converges faster and achieves lower norms. Lastly, for Min s - t Cut, SUPERGREEDY++ and MNP perform comparably, with SUPERGREEDY++ slightly outperforming, and both approaches consistently exceeding FW.

I.1 DSG

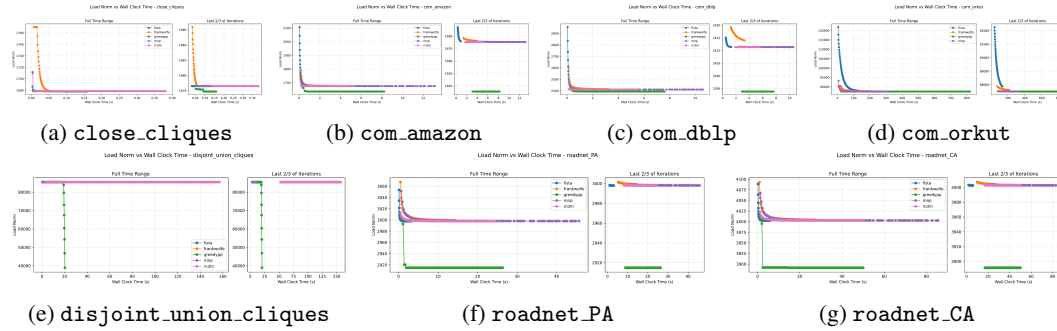


Figure 17: DSG load norm over time

I.2 Anchored DSG

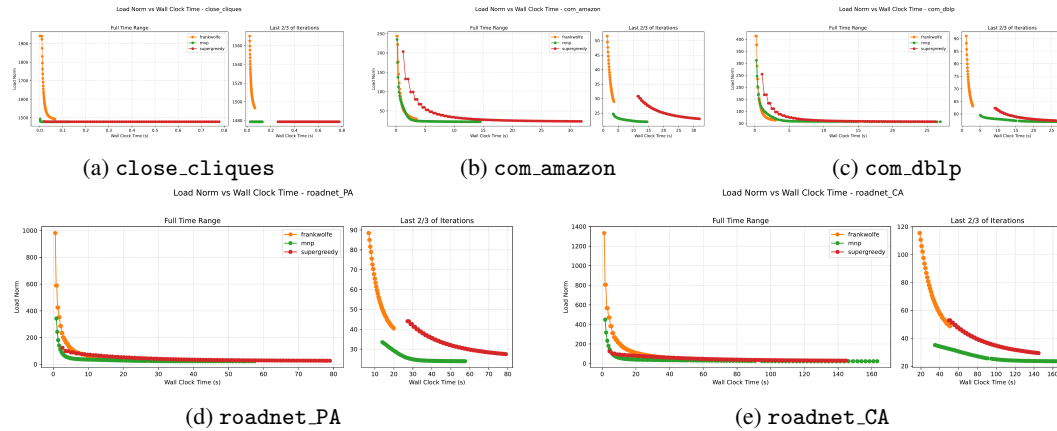


Figure 18: Anchored DSG load norm over time

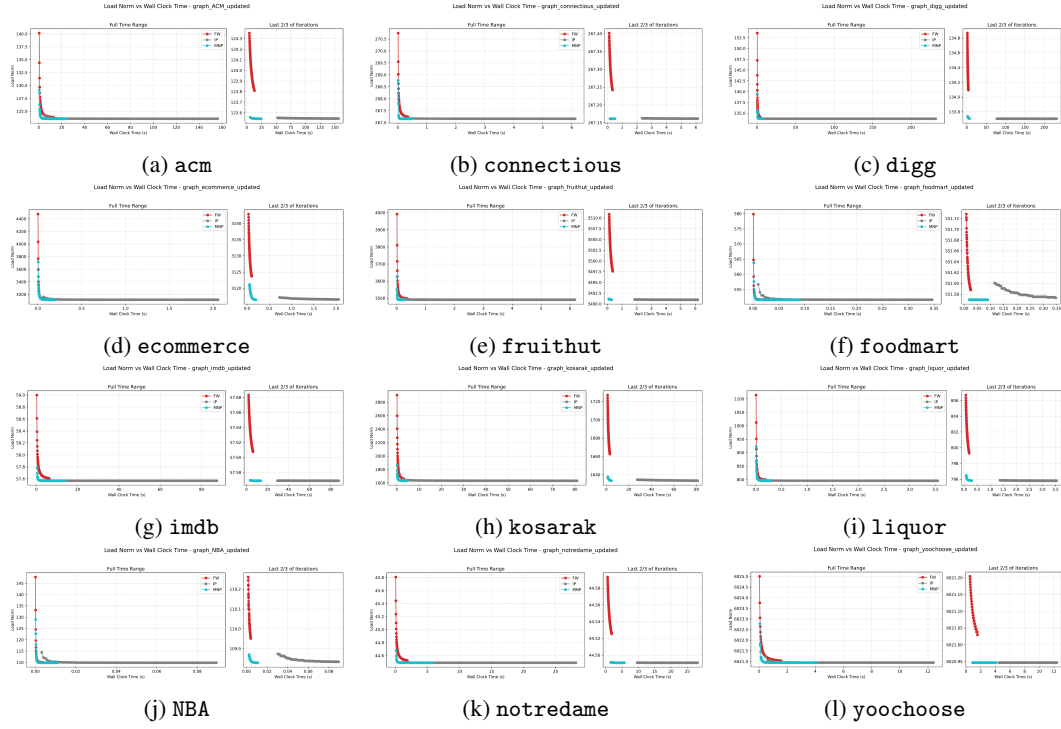
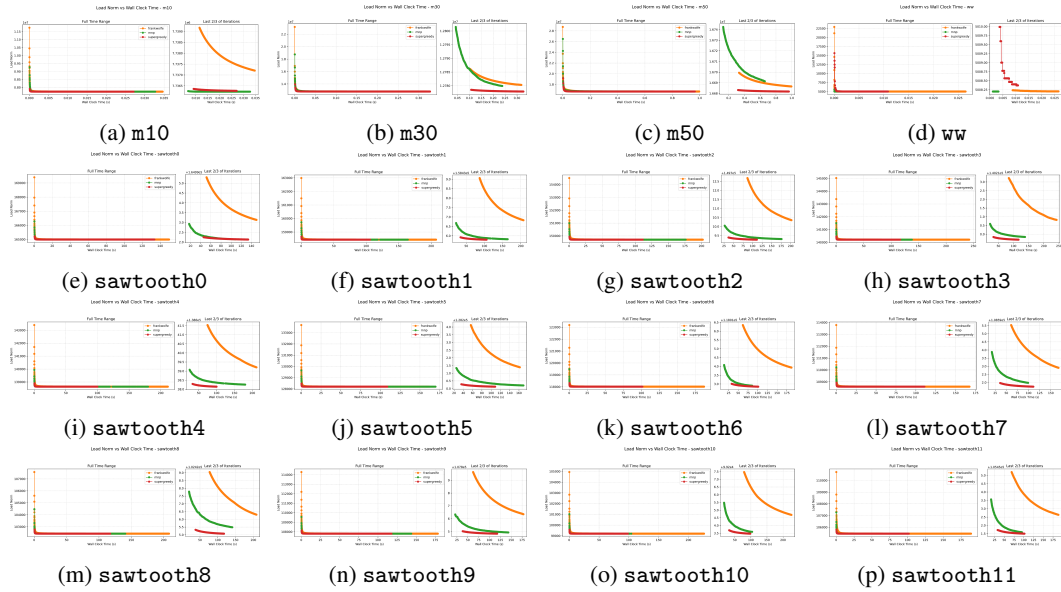


Figure 19: HNSN load norm over time



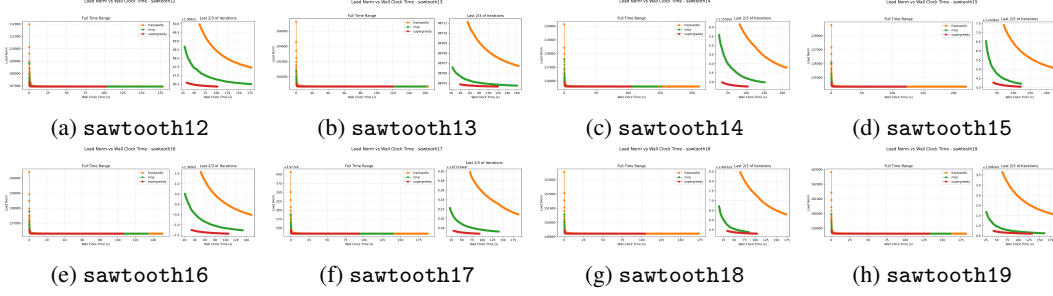


Figure 21: Min s - t Cut load norm over time

1375 I.5 Generalized p -mean DSG

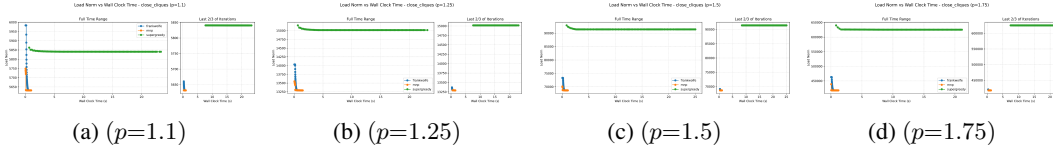


Figure 22: DSS load norm over time - close_cliques

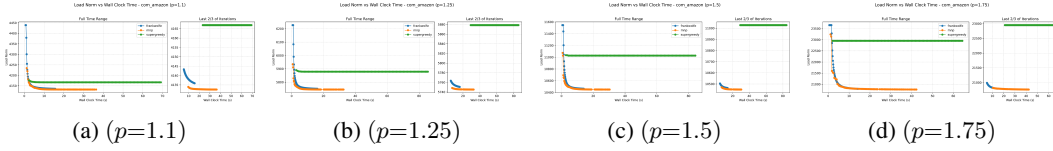


Figure 23: DSS load norm over time - com_amazon

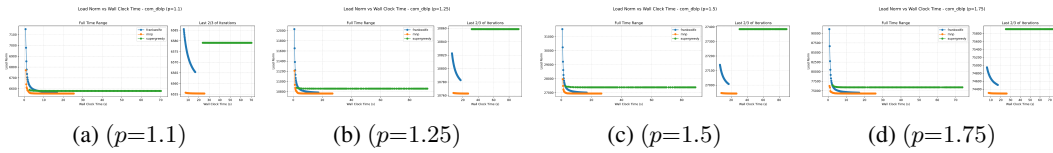


Figure 24: DSS load norm over time - com_db1p

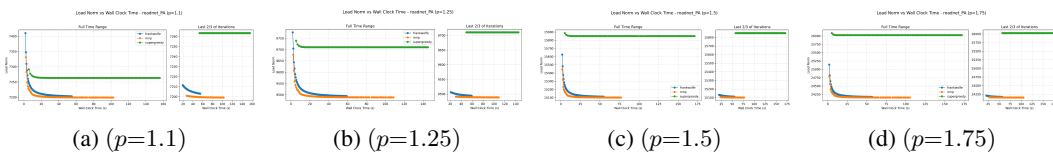


Figure 25: DSS load norm over time - roadnet_PA

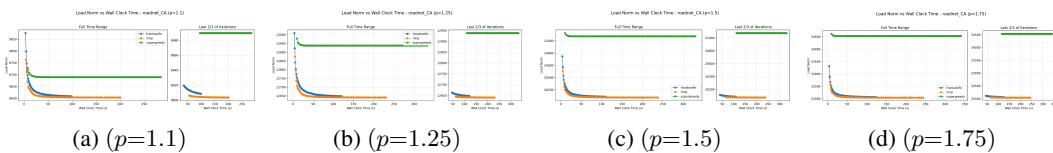


Figure 26: DSS load norm over time - roadnet_CA

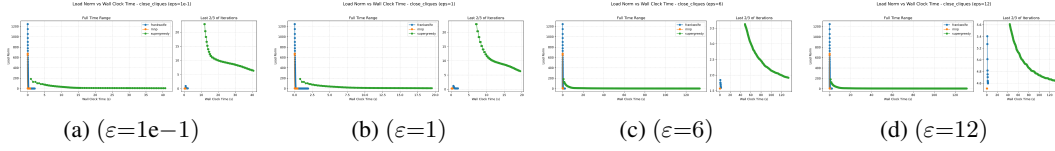


Figure 27: Contrapolymatroid Membership load norm over time - close_cliques

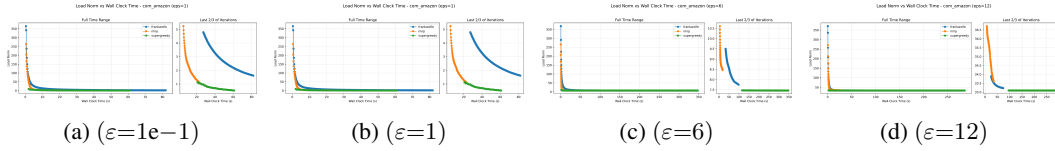


Figure 28: Contrapolymatroid Membership load norm over time - com_amazon

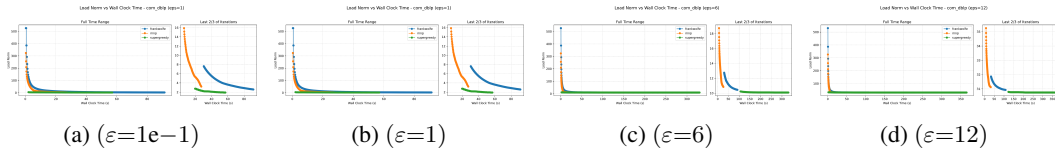


Figure 29: Contrapolymatroid Membership load norm over time - com_dblp

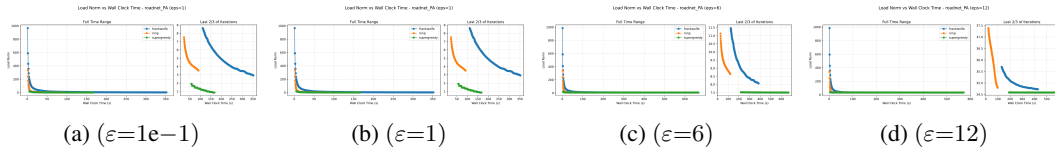


Figure 30: Contrapolymatroid Membership load norm over time - roadnet_PA

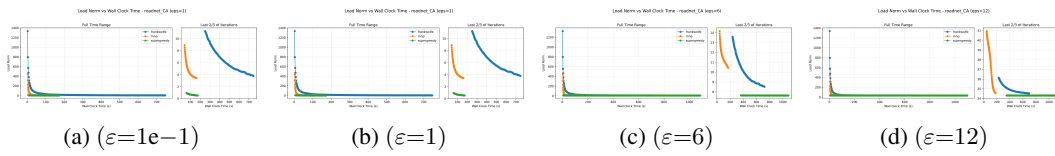


Figure 31: Contrapolymatroid Membership load norm over time - roadnet_CA