Combinatorial Optimization using Comparison Oracles

Vincent Cohen-Addad[*] Tommaso d'Orsi[†] Anupam Gupta[‡] Guru Guruganesh[*]
Euiwoong Lee[§] Renato Paes Leme[*] Debmalya Panigrahi[¶]
Madhusudhan Reddy Pittu[‡] Jon Schneider[*] David P. Woodruff[‖]

Abstract

In a linear combinatorial optimization problem, we are given a family $\mathcal{F} \subseteq 2^U$ of *feasible subsets* of a ground set U of n elements, and our goal is to find $S^* = \arg\min_{S\in\mathcal{F}}\langle w, \mathbb{1}_S\rangle$. Traditionally, we are either given the weight vector up-front, or else we are given a *value oracle* which allows us to evaluate $w(S) := \langle w, \mathbb{1}_S\rangle$ for any $S \in \mathcal{F}$. Motivated by recent practical interest in solving problems via pairwise comparisons, and also by the intrinsic theoretical quest to understand fundamental computational models, we consider the weaker and more robust *comparison oracle*, which for any two feasible sets $S, T \in \mathcal{F}$, reveals only if $w(S)$ is less than/equal to/greater than $w(T)$. We ask:

> When can we find the optimal feasible set $S^* = \arg\min_{S\in\mathcal{F}} w(S)$ using a small number of comparison queries? If so, when can we do this efficiently?

We present three main contributions:

1. Our first main result is a surprisingly general answer to the query complexity. We establish that the query complexity for the above problem over *any arbitrary set system* $\mathcal{F} \subseteq 2^U$ is $\widetilde{O}(n^2)$. This result leverages the inference dimension framework, and demonstrates a fundamental separation between information complexity and computational complexity, as the runtime may still be exponential for NP-hard problems (assuming the Exponential Time Hypothesis).
2. Having resolved the query complexity, we turn to the algorithmic question. Our second result is a novel *Global Subspace Learning (GSL) framework* tailored for objective functions with discrete integer weights bounded by B. We give an algorithm to sort all feasible sets by objective value using only $O(nB\log(nB))$ queries, improving upon the $\widetilde{O}(n^2)$ bound when $B = o(n)$. Moreover, we show that this framework can be efficiently implemented for linear matroids using algebraic techniques, yielding efficient algorithms with improved query complexity for problems such as k-SUM, SUBSET-SUM, and $A{+}B$ sorting.
3. Our third set of results give the first polynomial-time, low-query algorithms for several classic combinatorial problems. We develop such algorithms for finding minimum cuts in simple graphs, minimum weight spanning trees (and matroid bases in general), bipartite matching (and matroid intersection), and shortest s-t paths.

Our work provides the first general query complexity bounds and efficient algorithmic results for this fundamental model, opening new directions for comparison-based optimization, and suggests many directions for future research.

[*]Google Research. Emails: {vcohenad,gurug,renatoppl,jschnei}@google.com
[†]Bocconi University. Email: tommasodorsi@gmail.com
[‡]New York University. Emails: {anupam.g,madhusudhan.p}@nyu.edu
[§]University of Michigan. Email: euiwoong@umich.edu
[¶]Duke University. Email: debmalya@cs.duke.edu
[‖]Carnegie Mellon University. Email: dwoodruf@andrew.cmu.edu

1 Introduction

In a (linear) combinatorial optimization problem, we are given a family $\mathcal{F} \subseteq 2^U$ of *feasible subsets* of a ground set U of n elements, and our goal is to find $S^* = \arg\min_{S \in \mathcal{F}} \langle w, \mathbb{1}_S \rangle$. Traditionally, we are either given the weight vector w explicitly, or else we are given a *value oracle* which allows us to evaluate $w(S) := \langle w, \mathbb{1}_S \rangle$ for any $S \in \mathcal{F}$. In this work, we pose the question:

> Which combinatorial optimization problems can we efficiently solve if we are only allowed *pairwise comparisons between feasible solutions?*

That is, we are given access to a comparison oracle that takes two feasible sets $S, T \in \mathcal{F}$, and outputs whether $w(S)$ is less than/equal to/greater than $w(T)$. This comparison query model is perhaps the weakest feedback model where we can hope to optimize efficiently.[1] As a concrete question, consider the min-cut problem in this model. Specifically:

> We are given an undirected simple graph $G = (V, E)$. In each query, we can compare any two (edge) cuts to each other. *Can we efficiently find a minimum cut in this graph?*

The min-cut problem has been intensively studied in the value-query model in recent years, leading to new insights and algorithms: can we replicate this success in the comparison-query model?

More generally, our high-level motivation for the model comes from recent interest in applications (e.g., from crowdsourcing, recommendation systems, and most recently, RLHF) which solicit feedback from users only via comparison queries. In these settings, users' notions of quality/utility is often difficult to quantify, or noisy, or only partially consistent, and hence they are asked to give *relative preferences*—choosing one alternative over another—rather than assigning precise numerical values. While these practical settings do not seek to solve precise optimization problems (like min-cuts or spanning trees), they do motivate the particularly clean query model we investigate in this work.

1.1 Our Results

1.1.1 Query Complexity

We first focus on the *query complexity* of the problem: how many comparison queries give us enough information to solve the problem, without worrying about computation? Recall that we can only query pairs of feasible solutions, so it is completely unclear whether we can identify the optimal solution using much less than $|\mathcal{F}|$ queries.

For this query complexity question, we show a surprisingly general *positive* result: *we can optimize over all Boolean families with a small number of queries!*

Theorem 1.1 (Boolean Linear Optimization)**.** *For any family $\mathcal{F} \subseteq 2^U$ and unknown weight function $w^* : U \to \mathbb{R}$, we can solve $\arg\min_{S \in \mathcal{F}} \sum_{e \in S} w^*_e$ using $O(n \log^2 n \cdot \log |\mathcal{F}|) = \widetilde{O}(n^2)$ comparison queries, where $n = |U|$.*

We find Theorem 1.1 remarkable: it shows that the number of comparison queries required to find the optimal solution is only $\widetilde{O}(n^2)$, *regardless of the complexity of* the set system $\mathcal{F}$. Indeed, we

[1] One can consider more general objective functions, such as XOS, submodular, subadditive, etc., where the function $w(\cdot)$ is typically exponential-sized, and hence we can only access it via value or demand oracles. Our question—what can you do with just comparison queries—is interesting in these settings as well, but we focus on linear optimization for now.

Problem	Past Work	Our Results
Any Boolean	-	$O(n^2 \log^2 n)$
optimization problem	-	$O(nB \log(nB))$

Table 1: Our general results for query complexity. The second row is when weights are integers in $[-B, B]$.

could consider $\mathcal{F}$ to represent set covers, independent sets, cliques, or other NP-hard problems for which finding the optimal set is believed to be computationally hard—nonetheless, the number of comparisons required is still nearly quadratic! Indeed, while the query complexity of the procedure in Theorem 1.1 is polynomial, its running time could be exponential (under standard complexity-theoretic assumptions). This shows a large gap between the information complexity and the computational complexity in this model.

The key to proving Theorem 1.1 is to approach this query-complexity question from the correct perspective: we adapt the powerful active classification framework of [KLMZ17] to the optimization setting. We show how to use their classification framework to also optimize over arbitrary point sets (and not just Boolean point sets) in $\mathbb{R}^d$, with the number of queries being related to their "conic dimension":[2]

Theorem 1.2 (General Linear Optimization)**.** *There is an algorithm that, for any point set* $\mathcal{P} \subseteq \mathbb{R}^d$ *with conic dimension* k *and unknown weights* $w^* \in \mathbb{R}^d$*, returns the minimizer* $x^* = \arg\min_{x \in \mathcal{P}} \langle w^*, x \rangle$ *using at most* $O(k \log k \log |\mathcal{P}|)$ *comparisons.*

Since the conic dimension of the Boolean hypercube $\{0,1\}^n$ is known to be $O(n \log n)$, this result immediately implies Theorem 1.1. Moreover, it gives bounds on the query complexity when the point sets have bounded representation size, and for approximate optimization when they have bounded norm. We discuss these results in Section 2.

1.1.2 Global Subspace Learning

Unfortunately, the approach of Theorem 1.1 requires explicitly maintaining the set of feasible solutions, which could have size exponential in n; we currently do not know efficient implementations for it, even for simple problems like finding the subset of size k of largest weight (a.k.a. optimizing over rank-k uniform matroids) for large k, which can be easily solved using other approaches.

In our quest for efficient algorithms, we develop another, different low-query-complexity algorithm for the case where the weights in the objective function are integers in the range $[-B, B]$. This algorithm is more intuitive, and in fact, sorts all the feasible sets by their weight (again, in $\text{poly}(|\mathcal{F}|, n)$ time for now):

Theorem 1.3 (Boolean Linear Optimization: Bounded-Weights)**.** *There is an algorithm that, for any family* $\mathcal{F} \subseteq 2^U$ *and unknown integer weight vector satisfying* $|w^*_e| \leq B$*, sorts the feasible sets according to their weight* $w^*(S)$ *(and hence solves the optimization problem) using* $O(nB \log(nB))$ *comparison queries, where* $n = |U|$.

The main insight behind this result is the following: two sets $S, T \in \mathcal{F}$ with the same weight satisfy $\langle w^*, \mathbb{1}_S - \mathbb{1}_T \rangle = 0$, and hence the vector $(\mathbb{1}_S - \mathbb{1}_T)$ gives us information about the subspace

[2] The conic dimension is a slightly weaker variant of the "inference dimension" from [KLMZ17], specialized for linear optimization. The name reflects its reliance on the notion of conic independence (based on conic spans).

Problem	Past Work	Our Results	
		Comparisons	Equality queries
k-SUM	$O(kn\log^2 n)$	$O((n+kB)\log(kB))$	$O(kB(n+kB))$
SUBSET-SUM	$O(n^2\log n)$	$O(nB\log(nB))$	$O(n^2B^2)$
Sorting $P+Q$	$O(n\log^2 n)$	$O((n+B)\log B)$	$O(B(n+B))$
Linear Matroids	-	$O(nB\log(nB))$	-
Min-cuts	-	$\tilde{O}(\|V\|)$	-
Graph reconstruction	-	$O(\|V\|^2, (\|E\|+\|V\|)\log\|V\|)$	-
Matroids	-	$\tilde{O}(n)$	-
Matroid Intersection	-	$O(n^4)$	-

Table 2: Efficient algorithms and query complexities for problems in the comparison decision tree model. The first set of results are for integer values in $[-B, B]$. Our algorithms are efficient, running in $\mathrm{poly}(n, B)$ and $\mathrm{poly}(n)$ time. The past work is from [KLM19].

orthogonal to w^*. Our algorithm maintains a subspace $\mathcal{A}$ spanned by these orthogonal directions $(\mathbb{1}_S - \mathbb{1}_T)$, and uses it to infer the weight class of other sets in $\mathcal{F}$ without making any additional comparisons; the details appear in Section 3. While Theorem 1.3 relies on weights being integral and bounded, it highlights how the structure of linear optimization can be exploited.

This explicit linear-algebraic structure allows us to get efficient algorithms from Theorem 1.3:

Theorem 1.4 (Efficient Optimization over Linear Matroids)**.** *Let $\mathcal{F}$ be the bases of a rank-k linear matroid $\mathcal{M} = (U, \mathcal{I})$, represented by $V \in \mathbb{R}^{k\times n}$ with entry representation size $\mathrm{poly}(n)$. For an unknown integer weight function w^* satisfying $|w^*_e| \le B$, we can sort the bases according to their weight $w^*(S)$, and hence find the minimum-weight basis $\arg\min_{S\in\mathcal{F}} w^*(S)$ in $\mathrm{poly}(n, B)$ time. The number of comparison queries required is $O(nB\log nB)$ and more generally, $O((n+C)\log C)$, where $C = |\{w^*(S) : S \in \mathcal{F}\}|$ is the number of distinct basis weights.*

Moreover, we get new algorithms for the Subset Sum, k-Sum and Sorting $P+Q$ problems studied by [KLM19], with better bounds for the case when $B = o(n)$; see Section 3.3 for details.

1.1.3 Efficient algorithms for Classical Combinatorial Families

While our general results provide broad query bounds, achieving computational efficiency for classic combinatorial problems often requires tailored approaches. We now turn our focus to several such problems, and show that despite the diversity in these optimization tasks, they admit very efficient algorithms using only comparison queries.

Graph Cuts. For the motivating problem of finding min-cuts in (unweighted) simple graphs, we extend the work of [RSW18] from the more informative *value oracle* model and match it in the *comparison oracle* model:

Theorem 1.5 (Minimum cut)**.** *There is a randomized algorithm that computes the exact minimum cut of a simple graph G with high probability using $\widetilde{O}(|V|)$ cut comparison queries and $\widetilde{O}(|V|^2)$ time.*

In addition to finding the minimum cut, we show how to recover the entire edge set of G *deterministically* using cut comparison queries, except for some degenerate cases, when the task is provably

impossible.

Theorem 1.6 (Graph recovery)**.** *A simple unweighted graph $G \notin \{K_2, \bar{K}_2, K_3, \bar{K}_3\}$ can be recovered using $O(\min\{(|E| + |V|) \log |V|, |V|^2\})$ cut comparison queries and in $\widetilde{O}(|V|^2)$ time.*

Finally, we show that cut comparisons suffice to construct sparsifiers for a broad class of graph problems called *heavy subgraph* problems (which include *densest subgraph* and *max-cut*) by combining our edge-sampling techniques with the results of [EHW15]; see Section 4.4 for details.

The min-cut problem illustrates a *qualitative difference between the value oracle and comparison oracle settings*: in the former, we can evaluate the expression $1/2(w(\partial u) + w(\partial v) - w(\partial\{u, v\}))$ to learn whether edge (u, v) exists (and what its weight is, if the graphs are weighted). In contrast, reconstructing a graph is not straightforward with cut comparisons. (E.g., all non-trivial cuts in K_3 have the same value, and the same holds for its complement graph $\bar{K}_3$, so we cannot distinguish them by comparison queries alone. Moreover, *weighted* graphs cannot be recovered at all using comparisons, even up to scaling (see Section C.4.2)! Given this, Theorem 1.6 comes as a surprise.

The main idea behind both results is the following: for a vertex v and set $S \not\ni v$, comparing $w(\partial(S + v))$ and $w(\partial(S))$ tells us whether more than half of v's incident edges go into S. We build on this observation to identify all edges incident to v: for a sequence of nested sets $\emptyset \neq S_0 \subseteq S_1 \subseteq \ldots \subseteq S_{n-1} \neq V \setminus \{v\}$, we ask the above question for each S_i to find a "tipping point"—a query where the sign of $w(\partial(S+v)) - w(\partial(S))$ changes—whereupon we find an edge. We then refine the process to make edge recovery more efficient, and to perform efficient sampling on G and get a randomized algorithm using only $\widetilde{O}(n)$ comparison queries; see Section 4.

Weighted Min-Cut. These techniques do not extend to weighted graphs. E.g., there are many weighted graphs which cannot be identified using just comparison queries. (E.g., adding tiny-weight edges to a graph changes the graph but not the comparisons.) Also, while our techniques can give *non-adaptive* algorithms using $\text{poly}(n)$ comparisons for min-cut/graph recovery in simple graphs, any non-adaptive algorithm *for the weighted case* must have exponential query complexity in the worst case. (See Section C.4 for a discussion of this and other bottlenecks.) That said, we give the following result for the case where the graph has a small number of distinct vertex degrees. (See Section C.3 for a proof.)

Theorem 1.7 (Weighted minimum cut)**.** *For a graph $G = (V, E)$ with unknown integer edge weights in the range $[0, B]$, the minimum cut can be found using $(nB)^{O(r^2 \log B)}$ queries and time, where $r = |\partial_w(u) : u \in V|$ is the number of distinct weighted degrees among all vertices. In the special case of degree-regular graphs, the complexity is $(nB)^{O(\log B)}$.*

Matroid Bases, Matroid Intersections, and Paths. Next, we consider other basic combinatorial objects in graphs: *spanning trees*, (bipartite) *matchings*, and *s-t* paths. Going beyond graphs, we also consider natural extensions of the first two problems to *matroid bases* and *matroid intersections*.

Theorem 1.8 (Matroids, Matchings, and Paths)**.** *The following combinatorial problems can be solved efficiently in the comparison oracle model:*

(i) the minimum-weight basis of a matroid on n elements can be found $O(n \log n)$ comparison queries in $\widetilde{O}(n^2)$ time,

(ii) the minimum-weight set in the intersection of two matroids on an n-element ground set can be found using $O(n^4)$ comparison queries/time, and

(iii) the minimum-length s-t path in a graph (or a negative cycle, if one exists) can be found using using $O(n^3)$ walk comparisons in $O(n^3)$ time.

Note that knowing the order of element weights is enough to optimize over matroids (using the greedy algorithm), but how can we compare element weights? We observe that if two elements share a circuit, then we can find two bases which differ on exactly these elements, and hence compare them. We then show that one can decompose any matroid so that we only need to compare such pairs of elements. (See Section D.1.)

For matroid intersection (and its special case, bipartite matchings), sorting the element weights is neither doable nor sufficient. Instead, we implement the shortest augmenting path algorithm (and its natural extension for matroid intersection) using only comparisons between matchings (i.e., common independent sets); see Section D.2. Finally, we show that the Bellman-Ford algorithm for shortest s-t paths can be implemented using comparisons between s-t walks to prove (iii).

Organization. In Section 2, we present our first result on query complexity for Boolean Linear Optimization. Next, Section 3 introduces our *Global Subspace Learning* framework, and gives efficient algorithms for specific classes of set systems. We then present efficient algorithms for other classic combinatorial problems: Section 4 is dedicated to min-cuts, while Section D covers minimum-weight matroid bases, matroid intersections, and shortest s-t paths. To improve flow, many technical proofs and discussions are deferred to the appendix.

1.2 Related Work

While our theoretical study of comparison oracles for linear discrete optimization problems is new, to the best of our knowledge, comparison oracles have been studied for submodular functions and beyond: [BVW16] show how to use few exact comparison queries to any submodular function to construct an approximate comparison oracle. [HGvL17] studied a comparison model for metric spaces, where an oracle compares distances of nodes to a fixed location i, and gave small-query algorithms under some expansion conditions; later work studied a similar access model for classification/regression in Euclidean space [HGL18]. [XZM+17] studied binary classification under noisy labeling and access to a pairwise comparison oracle. [KLMZ17] studied active classification using label and pairwise comparison queries between the data points to recover a (linear) classification boundary. This last result plays an central role in our work on general linear optimization. The techniques of [KLM19] can be used to derandomize the query strategy for general linear optimization.

Minimum Cuts: The study of value (and the more powerful demand) queries has long been used for general submodular functions, given their natural representation is exponential sized. The graph min-cut problem can be viewed both as submodular optimization on vertex sets, or linear optimization over the feasible edge sets. Min-cuts have been intensively studied in the value query model in recent years, first by [RSW18] who gave an algorithm using $\widetilde{O}(n)$ queries in simple graphs. This was extended by [MN20, AEG+22] to the case of weighted graphs. These algorithms are randomized; the best deterministic bound is $O(n^2/\log n)$ [GK00] which allows full reconstruction of the input graph. There are also lower bounds: [GPRW20] define the *cut dimension* and use it to give a lower bound of $3n/2 - 2$ queries for weighted graphs; [LLSZ21] improve the lower bound to $2n - 2$, which remains the best deterministic lower bound for this problem. The best randomized lower bound for the problem is $\Omega(n \log\log n/\log n)$ queries [RS95, AD21].

There is earlier work on graph reconstruction using cut-value queries: see, e.g., [GK00, BM11, Cho13]. Recently, there is work on solving other optimization problems (e.g., minimal spanning

forests) using cut value queries; see, e.g., [Har08, AL21, LC24]. While we do not consider these kinds of "improper" optimization problems in this work. Further afield are more general query models, which allow for "independent set" queries, see, e.g., [LC24] for references.

Continuous Optimization: There is a large body of work in *continuous optimization* using comparisons between points, where the algorithm can query points $x, y \in \mathbb{R}^d$ and get back the sign of $f(x) - f(y)$ [JNR12, CSC+20, ZL24] (also see the work on *dueling bandits* [YBKJ12]). However, in our model we are allowed to only query discrete, feasible solutions and not points in the ambient space outside $\mathcal{F}$, so the ideas used in gradient-based methods seem inapplicable. E.g., one can try writing each interior point as a convex combination of extreme points, but comparisons between these extreme points do not give us comparisons between the fractional points. Other approaches, e.g., estimating gradients or learning cutting-planes, also seem difficult to implement. For the special case of min-cuts, the Lovász extension initially seems promising: we can learn the signs of the gradient coordinates, but this information is insufficient. Moreover, the Lovász extension is not smooth; one can try Gaussian smoothing, but again that requires evaluation at points outside of our feasible set.

Comparison Models more broadly: Ordinal preferences are considered in economics [Par19], recommendation systems and crowd-sourcing [CBCTH13], modeling consumer behavior [Bar03], statistics [BT52], and the social sciences [HH10]. More recently, these have become important to reinforcement learning with human feedback (RLHF), and particularly direct preference optimization methods [CCYL25, TRC24].

2 Linear Optimization for General Set Systems

We begin our investigation with the question of query complexity for linear optimization in a very general setting: in the *general linear optimization* (GLO) problem, we are given a set of N points $\mathcal{P} = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^d$, and an unknown weight vector $w^* \in \mathbb{R}^d$. We are only allowed comparison queries: given $x, y \in \mathcal{P}$, the comparison oracle responds with $\operatorname{sign}(\langle w^*, x - y\rangle)$, i.e., the relative ordering of x and y in the ordering on $\mathcal{P}$ induced by w^*. The goal is to identify $x^* := \arg\min_{x\in\mathcal{P}}\langle w^*, x\rangle$ using as few queries as possible. The combinatorial optimization setting is obtained when we restrict to some point set in $\{0,1\}^d$.

Our main result for general linear optimization depends on the complexity of these point sets, via the notion of the *conic dimension*, which we define next.

2.1 Envelopes and the conic dimension of Point Sets

We begin by defining the *envelope* and *conic dimension* of a point set. Recall that a cone of a point set is the collection of all positive linear combinations of these points. The idea behind the envelope is simple: if σ is the total order induced by w^* on the points $\{y_1, \ldots, y_n\}$ (i.e., if $\langle w^*, y_i - y_j\rangle \geq 0$ for $i < j$, then taking the envelope of σ gives us all the "implications" of this ordering, and allows us to derive implications without performing any further calculations.

Definition 2.1. For a sequence of points $\sigma = (y_1, \ldots, y_k) \in (\mathbb{R}^d)^k$:

1. The *envelope* of σ is
$$\operatorname{env}(\sigma) := \operatorname{cone}(\{y_j - y_i\}_{1\leq i<j\leq k}) = \operatorname{cone}(\{y_{i+1} - y_i\}_{1\leq i<k}).$$
2. The points are *conically independent* if, for each $t \in \{2, \ldots, k\}$,
$$y_t - y_1 \notin \operatorname{env}(\sigma^{1:t-1}),$$

where $\sigma^{1:t}$ denotes the prefix consisting of the first t elements of σ.

Definition 2.2. The *conic dimension* of a set $\mathcal{Y} \subseteq \mathbb{R}^d$, denoted $\text{ConicDim}(\mathcal{Y})$, is the largest integer k for which there exists a length-k conically independent sequence $\sigma = (y_1, \ldots, y_k) \in \mathcal{Y}^k$.

In other words, the conic dimension is the largest number of comparisons $\langle w^*, y_t \rangle$ vs. $\langle w^*, y_1 \rangle$ such that each one cannot be inferred from the complete sorted ordering of the preceding points.

2.2 An Algorithm for Finding a Minimizer

Given these definitions, it is possible to show that if a point set $\mathcal{P}$ has conic dimension k, then for the optimal point $y^* \in \mathcal{P}$ there exists proofs/certificates of optimality of size at most $k-1$ (see Section A.1). Since we are interested in the query complexity and not the certificate complexity of optimization, we show the following:

Theorem 1.2 (General Linear Optimization)**.** *There is an algorithm that, for any point set $\mathcal{P} \subseteq \mathbb{R}^d$ with conic dimension k and unknown weights $w^* \in \mathbb{R}^d$, returns the minimizer $x^* = \arg\min_{x \in \mathcal{P}} \langle w^*, x \rangle$ using at most $O(k \log k \log |\mathcal{P}|)$ comparisons.*

The proof of this remarkable result relies on adapting the powerful active classification framework of [KLMZ17] to the optimization setting; we believe that this connection between the active learning/classification and optimization settings should be interesting more generally. The algorithm is also surprisingly simple:

Algorithm 1: Iterative Sieving Algorithm

1.1 **while** $|\mathcal{P}|$ *is more than* $O(k)$ **do**
1.2 Independently include each point of $\mathcal{P}$ in a subset $\mathcal{Y}$ with probability $2k/N$
1.3 $\sigma \leftarrow$ result of sorting $\mathcal{Y}$ using $O(k \log k)$ expected comparisons
1.4 $y \leftarrow$ first (smallest) element of σ
1.5 Eliminate all points $x \in \mathcal{P} \setminus \{y\}$ such that $x - y \in \text{env}(\sigma)$
1.6 Sort $\mathcal{P}$ using a brute-force algorithm.

The proof of the algorithm uses from the next lemma, which shows that each iteration reduces the number of points in $\mathcal{P}$ by half in expectation. We defer the proof to Section A.1.2.

Lemma 2.3 (Sieving Lemma)**.** *The expected number of points that are eliminated from $\mathcal{P}$ at step 1.4 is at least $|\mathcal{P}|/2$.*

Since Boolean point sets have small conic dimension [KLMZ17] (see Section A.2 for a proof), we get our main query complexity theorem for Boolean optimization:

Theorem 1.1 (Boolean Linear Optimization)**.** *For any family $\mathcal{F} \subseteq 2^U$ and unknown weight function $w^* : U \to \mathbb{R}$, we can solve $\arg\min_{S \in \mathcal{F}} \sum_{e \in S} w^*_e$ using $O(n \log^2 n \cdot \log |\mathcal{F}|) = \widetilde{O}(n^2)$ comparison queries, where $n = |U|$.*

As mentioned in Section 1.1, this result is very surprising to us: it shows that a small number of very structured comparison queries suffice to find the optimal solution, even for NP-hard optimization problems for which the computational complexity is exponential (assuming the ETH).

Can we hope to implement Algorithm 1 efficiently for "nice" Boolean problems? E.g., can we do this for problems which admit an optimization oracle, and also a sampling oracle, so that we can

implement step 1.1 efficiently? Matroids are one such class of problems to keep in mind. Note that the above algorithm restricts to some subset of the points in step 1.4; it remains unclear how to efficiently sample from these restrictions. To sidestep this difficulty, we give a different algorithm in the next section, and show that which we can indeed implement for several "nice" problems.

3 Global Subspace Learning

We now develop a different algorithm tailored to cases where the weight vector w^* in the objective function uses integers bounded by B, i.e., $w^* \in \mathbb{Z}^n$ and $\|w^*\|_\infty \leq B$. The resulting algorithm has a cleaner structure which we can use to obtain efficient algorithms for several natural set systems $\mathcal{F}$.

Recall that the algorithm from the previous section was based on maintaining a collection of potentially optimal solutions, sampling from them and sorting the samples, and using them to filter out the dominated solutions. The main conceptual hurdle is to get a handle on the structure of the solutions still in play. Our new *Global Subspace Learning (GSL)* algorithm is more intuitive and demonstrates the idea of inference more naturally. It achieves more than the previous algorithm: it actually sorts all feasible sets by their weight, using only $\text{poly}(nB)$ comparisons—a task which is feasible in the bounded weights case, because there are at most $O(nB)$ possible distinct weight classes.

3.1 The Algorithm

As in the previous section, we work with a set $\mathcal{P}$ of points, which are indicator vectors $\mathbb{1}_S$ of the feasible sets in $S \in \mathcal{F}$. For brevity, we will use $x, y \in \mathcal{P}$ to refer both to points in $\mathbb{R}^n$, and also the corresponding sets in $\mathcal{F}$. The weight of a point x is $w^*(x) := \langle w^*, x \rangle$.

The technical core of the algorithm is the following observation: if two points $x, y \in \mathcal{P}$ have the same weight, they imply the linear equality $\langle w^*, x - y \rangle = 0$. Concretely, the *Global Subspace Learning* algorithm does the following:

3.1.1 Global Subspace Learning Algorithm

1. **Buckets:** At the end of step t, the algorithm has considered t points $y_1, \ldots, y_t$ from $\mathcal{P}$ which are divided into some number of buckets $B_1, B_2, \ldots, B_\ell$. These buckets are indexed in increasing order of weight, and each bucket contains points of the same weight. For each bucket B_i, we have chosen an arbitrary *representative* r_i, say the first point in that bucket.

   Moreover, the algorithm also maintains a subspace

   $$\mathcal{A} = \text{span}\left(\cup_{i \in [\ell]} \{x - y \mid x, y \in B_i\}\right)$$

   that is spanned by directions $x - y$ known to be orthogonal to w^*. By construction, if a pair $x', y' \in \mathcal{F}$ satisfies $x' - y' \in \mathcal{A}$, we can infer that $\langle w^*, x' - y' \rangle = 0$ and hence $w^*(x') = w^*(y')$. Hence, each bucket B_i has a corresponding affine subspace $\mathcal{A}_i = \{r_i + v \mid v \in \mathcal{A}\}$ defined using the representative $r_i \in B_i$, and $\mathcal{P} \cap \mathcal{A}_i$ is precisely the set of points which can be inferred to have the same weight as the points in B_i at this time.

   To begin, at time $t = 1$, we choose y_1 as an arbitrary point in $\mathcal{P}$, have a single bucket $B_1 = \{y_1\}$ and $r_1 = y_1$, and the subspace $\mathcal{A}$ is the trivial subspace of dimension zero.

2. **Separation Step:** At step $t+1$, the algorithm selects a point $y_{t+1} \in \mathcal{P}$ that does not belong to $\bigcup_{i \in [\ell]} \mathcal{A}_i$ if such a point exists, else it terminates. In other words, it selects a point whose weight cannot be inferred to equal that of any representative r_i, based on the knowledge

accumulated so far. (This is the central *computational* challenge of the algorithm, which will return to in the next section.)

3. **Update Step:** It uses binary search to locate the position of the weight $w^*(y_{t+1})$ with respect to the ordered sequence $w^*(r_1) < \cdots < w^*(r_\ell)$.

   If $w^*(y_{t+1}) = w^*(r_i)$ for some $i \in [\ell]$, then it inserts y_{t+1} into B_i, and updates $\mathcal{A}$ to include the new direction $(y_{t+1} - r_i)$ in its span, thereby increasing its dimension. This also changes the definitions of all the affine subspaces $\mathcal{A}_j$. Else the weight $w^*(y_{t+1})$ is a weight it has not seen before, so the algorithm starts a fresh singleton bucket containing y_{t+1} as its representative.

Lemma 3.1. *The number of iterations of the Global Subspace Learning algorithm is at most* $2nB+n$.

Proof. The potential $\phi = \dim(\mathcal{A}) + \ell$, starts at zero and is upper-bounded by $(n-1)+(2nB+1) = 2nB+n$. It is sufficient to prove that ϕ increases by exactly 1 each time a new point is encountered. We have two cases when a new point y_{t+1} is considered:

1. If $w^*(y_{t+1}) = w^*(r_i)$ for some $i \in [\ell]$, the number of buckets ℓ remains unchanged, as y_{t+1} is inserted into B_i. This vector is guaranteed to be linearly independent of $\mathcal{A}$ thus increasing the dimension because $y_{t+1} - r_i \notin \mathcal{A}$, which follows directly from the condition $y_{t+1} \notin \mathcal{A}_i$.

2. Else, a new weight value is discovered. A new bucket is opened, which increases ℓ by one, while $\mathcal{A}$ remains unchanged.

In both cases, the potential function ϕ increases by exactly one. This completes the proof. □

Theorem 3.2 (Boolean Linear Optimization: Bounded-Weights)**.** *For any family* $\mathcal{F} \subseteq 2^U$ *and unknown integer weight function satisfying* $|w_e^*| \leq B$*, we can sort the feasible sets according to their weight* $w^*(S)$ *and hence solve* $\arg\min_{S\in\mathcal{F}} w^*(S)$ *using* $O(nB \log nB)$ *comparison queries, where* $|U| = n$*. In fact, the number of comparison queries required is* $O((n+C)\log C)$ *if* C *is the number of distinct weight values realized by the feasible set.*

Proof. Algorithm 3.1.1 maintains a structured representation which allows us to identify the weight class of any point $x \in \mathcal{P}$. We can decide that $w^*(x)$ is the ith smallest weight by checking if $x \in \mathcal{A}_i$ (or equivalently, $x - r_i \in \mathcal{A}$), simply because the algorithm's termination condition ensures that $\mathcal{P} \subseteq \bigcup_{i\in[\ell]} \mathcal{A}_i$. This serves as a compact representation for the sorted order of all points by their weights $w^*(x)$. Moreover, r_1 (or any other point in B_1) is a point of minimum weight.

The number of comparison queries per point encountered is $O(\log \ell) = O(\log nB)$ due to binary search over at most $O(nB)$ weights, summing up to $O(nB \log(nB))$ queries over its entire execution. This bound can be improved to $O((n + C)\log C)$ if C is the number of distinct weight values realized by sets in $\mathcal{F}$, because $O(\log \ell) = O(\log C)$ and the number of points encountered is at most $n - 1 + C$. □

The GSL framework provides a powerful information-theoretic bound. More importantly, it provides a clear path toward computationally efficient algorithms, which we turn to in the next section. We make two final observations:

1. A conceptual difference between Theorem 1.1 and Theorem 3.2 is that the former uses *conic spans* while the second result uses only *linear spans* for inference. The use of linear spans makes it easier to reason about, and as we will see, to obtain efficient algorithms.

2. The GSL algorithm can be implemented even when restricted to only performing equality queries—i.e., queries that only tell us whether $w^*(S) = w^*(T)$ or $w^*(S) \neq w^*(T)$. In this setting, the buckets would be unordered, and identifying whether $w^*(y_{t+1}) = w^*(r_i)$ would require $O(nB)$ equality queries, increasing the query complexity to $O(n^2B^2)$. More generally, $O((n + C) \cdot C)$ equality queries if C is the number of distinct weight values realized by the feasible set.

3.2 Efficient Implementation of Global Subspace Learning

The main computational bottleneck in the GSL algorithm is in finding a point $y \in \mathcal{P}$ such that $y \notin \bigcup_{i\in[\ell]} \mathcal{A}_i$—i.e., the separation problem over the union of the affine subspaces $\mathcal{A}_i$. We show that we can solve this separation problem when $\mathcal{F}$ is the family of all k-subsets $\binom{U}{k}$, or more generally, when $\mathcal{F}$ is the set of bases of a linear matroid representable over $\mathbb{R}$.

Recall that the subspace $\mathcal{A}$ is the span of directions orthogonal to w^*; since w^* has integer coordinates, it must lie in the orthogonal lattice $\mathcal{W} := \mathcal{A}^\perp \cap \mathbb{Z}^n$. Let $A \in \mathbb{Z}^{m\times n}$, where $m = \dim(\mathcal{A}) \leq n - 1$, be the matrix whose rows form a basis for $\mathcal{A}$. By definition, the orthogonal complement $\mathcal{A}^\perp$ is the nullspace of A. Therefore, $\mathcal{W}$ is precisely the *integer nullspace* of A.

An integer basis for $\mathcal{W}$ can be computed using the *Smith Normal Form* (SNF) of the $m \times n$ matrix A, which returns a decomposition $D = UAV$, where $U \in \mathbb{Z}^{m\times m}$ and $V \in \mathbb{Z}^{n\times n}$ are unimodular matrices, and D is an $m \times n$ diagonal matrix with m non-zero entries. The last $s := n - m$ columns of V, which correspond to the zero columns of D, form the required integer basis for $\mathcal{W}$. Moreover, the decomposition U, D, V can be found in polynomial time.

Lemma 3.3 (Integer Nullspace via SNF, [KB79, Sch86])**.** *Let $A \in \{-1, 0, 1\}^{m\times n}$. An integer basis for the integer nullspace*

$$\ker_{\mathbb{Z}}(A) = \{x \in \mathbb{Z}^n : Ax = 0\}$$

can be computed in polynomial time in n and m. The entries of the resulting basis vectors are bounded in bit-length by $\mathrm{poly}(n, m)$.

Note that our matrix entries are of the form $\mathbb{1}_S - \mathbb{1}_T$ where $S, T \in \mathcal{F}$, and hence they lie in $\{-1, 0, 1\}$. Since $m \leq n$, Lemma 3.3 implies that the resulting integer basis vectors have entries bounded by $2^{\mathrm{poly}(n)}$. Let $W \in \mathbb{Z}^{s\times n}$ denote the matrix whose rows form this basis spanning the orthogonal lattice $\mathcal{W}$. The next implication is immediate:

Observation 3.4. Point x belongs to $\mathcal{A}_i$ for some $i \in [\ell]$ if and only if $W(x - r_i) = 0$. This implies that $y \notin \bigcup_{i\in[\ell]} \mathcal{A}_i$ is equivalent to $Wy \notin \{Wr_1, \ldots, Wr_\ell\}$, where the r_i vectors are the bucket representatives.

Let $W = [w_1, \ldots, w_n]$ where $w_i \in \mathbb{Z}^s$ is the ith column of W and $Z = \{z_1, \ldots, z_\ell\}$ with $z_i = Wr_i$. We will use $w(S) := \sum_{i\in S} w_i$. Henceforth, the following problem is the one we focus on.

Definition 3.5 (Separation Problem)**.** Given vectors $w_1, w_2, \ldots, w_n \in \mathbb{Z}^s$, and a collection $Z \subseteq \mathbb{Z}^s$ with some ℓ vectors, identify a subset $S \in \mathcal{F}$ such that $w(S) \notin Z$.

3.2.1 Efficient Separation via Dynamic Programming

As a quick warm-up, we show an efficient algorithm for the separation problem using dynamic programming, when the feasible set $\mathcal{F}$ is 2^U. (Note that even these cases were not efficiently implementable using the Sieving algorithm of Section 2).

Define

$$\mathcal{V}_i = \{w(S) : S \subseteq [i]\}, \quad 0 \le i \le n,$$

as the set of vector values obtained by adding a subset of the first i columns of W. We can compute $\mathcal{V}_i = \mathcal{V}_{i-1} \cup \{w_i + v : v \in \mathcal{V}_{i-1}\}$ starting from $V_0 = \emptyset$ and increasing i stopping when $|\mathcal{V}_i| > \ell$. This implies that there exists a value $v \in V_i$ that is not in Z as $|Z| = \ell$.

The set S such that $w(S) = v$ can be constructed by tracking the computation of V_i.

3.2.2 Efficient Separation for Linear Matroids via Algebraic Techniques

In this section, we show how to solve the separation problem for $\mathcal{F}$ being the bases of a linear matroid $\mathcal{M} = (U, \mathcal{I})$. Recall that a linear matroid of rank k is represented by vectors $V = \{v_1, \ldots, v_n\} \subset \mathbb{R}^k$, where each v_i corresponds to an element in U. The independent sets $\mathcal{I}$ are those subsets of U whose corresponding vectors are linearly independent, and the bases $\mathcal{F}$ are the independent sets of size k.

In order to solve the separation problem for vector costs w_i, we proceed in two steps:

1. First, we solve the problem when the dimension of the vectors in the separation problem is $s = 1$, i.e., we are given a collection of scalars $c_1, \ldots, c_n$, and a collection Z of scalars, and want to find a feasible set $S \in \mathcal{F}$ such that $\sum_{i \in S} c_i \notin Z$.
2. Secondly, we encode each vector w_i into a unique non-negative scalar $\widehat{w}_i$ (and each vector in Z also to a scalar), and apply the result from the first step.

The scalar encoding in the second step is easy to describe. For a vector $v \in [-M, M]^s$, define its encoding

$$g(v) := (3nM)^s + \sum_{j=1}^{s} (3nM)^{j-1} v_j.$$

Lemma 3.6 (Scalar Encoding)**.** *For any vector $v \in [-M, M]$, its encoding $g(v)$ takes on values in $[0, (3nM)^{s+1}]$. Moreover, for any two subsets $S, S' \subseteq U$, the sums of the encoded values are equal if and only if the sums of the original vectors are equal: i.e., $\sum_{i \in S} g(w_i) = \sum_{i \in S'} g(w_i) \iff \sum_{i \in S} w_i = \sum_{i \in S'} w_i$.*

Given this, we turn to the first step, and give an algorithm for the separation problem when the columns w_i of W are scalars:

Lemma 3.7 (Separation for Linear Matroids)**.** *Given a linear matroid $\mathcal{M} = (U, \mathcal{I})$, non-negative integers $c_1, c_2, \ldots, c_n$ for elements in U, and a set of values $Z = \{z_1, \ldots, z_\ell\}$, we can identify a basis $S \in \mathcal{F}$ of $\mathcal{M}$ such that $c(S) \notin Z$ (or decide that no such basis exists) in $\mathrm{poly}(n, \ell, \log M)$ time, where $M = \max_{i \in [n], j \in [\ell]} \{c_i, z_j\}$.*

To give the main idea behind the lemma, let us first give a proof that yields a running time of $\mathrm{poly}(n, M)$. The full proof showing a $\mathrm{poly}(n, \log M)$ runtime is deferred to the appendix.

Warmup Proof of Lemma 3.7. The main idea is to use a specific determinant as a generating function for the set of basis costs, $\{c(S) : S \in \mathcal{F}\}$. For a linear matroid of rank k represented by $V = \{v_1, \ldots, v_n\} \subseteq \mathbb{R}^k$, we use the Cauchy-Binet identity to infer:

$$\det\left(\sum_{i=1}^{n} x_i v_i v_i^T\right) = \sum_{S \in \mathcal{F}} \left(\prod_{i \in S} x_i\right) \det(V_S)^2,$$

where V_S is the $k \times k$ matrix with columns indexed by S, and $\det(V_S) \neq 0$ if and only if $S \in \mathcal{F}$. We now define a polynomial $P(x)$ by setting $x_i = x^{c_i}$, where c_i's are the given scalars: i.e.,

$$P(x) = \det\left(\sum_{i=1}^{n} x^{c_i} v_i v_i^T\right) = \sum_{S \in \mathcal{F}} x^{\sum_{i \in S} c_i} \det(V_S)^2 = \sum_{S \in \mathcal{F}} x^{c(S)} \det(V_S)^2. \tag{1}$$

The exponents with non-zero coefficients in $P(x)$ are precisely the basis costs $\{c(S) \mid S \in \mathcal{F}\}$. The degree of $P(x)$ is bounded by nM. We can therefore find $P(x)$ exactly using polynomial interpolation by evaluating it at $O(nM)$ points. This process runs in $\text{poly}(n, M)$ time. Once the polynomial $P(x)$ is known, we can inspect its coefficients to find if any exponent $c \notin Z$ has a non-zero coefficient.

Furthermore, the associated a basis can be found using self-reducibility for matroids. For each element $j \in U$, we compute the "edge-deletion" polynomial $P_j(x) = \det(\sum_{i \neq j} x^{c_i} v_i v_i^T)$: if c appears as an exponent in $P_j(x)$, a basis with cost c exists without j. We delete j and recurse on the smaller problem. On the other hand, if c does not appear in $P_j(x)$, then j must be in every basis S with $c(S) = c$. We select j and recurse, searching for a basis with cost $c - c_j$ in the problem on $U/\{j\}$ (i.e., using an "edge-contraction" polynomial $P_{/j}(x)$) □

When the bound M is large, the $\text{poly}(n, M)$ time bound is infeasible, as the degree of $P(x)$ is too high for efficient interpolation. The full proof, which appears in Section B, avoids this problem by remaindering; i.e., working with the polynomial $P(x) \pmod{(x^p - 1)}$ for a suitable prime $p \in \text{poly}(n, \ell, \log M)$. This prime is chosen so that the residues modulo p of all values in Z and at least one basis cost $c \notin Z$ (if such a basis exists) are distinct. This allows for efficient interpolation and reconstruction within the required time bounds. Now we can use Lemma 3.7 to prove Theorem 1.4.

Theorem 1.4 (Efficient Optimization over Linear Matroids)**.** *Let $\mathcal{F}$ be the bases of a rank-k linear matroid $\mathcal{M} = (U, \mathcal{I})$, represented by $V \in \mathbb{R}^{k \times n}$ with entry representation size $\text{poly}(n)$. For an unknown integer weight function w^* satisfying $|w_e^*| \leq B$, we can sort the bases according to their weight $w^*(S)$, and hence find the minimum-weight basis $\arg\min_{S \in \mathcal{F}} w^*(S)$ in $\text{poly}(n, B)$ time. The number of comparison queries required is $O(nB \log nB)$ and more generally, $O((n+C) \log C)$, where $C = |\{w^*(S) : S \in \mathcal{F}\}|$ is the number of distinct basis weights.*

Proof. From Algorithm Section 3.1.1, the stated query complexity follows directly from Theorem 3.2. It remains to bound the total running time required to implement the separation steps.

The maintenance of buckets, affine subspaces, and the corresponding updates can each be carried out in $\text{poly}(n, B)$ time overall. By Lemma 3.1, the number of separation steps is bounded by $O(nB)$. Each separation step invokes the procedures in Lemma 3.6 and Lemma 3.7, which take $\text{poly}(n, \log M)$ and $\text{poly}(n, \ell, \log M)$ time, respectively. Summing over all separation steps gives a total time of $\text{poly}(n, B)$, using the facts that $\ell = O(nB)$ and $M = 2^{\text{poly}(n)}$, the latter following from Lemma 3.3. □

3.3 Other Applications

In this section, we present several applications of Theorem 1.4 and Theorem 3.2 that yield low-query and efficient algorithms for well-studied problems such as k-SUM, SUBSET-SUM, and sorting sumsets $A + B$ (see [KLM19]). When the elements of the ground set take integer values in the range $[-B, B]$, our algorithms achieve better query complexity while also being the first efficient algorithms in this setting.

3.3.1 Low-query and efficient comparison decision trees

1. **k-SUM:** In the k-SUM problem, the input elements in U have unknown values w^*_e, and the goal is to decide whether there exist k distinct elements whose sum is zero. The comparison model considered in [KLM19] allows comparisons between $w^*(S)$ and $w^*(T)$ for $S, T \in \binom{U}{k}$ and between $w^*(S)$ and 0. They achieve a query complexity (decision tree depth) of $O(kn \log^2 n)$.

   Using Theorem 1.4 with $C = kB$, we obtain an algorithm with query complexity $O((n + kB) \log(kB))$ and running time $\text{poly}(n, B)$, which improves both measures when $B = o(n)$. Our algorithm first sorts all feasible k-sets by their weights using only comparison queries between solutions, and then performs a binary search using comparisons with 0 to check whether a subset of size k has total weight zero.

2. **SUBSET-SUM:** In the SUBSET-SUM problem, the goal is to determine whether there exists a subset $S \subseteq U$ such that $w^*(S) = t$ for a given target t. Similar to k-SUM, the model allows comparisons between feasible sets ($\mathcal{F} = 2^U$) and between a feasible set and the target t. The algorithm in [KLM19] achieves query complexity $O(n^2 \log n)$.

   Using Algorithm Section 3.1.1 with the efficient separation step stated in Section 3.2.1, we obtain an algorithm with query complexity $O(nB \log(nB))$ and running time $\text{poly}(n, B)$, which again improves both parameters when $B = o(n)$. As before, the algorithm first sorts all subset weights using comparison queries and then performs a binary search using comparisons with t to determine whether a subset with total weight t exists.

3. **Sorting sumsets $A + B$:** In the $A + B$ sorting problem, we are given two sets A and B, each of cardinality n, with unknown element values w^*_a and w^*_b for $a \in A$ and $b \in B$. The goal is to sort the set $A + B = \{a + b : a \in A, b \in B\}$ using only comparisons between sums of the form $w^*(a_1) + w^*(b_1)$ and $w^*(a_2) + w^*(b_2)$.

   The model in [KLM19] even allows stronger comparisons—such as comparing sums of four elements $w^*(a_1) + w^*(b_1) + w^*(a_2) + w^*(b_2)$ versus $w^*(a_3) + w^*(b_3) + w^*(a_4) + w^*(b_4)$—which we do *not* use. Their algorithm achieves a query complexity of $O(n \log^2 n)$.

   To fit this problem into our framework, we set $U = A \cup B$, and let $\mathcal{F}$ consist of the bases of a partition matroid over U with parts A and B, each having capacity one. Then comparing $w^*(a_1) + w^*(b_1)$ with $w^*(a_2) + w^*(b_2)$ corresponds exactly to comparing the weights of two feasible sets $\{a_1, b_1\}$ and $\{a_2, b_2\}$. Since partition matroids are representable over the reals, and the element values lie in $[-B, B]$, we can apply Theorem 1.4 with $C = 2B$ to obtain an algorithm with query complexity $O((n + B) \log B)$ and running time $\text{poly}(n, B)$. This improves on previous bounds when $B = o(n)$ and uses only valid pairwise comparisons among elements of $A + B$.

Moreover, our results extend to an even weaker model in which only *equality* queries are allowed. In this setting, the query complexities for the three problems above are $O(kB(n+kB))$, $O(n^2B^2)$, and $O(B(n + B))$, respectively, while still maintaining a running time of $\text{poly}(n, B)$. See Section 1.1.2 for a summary comparing the results.

4 Minimum Cut using Cut Comparisons

We now turn to specific combinatorial problems, and give efficient algorithms using their structural properties. We begin with the min-cut problem on graphs and prove Theorems 1.5 and 1.6, which we restate for convenience.

Theorem 1.5 (Minimum cut). *There is a randomized algorithm that computes the exact minimum cut of a simple graph G with high probability using $\widetilde{O}(|V|)$ cut comparison queries and $\widetilde{O}(|V|^2)$ time.*

Theorem 1.6 (Graph recovery). *A simple unweighted graph $G \notin \{K_2, \bar{K}_2, K_3, \bar{K}_3\}$ can be recovered using $O(\min\{(|E|+|V|)\log|V|, |V|^2\})$ cut comparison queries and in $\widetilde{O}(|V|^2)$ time.*

In order to prove the two results, we draw on the following useful primitives that can be obtained using the cut comparison queries:

Theorem 4.1 (Structural Primitives). *When $n \geq 4$, for any vertex $u \in U$:*

(i) Given a set of k vertices $A = \{v_1, \ldots, v_k\} \subseteq U \setminus \{u\}$, the neighbors of u in A can be determined using $k + O(\log n)$ queries and $\widetilde{O}(n+k)$ time.

(ii) Given an ordered subset $T = (v_1, \ldots, v_t) \subseteq U \setminus \{u\}$, either the neighbor $v_i \in N(u)$ of u with the smallest index $i \in [t]$ (or a certificate that $N(u) \cap T = \emptyset$) can be found using $O(\log n)$ queries and $\widetilde{O}(n)$ time.

Remark 4.2 (A Note on Runtimes). We assume that the query input (S, T) to the comparison oracle is given as two bit vectors encoding the sets S and T respectively. The running time bounds are the total computation done to process the query information, where oracle calls are assumed to take unit time. The dominant cost is writing down the input bit vectors for queries, which takes $O(n)$ time per query. In addition, we separately account for standard algorithmic overhead, such as binary search or other data structure updates.

4.1 Graph Reconstruction

With the power of Theorem 4.1 behind us, the graph recovery problem is easily solved. Indeed, using Theorem 4.1(i), we can discover the neighborhood of node $u \in V$ using $n-1+O(\log n)$ queries and $\widetilde{O}(n)$ time; summing over all nodes gives us the graph G in at most $O(n^2)$ queries and $\widetilde{O}(n^2)$ time.

To get the better bound of $O(m \log n)$ for sparse graphs, we use Theorem 4.1(ii) to prove:

Lemma 4.3 (Edge Extraction). *When $n \geq 4$, for any vertex u, a subset $T \subseteq U - u$ of vertices, and any integer k, we can extract $r = \min(k, \partial(u, T))$ edges from $\partial(u, T)$ using $O(r \log n)$ queries and $\widetilde{O}(n)$ time.*

Proof. Start with $T_0 := T$ and extract a neighbor $v_i \in N(u) \cap T_i$ (if any) from $\partial(u, T_i)$ using Theorem 4.1(ii); set $T_{i+1} := T_i - v_i$, and repeat. Each step requires $O(\log n)$ queries and $\widetilde{O}(n)$ time. Repeating this for $r = \min(k, \partial(u, T))$ neighbors gives a total of $O(r \log n)$ queries and $\widetilde{O}(nr)$ time. A more efficient implementation, which amortizes the search and is detailed in Section C, achieves the claimed $\widetilde{O}(n)$ time. □

Given this efficient edge-extraction claim, we now prove the claimed bounds for graph recovery.

Proof of Theorem 1.6. When $n \geq 4$, using Lemma 4.3, we can discover all the neighbors of u using $|\partial(u)| \cdot O(\log n)$ queries and $\widetilde{O}(n)$ time. Summing over all u gives a total of at most $O((m+n)\log n)$ queries and $\widetilde{O}(n^2)$ time. Hence, we start by running this procedure until the number of edges discovered is at least $n^2/\log n$. At this point, we switch to the algorithm using Theorem 4.1(i) to discover the remaining edges using at most $O(n^2)$ queries. This guarantees that we use at most $O(\min((m+n)\log n, n^2))$ queries and $\widetilde{O}(n^2)$ time.

When $n \leq 3$ and $G \notin \{K_2, \bar{K}_2, K_3, \bar{K}_3\}$, sorting the degrees of the vertices reveals the graph. □

4.2 Finding Minimum Cuts

We now turn to finding minimum cuts. We start off by observing that when $n \leq 4$, we can bruteforce the minimum cut and otherwise, use Theorem 1.6 to optimize over min-cuts: we reconstruct the graph using $O(\min((m+n)\log n, n^2))$ queries and $\widetilde{O}(n^2)$ time; we can then optimize over it in $\widetilde{O}(n^2)$ time. However, the number of cut queries incurred this way is quadratic and not near-linear, as promised in Theorem 1.5.

However, we can use ideas from [RSW18] to do better. Suppose G' is a contracted graph (i.e., where some nodes in G have been contracted into each other), and $G'(p)$ is the "edge-percolation" graph obtained by sub-sampling each edge of G' independently with probability p. Distilling the results of [RSW18] shows that if, for any contraction G' of G, we can obtain a sample from $G'(p)$ using $\alpha \cdot p \cdot |E(G')|$ queries, then we can compute the min-cut of G using $\alpha \cdot \widetilde{O}(n)$ queries and in $\widetilde{O}(n^2)$ expected runtime. (A proof appears in Section C.2.)

To use this result, we show how to sample percolations of contractions with $\alpha = \widetilde{O}(1)$.

Lemma 4.4 (Contracted Graphs)**.** *When $n \geq 4$, for any contraction $G' = (U', E')$ of G, with its vertex set U' represented as a partition of the original vertex set U, we can sample a subgraph $G'(p)$ where each edge in G' is sampled independently with probability p using $|E'| \cdot O(p \log n) + \widetilde{O}(n)$ queries and $\widetilde{O}(n^2)$ time in expectation.*

The sampling can be done more generally for any induced subgraph $G'' = G'[S]$ for some $S \subseteq U'$ with $|E'(S)| \cdot O(p \log n) + \widetilde{O}(n)$ queries and $\widetilde{O}(n^2)$ time.

Proof. For any regular vertex u, if V_u is the super vertex that contains it, we will show how to sample edges from $\partial(u, U - V_u)$ with each edge sampled independently with bias $q \in [0, 1]$. Call this random subgraph $\partial_{G'}(u, q)$. Given the primitive to sample $\partial_{G'}(u, q)$, taking the union $\cup_{u \in U} \partial_{G'}(u, q)$ of such samples for each vertex $u \in U$ with $q = 1 - \sqrt{1-p}$ (so that $1 - (1-q)^2 = p$) gives $G'(p)$. Because, the probability of any edge $e \in E'$ to be selected is $1 - (1-q)^2 = p$ and the independence is trivial.

To sample $\partial_{G'}(u, q)$, let T be a random subset of $U - V_u$ with each element sampled independently with probability q. Reveal all the edges of u going into T using Lemma 4.3. This takes $|\partial(u, T)| \cdot O(\log n)$ queries which is $q \cdot |\partial(u, U - V_u)| \cdot O(\log n)$ in expectation and $\widetilde{O}(n)$ time. Summing over all $u \in U$ gives a total of $|E'| \cdot O(p \log n) + \widetilde{O}(n)$ queries and $\widetilde{O}(n^2)$ time in expectation. An edge $\{u, v\}$ for $v \in U \backslash V_u$ is sampled iff $v \in T$. This implies that each edge is sampled independently with probability q. Simply replacing U with S gives the extension to induced subgraphs. □

Combining Lemma 4.4 with the abstraction of [RSW18] discussed above completes our algorithm for min-cuts (Theorem 1.5).

4.3 Proof of the Structure Primitives Theorem

Finally, we turn to the proof of the Structure Primitives result in Theorem 4.1. We will provide a proof sketch here for the query complexity bounds; see Section C for the full proof including the running time bounds.

Proof sketch of Theorem 4.1. Recall that we want to identify the edges coming out of u. The key observation is that, for any set S not containing u, the sign of $\partial(S + u) - \partial(S)$ tells us whether less, equal to, or more than half of the edges incident to u go into S (if the sign is positive, zero, or negative).

Suppose we find a set S such that *just below* half of the edges from u go into S (so that expression has a positive sign)—such an S is called a *median set for* u. Now if we add v to the median set S, and the sign of $\partial(S+v+u)-\partial(S+v)$ changes, we know that v is a neighbor of u. In summary, once we have a median set S, we can find all the neighbors of u not in S.

How do we find a median set for u? Order all the vertices other than u (say this is $v_1, v_2, \ldots, v_{n-1}$, define S_i to be the first i elements. If u has at least one edge incident to it, the query $\partial(S_0+u)-\partial(S_0)$ has a positive sign, whereas $\partial(S_{n-1}+u)-\partial(S_{n-1})$ has a negative sign (note that we use $S_0=\emptyset$ and $S_{n-1}+u=U$ here for the sake of exposition, but such trivial cuts are not allowed in the actual comparison model. The full proof in Section C handles this correctly and with full rigor). So we can perform binary search to find a point where the sign changes, which can be used to find a median set in $\log_2 n$ comparisons. (In fact, we can find two disjoint median sets—a prefix and a suffix of this ordering—and use the two to find all the neighbors in some k-sized set with $k+O(\log n)$ queries.)

For part (ii), the argument builds on the use of such a median set S. Given an ordered set $T=\{v_1,\ldots,v_r\}$ where v_j is the first neighbor of u in the ordering, we first compute a median set S that is guaranteed to exclude the prefix $\{v_1,\ldots,v_j\}$. Then, we binary search over the chain

$$S \subseteq S\cup\{v_1\} \subseteq \cdots \subseteq S\cup(T\setminus S)$$

to locate the point at which the marginal changes sign. This transition occurs precisely at the set $S\cup\{v_1,\ldots,v_{j-1}\}$, thus identifying v_j as the first neighbor of u in T. □

4.4 Sparsifiers for Heavy Subgraph Problems

The work of [EHW15] considers a broad class of graph optimization problems, which they call *heavy subgraph* problems; these include densest subgraph, densest bipartite subgraph, and d-max cut. For these problems, they show that uniformly sampling $\widetilde{O}(n/\varepsilon^2)$ edges without replacement from the underlying graph G produces a subgraph H such that running any α-approximation algorithm for the problem on H yields (after appropriate rescaling) an $(\alpha-\varepsilon)$-approximate solution for the original graph, with high probability. In other words, the sampled subgraph preserves the structure of the problem up to a small loss in accuracy, and approximate solutions on the sparsifier H translate to approximate solutions on the full graph G. We show how to sample a fixed number of edges from G uniformly using the following lemma:

Lemma 4.5 (Sampling Lemma)**.** *When $n\geq 4$, given cut comparison queries, for any $k=\widetilde{\Omega}(1)$, we can sample k edges uniformly without replacement using $\widetilde{O}(n+k)$ queries and $\widetilde{O}(n^2)$ time with high probability.*

Proof. Consider the process that at time step t (starting from $t=0$) samples $G(p_t)$ for $p_t := 2^t/n^2$ until the sample $G(p_t)$ at $t=r$ has at least k edges. Then, subsample k edges uniformly from the sample $G(p_r)$.

Because edges are sampled i.i.d. from the Bernoulli $\mathrm{Ber}(p_t)$ distribution at each step t, the symmetry implies that the subsample of edges is a uniform sample of k edges. Using Lemma 4.4, each sample $G(p_t)$ takes $p_t\cdot\widetilde{O}(m)+\widetilde{O}(n)$ queries and $\widetilde{O}(n^2)$ time in expectation. Summing this over $0\leq t\leq r$ gives an upper bound of $2p_r\cdot\widetilde{O}(m)+\widetilde{O}(nr)$ queries and $\widetilde{O}(n^2 r)$ time. Using the fact that the probability that $G(p)$ has less than k edges for $p=2k/m$, is at most $e^{-k/4}$, we can conclude that $p_r\leq 4k/m$ with probability at least $1-e^{-k/4}$. So the expected number of queries is upper bounded by $\widetilde{O}(nr+k)=\widetilde{O}(n+k)$ and the running time by $\widetilde{O}(n^2 r)=\widetilde{O}(n^2)$ using the fact that $r\leq O(\log n)$. □

Using Lemma 4.5 with $k = \widetilde{O}(n/\varepsilon^2)$, we can obtain a random subgraph H of the unknown graph G to get such a sample of edges using $\widetilde{O}(n/\varepsilon^2)$ queries, and hence apply existing approximations for all heavy subgraph problems.

The results in this section do not generalize to the weighted case when the edges of the graph have unknown non-negative weights. We address this issue by giving some preliminary results under additional structural assumptions and slightly stronger oracle models—see Section C.3—but the general question remains wide open.

See Section C.4 for some illustrative examples which show some separation between the capabilities of the cut comparison and cut value query oracles.

Appendix

A Omitted Content from Section 2

In this section, we give omitted proofs and other discussion from Section 2. In particular, we show that the "certification" problem, where we have to write the shortest certificate of some point y^* being the optimizer of $\langle w^*, x\rangle$ over points $x \in \mathcal{P}$, can be directly related to the conic dimension. We present this discussion since it helps motivate and demystify the definition further. We then give the proofs omitted from Section 2.

The notion of conic dimension is intimately related the notion of *inference dimension* given in [KLMZ17]. They defined it as being the size of a largest "inference-independent" set where none of the inequalities are implied by the others. Our definition is very similar to theirs, with two minor differences: (a) we restrict to linear optimization, and (b) we consider sequences where each comparison is not implied by the previous ones. Since implications using linear inequalities has a natural geometric visualization as a cone, we call it *conic dimension*.

A.1 The Certification Problem

Before addressing the optimization problem, we ask whether it is even possible to certify optimality efficiently. Suppose we wish to prove that some point $y \in \mathcal{P}$ is the optimal solution, i.e., $y = \arg\min_{x\in\mathcal{P}}\langle w^*, x\rangle$, or equivalently, that

$$\langle w^*, x - y\rangle \geq 0 \quad \forall x \in \mathcal{P}. \tag{2}$$

We want to use as few comparisons as possible.

1. A *y-certificate* is a collection $\mathcal{C}$ of index pairs (i, j) such that for all $w \in \mathbb{R}^d$, we have that
$$(\langle w, x_i - x_j\rangle \geq 0 \quad \forall (i,j) \in \mathcal{C}) \Rightarrow (\langle w, x - y\rangle \geq 0 \quad \forall x \in \mathcal{P}) \tag{3}$$
2. A certificate $\mathcal{C}$ is *valid* if $\langle w^*, x_i - x_j\rangle \geq 0$ for all $(i, j) \in \mathcal{C}$.

Hence a valid y-certificate implies that y is a minimizer of $\langle w^*, x\rangle$ among points in $\mathcal{P}$. In fact, the minimum number of queries needed to find an optimal solution is lower bounded by the minimum size of a valid certificate.

Recall that given a set $V \subseteq \mathbb{R}^d$ of vectors, the cone generated by V is the set of all vectors that can be written as positive linear combinations of vectors from V; this is denoted by $\text{cone}(V)$. Formally,

$$\text{cone}(V) = \left\{ x \in \mathbb{R}^d : x = \sum_{v\in V} \alpha_v\, v,\ \alpha_v \geq 0 \right\}.$$

The following lemma gives a characterization of y-certificates based only on the geometry of the point set $\mathcal{P}$.

Lemma A.1. *A set $\mathcal{C} \subseteq [N] \times [N]$ is a y-certificate iff $\mathcal{P} \subseteq y + \text{cone}\left(\{x_i - x_j\}_{(i,j)\in\mathcal{C}}\right)$.*

Proof. Suppose $\mathcal{C}$ is a y-certificate. Then, the definition implies that the system:

$$\begin{aligned} &\langle w, x_i - x_j\rangle \geq 0 \quad \forall (i,j) \in \mathcal{C} \\ &\langle w, x - y\rangle < 0 \end{aligned} \tag{4}$$

is infeasible for any $x \in \mathcal{P}$. By Farkas' lemma, this implies that $x - y \in \text{cone}\left(\{x_i - x_j\}_{(i,j)\in\mathcal{C}}\right)$. Conversely, if $x-y \in \text{cone}\left(\{x_i - x_j\}_{(i,j)\in\mathcal{C}}\right)$ and $\langle w, x_i - x_j\rangle \geq 0$ for all $(i,j) \in \mathcal{C}$, then $\langle w, x-y\rangle \geq 0$ follows immediately. □

Definition A.2. For a sequence $\sigma = (x_{i_1}, x_{i_2}, \ldots, x_{i_k})$, the *induced certificate* is defined as $\mathcal{C}_\sigma = \{(i_\ell, i_{\ell+1}) : 1 \leq \ell \leq k-1\}$

A.1.1 Basic Subsequences

For a set $\mathcal{Y} \subseteq \mathbb{R}^d$ with $\text{ConicDim}(\mathcal{Y}) = k$ and any sequence $\sigma = (y_1, \ldots, y_n) \in \mathcal{Y}^n$, consider a process that maintains a subsequence π_t of σ for each $0 \leq t \leq n$. We start with the empty sequence $\pi_0 = \langle\rangle$, and at step $t+1$, we update

$$\pi_{t+1} = \begin{cases} \pi_t \circ y_{t+1}, & \text{if } y_{t+1} - y_1 \notin \text{env}(\pi_t) \\ \pi_t, & \text{otherwise.} \end{cases}$$

An analogy can be made to Kruskal's algorithm, where we add the next element in the sequence precisely when it is not conically spanned by the envelope of the current set. We observe:

1. the sequence π_t is a prefix of π_{t+1}, and moreover, π_{t+1} contains at most one more element.
2. $|\pi_n| \leq k$.

The first observation is immediate from the construction, and the second follows by contradiction: if $|\pi_n| > k$, then some subsequence of size $k+1$ contradicts the definition of conic dimension. Let $B(\sigma) := \pi_n$ be the subsequence of σ obtained by this process; we call this the "basis" of σ.

Lemma A.3. *Given a sequence $\sigma = (y_1, \ldots, y_n) \in \mathcal{Y}^n$, we have:*

$$\text{cone}(\{y_j - y_1\}_{1\leq j\leq n}) \subseteq \text{env}(B(\sigma)) \subseteq \text{env}(\sigma). \tag{5}$$

Proof. For the rightmost inclusion, suppose $B(\sigma) = (y_{i_1}, \ldots, y_{i_k})$. Then,

$$\text{env}(B(\sigma)) = \text{cone}(\{y_{i_{\ell+1}} - y_{i_\ell}\}_{1\leq\ell\leq k-1}) \subseteq \text{cone}(\{y_j - y_i\}_{1\leq i\leq j\leq n}) = \text{env}(\sigma). \tag{6}$$

For the leftmost inclusion, let $\pi_t := B(\sigma^{1:t})$ denote the basis of the prefix at time t. Consider any $y_t \in \sigma$:

- If $y_t \notin \pi_n = B(\sigma)$, then it was excluded by the base construction algorithm, meaning:

$$y_t - y_1 \in \text{env}(\pi_{t-1}) \subseteq \text{env}(\pi_n) = \text{env}(B(\sigma)).$$

- If $y_t \in \pi_n$, then it is one of the elements of the basis, and we can write:

$$y_t - y_1 = \sum_{\ell : i_\ell \le t} (y_{i_{\ell+1}} - y_{i_\ell}) \in \text{cone}(\{y_{i_{\ell+1}} - y_{i_\ell}\}_{1 \le \ell \le k-1}) = \text{env}(B(\sigma)).$$

Hence, in both cases, $y_t - y_1 \in \text{env}(B(\sigma))$, completing the proof. □

We can now relate the size of valid x^*-certificates to the conic dimension of the underlying point set.

Lemma A.4. *If some point set $\mathcal{Y} \subseteq \mathbb{R}^d$ has conic dimension k, and $y^* = \arg\min_{y \in \mathcal{Y}} \langle w^*, y \rangle$, then there exists a valid y^*-certificate $\mathcal{C}$ of length $|\mathcal{C}| \le k - 1$.*

Proof. Order the points in $\mathcal{Y}$ as the sequence $\sigma = \langle y^* = y_1, y_2, \ldots, y_n \rangle$ such that $\langle w^*, y_i \rangle \le \langle w^*, y_{i+1} \rangle$ for all i, and let $B(\sigma) = (y_{i_1}, \ldots, y_{i_k})$ be the basic subsequence corresponding to σ. We claim that the induced certificate $\mathcal{C} := \mathcal{C}_{B(\sigma)}$ is a valid y^*-certificate. Notice that $\mathcal{C}$ is defined such that

$$\text{env}(B(\sigma)) = \text{cone}(\{y_i - y_j\}_{(i,j) \in \mathcal{C}}).$$

Using Lemma A.3, we have $\mathcal{Y} \subseteq y_1 + \text{cone}(\{y_i - y_j\}_{(i,j) \in \mathcal{C}})$, which means that $\mathcal{C}$ is a valid certificate using Lemma A.1. Moreover, by construction, $(i, j) \in C$ means $\langle w^*, y_j - y_i \rangle \ge 0$, which in turn means that the certificate is valid. □

A.1.2 Proof of Lemma 2.3

Lemma 2.3 (Sieving Lemma)**.** *The expected number of points that are eliminated from $\mathcal{P}$ at step 1.4 is at least $|\mathcal{P}|/2$.*

Proof of Lemma 2.3. For the analysis, consider a sequence $(x_1, \ldots, x_N)$ be such that $\langle w^*, x_i \rangle \le \langle w^*, x_{i+1} \rangle$ for all $1 \le i \le N-1$. Note that σ is distributed as a random subsequence of $(x_1, \ldots, x_N)$, where each element is selected independently with probability $2k/N$.

Consider a slightly modified, more lenient version of the algorithm that eliminates a point $x_t \in \mathcal{P} \setminus \{y\}$ in Step 1.4 only if $x_t - y \in \text{env}(B(\sigma_{t-1}))$ where σ_t is the prefix of σ restricted to elements $\{x_i\}_{1 \le i \le t}$; recall the definition of the basic subsequence $B(\cdot)$ from Section A.1.1. This modification ensures that elimination of x_t depends only on the coin tosses at indices in $[t-1]$ and the sequence $(x_1, \ldots, x_t)$. This modified algorithm eliminates no more points than the original algorithm because $\text{env}(B(\sigma_{t-1})) \subseteq \text{env}(\sigma_{t-1}) \subseteq \text{env}(\sigma)$—each inequality just uses that the cone of a smaller set of vertices is smaller. Hence, it suffices to prove that the expected number of points eliminated by the modified algorithm is at least $N/2$.

We now use the principle of deferred randomness to analyze this version of the algorithm. Let E_t be the "evolving set" of elements in $\{x_i\}_{1 \le i \le t}$ that are not eliminated by the modified algorithm after it has seen the elements $x_1, x_2 \ldots, x_t$ in that order. Remember that σ_t is the subsequence of $(x_1, \ldots, x_t)$ corresponding to elements that are sampled. We can simulate the process of generating the sets E_t and σ_t in the following way:

1. Start with $E_0, \sigma_0 = \langle\rangle$.
2. At step $1 \le t \le N$, toss a biased coin with success probability $2k/N$. Update

$$\sigma_t = \begin{cases} \sigma_{t-1} \circ x_t, & \text{if coin flip succeeds at t} \\ \sigma_{t-1}, & \text{otherwise.} \end{cases}$$

and

$$E_t = \begin{cases} E_{t-1} \circ x_t, & \text{if } x_t - y \notin \mathrm{env}(B(\sigma_{t-1})) \\ E_{t-1}, & \text{otherwise.} \end{cases}$$

where y is the first element sampled.

Observing that

$$\begin{aligned} |B(\sigma_t)| - |B(\sigma_{t-1})| &= \mathbb{1}[x_t - y \notin \mathrm{env}(B(\sigma_{t-1})) \wedge \text{coin flip succeds at } t] \\ &= (|E_t| - |E_{t-1}|) \cdot \mathbb{1}[\text{coin flip succeds at } t]. \end{aligned}$$

Taking expectations on both sides gives

$$\mathbb{E}[|E_t| - |E_{t-1}|] = \frac{N}{2k} \cdot \mathbb{E}[|B(\sigma_t)| - |B(\sigma_{t-1})|].$$

Summing this over $1 \le t \le N$ gives

$$\mathbb{E}[|E_N|] = \frac{N}{2k} \cdot \mathbb{E}[|B(\sigma)|] \le \frac{N}{2k} \cdot k = \frac{N}{2}.$$ □

A.2 The conic dimension of the Boolean Point Sets

An important special case is that of point sets consisting of Boolean vectors in $\{0,1\}^d$, i.e., of subsets of the hypercube. For completeness, we present a repackaging of the bound on the conic dimension from [KLMZ17]. To begin, we recall the fact that all subset sums of any linearly independent set of vectors are distinct, and extend this fact to the case of conic independence. The bound on the conic dimension of Boolean point sets then follows by a counting argument.

Lemma A.5. *If $\sigma = (y_1, y_2, \ldots, y_{k+1})$ is a conically independent sequence, then all subset sums*

$$\sum_{i \in S} (y_{i+1} - y_i), \quad S \subseteq [k],$$

are distinct.

Proof. Suppose for contradiction that the claim fails, and let k be the smallest integer for which a counterexample exists. The statement is trivial for $k = 1$, so $k \ge 2$. Let $\sigma = (y_1, \ldots, y_{k+1})$ be such a minimal counterexample. Then there exist distinct subsets $S, T \subseteq [k]$ satisfying

$$\sum_{i \in S} (y_{i+1} - y_i) = \sum_{i \in T} (y_{i+1} - y_i).$$

By minimality, the symmetric difference $S \triangle T$ must contain k; otherwise, restricting to the first $k-1$ elements would yield a smaller counterexample. Without loss of generality, assume $k \in S$ and define $S' = S \setminus \{k\}$.

Rearranging gives

$$y_{k+1} - y_k = \sum_{i \in [k-1]} \big(\mathbb{1}_{S'}(i) - \mathbb{1}_T(i)\big)(y_{i+1} - y_i).$$

Substituting into the telescoping sum

$$y_{k+1} - y_1 = (y_{k+1} - y_k) + \sum_{i \in [k-1]} (y_{i+1} - y_i),$$

we obtain

$$y_{k+1} - y_1 = \sum_{i \in [k-1]} \big(1 + \mathbb{1}_{S'}(i) - \mathbb{1}_T(i)\big)(y_{i+1} - y_i),$$

so $y_{k+1} - y_1$ lies in the conic hull of $\{y_2 - y_1, \ldots, y_k - y_{k-1}\}$, contradicting the conic independence of σ. □

Lemma A.6 ([KLMZ17]). *The conic dimension of any subset of $\{0,1\}^d$ is at most $O(d \log d)$. In fact, it is the smallest k such that $2^k > (2k+1)^d$.*

Proof. Let $\sigma = (y_1, \ldots, y_{k+1})$ be a conically independent sequence of Boolean points. By Lemma A.5, the 2^k subset sums

$$\sum_{i \in S} (y_{i+1} - y_i), \quad S \subseteq [k],$$

are all distinct. Each such sum is an integer vector in $[-k, k]^d$, of which there are at most $(2k+1)^d$ possibilities. Hence $2^k \leq (2k+1)^d$, implying that $k = O(d \log d)$. □

A.3 Beyond the Hypercube

In this section, we consider extensions of the above claim for the Boolean hypercube: to the setting of hypergrids, and to getting ε-approximate solutions for other bounded sets.

Hypergrids and Bounded-Precision Solutions A more general version of Lemma A.6 when points belong to $\{0, 1, \ldots, N-1\}^d$ shows that the conic dimension is shown to be at most $O(d \log(Nd))$. The proof is very similar to that of Lemma A.6 above; see [KLMZ17, Lemma 4.2].

ε-Approximate Solutions for Bounded Sets For bounded sets $\mathcal{P} \subset \mathbb{B}_d = \{x \in \mathbb{R}^d; \|x\|_2 \leq 1\}$ it is possible to bound an approximate version of the conic dimension, which leads to an algorithm that returns an ε-approximate point. Given a comparison oracle, the algorithm returns $\widehat{x} \in \mathcal{P}$ such that $\langle w^*, \widehat{x} \rangle \leq \langle w^*, x \rangle + \varepsilon$ for all $x \in \mathcal{P}$.

For the following definition, we will use the notion of the distance: given a set $S \subset \mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, define $\operatorname{dist}(x, S) = \min_{y \in S} \|x - y\|_2$.

Definition A.7. The *ε-approximate conic dimension* of a *set* $\mathcal{Y} \subseteq \mathbb{R}^d$ denoted by $\operatorname{ConicDim}_\varepsilon(\mathcal{Y})$ is the largest integer k for which there exists a length-k sequence $\sigma = (y_1, \ldots, y_k) \in (\mathcal{Y})^k$ of points from $\mathcal{Y}$, such that for each $t \in \{2, \ldots, k\}$ we have

$$\operatorname{dist}(y_t - y_1, \operatorname{env}(\sigma^{1:t-1})) \geq \varepsilon. \tag{7}$$

All the notions previously defined extend to the approximate conic dimension with the natural changes. We can extend the notion of a basic subsequence $B_\varepsilon(\sigma)$ following the same Kruskal-like procedure: we add an element to the basic subsequence if its distance to the previously spanned elements is at least ε.

If $k = \operatorname{ConicDim}_\varepsilon(\mathcal{Y})$ we can apply Algorithm 1 with two modifications: after the first iteration, always include the minimum point from the previous iteration in the sample; in step 2 eliminate all points $x \in \mathcal{P} \setminus \{y\}$ such that $\operatorname{dist}(x - y, \operatorname{env}(\sigma)) < \varepsilon$ obtaining the following results:

Corollary A.8. *If $\|w^*\| \le 1$ and $k = \mathrm{ConicDim}_\varepsilon(\mathcal{Y})$, the the variant of Algorithm 1 described above returns a point $\hat{x}$ such that $\langle w^*, \hat{x}\rangle \le \langle w^*, x\rangle + \varepsilon$ for all $x \in \mathcal{P}$ using $O(k \log k \log |\mathcal{P}|)$ comparisons.*

Proof. Observe that if y is the smallest point in the sampled subsequence then if $\mathrm{dist}(x-y, \mathrm{env}(\sigma)) < \varepsilon$ then let $z \in y + \mathrm{env}(\sigma)$ be a point such that $\|x - z\|_2 \le \varepsilon$. We know that

$$\langle w^*, y\rangle \le \langle w^*, z\rangle = \langle w^*, x\rangle + \langle w^*, z - x\rangle \le \langle w^*, x\rangle + \varepsilon$$

Hence we only eliminate points that are at no better than the point y by more than ε. The modification in step 1 guarantees that the points y selected in each iteration as increasingly better. The remainder of the proof works without any change. □

Finally we bound the approximate conic dimension:

Lemma A.9. *The ε-approximate conic dimension of any set of points $\mathcal{P} \subset \mathbb{B}_d$ is the smallest k such that $2^k(\varepsilon/2)^d > (2k+1)^d$.*

Proof. The proof follows by replacing the pigeonhole argument in Lemma A.6 by a volumetric argument. We again consider the set:

$$\left\{ \sum_{i\in[k]} \beta_i(y_{i+1} - y_i) : \beta_i \in \{0,1\} \right\} \subseteq 2k \cdot \mathbb{B}_d \tag{8}$$

i.e, the ball of radius $2k$ in $\mathbb{R}^d$, which has volume $(2k)^d \cdot \mathrm{Vol}(\mathbb{B}_d)$. Now, for each vector $\beta \in \{0,1\}^k$, consider the ball of radius $\varepsilon/2$ around $\sum_{i\in[k]} \beta_i(y_{i+1} - y_i)$, which have total volume $2^k(\varepsilon/2)^d \cdot \mathrm{Vol}(\mathbb{B}_d)$. Each of those balls is contained in the ball of radius $2k+1$ around zero. If $2^k(\varepsilon/2)^d > (2k+1)^d$ then two of the smaller balls must intersect, so there exist vectors $\beta, \gamma \in \{0,1\}^k$ such that:

$$\sum_{i\in[k]} (\beta_i - \gamma_i)(y_{i+1} - y_i) = z, \tag{9}$$

with $\|z\|_2 < \epsilon$. Let t_0 be the largest index where $\beta_{t_0} \ne \gamma_{t_0}$, and assume $\beta_{t_0} = 0$, $\gamma_{t_0} = 1$. Doing the same manipulations as in Lemma A.6 we obtain:

$$z + y_{t_0+1} - y_1 = \sum_{i\in[t_0-1]} (\beta_i - \gamma_i + 1)(y_{i+1} - y_i) \tag{10}$$

Since $\beta_i, \gamma_i \in \{0,1\}$, the coefficients are non-negative, so

$$\mathrm{dist}(y_{t_0+1} - y_1, \mathrm{env}(y_1, \ldots, y_{t_0})) \le \|z\|_2 < \varepsilon$$

which shows that that ε-approximate conic dimension is at most k. □

B Proofs from Section 3

Proof of Lemma 3.6. The upper bound on the value is immediate, and the non-negativity follows since

$$\sum_{j=1}^{s} (3nM)^{j-1} w_j \ge -M \sum_{j=1}^{s} (3nM)^{j-1} > -(3nM)^s.$$

For any $T \subseteq U$,

$$\sum_{i\in T} g(w_i) = (3nM)^s \cdot |T| + \sum_{j=1}^{s} (3nM)^{j-1} \cdot \left(\sum_{i\in T} w_{i,j} \right).$$

Here $|T| \in [0, n]$ and each $\sum_{i \in T} w_{i,j} \in [-nM, nM]$. Thus, $\sum_{i \in T} g(w_i)$ is a base-$(3nM)$ encoding of the tuple $(|T|, \sum_{i \in T} w_{i,1}, \ldots, \sum_{i \in T} w_{i,s})$, which can be uniquely decoded. Hence, equality of the encoded sums holds iff the corresponding vector sums are equal. □

Lemma 3.7 (Separation for Linear Matroids)**.** *Given a linear matroid $\mathcal{M} = (U, \mathcal{I})$, non-negative integers $c_1, c_2, \ldots, c_n$ for elements in U, and a set of values $Z = \{z_1, \ldots, z_\ell\}$, we can identify a basis $S \in \mathcal{F}$ of $\mathcal{M}$ such that $c(S) \notin Z$ (or decide that no such basis exists) in $\mathrm{poly}(n, \ell, \log M)$ time, where $M = \max_{i \in [n], j \in [\ell]} \{c_i, z_j\}$.*

Proof of Lemma 3.7. Consider the polynomial $P(x)$ defined in (1):

$$P(x) = \det\left(\sum_{i=1}^{n} x^{c_i} v_i v_i^T\right) = \sum_{S \in \mathcal{F}} x^{\sum_{i \in S} c_i} \det(V_S)^2 = \sum_{S \in \mathcal{F}} x^{c(S)} \det(V_S)^2.$$

As established, the set of exponents with non-zero coefficients in $P(x)$ is precisely the set of basis costs, $\{c(S) \mid S \in \mathcal{F}\}$.

Assume there exists a basis S^* such that its cost $c = c(S^*)$ is not in Z. Our goal is to find such a c. Instead of explicitly computing $P(x)$ which has degree can be $O(nM)$, we compute $Q(x) = P(x) \pmod{x^p - 1}$ for a carefully chosen prime p. Indeed, we would like a prime p that does not divide any of the integers in the set $D = \{z_i - z_j \mid i \neq j \in [\ell]\} \cup \{c - z_j \mid j \in [\ell]\}$. If p satisfies this property, then all residues $\{z_i \pmod p\}_{i \in [\ell]}$ are distinct, and $c \pmod p$ is distinct from all of them.

This reduces the original problem to a smaller one: finding an exponent c' in $Q(x)$ (which has degree at most $p-1$) such that $c' \notin \{z_i \pmod p\}_{i \in [\ell]}$. This new problem can be solved in $\mathrm{poly}(n, p)$ time from the warmup proof, which also recovers the witness basis S using the self-reducibility approach.

The final step is to show that a "small" prime p with this property exists and can be found efficiently. The set D contains $O(\ell^2)$ integers. Each integer in D is bounded in magnitude by $O(nM)$ (since $c_i, z_j \leq M$). The product of all these non-zero integers has at most $L := O(\ell^2 \log(nM))$ distinct prime factors.

By a simple pigeonhole argument, a prime p that does not divide any of these integers must exist among the first $L + 1$ primes. The prime number theorem tells us that the t-th prime has magnitude $O(t \log t)$, the smallest such p has a value of $O(L)$. This prime can be found deterministically in $\mathrm{poly}(L)$ time. Hence, the overall runtime of the self-reduction algorithm is $\mathrm{poly}(L) = \mathrm{poly}(n, \ell, \log M)$, as required. Conversely, if the search through all primes in this range yields no such exponent, we can conclude that no basis S with $c(S) \notin Z$ exists. □

C Proofs from Section 4

We begin with a simple observation allowing us to test, for a vertex u and any non-empty set S that is a proper subset of $U - u$, whether a majority of u's neighbors lie inside S.

Observation C.1. Suppose we perform a cut comparison query on the pair S and $S + u$. Then the result of the query is:

$$\mathrm{sign}(|\partial(S)| - |\partial(S + u)|) = \mathrm{sign}\left(|\partial(u, S)| - \frac{1}{2}|\partial(u)|\right). \tag{11}$$

Proof. Indeed,

$$|\partial(S)| - |\partial(S + u)| = |\partial(u, S)| - |\partial(u, U \setminus (S + u))|$$

$$= 2|\partial(u, S)| - |\partial(u)|.$$

Therefore, the sign of the query $(S, S + u)$ indicates whether a majority of u's edges go into S. □

The above observation allows us to reason about neighborhood structure. We now formalize how to use this to isolate individual neighbors.

Lemma C.2 (tipping point)**.** *For $u \in U$, consider a set $\emptyset \neq S \subseteq U \setminus \{u\}$, and a sequence $v_1, v_2, \ldots, v_t$ of vertices in $U \setminus (S \cup \{u\})$. Define $S_i = S \cup \{v_1, \ldots, v_i\}$. Suppose $S_t \neq U - u$ and the comparisons of the queries $(S_0, S_0 + u)$ and $(S_t, S_t + u)$ differ. Then there exists an index $i \in [t]$ such that*

$$\text{sign}(|\partial(S_{i-1})| - |\partial(S_{i-1} + u)|) < \text{sign}(|\partial(S_i)| - |\partial(S_i + u)|). \tag{12}$$

The corresponding vertex v_i must be a neighbor of u, and can be identified using $O(\log t)$ queries and $\widetilde{O}(n)$ time via binary search.

Proof. By Observation C.1, the $\text{sign}(|\partial(S)| - |\partial(S + u)|)$ is monotonic in S, since the quantity $|\partial(u, S)|$ increases as S grows. Therefore, if the sign changes between S_0 and S_t, then by monotonicity, it must strictly increase at some index i, which proves (12). Moreover, using the equality in (11) again, (12) can be rewritten as

$$|\partial(u, S_{i-1})| < |\partial(u, S_i)|, \tag{13}$$

and hence u is adjacent to the vertex v_i. Finally, a binary search over the chain finds such an i using $O(\log t)$ queries. The binary search performs $O(\log t)$ queries, each requiring $O(n)$ time to write bit vectors, for a total of $O(n \log t) = \widetilde{O}(n)$. The additional binary search logic takes $O(\log t)$ time. □

We now apply these ideas to construct balanced sets around each vertex.

Lemma C.3 (median sets)**.** *For any vertex u with $\partial(u) \neq \emptyset$, and an ordering $v_1, \ldots, v_{n-1}$ of the vertices in $U - u$, there exist disjoint sets $S_u^-, S_u^+ \subseteq U \setminus \{u\}$ that are a prefix and a suffix of the sequence such that*

$$|\partial(u, S_u^-)| = |\partial(u, S_u^+)| = \left\lceil \frac{|\partial(u)|}{2} \right\rceil - 1. \tag{14}$$

These sets can be found using $O(\log n)$ queries and $\widetilde{O}(n)$ time.

Proof. Let $L_j = \{v_1, v_2, \ldots\ldots, v_j\}$ and $R_j = \{v_j, v_{j+1}, \ldots, v_{n-1}\}$ for $j \in [n-1]$. First, check if $\partial(v_1) > \partial(v_1 + u)$ in which case, we can conclude $S_u^- = \emptyset, S_u^+ = \{v_2, v_3, \ldots, v_{n-1}\}$ because $\{u, v_1\}$ has to be the only edge using Observation C.1. Similarly for v_{n-1}. So from now on, assume that $\partial(v_1) \leq \partial(v_1 + u)$ and $\partial(v_{n-1}) \leq \partial(v_{n-1} + u)$. Equivalently, this can also be written as $\partial(R_2) \geq \partial(R_2 + u), \partial(L_{n-2}) \geq \partial(L_{n-2} + u)$.

In what follows, we will first describe how to compute S_u^- and then use a similar argument to compute S_u^+. If $\partial(v_1) = \partial(v_1 + u)$, then again we can simply conclude $S_u^- = \emptyset$ because this implies that $|\partial(u)| = 2$ and $\{u, v_1\}$ is an edge. Otherwise, if $\partial(v_1) < \partial(v_1 + u)$, we can search for S_u^- as a tipping point for the chain defined by the sequence $v_1, v_2, \ldots, v_{n-2}$ using Lemma C.2 because $\partial(v_1) < \partial(v_1 + u)$ and $\partial(v_1, \ldots, v_{n-2}) \geq \partial(v_1, \ldots, v_{n-2} + u)$. By the arguments in Lemma C.2, there exists $i \in [n-2]$ such that $v_i \in N(u)$ and

$$|\partial(u, L_{i-1})| < \tfrac{1}{2}|\partial(u)|, \quad |\partial(u, L_i)| \geq \tfrac{1}{2}|\partial(u)|. \tag{15}$$

This implies that $S_u^- := L_{i-1}$ satisfies $|\partial(u, S_u^-)| = \lceil \frac{|\partial(u)|}{2} \rceil - 1$. We can use the same argument to get S_u^+. Moreover, since S_u^- and S_u^+ are prefix and suffix of the same ordering, and contain no more than half the neighbors of u, they must be disjoint. The proof makes two calls to Lemma C.2, one for each ordering, with $O(n)$ preprocessing. Each call takes $\widetilde{O}(n)$ time, so the total running time is $\widetilde{O}(n)$. □

C.1 Proof of Structural Primitives Lemma

We can now prove the main structural result Theorem 4.1, which we restate for convenience:

Theorem 4.1 (Structural Primitives)**.** *When $n \geq 4$, for any vertex $u \in U$:*

(i) Given a set of k vertices $A = \{v_1, \ldots, v_k\} \subseteq U \setminus \{u\}$, the neighbors of u in A can be determined using $k + O(\log n)$ queries and $\widetilde{O}(n + k)$ time.

(ii) Given an ordered subset $T = (v_1, \ldots, v_t) \subseteq U \setminus \{u\}$, either the neighbor $v_i \in N(u)$ of u with the smallest index $i \in [t]$ (or a certificate that $N(u) \cap T = \emptyset$) can be found using $O(\log n)$ queries and $\widetilde{O}(n)$ time.

Before we prove the theorem, we first prove a primitive that shows how to identify if a vetrex is isolated using constant queries when $n \geq 4$:

Lemma C.4 (Home Alone)**.** *For any three vertices $u, v, w \in U$, vertex u is isolated iff*

$$|\partial(v)| = |\partial(u + v)|, |\partial(w)| = |\partial(u + w)|, |\partial(v + w)| = |\partial(u + v + w)|. \tag{16}$$

Proof. The three equalities are trivial when u is isolated. Given the equalities, we observe that they can be written equivalently as

$$|\partial(u, v)| = \tfrac{1}{2}|\partial(u)|, |\partial(u, w)| = \tfrac{1}{2}|\partial(u)|, |\partial(u, v + w)| = \tfrac{1}{2}|\partial(u)|. \tag{17}$$

Using the fact that $|\partial(u, v + w)| = |\partial(u, v)| + |\partial(u, w)|$ gives $|\partial(u)| = 0$. □

Proof of Theorem 4.1. We first check if $\partial(u) = \emptyset$ using Lemma C.4. If $\partial(u) = \emptyset$, there are no neighbors of u in A in part (i) and we can certify that $N \cap T = \emptyset$ in part (ii). Assume $\partial(u) \neq \emptyset$ from here on.

In proving part (i), and determining which of $v_1, \ldots, v_k \in U \setminus \{u\}$ are neighbors of u, we begin by computing the disjoint sets $S_u^-, S_u^+ \subseteq U \setminus \{u\}$ with exactly $\lceil \frac{|\partial(u)|}{2} \rceil - 1$ neighbors of u; this takes $O(\log n)$ queries and $\widetilde{O}(n)$ time due to Lemma C.3.

For each v_i we choose S to be one of S_u^- or S_u^+ in which v_i does not appear—say $S := S_u^+$. Since $|\partial(u, S)| < \frac{1}{2}|\partial(u)|$, we know from Observation C.1 that $\text{sign}(|\partial(S)| - |\partial(S + u)|) = -1$. We then add v_i to S and ask for $\text{sign}(|\partial(S + v_i)| - |\partial(S + v_i + u)|)$. Since $\partial(S, u)$ is just one edge short of reaching half the degree of u, this sign increases if and only if $v_i \in N(u)$. Thus, each test requires one query, and the total query complexity is $k + O(\log n)$. After computing the median sets S_u^- and S_u^+ in $\widetilde{O}(n)$ time, we partition the vertices $v_1, \ldots, v_k$ into two groups based on whether each v_i lies in S_u^- or S_u^+; this takes $O(k)$ time using bit vector lookups. For each group, we fix the corresponding set $S := S_u^+$ or S_u^- and write its bit vector once in $O(n)$ time. Each query $(S + v_i, S + v_i + u)$ is then constructed in $O(1)$ time via bit flipping. The total running time is $\widetilde{O}(n + k)$.

This proves Theorem 4.1(i).

For part (ii) of the lemma, consider the ordered sequence $T = (v_1, \ldots, v_t) \subseteq U \setminus \{u\}$ and extend it to a total ordering of $U \setminus \{u\}$ by appending the remaining vertices arbitrarily, forming the sequence $(v_1, \ldots, v_t, v_{t+1}, \ldots, v_{n-1})$. Define suffixes $R_k := \{v_k, \ldots, v_{n-1}\}$ for each k; by Lemma C.3, there exists some k such that

$$|\partial(u, R_k)| = \left\lceil \frac{|\partial(u)|}{2} \right\rceil - 1, \tag{18}$$

and such a suffix $S := R_k$ can be found using $O(\log n)$ queries and $\widetilde{O}(n)$ time via Lemma C.3. To verify that $|\partial(u, T)| > 0$, consider the sequence $(v_{t+1}, \ldots, v_{n-1}, v_1, \ldots, v_t)$ with T placed all the way at the end. The suffix median of this ordering computed using Lemma C.3 is the same as S iff $|\partial(u, T)| > 0$. Otherwise, define the nested sequence

$$S_0 := S, \tag{19}$$
$$S_i := S \cup \{v_1, \ldots, v_i\} \quad \text{for } i = 1, \ldots, \min(t, k). \tag{20}$$

(Recall that $|T| = t$ and $|S| = n - k + 1$.) We have:

$$|\partial(u, S_0)| = |\partial(u, S)| = \left\lceil \frac{|\partial(u)|}{2} \right\rceil - 1, \tag{21}$$
$$|\partial(u, S_{\min(t,k)})| = |\partial(u, S)| + |\partial(u, T \setminus S)| \geq \tfrac{1}{2}|\partial(u)|. \tag{22}$$

To justify the inequality in (22): if $|\partial(u, T)| > 0$, then $|\partial(u, T \setminus S)| > 0$, because otherwise if $|\partial(u, T \setminus S)| = 0$, then $N(u) \subseteq S$. But this would imply $|\partial(u, S)| = |\partial(u)|$, contradicting the fact that less than half the edges incident to u go into S (by the choice of S). Hence, $T \setminus S$ must contain a neighbor of u.

By Lemma C.2, there exists an index i such that the sign of the query $(S_{i-1}, S_{i-1} + u)$ is strictly smaller than that of $(S_i, S_i + u)$. The corresponding vertex v_i is the first vertex in T adjacent to u, and binary search over $i \in [\min(t, k)]$ finds it using $O(\log n)$ queries and $\widetilde{O}(n)$ time. □

Lemma 4.3 (Edge Extraction)**.** *When $n \geq 4$, for any vertex u, a subset $T \subseteq U - u$ of vertices, and any integer k, we can extract $r = \min(k, \partial(u, T))$ edges from $\partial(u, T)$ using $O(r \log n)$ queries and $\widetilde{O}(n)$ time.*

Proof. We assume that $|\partial(u, T)| \leq \left\lfloor \frac{1}{2}|\partial(u)| \right\rfloor + 1$; otherwise, we split T into $T_+ := T \cap S_u^+$ and $T_- := T \cap S_u^-$ using the median sets from Lemma C.3, and recursively extract edges from each. Both subsets satisfy the same degree bound, and the total query and runtime complexity remains the same.

Let S be a median set disjoint from T, which exists by construction and can be found using $O(\log n)$ queries and $\widetilde{O}(n)$ time. Let $T = \{v_1, \ldots, v_t\}$, and define the nested sets $S_j := S \cup \{v_1, \ldots, v_j\}$. We maintain two bit vectors: one for the fixed set S, and one for the growing prefix of T.

In each iteration, we find the smallest index j such that the sign of the query $(S_{j-1}, S_{j-1} + u)$ is strictly smaller than that of $(S_j, S_j + u)$, indicating that $v_j \in N(u)$. We locate this index using the doubling version of binary search. Once v_j is found, we report the edge $\{u, v_j\}$, remove $v_1, \ldots, v_j$ from T, and repeat. Since we extract at most r edges, we perform at most r such searches, each requiring $O(\log n)$ queries, for a total of $O(r \log n)$ queries overall.

Running time. Constructing the median set S takes $\widetilde{O}(n)$ time. To construct the query sets, we reuse the bit vector for S and build each query by appending a prefix $\{v_1, \ldots, v_i\} \subseteq T$. In each iteration, the doubling search explores prefixes of geometrically increasing size until it finds a

neighbor at index j. The total size of all prefixes queried in that iteration is $O(j \log t)$, so the time spent constructing queries is $O(j \log t)$. Deleting $v_1, \ldots, v_j$ from T also takes $\widetilde{O}(j)$ time. The values $j_1, \ldots, j_r$ across all r iterations sum to at most n as these are gaps between the edge indices, so the overall total time taken is $O\left(\sum_{i=1}^{r} j_i \log n\right) = \widetilde{O}(n)$. □

C.2 Cut sparsifiers using $\widetilde{O}(n)$ queries

Using Lemma 4.4, we show that we can implement the sparsifier construction from [RSW18, Algorithm 3.4] and the min-cut computation from [RSW18, Algorithm 4.1] using $\widetilde{O}(n/\varepsilon^2)$ cut comparison queries. The mentioned algorithms and relevant theorems from [RSW18] are mentioned verbatim below:

Algorithm 2: (Approximating Edge Strengths and Sampling a Sparsifier H)

2.1 Input: An accuracy parameter ε, and a cut-query oracle for graph G.

1. Initialize an empty graph H on n vertices and $G_0 \leftarrow G$.
2. For $j = 0, \ldots, \log n$, set $\kappa_j = n2^{-j}$ and:
   (a) Subsample G'_j from G_j by taking each edge of G_j with probability
   $$q_j = \min\left(\frac{100 \cdot 40 \cdot \ln n}{\kappa_j}, 1\right).$$
   (b) In each connected component of G'_j:
      i. While there exists a cut of size $\leq q_j \cdot \frac{4}{5}\kappa_j$, remove the edges from the cut. Let the connected components induced by removing the cut edges be $C_1, \ldots, C_r$.
      ii. For every $i \in [r]$ and every edge (known or unknown) with both endpoints in C_i, set the approximate edge strength $k'_e := \frac{1}{2\kappa_j}$ (alternatively, subsample every edge in $C_i \times C_i$ with probability $2q_j/\varepsilon^2$ and add it to H with weight $\varepsilon^2/2q_j$).
      iii. Set G_{j+1} as the graph obtained by contracting C_i for each $i \in [r]$ in G_j.

Output: The edge strength approximators $\{k'_e\}$ (or the sparsifier H).

Theorem C.5 (Theorem 3.5 from [RSW18])**.** *For each edge $e \in G$, the approximate edge strength given in Algorithm 2 satisfies $\frac{1}{4}k_e \leq k'_e \leq k_e$. Furthermore, the algorithm requires $\widetilde{O}(n/\varepsilon^2)$ oracle queries to produce a sparsifier H with the following properties:*

- *H has $O(n \log n/\varepsilon^2)$ edges.*
- *The maximum weight of any edge e in H is $O(\varepsilon^2 k_e/\log n)$.*
- *Every cut in H approximates the corresponding cut in G within a $(1 \pm \varepsilon)$ factor.*

Theorem C.6 (Theorem 4.2 from [RSW18])**.** *Algorithm 3 uses $\widetilde{O}(n)$ queries and finds the exact minimum cut in G with high probability.*

Proof of Theorem 1.5. For each $0 \leq j \leq \log n$, step 2(a) of Algorithm 2 can be implemented using Lemma 4.4 with $O(|E_j| \cdot q_j \cdot \log n) + \widetilde{O}(n)$ comparison queries in expectation; which is shown to be $\widetilde{O}(n)$ in the proof of Theorem C.5. The runtime bound of the same step is $\widetilde{O}(n^2)$. Step 2(b)(ii) of Algorithm 2 also takes $\widetilde{O}(n/\varepsilon^2)$ queries and $\widetilde{O}(n^2)$ time because the number of edges sampled and added to H is shown to be $\widetilde{O}(n/\varepsilon^2)$ (see proof of Theorem 4.2, [RSW18]) and this sub-sampling can be done again using Lemma 4.4. In Algorithm 3, since step (1) is only used to compare the singleton

Algorithm 3: Simpler global Min Cut with $\widetilde{O}(n)$ oracle queries

3.1 Input: Oracle access to the cut values of an unweighted simple graph G.

1. Compute all of the single-vertex cuts.
2. Compute a sparsifier H of G using Algorithm 2 with small constant ε.
3. Find all non-singleton cuts of size at most $(1+3\varepsilon)$ times the size of the minimum cut in H, and contract any edge which is not in such a cut. Call the resulting graph G'.
4. If the number of edges between the super-vertices of G' is $O(n)$, learn all of the edges between the super-vertices of G' and compute the minimum cut.

Output: Return the best cut seen over the course of the algorithm.

cuts to output the minimum cut seen by the algorithm, we can implement this using comparison in $O(n)$ queries and time. In step (3), all the approximate cuts can be computed in $\widetilde{O}(n^2)$ time using [HHS24]. In step (4), all the edges of G' can be learned using Lemma 4.4 with $p = 1$. The proof of theorem C.6 shows that the number of edges in G' is at most $O(n)$ with high probability. This implies that all the steps in the algorithms take $\widetilde{O}(n/\varepsilon^2)$ queries and $\widetilde{O}(n^2)$ time. There are at most $\log n$ steps in any algorithm which proves the desired bounds. □

C.3 Weighted Min-Cut: Special Cases and Stronger Oracles

In the weighted setting, where the edges of the graph $G = (V, E)$ have unknown non-negative weights, the techniques developed for the unweighted case (where weights are 0 or 1; see Section 4) no longer apply. Even when the weights are restricted to integers in $[0, B]$, little is known.

C.3.1 Weighted regular graphs

We consider the case when the graph satisfies the *weighted regularity* condition, i.e., $|w^*(\partial u)|$ is identical for all $u \in V$. Under this assumption,

$$w^*(\{u,v\}) = \tfrac{1}{2}\big(w^*(\partial u) + w^*(\partial v) - w^*(\partial\{u,v\})\big), \tag{23}$$

and hence the relative order of edge weights $w^*_{\{u,v\}}$ is simply the reverse of the order of the cuts $\partial\{u,v\}$ sorted by their cut values. Since these comparisons can be performed using only cut queries, we can recover the rank order of all edge weights.

Because the weights are bounded by B, there are at most $\ell := n^2B + 1$ distinct cut values, corresponding to ℓ ordered *buckets* of edges. We wish to classify these buckets into $O(\log B)$ weight ranges:

$$0, [2^0, 2^1), [2^1, 2^2), \ldots, [2^{\lfloor \log_2 B \rfloor}, 2^{\lfloor \log_2 B \rfloor + 1}).$$

A naive assignment of each bucket to one of these $O(\log B)$ classes would yield $(O(\log B))^\ell$ possibilities. However, since the buckets are ordered, we can choose the $O(\log B)$ boundary points separating consecutive classes along this sequence. Thus, the number of possible partitions is $O(\ell^{O(\log B)})$, and one such partition corresponds to a 2-approximation of the true edge weights.

Using these approximate weights, we enumerate all 2-approximate cuts. By Karger's result, there are only $n^{O(1)}$ such cuts, and these can be enumerated in $\widetilde{O}(n^2)$ time (see [HHS24]). Since our guessed weights form a 2-approximation, the true minimum cut (which is a 2-approximate cut

under the guessed weights) appears among them. Comparing all enumerated cuts across all weight guesses and returning the smallest yields a total running time and number of cut comparisons of $O((nB)^{O(\log B)})$.

The guessing step can be extended to the more general case where there are r distinct weighted degrees. In this case, we first classify the vertices into $V_1, \ldots, V_r$ according to their degrees (this can be done by comparing their degree cuts). Each edge then lies between some pair (V_i, V_j), and we accordingly classify all edges into $r + \binom{r}{2}$ classes $E_{\{i,j\}}$.

Observe that for two edges $\{u_1, v_1\}, \{u_2, v_2\} \in E_{\{i,j\}}$ within the same class, their weights can be compared by comparing the cuts $\partial\{u_1, v_1\}$ and $\partial\{u_2, v_2\}$ using Equation (23). Hence, we apply the same guessing procedure as before independently within each class, resulting in a total of $O(\ell^{O(r^2 \log B)})$ guesses. We can then proceed exactly as in the uniform-degree case. Consequently, the overall running time and number of cut comparisons are $O((nB)^{O(r^2 \log B)})$.

C.3.2 Stronger Oracles

An alternative to imposing structural assumptions is to consider stronger oracle models. We highlight two natural extensions:

1. **Comparing marginals:** Instead of comparing the objective values of feasible sets, i.e., $w(S)$ and $w(T)$ for $S, T \in \mathcal{F}$, we allow comparisons between *marginal differences* such as $w(S) - w(S')$ and $w(T) - w(T')$ for $S, S', T, T' \in \mathcal{F}$.
2. **Comparing arbitrary sets:** We relax the restriction that comparisons must involve only feasible sets, and instead allow comparing $w(S)$ and $w(T)$ for any $S, T \subseteq U$, while the optimization goal remains finding $\arg\min_{S \in \mathcal{F}} w(S)$.

In both of these stronger models, the minimum cut can be found efficiently, even in the weighted setting with arbitrary non-negative weights. The key observation is that the Nagamochi–Ibaraki algorithm [NI92] for minimum cuts only requires comparisons of *differences* of cut values. More specifically, at each step it only needs to find

$$\arg\min_{u \in V \setminus S} \big(w(\partial(S+u)) - w(\partial u)\big),$$

where the graph at this point is a contraction of the original, so the current vertex set V may consist of disjoint subsets of the original vertices (which poses no issue). The algorithm then outputs the best cut among a small collection of candidate cuts, which can be selected using cut comparisons.

Notably, the identity

$$\arg\min_{u \in V \setminus S} \big(w(\partial(S+u)) - w(\partial u)\big) = \arg\min_{u \in V \setminus S} w(\partial(S,u))$$

implies that the Nagamochi–Ibaraki algorithm can also be implemented under the second, arbitrary-set comparison model. Finally, the Nagamochi–Ibaraki algorithm has a natural extension to general non-negative symmetric submodular functions, as shown by [Que98].

C.4 Illustrative Examples

C.4.1 Separations between Comparison and Value Queries

In the weighted setting, consider taking two disjoint cycles C_1 and C_2 of $k = n/2$ vertices and with unit-weight edges, and adding in a perfect matching of size k between them, with edges of cost ε. This is the *circular ladder graph*, with cross-edge weight ε.

Claim C.7. *There exist non-adaptive algorithms making $O(n^2)$ value queries to sets of size one and two that can reconstruct any graph, but for any non-adaptive algorithm in the comparison-query model making queries to sets of constant size (or even $\ll n/2$ size), there exist weighted n-vertex graphs G^+, G^- which cannot find the minimum cut.*

Proof. Take the circular ladder graph with $\varepsilon = \frac{2}{k-1} \pm \gamma$ for some tiny $\gamma > 0$; the sign in front of the γ will determine whether the min-cut is a single-vertex cut with cost $2+\varepsilon$, or the bisection (C_1, C_2) with cost $k\varepsilon$. However, we cannot distinguish these two cases if we compare sets S, T of size $\ll n/2$. (This is true even if we know the edges of the graph and the entire construction, except the sign in front of γ. This is in contrast to the value query setting, where asking the value of $f(S)$ for sets of size 1 and 2 allows us to completely reconstruct the graph. □

Claim C.8. *There exist non-adaptive algorithms making $O(n^2)$ value queries to reconstruct the graph, but for any non-adaptive algorithm in the comparison-query model, there exist weighted n-vertex graphs G^+, G^- which require $\exp(n)$ comparison queries to have high success probability.*

Proof. Again, take the circular ladder graph, and rename the vertices uniformly at random. Assume that the cross-edge weight is $\varepsilon = \frac{2}{k-1} \pm \gamma$ for some tiny $\gamma > 0$; the sign in front of the γ will determine whether the degree cut or the partition (C_1, C_2) is optimal. Again, for value queries we can non-adaptively reconstruct the graph using $O(n^2)$ queries. But for comparison queries, the only way to learn the sign in front of γ would be to query the correct partition with a degree cut. Now if the queries are non-adaptive, and the vertices of the graph have been uniformly and randomly renumbered, the number of non-adaptive queries would have to be exponential in n in order to have a high probability of success. □

C.4.2 Non-Reconstructable Graphs

The unweighted graph pairs $(K_2, \bar{K_2})$ and $(K_3, \bar{K_3})$ are not distinguishable, since essentially the only feasible sets S are singleton sets (recall that by symmetry, $f(S) = f(\bar{S})$), and all of them have the same cut value. However, as we showed in Theorem 1.6, we can recover all unweighted simple graphs other than $K_2, \bar{K_2}, K_3, \bar{K_3}$ using comparison queries.

In contrast, many connected weighted graphs cannot be recovered using just comparison queries. This is not surprising: we cannot distinguish between a graph with weights w_e, and those with weight αw_e for all $\alpha > 0$. But there are other kinds of barriers apart from a simple scaling of weights—here is an example. Given any graph $G = (V, E)$ with unit weight edges, add another spanning tree $T = (V, F)$ on the same vertices, and give the i^{th} edge $e_i \in F$ a weight of $w_T(e_i) = n^2 \cdot 2^i$. This weight function ensures that the F-weights of all cuts are distinct. Moreover, for a set $S \subseteq V$, $w(\partial S) = |\partial_G S| + w_T(\partial_T S)$. Since the w_T weights are scaled by n^2 (and hence much larger than the contribution due to the unit-weight edges), we know that

$$w(\partial S) > w(\partial S') \iff w_T(\partial_T S) > w_T(\partial_T S)$$

and hence we cannot infer the structure of the unit-weight edges.

C.4.3 Finding Heaviest Edge Incident to a Vertex

We can show that comparison queries do not allow us to identify, for a vertex v, the heaviest edge adjacent to it. Finding this edge would be useful, e.g., in the min-cut algorithm of [SW94]. Indeed, consider the 4-vertex graph $\{a, b, c, d\}$ and the 6 edge weight variables. Here are two universes:

Edge Weight	Universe 1 ($w_{ab} > w_{ac}$)	Universe 2 ($w_{ab} < w_{ac}$)
w_{ab}	**1000.01**	**999.99**
w_{ac}	**999.99**	**1000.01**
w_{ad}	**10**	**10**
w_{bc}	100	100
w_{bd}	50	50
w_{cd}	1	1

In both scenarios, $w_{ad} = 10$ is much less than w_{ab} and w_{ac} (which are both $\sim$1000). However, a calculation shows that the order of the cuts is **identical** in both universes:

$$f(\{a,d\}) > f(\{a\}) > f(\{a,b\}) > f(\{b\}) > f(\{a,c\}) > f(\{c\}) > f(\{d\}).$$

Hence, given this specific ordering of cuts, we cannot distinguish between Case 1 (where (a,b) is the max-weight edge adjacent to a) or Case 2 (where (a,c) is).

D Matroids, Matchings, and Paths

In this section, we give efficient comparison-based algorithms to optimize over matroids, matroid intersections, and shortest paths.

D.1 Matroid Bases

We now give our algorithm to compute a minimum-weight basis of a matroid $\mathcal{M} = (U, \mathcal{I})$ with $|U| = n$ elements, where we are only allowed to compare matroid bases.

Our main result for matroid bases is the following:

Theorem D.1 (Matroid Bases)**.** *There is an algorithm outputs the minimum-weight basis of a matroid on n elements using $O(n \log n)$ comparison queries in $\widetilde{O}(n^2)$ time.*

Proof. First, let us consider graphical matroids, where the bases are spanning trees of a graph. Given any graph with edge set E, we can first partition the graph into its 2-vertex-connected components $E_1, \ldots, E_k$ (note that these are the edge sets of the components); the edge set of the minimum-weight spanning tree (MST) for E is the union of those for the E_i's, so it suffices to focus on a 2-connected component induced by the edges in some E_i. We claim we can simulate Kruskal's algorithm for MST in each of these components, which just compares weights of edges: indeed, if this algorithm compares edges $e, e' \in E_i$, we can find a base B containing e, such that $B' := (B \setminus e) \cup e'$ is also a base. (We defer the proof, since it follows from the result below.) Now comparing e, e' has the same answer as comparing B, B'.

The same argument holds for arbitrary matroids, where we recall that the notion of 2-vertex connectivity in graphs extends to matroids [Oxl11]:

Lemma D.2. *For any matroid $\mathcal{M} = (\mathcal{I}, U)$, let $\mathcal{B}(\mathcal{M})$ and $\mathcal{C}(\mathcal{M})$ be the collection of all bases and circuits in the matroid.*

1. *If elements $e, e' \in E$ share a circuit $C \in \mathcal{C}(\mathcal{M})$, then there exist bases $B, B' \in \mathcal{B}(\mathcal{M})$ such that $B' = B + e' - e$. Call such pair of elements e, e' **comparable**.*
2. *The binary relation $\gamma_{\mathcal{M}}$ on the elements of the matroid such that $(e, e') \in \gamma_{\mathcal{M}}$ iff either $e = e'$ or e, e' are comparable is an equivalance relation.*

Proof. For the first part, let B be the basis that extends the independent set $C - e'$. This means that the fundamental circuit in $B + e'$ is C implying that $B' := B + e' - e$ is also a basis. The second part is proved in [Oxl11, Proposition 4.1.2]. The equivalence classes in lemma D.2(2) are called the *connected components* of $\mathcal{M}$. □

Now we can run the analog of Kruskal's greedy algorithm for matroids on the connected components of $\mathcal{M}$; this uses $O(|U_i| \log |U_i|)$ element (and hence base) comparisons for each component; summing over all components gives us $O(|U| \log |U|)$, as claimed.

We now give details on efficient algorithms for decomposition of the matroid into connected components and obtaining two bases B, B' such that $B \Delta B' = \{x, y\}$ for every pair x, y compared by the algorithm, and overall running time bounds: The running time consists of the time taken to decompose the matroid into its connected components, and obtaining two bases B, B' such that $B \Delta B' = \{x, y\}$ for every pair x, y compared by the algorithm. For this, we use the results from [Kro77] that connect the *basis exchange graph* with the connected components of the matroid. The following are a definition and a lemma:

For any basis B of $\mathcal{M}$, the basis exchange graph $H(B)$ is defined as the bipartite graph with color classes B and $U \backslash B$. For $x \notin B, y \in B$, we have $\{x, y\} \in H$ iff $B + x - y$ is also a basis. The following lemma connects this exchange graph with matroid connectivity:

Lemma D.3 (Section 6, [Kro77])**.** *For any basis $B \in \mathcal{M}$,*

1. *The nodes in the connected components (in the graphic sense) of $H(B)$ are the connected components (in the matroid sense) of matroid $\mathcal{M}$.*
2. *For x, y in the same connected component of $H(B)$ (or equivalently $\mathcal{M}$) such that $x \notin B$ and $y \in B$, if P is the shortest path connecting x and y in $H(B)$, then $B' := B \Delta P$ is a basis such that $B' + y - x$ is a basis.*

Given Lemma D.3, the matroid decompostion step requires $O(n^2)$ matroid independence calls (in fact, the calls are always about whether a set is a basis or not) to build the exchange graph using Lemma D.3 part 1 and $O(n^2)$ time to identify connected components.

Next, to simulate any pairwise comparison between elements x, y, we obtain the two bases $B \ni x, B' \ni y$ such that $B \Delta B' = \{x, y\}$ using part 2 of Lemma D.3. The implementation of Kruskal's algorithm requires sorting the elements in each connected component which requires $O(c_i \log c_i)$ comparisons in each component with c_i elements. Each such comparison takes $O(n)$ time to find the path and write down the bases to input to the comparison oracle. Adding everything, we get the $O(n^2 \log n)$ time bound as claimed. □

D.2 Matroid Intersection

We now turn to finding *common independent sets* $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ between two matroids $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$. Our main result in this case is the following:

Theorem D.4 (Matroid Intersection)**.** *Let $\mathcal{M}_1 = (U, \mathcal{I}_1)$ and $\mathcal{M}_2 = (U, \mathcal{I}_2)$ be two matroids defined on a ground set U containing n elements. Then, there is an algorithm that outputs the minimum-weight set that is in both $\mathcal{I}_1, \mathcal{I}_2$ using $O(n^4)$ comparison queries in $O(n^4)$ time.*

To prove this, we show that Edmond's classical weighted matroid intersection algorithm, as presented in [Sch03, Section 41.3], can be implemented using only comparisons of common independent sets. As in the previous section, we first consider the simpler bipartite matching case, where the shortest

augmenting paths algorithm constructs extremal matchings via shortest paths in an exchange graph. Our key observation is that these shortest-path steps—e.g., via Bellman–Ford—can themselves be implemented using only comparisons between matchings.

Formally, let $G = (V, E)$ be a bipartite graph with color classes $V = L \cup R$. Given an *extremal matching* M of size k (i.e., a min-cost matching with k edges), we describe a primitive to obtain an extremal matching M' of size $k + 1$.

The *M-exchange graph* D orients matching edges from R to L and non-matching edges from L to R in G. Let L' and R' be the unmatched vertices in L and R. An *M-augmenting path* is a directed path in D from a vertex in L' to one in R'. Define the edge length function $\ell : E \to \mathbb{R}$ as: $\ell(e) := -w(e)$ if $e \in M$, and $w(e)$ otherwise. The following result is standard (see, e.g., Theorem 3.5, Proposition 1 in Section 3.5 of [Sch90]).

Lemma D.5. *Let M be an extremal matching. Then:*

1. *The exchange graph D has no negative-length cycles.*
2. *If P is a minimum-length M-augmenting path, then $M' := M \triangle P$ is also extremal.*

By iteratively applying Lemma D.5, we construct extremal matchings of increasing size, from size 0 to $n/2$, and return the one with minimum total weight. Moreover, to implement this using only comparisons between matchings, we describe a variant of the Bellman-Ford algorithm. For each $u \in L$, let $p_u^{(k)}$ be the shortest M-alternating path from L' to u using at most k matching edges. For matched $u \in L$, let $u' \in R$ be its partner. We initialize:

$$p_u^{(1)} = \begin{cases} \emptyset & \text{if } u \in L', \\ s \to u' \to u & \text{for } s = \arg\min_{s \in L'} \ell(s \to u' \to u). \end{cases}$$

Moreover, we recursively define $p_u^{(k+1)} = \min_{v \in L} \left(p_u^{(k)},\ p_v^{(k)} \to u' \to u \right)$, where the min operator returns the path with minimum total length $\ell(p)$, ties broken arbitrarily. Each such path P yields a matching $M \triangle P$, and satisfies $\ell(P) = w(M \triangle P) - w(M)$, so comparing path lengths reduces to comparing weights of the resulting matchings.

To reach a free vertex $t \in R'$, compute: $p_t = \min_{v \in L} \left(p_v^{(n)} \to t \right)$, which again defines an alternating path. Hence, the full algorithm can be implemented using only comparisons between matchings.

This insight extends naturally to the matroid intersection setting, where shortest augmenting paths must also be minimal (i.e., having fewest arcs among paths of equal weight), a property standard algorithms like Bellman–Ford already ensure. We show that the path-finding steps can again be carried out using only comparisons of common independent sets. The details appear below:

D.3 Generalization to Matroid Intersection

We now extend the concepts from bipartite matching to matroid intersection. Let $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ be matroids over a common ground set E, and let $Y \in \mathcal{I}_1 \cap \mathcal{I}_2$ be a common independent set. Define the *exchange graph* $H(\mathcal{M}_1, \mathcal{M}_2, Y)$ as a directed bipartite graph with color classes Y and $E \setminus Y$. For $y \in Y$ and $x \in E \setminus Y$:

$$(y, x) \in H \iff Y - y + x \in \mathcal{I}_1,$$
$$(x, y) \in H \iff Y - y + x \in \mathcal{I}_2.$$

Let $H(\mathcal{M}_1, Y), H(\mathcal{M}_2, Y)$ be the subgraphs of H with edges corresponding to $\mathcal{M}_1, \mathcal{M}_2$ respectively.

We know the following lemma (see [Sch90, Section 10.4]):

Lemma D.6 (Matching lemmas)**.** *For $Y \in \mathcal{I}_1$, let $H_1 = H(\mathcal{M}_1, Y)$.*

(a) If $Z \in \mathcal{I}_1$ with $|Y| = |Z|$, then H_1 contains a perfect matching on $Y \Delta Z$.
(b) If $Z \subseteq E$ such that H_1 contains a unique perfect matching on $Y \Delta Z$, then $Z \in \mathcal{I}_1$.

Let $X_1 := \{x \in E \setminus Y : Y + x \in \mathcal{I}_1\}$ and $X_2 := \{x \in E \setminus Y : Y + x \in \mathcal{I}_2\}$. Given a weight function $w : E \to \mathbb{R}$, define the length of any path (or cycle) P in H by:

$$\ell(P) := -\sum_{e \in P \cap Y} w_e + \sum_{e \in P \setminus Y} w_e.$$

A common independent set $Y \in \mathcal{I}_1 \cap \mathcal{I}_2$ is *extreme* if it minimizes weight among all common independent sets of the same size. We know the following lemmas (see Theorem 10.7, Theorem 10.11, Theorem 10.12 in [Sch90])

Lemma D.7. *The subset $Y \in \mathcal{I}_1 \cap \mathcal{I}_2$, is a maximum cardinality common independent set iff there is no directed path from X_1 to X_2 in H.*

Lemma D.8. *Consider a common independent set $Y \in \mathcal{I}_1 \cap \mathcal{I}_2$.*

(i) Y is extreme if and only if $H(\mathcal{M}_1, \mathcal{M}_2, Y)$ contains no directed cycle of negative length.
(ii) If Y is extreme, and P is a shortest (minimal-hop) path from X_1 to X_2 in H, then $Y \triangle P$ is also extreme.

To mimic the Bellman–Ford algorithm, we generalize the notion of augmenting paths using *minimal paths and cycles*, defined with respect to both weight and hop-count.

Minimal Cycles and Paths. A cycle C in H is *minimal* if no proper subcycle C' satisfies $\ell(C') \leq \ell(C)$. Similarly, for $y \in Y$, define $\mathcal{P}(y, t)$ to be the set of directed paths from X_1 to y in H using at most $2t - 1$ arcs. A path $P \in \mathcal{P}(y, t)$ is *minimal* if it is lexicographically minimal with respect to length and number of arcs:

$$\text{For all } P' \in \mathcal{P}(y, t), \quad (\ell(P'), |P'|) \succeq (\ell(P), |P|).$$

In the following lemma, we prove our main observation that augmenting an extreme common independent by a minimal path gives a common independent set.

Lemma D.9 (Minimal paths and cycles)**.** *If Y is extreme:*

1. *For any minimal cycle C in H, we have $Y \triangle C \in \mathcal{I}_1 \cap \mathcal{I}_2$.*
2. *For any minimal path $P \in \mathcal{P}(y, t)$, we have $Y \triangle P \in \mathcal{I}_1 \cap \mathcal{I}_2$.*

Proof of Lemma D.9. For the proof of part (1), suppose $Z := Y \Delta C \notin \mathcal{I}_1 \cap \mathcal{I}_2$, assume by symmetry that $Z \notin \mathcal{I}_1$. Let N, M be the matching edges in C corresponding to $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively. Using Lemma D.6(a), we know that there exists a different matching $N' \neq N$ between the vertices $VC \cap Y$ and $VC \setminus Y$ with edges corresponding to $\mathcal{M}_1$.

Consider the multiset union of edges $N \cup N' \cup 2M$ which takes two copies of every edge from M and a copy of every edge from N and N'. Since these set of edges enter and leave every vertex exactly twice, the set of edges can be decomposed into a collection of cycles $C_1, \ldots, C_p$ for some $p \geq 2$ such that some of the cycles have strictly fewer vertices than C. We also have $\sum_{i \in p} \ell(C_i) = 2\ell(C)$. Using Lemma D.8(i), we know $\ell(C_i) \geq 0$ and since $p \geq 2$, there exists a cycle C_i with lesser weight or the same weight with lesser number of vertices as C contradicting the minimality of C.

The proof of part (2) proceeds in a few steps:

(2a) (**No re-entry**) First, we prove that a minimal path in $\mathcal{P}(y,t)$ never re-enters X_1 again. If $P = (x_0, y_1, x_1, \ldots, y_k, x_k, \ldots, x_{t-1}, y_t)$ with $x_k \in X_1$, then observe that there is an edge from y_k to x_0 because $Y + x_0 \in \mathcal{I}_1$ as $x_0 \in \mathcal{I}_1$ which implies $Y + x_0 - y_k \in \mathcal{I}_1$ and hence the edge. If we let $C = (x_0, y_1, x_1, \ldots, y_k, x_0)$, and $P' = (x_k, y_{k+1}, \ldots, x_{t-1}, y_t)$, we have $\ell(P) = \ell(C) + \ell(P')$ implying that P' has smaller length and lesser number of arcs than P as $\ell(C) \geq 0$ from Lemma D.8(i) contradicting the minimality of P. An identical argument can be used to show that vertices are not revisited again in minimal paths.

(2b) ($Y \Delta P \in \mathcal{I}_1$) Next, we show that $Y \Delta P \in \mathcal{I}_1$. Let $Z = \{y_1, x_1, \ldots, y_{t-1}, x_{t-1}\}$. We first show that $Y \Delta Z \in \mathcal{I}_1$. Let N be the set of matching edges (y_i, x_i) corresponding to $\mathcal{M}_1$ for $1 \leq i \leq t-1$ and M be the set of matching edges (x_{j-1}, y_j) corresponding to $\mathcal{M}_2$ for $1 \leq j \leq t$. If $Y \Delta Z \notin \mathcal{I}_1$, then there is a different matching N' between $Z \backslash Y$ and $Z \cap Y$ using Lemma D.6(a). Consider the multiset union of edges $N \cup N' \cup 2M \cup 2(y_k, x_0)$.

Since these set of edges enter and leave every vertex exactly twice, the set of edges can be decomposed into a collection of cycles $C_1, \ldots, C_p$ for some $p \geq 2$ such that some of the cycles have strictly fewer vertices than C. We also have $\sum_{i \in p} \ell(C_i) = 2\ell(C)$. Using Lemma D.8(i), we know $\ell(C_i) \geq 0$. Consider the two different cycles (say) C_1, C_2 that each contain a copy of the edge (y_t, x_0). We know that one of these cycles (say) C_1 should have $\ell(C_1) \leq \ell(C)$ and $|C_1| < |C|$. Removing the (y_t, x_0) edge from C_1 gives a path (say) P_1 such that $(\ell(P), |P|) \succeq (\ell(P_1), |P_1|)$ contradicting the minimality of P.

Now we will show that $Y \Delta Z \in \mathcal{I}_1 \Rightarrow Y \Delta P \in \mathcal{I}_1$. First, observe that $r_{\mathcal{M}_1}(Y \cup Z) = |Y|$ because $r_{\mathcal{M}_1}(Y + z) = r_{\mathcal{M}_1}(Y) = |Y|$ for every $z \in Z$ as we showed $Z \cap X_1 = \emptyset$ in Item (2a). On the other hand, the rank $r_{\mathcal{M}_1}(Y \cup P) \geq |Y| + 1$ because $Y \cup P \supseteq Y + x_0 \in \mathcal{I}_1$. Since $Y \Delta Z \subseteq Y \cup P$, there is an element in $\{x_0, y_1, y_2, \ldots, y_{t-1}\}$ that extends $Y \Delta Z$. This has to be x_0 because we know $r_{\mathcal{M}_1}(Y \cup Z) = |Y|$. This implies that

$$(Y \Delta Z) + x_0 = (Y \Delta P) + y_t \in \mathcal{I}_1 \Rightarrow (Y \Delta P) \in \mathcal{I}_1.$$

(2c) ($Y \Delta P \in \mathcal{I}_2$) Finally, we show that $Y \Delta P \in \mathcal{I}_2$. Let N be the set of matching edges (y_i, x_i) corresponding to M_1 for $1 \leq i \leq t-1$ along with (y_t, x_0). Let M be the set of matching edges (x_{j-1}, y_j) corresponding to $\mathcal{M}_2$ for $1 \leq j \leq k$. If $Y \Delta P \notin \mathcal{I}_2$, then there is an alternate matching M' from vertices in $P \backslash Y$ to $Y \backslash P$. Consider the multiset union of edges $M \cup M' \cup 2N$. Similar to the argument in parts (1) and (2a) above, we find a cycle (say) C_1 containing the edge (y_t, x_0) such that $\ell(C_1) \leq \ell(C)$ and $|C_1| < |C|$. Deleting the edge (y_t, x_0) gives a path P_1 that contradicts the minimality of P. □

Lemma D.9 allows us to compare lengths of minimal paths using only comparisons of matroid intersections:

$$\ell(P_1) - \ell(P_2) = w(Y \triangle P_1) - w(Y \triangle P_2),$$

when P_1 and P_2 are minimal paths or cycles. In order to optimize over minimal paths, we modify the Bellman-Ford algorithm in the natural way.

Modified Bellman-Ford Algorithm. Let $p_y^{(t)}$ denote the minimal path in $\mathcal{P}(y,t)$. We compute it recursively:

- **Initialization:** For each $y \in Y$,

$$p_y^{(1)} = \min_{x \in X_1} (x \to y).$$

- **Update:** For $t \geq 1$,
$$p_y^{(t+1)} = \min\left\{p_y^{(t)},\ p_z^{(t)} \to x \to y\right\},$$
where $(z, x) \in Y \times (E \setminus Y)$ satisfies:
  1. (z, x) and (x, y) are arcs in H, and
  2. $Y \triangle p_z^{(t)} + x - y \in \mathcal{I}_1 \cap \mathcal{I}_2$.
- **Shortest Augmenting path:** After computing paths $p_y^{(t)}$ for all $y \in Y$ and $t = |Y|$, we extract a minimal shortest path P from X_1 to X_2 such that $Y \triangle P \in \mathcal{I}_1 \cap \mathcal{I}_2$, and update $Y \leftarrow Y \triangle P$.

Proof of Theorem D.4. Let Y_t be the extreme common independent set of size exactly t starting with $Y_0 = \emptyset$. Use the modified Bellman-Ford Algorithm to find the shortest augmenting path P_t in the exchange graph $H(\mathcal{M}_1, \mathcal{M}_2, Y_t))$. Using Lemma D.8 part (i), the exchange graph has no negative cycles. The shortest path computation takes $O(n^3)$ comparison queries and time (see section D.4). After every update, we know that $Y_{t+1} := Y_t \Delta P_t$ is an extreme set using Lemma D.8 part (ii). We continue for n steps as long as there is a directed path from X_1 to X_2 in the exchange graph. Using Lemma D.7, we know that the maximum cardinality intersection is reached when an augmenting path is not present anymore. Finally, we can compare the weights of all the extreme sets obtained to output the minimum weight common intersection $O(n^4)$ comparisons and $O(n^4)$ time. □

D.4 Shortest paths

Finally, when the combinatorial objects of interest are $s - t$ walks, we show that the Bellman-Ford shortest path algorithm can be used to find the shortest $s - t$ walk (which will be a path) using $O(n^3)$ many $s - t$ walk comparisons.

Theorem D.10 (s-t walks)**.** *There is an algorithm that finds the minimum-length s-t path in a graph G (or a negative cycle, if one exists) using $O(n^3)$ s-t walk comparisons and $O(n^3)$ time.*

Proof. Assume that for every vertex $v \in V$, there exists a path from s to v and from v to t, since otherwise no s-t walk can pass through v, and we may ignore such vertices. For each $v \in V$, fix an arbitrary v-t walk t_v. Let $s_v^{(k)}$ denote the shortest s-v walk with at most k edges, where $s_v^{(0)} = \emptyset$.

Define the recurrence:

$$s_v^{(k+1)} = \min_{u \in V}\left(s_u^{(k)} \circ (u, v)\right), \tag{24}$$

where ties are broken arbitrarily. We can compute this because for any two walks $s_{u_1}^{(k)} \circ (u_1, v)$ and $s_{u_2}^{(k)} \circ (u_2, v)$, we can compare them by extending each with t_v and comparing total weights.

If w has no negative cycles, then $s_t^{(n-1)}$ is the shortest s-t path. To detect a negative cycle, check whether $w(s_u^{(n)}) < w(s_u^{(n-1)})$ for any $u \in V$.

Since there are $O(n^2)$ entries $s_u^{(k)}$, and each is computed by comparing $O(n)$ candidate walks, the total number of comparisons is $O(n^3)$. Although each comparison naively takes $O(n)$ time, we can amortize this using shared subwalks, leading to $O(n)$ time per table entry and total runtime $O(n^3)$. □

References

[AD21] Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 172–180. SIAM, 2021.

[AEG+22] Simon Apers, Yuval Efron, Pawel Gawrychowski, Troy Lee, Sagnik Mukhopadhyay, and Danupon Nanongkai. Cut Query Algorithms with Star Contraction . In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 507–518, Los Alamitos, CA, USA, November 2022. IEEE Computer Society.

[AL21] Arinta Auza and Troy Lee. On the query complexity of connectivity with global queries. *arXiv preprint arXiv:2109.02115*, 2021.

[Bar03] William Barnett. The modern theory of consumer behavior: Ordinal or cardinal? *The Quarterly Journal of Austrian Economics*, 6(1):41–65, 2003.

[BM11] Nader H Bshouty and Hanna Mazzawi. On parity check (0, 1)-matrix over $\mathbb{Z}_p$. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1383–1394. SIAM, 2011.

[BT52] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[BVW16] Maria-Florina Balcan, Ellen Vitercik, and Colin White. Learning combinatorial functions from pairwise comparisons. In *Conference on Learning Theory*, pages 310–335. PMLR, 2016.

[CBCTH13] Xi Chen, Paul N Bennett, Kevyn Collins-Thompson, and Eric Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 193–202, 2013.

[CCYL25] Peter Chen, Xi Chen, Wotao Yin, and Tianyi Lin. Compo: Preference alignment via comparison oracles, 2025.

[Cho13] Sung-Soon Choi. Polynomial time optimal query algorithms for finding graphs with arbitrary real weights. In *Conference on Learning Theory*, pages 797–818. PMLR, 2013.

[CSC+20] Minhao Cheng, Simranjit Singh, Patrick H. Chen, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. Sign-opt: A query-efficient hard-label adversarial attack. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[EHW15] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P. Woodruff. Applications of uniform sampling: Densest subgraph and beyond, 2015.

[GK00] Vladimir Grebinski and Gregory Kucherov. Optimal reconstruction of graphs under the additive model. *Algorithmica*, 28(1):104–124, 2000.

[GPRW20] Andrei Graur, Tristan Pollner, Vidhya Ramaswamy, and S. Matthew Weinberg. New query lower bounds for submodular function minimization. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 64:1–64:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Har08] Nicholas James Alexander Harvey. *Matchings, matroids and submodular functions*. PhD thesis, Massachusetts Institute of Technology, 2008.

[HGL18] Siavash Haghiri, Damien Garreau, and Ulrike Luxburg. Comparison-based random forests. In *International Conference on Machine Learning*, pages 1871–1880. PMLR, 2018.

[HGvL17] Siavash Haghiri, Debarghya Ghoshdastidar, and Ulrike von Luxburg. Comparison-based nearest neighbor search. In *Artificial Intelligence and Statistics*, pages 851–859. PMLR, 2017.

[HH10] Sandra Heldsinger and Stephen Humphry. Using the method of pairwise comparison to obtain reliable teacher assessments. *The Australian Educational Researcher*, 37(2):1–19, 2010.

[HHS24] Zhongtian He, Shang-En Huang, and Thatchaphol Saranurak. Cactus representation of minimum cuts: Derandomize and speed up, 2024.

[JNR12] Kevin G. Jamieson, Robert D. Nowak, and Benjamin Recht. Query complexity of derivative-free optimization. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2681–2689, 2012.

[KB79] Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the smith and hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–507, 1979.

[KLM19] Daniel M Kane, Shachar Lovett, and Shay Moran. Near-optimal linear decision trees for k-sum and related problems. *Journal of the ACM (JACM)*, 66(3):1–18, 2019.

[KLMZ17] Daniel M. Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active Classification with Comparison Queries . In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 355–366, Los Alamitos, CA, USA, October 2017. IEEE Computer Society.

[Kro77] Stein Krogdahl. The dependence graph for bases in matroids. *Discrete Mathematics*, 19(1):47–59, 1977.

[LC24] Hang Liao and Deeparnab Chakrabarty. Learning spanning forests optimally in weighted undirected graphs with cut queries. In Claire Vernade and Daniel Hsu, editors, *Proceedings of The 35th International Conference on Algorithmic Learning Theory*, volume 237 of *Proceedings of Machine Learning Research*, pages 785–807. PMLR, 25–28 Feb 2024.

[LLSZ21] Troy Lee, Tongyang Li, Miklos Santha, and Shengyu Zhang. On the cut dimension of a graph. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 15:1–15:35. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[MN20] Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 496–509, New York, NY, USA, 2020. Association for Computing Machinery.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discret. Math.*, 5(1):54–66, February 1992.

[Oxl11] James Oxley. *Matroid Theory*. Oxford University Press, 02 2011.

[Par19] Vilfredo Pareto. *Manuale di economia politica con una introduzione alla scienza sociale*, volume 13. Società editrice libraria, 1919.

[Que98] Maurice Queyranne. Minimizing symmetric submodular functions. *Mathematical Programming*, 82(1):3–12, 1998.

[RS95] Ran Raz and Boris Spieker. On the "log rank"-conjecture in communication complexity. *Combinatorica*, 15(4):567–588, 1995.

[RSW18] Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing Exact Minimum Cuts Without Knowing the Graph. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., USA, 1986.

[Sch90] Alexander Schrijver. A course in combinatorial optimization. 1990.

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume B. 01 2003.

[SW94] Mechthild Stoer and Frank Wagner. A simple min cut algorithm. In Jan van Leeuwen, editor, *Algorithms - ESA '94, Second Annual European Symposium, Utrecht, The Netherlands, September 26-28, 1994, Proceedings*, volume 855 of *Lecture Notes in Computer Science*, pages 141–147. Springer, 1994.

[TRC24] Zhiwei Tang, Dmitry Rybin, and Tsung-Hui Chang. Zeroth-order optimization meets human feedback: Provable learning via ranking oracles, 2024.

[XZM+17] Yichong Xu, Hongyang Zhang, Kyle Miller, Aarti Singh, and Artur Dubrawski. Noise-tolerant interactive learning using pairwise comparisons. *Advances in neural information processing systems*, 30, 2017.

[YBKJ12] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. The k-armed dueling bandits problem. *J. Comput. Syst. Sci.*, 78(5):1538–1556, 2012.

[ZL24] Chenyi Zhang and Tongyang Li. Comparisons are all you need for optimizing smooth functions. *CoRR*, abs/2405.11454, 2024.