Optimization Everywhere: Convex, Combinatorial, and Economic

By

Chiu Wai Wong

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Prasad Raghavendra, Chair
Professor Satish Rao
Professor Nikhil Srivastava

Fall, 2018

Optimization Everywhere: Convex, Combinatorial, and Economic

Abstract

Optimization Everywhere: Convex, Combinatorial, and Economic

By

Chiu Wai Wong

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Prasad Raghavendra, Chair

In this thesis we study fundamental problems that arise in optimization and its applications. We present provably efficient algorithms that achieve better running times or approximation guarantees than previously known. Our method draws on the toolkit from convex and combinatorial optimization as well as economics. By intertwining techniques from these disciplines, we are able to make progress on multiple old and new problems, some of which have stood open for many years. Main results of this thesis include the following:

- **Convex Programming**: We show how to solve convex programming with an expected $O(n\log(nR/\epsilon))$ evaluations of the separation oracle and additional time $O(n^3\log^{O(1)}(nR/\epsilon))$. This matches the oracle complexity and improves upon the $O(n^{\omega+1}\log(nR/\epsilon))$ additional time of the previous fastest algorithm achieved over 25 years ago for the current value of the matrix multiplication constant $\omega < 2.373$ when $R/\epsilon = O(\mathsf{poly}(n))$.

- **Submodular Function Minimization**: We provide new weakly and strongly polynomial time algorithms with a running time of $O(n^2\log nM\cdot\text{EO}+n^3\log^{O(1)}nM)$ and $O(n^3\log^2 n\cdot\text{EO}+n^4\log^{O(1)}n)$, improving upon the previous best of $O((n^4\cdot\text{EO}+n^5)\log M)$ and $O(n^5\cdot\text{EO}+n^6)$ respectively. We also provide the first subquadratic time algorithm for computing an approximately optimal solution.

- **Matroid Intersection**: We provide new algorithms with a running time of $O(nr\mathcal{T}_{\text{rank}}\log n\log(nM)+n^3\log^{O(1)}nM)$ and $O(n^2\mathcal{T}_{\text{ind}}\log(nM)+n^3\log^{O(1)}nM)$, achieving the first quadratic bound on the query complexity for the independence and rank oracles. In the unweighted case, this is the first improvement since 1986 for independence oracle.

- **Submodular Flow**: We obtain a faster weakly polynomial runtime of $O(n^2\log(nCU)\cdot\text{EO}+n^3\log^{O(1)}(nCU))$, improving upon the previous best of $O(mn^5\log(nU)\cdot\text{EO})$ and $O\left(n^4h\min\{\log C,\log U\}\right)$ from 15 years ago by a factor of $\tilde{O}(n^4)$.

- **Semidefinite Programming**: We obtain a running time of $\tilde{O}(n(n^2 + m^\omega + S))$, improving upon the previous best of $\tilde{O}(n(n^\omega + m^\omega + S))$ for the regime S is small.
- **Market Equilibrium**: We present the first polynomial time algorithm for computing market equilibrium in an economy with indivisible goods and general buyer valuations having only access to an aggregate demand oracle.
- **Vertex Cover with Hard Capacity**: We give a f-approximation algorithm for the minimum unweighted Vertex Cover problem with Hard Capacity constraints on f-hypergraphs This improves over the previous $2f$-approximation and is the best possible assuming the unique game conjecture.
- **Network Design for Effective Resistance**: We initiate the study of network design for s-t effective resistance. Among other results we present a constant factor approximation by applying classic techniques to a convex quadratic programming relaxation.

Acknowledgments

First and foremost, my deepest gratitude goes to my advisor Christos Papadimitriou for bearing my immaturity both as a person and as a researcher. His patience and honest feedback have shaped my research style. The simple-yet-powerful type of results that are the trademark of his works has led me into asking and answering various questions. Despite not having a joint paper with him (yet), his influence is ubiquitous in almost all of my PhD works. For this I am forever indebted to him. I am also fortunate to be bestowed with a high degree of freedom in my adventure into unlocking the secrets of math and nature. Perhaps his only shortcoming is his patience and generosity for putting up with my hastily prepared presentations and unscheduled invasions into his office.

Beyond Christos more people than I can remember have played a role in my development, from the very day I was born on to this moment. Thanks to my family and friends for emotional support. Thanks to Satish, Prasad and Nikhil for serving on my committee. Thanks to my collaborators whom I learned a great deal from. Sorry for being partly or solely responsible for the late night paper writing marathon. They are (in chronological order) Yajun, Wang Chi, Michel, Vahab, Hossein, Jon, Yin Tat, Aaron, Renato, Adrian, Deeparnab, Aaron, Lap Chi, Hong, Pak Hay, and Sahil.

I am fortunate to have the privilege of being a research intern at Google and IBM. Thanks to Vahab, Hossein, Jon, and Renato for hosting and mentoring me at Google; and Jayram and Nimrod at IBM. I am also grateful to Google for the PhD fellowship.

It is my lifetime honor to belong to the Berkeley theory group. To the prospective students or postdocs who may be reading this, our group is a community uniquely characterized by the abundance of support and friendship. This is truly the best stress-free place to do quality research. My apology to my fellow theorists for not naming you here because there are simply too many to list.. You know who you are.

Thanks to the anonymous reviewers who have reviewed my papers in case you're reading this.

Finally, I am equally sad that my days at Berkeley are numbered and excited about my next journey to the Northwest.

Contents

1 Introduction

Over the century optimization has established itself as an important branch of mathematics with relevance to engineering as well as natural and social sciences. As the invisible backbone behind numerous applications from airline scheduling to internet commerce, optimization has continued to evolve its role thanks to developments in both the basic theory and the application to other domains. Our ever greater demand for efficiency of time, energy and resources means that sustained research efforts on optimization are imperative to serve our present and future needs. We envision a future with optimization being an omnipresent technology.

In this thesis we study various fundamental problems in optimization, both combinatorial and continuous, and develop new provably efficient algorithms for them. An overarching theme of our works is the interplay between convex optimization and problems from combinatorics and economics. New and improved algorithms are obtained by synthesizing ideas and tools from these separate research areas in nontrivial manners. While this thesis is hardly the first to explore their connections, (some of) our approaches substantially deviate from the traditional thinking on the role of convex optimization in these areas. In the broader context, our results add to the recent growing body of works that establish convex optimization as a core tool in algorithm design.

In this Chapter, we present our new results and explain their significance in their respective research areas. To this end we describe how convex optimization is classically employed in Section 1.1. In Section 1.2, we provide a high-level summary of our results, obtained partly by applying convex optimization differently from before.

1.1 Overview

We assume general familiarity with optimization, the study of maximization or minimization problems under certain given constraints.

1.1.1 Opening the black box of convex optimization

The first half of this thesis is on the application of convex optimization in other research areas, primarily for designing fast algorithms.

Convex optimization is historically used in algorithm design via a *black-box* manner:

1. Formulate the given problem as a special case of convex minimization
2. Apply convex optimization algorithm to solve the problem

While black-box applications abstract the technical details and make those algorithms accessible to more researchers, much of the intricacies of convex optimization are sacrificed in the abstraction. As is common in mathematics, textbook theorems are sometimes too general to address all scenarios adequately. Faster algorithms can be expected by tailor-making those general results in view of the problem-specific structures, i.e. opening the black box. To execute this agenda, familiarity with both convex optimization and the application domain is needed in order to recognize and combine the necessary ingredients from both disciplines.

A prominent example is the recent nearly linear algorithms for solving approximate maximum flow [235, 148]. The algorithms, which massively outperform any black-box application of iterative methods, intertwine tools from structural graph theory and iterative methods in convex optimization. Indeed, since the celebrated result of Spielman and Teng on Laplacian solvers, a great body of works have been devoted to adapt the power of convex optimization for cut and flow problems in graphs.

We expand the reach of convex optimization to more fundamental problems such as submodular function minimization and market equilibrium computation. We will see that existing tools in convex optimization have to be extended and strengthened before they can be applied to our problems. This is in fact a two-way process: not only did optimization enable us to make progress in other fields, this effort has also contributed back positively by driving advances in convex optimization (e.g. cutting plane methods).

As an example, we use the cutting plane methods to explain our philosophy. The class of cutting plane methods include the famous ellipsoid method as a special case. For a given problem, cutting plane methods are classically employed only to show the existence of polynomial time algorithms due to the stigma that this class of methods have poor running times both in theory and in practice. We challenge this conventional wisdom by exhibiting several problems where cutting plane methods, coupled with appropriate modifications (i.e. opening the black box) and problem-specific properties, produce the fastest known algorithms for these problems. The interplay between cutting plane methods (along with other techniques in continuous optimization) and the problem-specific properties are so intertwined that a black box application of the original guarantee of the cutting plane method would not yield the same result.

1.1.2 Rounding linear and convex programs

The second half of this thesis studies approximation algorithms for NP-hard problems based on rounding of their linear or convex programming relaxation.

The use of linear or convex programming relaxation is one of the most popular paradigms for designing approximation algorithms [259, 253]. It consists of two ingredients:

- a linear or convex relaxation of our problem whose optimal fractional solution can be computed in polynomial time via general convex optimization algorithms

- a rounding algorithm which converts the fractional solution to an integral solution

A wide variety of techniques and results have been developed for approximation algorithms in the literature. In this thesis we study two NP-hard combinatorial problems and present companion approximation algorithms that add new twists to this diverse toolkit. Both of them are graph optimization problems with a capacity constraint. More specifically:

Our algorithm for the minimum vertex cover with capacity problem employs *iterative rounding*, which incrementally constructs an integral solution by repeatedly rounding certain variables and solving the updated linear relaxation until all variables are integral. This is in contrast to previous works which solve only the relaxations once or twice.

Our algorithm for the minimum effective resistance with capacity problem involves a *convex quadratic program* relaxation and a corresponding rounding algorithm that heavily leverages the optimality conditions of the relaxation. To the best of our knowledge this is the first time the optimality conditions of a convex quadratic program play a major role in designing approximation algorithms for graph problems.

1.2 New results and thesis organization

Chapter 2: Basic concepts and problems

In Chapter 2 we give a summary of the problems addressed in this thesis along with a crash course on the technical background.

Part I-V: Fast Algorithms via Convex Optimization

Part I: Cutting plane methods

We improve upon the running time for finding a point in a convex set given a separation oracle. In particular, given a separation oracle for a convex set $K \subset \mathbb{R}^n$ that is contained in a box of radius R we show how to either compute a point in K or prove that K does not contain a ball of radius ϵ using an expected $O(n \log(nR/\epsilon))$ evaluations of the oracle and additional time $O(n^3 \log^{O(1)}(nR/\epsilon))$. This matches the oracle complexity and improves upon the $O(n^{\omega+1} \log(nR/\epsilon))$ additional time of the previous fastest algorithm achieved over 25 years ago by Vaidya [248] for the current value of the matrix multiplication constant $\omega < 2.373$ [258, 97] when $R/\epsilon = O(\text{poly}(n))$.

Using a mix of standard reductions and new techniques we show how our algorithm can be used to improve the running time for solving classic problems in continuous and combinatorial optimization in the next three Parts.

This Part is based on joint works with Yin Tat Lee and Aaron Sidford.

Part II: Convex minimization and the intersection problem

We explain how the cutting plane method can be applied to solve convex minimization and the intersection problem.

The intersection problem involves optimizing over the intersection of two convex bodies provided that we can optimize over each of them. Our algorithm is faster than previous and inspired by *regularization*, a classic optimization technique. In the special case of matroid intersection, we achieve an oracle complexity of $\widetilde{O}(nr)$, an improvement over the previous $\widetilde{O}(nr^{1.5})$ from 1986 [51]. Additional applications include submodular flow and semidefinite programming.

This Part is based on joint works with Yin Tat Lee and Aaron Sidford.

Part III: Submodular function minimization (SFM)

Submodualr minimization is the problem of minimizing submodular functions, which are the discrete analogue to convex functions. Suppose n is the size of the ground set, M is the maximum absolute value of function values and EO is the time for function evaluation.

Our weakly and strongly polynomial time algorithms have a running time of $O(n^2 \log nM \cdot \mathrm{EO} + n^3 \log^{O(1)} nM)$ and $O(n^3 \log^2 n \cdot \mathrm{EO} + n^4 \log^{O(1)} n)$, improving upon the previous best of $O((n^4 \cdot \mathrm{EO} + n^5) \log M)$ and $O(n^5 \cdot \mathrm{EO} + n^6)$ respectively. At the technical level our algorithms rely on the interplay between a new duality guarantee of the cutting plane method as well as classic combinatorial properties of submodular functions.

This Part is based on joint works with Yin Tat Lee and Aaron Sidford.

Part IV: Market equilibrium computation

We present the first polynomial time algorithm for computing Walrasian equilibrium in an economy with indivisible goods and *general* buyer valuations having only access to an *aggregate demand oracle*, i.e., an oracle that given prices on all goods, returns the aggregated demand over the entire population of buyers. For the important special case of gross substitute valuations, our algorithm queries the aggregate demand oracle $\widetilde{O}(n)$ times and takes $\widetilde{O}(n^3)$ time, where n is the number of goods. At the heart of our solution is a method for exactly minimizing certain convex functions which cannot be evaluated but for which the subgradients can be computed.

We also give the fastest known algorithm for computing Walrasian equilibrium for gross substitute valuations in the *value oracle model*. Our algorithm has running time $\widetilde{O}((mn + n^3)T_V)$ where T_V is the cost of querying the value oracle. A key technical ingredient is to regularize a convex programming formulation of the problem in a way that subgradients are cheap to compute. En route, we give necessary and sufficient conditions for the existence of *robust Walrasian prices*, i.e., prices for which each agent has a unique demanded bundle and the demanded bundles clear the market. When such prices exist, the market can be perfectly coordinated by solely using prices.

This Part is based on joint works with Renato Paes Leme.

Part V: Subquadratic submodular function minimization (SFM)

Having explored the cutting plane method for SFM, in this Part we turn to the application of first-order methods to SFM.

We present *subquadratic time* SFM algorithms. For integer-valued submodular functions, we give an SFM algorithm which runs in $O(nM^3 \log n \cdot \mathrm{EO})$ time giving the first *nearly linear* time algorithm in any known regime. For real-valued submodular functions with range in $[-1, 1]$, we give an algorithm which in $\tilde{O}(n^{5/3} \cdot \mathrm{EO}/\epsilon^2)$ time returns an ϵ-additive approximate solution. At the heart of it, our algorithms are projected stochastic subgradient descent methods on the Lovasz extension of submodular functions where we crucially exploit submodularity and data structures to obtain fast, i.e. sublinear time, subgradient updates.

This Part is based on joint works with Deeparnab Chakrabarty, Yin Tat Lee and Aaron Sidford.

Part VI-VII: Approximation Algorithms via Linear and Convex Relaxations

Part VI: Vertex cover with hard capacity

We give a f-approximation algorithm for the minimum unweighted Vertex Cover problem with Hard Capacity constraints (VCHC) on f-hypergraphs. This problem generalizes standard vertex cover for which the best known approximation ratio is also f and cannot be improved assuming the unique game conjecture. Our result is therefore essentially the best possible. This improves over the previous 2.155 (for $f = 2$) and $2f$ approximation algorithms by Cheung, Goemans and Wong [40].

At the heart of our approach is to apply iterative rounding to a natural LP relaxation that is slightly different from prior works which used (non-iterative) rounding. Our algorithm is significantly simpler and offers an intuitive explanation why f-approximation can be achieved for VCHC.

No coauthor for this Part.

Part VII: Minimum effective resistance with capacity

We consider a new problem of designing a network with small s-t effective resistance. In this problem, we are given an undirected graph $G = (V, E)$ where each edge e has a cost c_e and a resistance r_e, two designated vertices $s, t \in V$, and a cost budget k. Our goal is to choose a subgraph to minimize the s-t effective resistance, subject to the constraint that the total cost in the subgraph is at most k. This problem is an interpolation between the shortest path problem and the minimum cost flow problem and has applications in electrical network design.

We present several algorithmic and hardness results for this problem. On the hardness side, we show that the problem is NP-hard, and is hard to approximate within a factor smaller than two assuming the small-set expansion conjecture. On the algorithmic side, we analyze a convex programming relaxation of the problem, and design a constant factor approximation algorithm when every edge has the same cost and the same resistance. The key of the rounding algorithm is a randomized path-rounding procedure based on the optimality conditions and a flow decomposition of the fractional solution.

This Part is based on joint works with Pak Hay Chan, Lap Chi Lau, Aaron Schild and Hong Zhou. My contribution includes formulating the problem and being the main architect behind an earlier $O(log^2 n)$ approximation cycle-pushing algorithm which was later improved to a constant factor approximation by my coauthors Lap Chi Lau and Hong Zhou.

2 Basic Concepts and Problems

We introduce the notions and problems studied throughout this thesis. This chapter is to give readers a glimpse into the flavor of this thesis; more in-depth discussions are deferred to later chapters.

2.1 General

Basics: We use $\mathrm{nnz}(\vec{x})$ or $\mathrm{nnz}(\mathbf{A})$ to denote the number of nonzero entries in a vector or a matrix respectively. Frequently, for $\vec{x} \in \mathbb{R}^d$ we let $\mathbf{X} \in \mathbb{R}^{d \times d}$ denote $\mathbf{diag}(\vec{x})$, the diagonal matrix such that $\mathbf{X}_{ii} = x_i$. For a symmetric matrix, $\mathbf{M}$, we let $\mathrm{diag}(\mathbf{M})$ denote the vector corresponding to the diagonal entries of $\mathbf{M}$, and for a vector, $\vec{x}$, we let $\|\vec{x}\|_{\mathbf{M}} \stackrel{\mathrm{def}}{=} \sqrt{\vec{x}^T \mathbf{M} \vec{x}}$.

Running Times : We typically use XO to denote the running time for invoking the oracle, where X depends on the type of oracle, e.g., SO typically denotes the running time of a separation oracle, EO denotes the running time of an evaluation oracle, etc. Furthermore, we use $\tilde{O}(f) \stackrel{\mathrm{def}}{=} O(f \log^{O(1)} f)$.

Spectral Approximations: For symmetric matrices $\mathbf{N}, \mathbf{M} \in \mathbb{R}^{n \times n}$, we write $\mathbf{N} \preceq \mathbf{M}$ to denote that $\vec{x}^T \mathbf{N} \vec{x} \leq \vec{x}^T \mathbf{M} \vec{x}$ for all $\vec{x} \in \mathbb{R}^n$ and we define $\mathbf{N} \succeq \mathbf{M}$, $\mathbf{N} \prec \mathbf{M}$ and $\mathbf{N} \succ \mathbf{M}$ analogously.

Misc: We let $\omega < 2.373$ [258] denote the matrix multiplication constant.

2.2 Convex Optimization

Basic concepts and definitions

Convex sets: A set $S \in \mathbb{R}^n$ is convex if for any $x, y \in S$, any point on the line between x and y also belongs to S, i.e. $tx + (1-t)y \in S$ for all $0 \leq t \leq 1$.

Balls: We let $B_p(r) \stackrel{\mathrm{def}}{=} \{\vec{x} : \|\vec{x}\|_p \leq r\}$ denote a ball of radius r in the ℓ_p norm. For brevity we refer to $B_2(r)$ as a a *ball of radius* r and $B_\infty(r)$ as a *box of radius* r.

Convex functions: Let S be convex. A function $f : S \longrightarrow \mathbb{R}$ is convex if for any $x, y \in S$, $f(x) + f(y) \geq f\left(\frac{x+y}{2}\right)$.

Gradients: The gradient of a differentiable function f at x is denoted by $\nabla f(\mathrm{x})$.

Subgradients: A subgradient of a convex function $f : S \longrightarrow \mathbb{R}$ at $x \in S$, denoted by $\partial f(x) \in \mathbb{R}^n$, satisfies $f(y) - f(x) \geq \partial f(x) \cdot (y - x)$ for any $y \in S$.

Problems

Convex minimization: The problem of solving $\min_{x \in S} f(x)$ for a convex function $f : S \longrightarrow \mathbb{R}$.
Algorithms for convex minimization generally proceed by generating a sequence of points that approach a minimizer based on the (sub)gradient.

Intersection problem: Let S, T be convex sets and f be a convex function defined on a set containing S and T. Given oracles for solving $\min_{x \in S} f(x)$ and $\min_{x \in T} f(x)$, compute $\min_{x \in S \cap T} f(x)$.

2.3 Separation Oracles

We frequently make assumptions about the existence of separation oracles for sets and functions. Here we formally define these objects as we use them throughout. Our definitions are possibly non-standard and chosen to handle the different settings that occur in this thesis.

Definition 1 (Separation Oracle for a Set)**.** Given a set $K \subset \mathbb{R}^n$ and $\delta \geq 0$, a δ*-separation oracle* for K is a function on $\mathbb{R}^n$ such that for any input $\vec{x} \in \mathbb{R}^n$, it either outputs "successful" or a half space of the form $H = \{\vec{z} : \vec{c}^T\vec{z} \leq \vec{c}^T\vec{x} + b\} \supseteq K$ with $b \leq \delta\|\vec{c}\|_2$ and $\vec{c} \neq \vec{0}$. We let $\mathrm{SO}_\delta(K)$ be the time complexity of this oracle.

The parameter δ indicates the accuracy of the oracle. For brevity we refer to a 0-separation oracle for a set as just a *separation oracle.*

Note that in Definition 1 we do not assume that K is convex. However, we remark that it is well known that there is a separation oracle for a set if and only if it is convex and that there is a δ separation oracle if and only if the set is close to convex in some sense.

Definition 2 (Separation Oracle for a Function)**.** For any function f, $\eta \geq 0$ and $\delta \geq 0$, a (η,δ)-separation oracle on a set Γ for f is a function on $\mathbb{R}^n$ such that for any input $\vec{x} \in \Gamma$, it either asserts $f(\vec{x}) \leq \min_{\vec{y}\in\Gamma} f(\vec{y}) + \eta$ or outputs a half space H such that

$$\{\vec{z} \in \Gamma : f(\vec{z}) \leq f(\vec{x})\} \subset H \overset{\text{def}}{=} \{\vec{z} : \vec{c}^T\vec{z} \leq \vec{c}^T\vec{x} + b\} \tag{2.1}$$

with $b \leq \delta\|\vec{c}\|$ and $\vec{c} \neq \vec{0}$. We let $\mathrm{SO}_{\eta,\delta}(f)$ be the time complexity of this oracle.

2.4 Combinatorial Optimization

Basic concepts and definitions

Submodular functions: A function $f : 2^V \longrightarrow \mathbb{Z}$ is *submodular* if $f(T+i) - f(T) \leq f(S+i) - f(S)$ for all $S \subseteq T$ and $i \in V \backslash T$.
Submodular functions can be viewed as the discrete analogue to convex functions, and generalize the graph cut function.

Matroids: Given a ground set E and a family of subsets $\mathcal{I} \subseteq 2^E$ (called *independent* sets), $M = (E, \mathcal{I})$ is a matroid if (1) any subset of an independent set is also independent; (2) for $I_1, I_2 \in \mathcal{I}$ with $|I_1| > |I_2|$, there is some $e \in I_1$ for which $I_2 + e \in \mathcal{I}$.
Matroids can be viewed as a generalization of linear independence.

Problems

Submodular function minimization: The problem of solving $\min_{A\subseteq V} f(A)$ for a submodular function $f : 2^V \longrightarrow \mathbb{Z}$.
This problem generalizes the well-known minimum s-t cut problem in graph theory.

Matroid Intersection: The problem of finding the maximum (weighted) set independent

in both of the two given matroids.
This problem generalizes the well-known maximum (weighted) bipartite matching problem in graph theory.

2.5 Economics

Basic concepts and definitions

Valuation functions: Given a set of items S, $v : 2^S \longrightarrow \mathbb{R}$ measures the valuation of an agent for each subset of items of S.

Utility functions: Given an agent with valuation function v and prices p_i on item i ($\forall i$), $u(S,p) = v(S) - \sum_{i \in S} p_i$ measures the utility ("satisfaction") of the agent if he/she purchases S.

Market equilibrium: Given an economy consisting of a number of agents and items, a market equilibrium is the state in which the prices on the items *clear* the market, i.e. there exists an allocation of the items to agents that maximizes the utility of every agent.

Problem

Market equilibrium computation: The problem of finding a market equilibrium under certain assumptions on what we know about the market.

2.6 Graph Theory

Basic concepts and definitions

Graphs: A graph $G = (V,E)$ consists of a (finite) set of vertices V and a set of edges E. Each edge $e \in E$ connects two vertices (formally, an unordered pair of vertices).

f-hypergraphs: A f-hypergraph $G = (V,E)$ consists of a (finite) set of vertices V and a set of edges E. Each edge $e \in E$ connects at most f vertices (formally, a subset of at most k vertices).

Vertex cover: Given (hyper)graph G, $S \subseteq V$ is a vertex cover if every edge has at least one endpoint in S.

s-t effective resistance: Given a graph $G = (V,E)$ with resistances on the edges, for $s,t \in V$ the s-t effective resistance is the potential difference between s and t when one unit of electrical flow is sent from s to t.

Problems

Vertex cover with hard capacity: Given a f-hypergraph $G = (V,E)$ and capacity $k_v \in \mathbb{N}$, find the minimum cardinality vertex cover S where each vertex $v \in S$ is only allowed to cover at most k_v edges, i.e. there is a mapping of E to S such that at most k_v edges are mapped to the same vertex v.

s-t **effective resistance network design**: Given a graph $G = (V, E)$ with resistances on the edges and capacity $k \in \mathbb{N}$, find the subgraph of most k edges with the minimum s-t effective resistance.

Part I
A Faster Cutting Plane Method

This Part is based on joint works with Yin Tat Lee and Aaron Sidford.

3 Introduction

Throughout Part I we study the following *feasibility problem*:

Definition 3 (Feasibility Problem)**.** Given a separation oracle for a set $K \subseteq \mathbb{R}^n$ contained in a box of radius R, either find a point $\vec{x} \in K$ or prove that K does not contain a ball of radius ϵ.

This feasibility problem is one of the most fundamental and classic problems in optimization. Since the celebrated result of Shor [238], Yudin and Nemirovski [262] and Khachiyan [152] in 1970s essentially proving that it can be solved in time $O(\mathsf{poly}(n) \cdot \mathrm{SO} \cdot \log(R/\epsilon))$, this problem has served as one of the key primitives for solving numerous problems in both combinatorial and convex optimization.

Despite the prevalence of this feasibility problem, the best known running time for solving this problem has not been improved in over 25 years. In a seminal paper of Vaidya in 1989 [248], he showed how to solve the problem in $\widetilde{O}(n \cdot \mathrm{SO} \cdot \log(nR/\epsilon) + n^{\omega+1}\log(nR/\epsilon))$ time. While there had been interesting generalizations and practical improvements [8, 226, 106, 13, 107, 210, 261, 108, 31], the best theoretical guarantees for solving this problem have not been improved since.

In Part I we show how to improve upon Vaidya's running time in certain regimes. We provide a cutting plane algorithm which achieves an expected running time of $O(n \cdot \mathrm{SO} \cdot \log(nR/\epsilon) + n^3 \log^{O(1)}(nR/\epsilon))$, improving upon the previous best running time for the current known value of $\omega < 2.373$ [258, 97] when $R/\epsilon = O(\mathsf{poly}(n))$.

We achieve our results by the combination of multiple techniques. First we show how to use techniques from the work of Vaidya and Atkinson to modify Vaidya's scheme so that it is able to tolerate random noise in the computation in each iteration. We then show how to use existing numerical machinery [249, 240, 177] in combination with some new techniques (Section 6.1 and Section 6.2) to implement each of these relaxed iterations efficiently. We hope that both these numerical techniques as well as our scheme for approximating complicated methods, such as Vaidya's, may find further applications.

While our work focuses on theoretical aspects of cutting plane methods, we achieve our results via the careful application of practical techniques such as dimension reduction and sampling. As such we hope that ideas in this work may lead to improved practical[1] algorithms for non-smooth optimization.

[1]Although cutting plane methods are often criticized for their empirical performance, recently, Bubeck, Lee and Singh [31] provided a variant of the ellipsoid method that achieves the same convergence rate as Nesterov's accelerated gradient descent. Moreover, they provided numerical evidence that this method can be superior to Nesterov's accelerated gradient descent, thereby suggesting that cutting plane methods can be as aggressive as first order methods if designed properly [31, 30].

3.1 Previous Work

Throughout, we restrict our attention to algorithms for the feasibility problem that have a polynomial dependence on SO, n, and $\log(R/\epsilon)$. Such "efficient" algorithms typically follow the following iterative framework. First, they compute some trivial region Ω that contains K. Then, the separation oracle is queried at some point $\vec{x} \in \Omega$. If $\vec{x} \in K$, the algorithm successfully solved the problem. Otherwise the separation oracle must return a half-space containing K. The algorithm then uses this half-space to shrink the region Ω while maintaining the invariant that $K \subseteq \Omega$. This process is then repeated until it finds a point $\vec{x} \in K$ or the region Ω becomes too small to contain a ball with radius ϵ.

Previous works on efficient algorithms for the feasibility problem all follow this iterative framework. They vary in terms of what set Ω they maintain, how they compute the center to query the separation oracle, and how they update the set. In Table 1, we list the previous running times for solving the feasibility problem. As usual SO indicates the cost of the separation oracle.

The first efficient algorithm for the feasibility problem is the *ellipsoid* method, due to Shor [238], Nemirovksii and Yudin [262], and Khachiyan [152]. The ellipsoid method maintains an ellipsoid as Ω and uses the center of the ellipsoid as the next query point. It takes $\Theta(n^2 \log(nR/\epsilon))$ calls of oracle which is far from the lower bound $\Omega(n \log(R/\epsilon))$ [209].

To alleviate the problem, the algorithm could maintain all the information from the oracle, i.e., the polytope created from the intersection of all half-spaces obtained so far. The center of gravity method [182] achieves the optimal oracle complexity using this polytope and its center of gravity as the next point. However, computing center of gravity is computationally expensive and hence we do not list its running time in Table 1. The Inscribed Ellipsoid Method [153] also achieved the optimal oracle complexity using this polytope as Ω but instead using the center of the maximal inscribed ellipsoid in the polytope to query the separation oracle. We listed it as occurring in year 1988 in Table 1 because it was [212] that yielded the first polynomial time algorithm to actually compute this maximal inscribed ellipsoid for polytope.

Vaidya [248] obtained a faster algorithm by maintaining an approximation of this polytope and using a different center, namely the volumetric center. Although the oracle complexity of this volumetric center method is very good, the algorithm is not extremely efficient as each iteration involves matrix inversion. Atkinson and Vaidya [13] showed how to avoid this computation in certain settings. However, they were unable to achieve the desired convergence rate from their method.

Bertsimas and Vempala [20] also gives an algorithm that avoids these expensive linear algebra operations while maintaining the optimal convergence rate by using techniques in sampling convex sets. Even better, this result works for a weaker oracle, namely the membership oracle. However, the additional cost of this algorithm is relatively high. We remark that while there are considerable improvements on the sampling techniques [187, 145, 177], the additional cost is still quite high compared to standard linear algebra.

Year	Algorithm	Complexity
1979	Ellipsoid Method [238, 262, 152]	$O(n^2\text{SO}\log\kappa + n^4\log\kappa)$
1988	Inscribed Ellipsoid [153, 212]	$O(n\text{SO}\log\kappa + (n\log\kappa)^{4.5})$
1989	Volumetric Center [248]	$O(n\text{SO}\log\kappa + n^{1+\omega}\log\kappa)$
1995	Analytic Center [13]	$O\left(\begin{array}{c} n\text{SO}\log^2\kappa + n^{\omega+1}\log^2\kappa \\ +(n\log\kappa)^{2+\omega/2}\end{array}\right)$
2004	Random Walk [20]	$\to O(n\text{SO}\log\kappa + n^7\log\kappa)$
2015	This work	$O(n\text{SO}\log\kappa + n^3\log^{O(1)}\kappa)$

Table 1: Algorithms for the Feasibility Problem. κ indicates nR/ϵ. The arrow $\to$ indicates that it solves a more general problem where only a membership oracle is given.

3.2 Challenges in Improving Previous Work

Our algorithm builds upon the previous fastest algorithm of Vaidya [250]. Ignoring implementation details and analysis, Vaidya's algorithm is quite simple. This algorithm simply maintains a polytope $P^{(k)} = \{x \in \mathbb{R}^n : \mathbf{A}\vec{x} - \vec{b} \geq \vec{0}\}$ as the current Ω and uses the *volumetric center*, the minimizer of the following *volumetric barrier function*

$$\text{argmin}_{\vec{x}}\frac{1}{2}\log\det\left(\mathbf{A}^T\mathbf{S}_{\vec{x}}^{-2}\mathbf{A}\right) \quad \text{where} \quad \mathbf{S}_{\vec{x}} \stackrel{\text{def}}{=} \mathbf{diag}(\mathbf{A}\vec{x} - \vec{b}) \tag{3.1}$$

as the point to query the separation oracle. The polytope is then updated by adding shifts of the half-spaces returned by the separation oracle and dropping unimportant constraints. By choosing the appropriate shift, picking the right rule for dropping constraints, and using Newton's method to approximately compute the volumetric center, a running time of $O(n \cdot SO \cdot \log\kappa + n^{1+\omega}\log\kappa)$ can be achieved.

While Vaidya's algorithm's dependence on SO is essentially optimal, the additional per-iteration costs of his algorithm could possibly be improved. The computational bottleneck in each iteration of Vaidya's algorithm is computing the gradient of $\log\det$ which in turn involves computing the so-called leverage scores $\vec{\sigma}(\vec{x}) \stackrel{\text{def}}{=} \text{diag}(\mathbf{S}_x^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{S}_x^{-1})$, a commonly occurring quantity in numerical analysis and convex optimization [240, 47, 183, 177, 176]. As the best known algorithms for computing leverage scores exactly in this setting take time $O(n^\omega)$, directly improving the running time of Vaidya's algorithm seems challenging.

However, since an intriguing result of Spielman and Srivastava in 2008 [240], it has been well known that using Johnson-Lindenstrauss transform these leverage scores can be computed up to a multiplicative $(1 \pm \epsilon)$ error by solving $O(\epsilon^{-2}\log n)$ linear systems involving $\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A}$. While in general this still takes time $O(\epsilon^{-2}n^\omega)$, there are known techniques for efficiently maintaining the inverse of a matrix so that solving linear systems takes amortized $O(n^2)$ time [249, 176, 177]. Consequently if it could be shown that computing *approximate* leverage scores sufficed, this would potentially decrease the amortized cost per iteration of Vaidya's method.

Unfortunately, Vaidya's method does not seem to tolerate this type of multiplicative error. If we compute gradients of (3.1) using this approximation of leverage scores, it seems that the point computed would be far from the true center. Moreover, without being fairly close to the true volumetric center, it is difficult to argue that such a cutting plane method would make sufficient progress.

To overcome this issue, it is tempting to directly use the recent work on faster linear programming [176]. In this work, the authors faced a similar issue where a volumetric, i.e. $\log\det$, potential function had the right analytic and geometric properties but was computational expensive to minimize. To overcome this issue the authors instead computed a weighted analytic center:

$$\text{argmin}_{\vec{x}} - \sum_{i\in[m]} w_i \log s_i(\vec{x}) \quad \text{where} \quad \vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b} \quad .$$

For carefully chosen weights this center provides the same convergence guarantees as the volumetric potential function, while each step can be computed by solving few linear systems (rather than forming the matrix inverse).

Unfortunately, it is unclear how to directly extend the work in [176] on solving an explicit linear program to the feasibility problem specified by a separation oracle. While it is possible to approximate the volumetric barrier by a weighted analytic center in many respects, proving that this approximation suffices for fast convergence remains open. In fact, the volumetric barrier function as used in Vaidya's algorithm is well approximated simply by the standard analytic center

$$\text{argmin}_{\vec{x}} - \sum_{i\in[m]} \log s_i(\vec{x}) \quad \text{where} \quad \vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b} \quad .$$

as all the unimportant constraints are dropped during the algorithm. However, despite decades of research, the best running times known for solving the feasibility problem using the analytic center are Vaidya and Atkinson algorithm from 1995 [13]. While the running time of this algorithm could possibly be improved using approximate leverage score computations and amortized efficient linear system solvers, unfortunately without further insight this would at best yield an algorithm which requires suboptimal $O(n\log^{O(1)}\kappa)$ queries to the separation oracle.

As pointed out in [13], the primary difficulty in using any sort of analytic center is quantifying the amount of progress made in each step. We still believe providing direct near-optimal analysis of weighted analytic center is a tantalizing open question warranting further investigation. However, rather than directly address the question of the performance of weighted analytic centers for the feasibility problem, we take a slightly different approach that side-steps this issue. We provide a partial answer that still sheds some light on the performance of the weighted analytic center while still providing our desired running time improvements.

3.3 Our Approach

To overcome the shortcoming of the volumetric and analytic centers we instead consider a hybrid barrier function

$$\text{argmin}_{\vec{x}} - \sum_{i\in[m]} w_i \log s_i(\vec{x}) + \log\det(\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{A}) \quad \text{where} \quad \vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b} \quad .$$

for careful chosen weights. Our key observation is that for correct choice of weights, we can compute the gradient of this potential function. In particular if we let $\vec{w} = \vec{\tau} - \vec{\sigma}(\vec{x})$ then the gradient of this potential function is the same as the gradients of $\sum_{i\in[m]} \tau_i \log s_i(\vec{x})$, which we can compute efficiently. Moreover, since we are using $\log\det$, we can use analysis similar to Vaidya's algorithm [248] to analyze the convergence rate of this algorithm.

Unfortunately, this simple observation does not immediately change the problem substantially. It simply pushes the problem of computing gradients of $\log\det$ to computing $\vec{w}$. Therefore, for this scheme to work, we would need to ensure that the weights do not change too much and that when they change, they do not significantly hurt the progress of our algorithm. In other words, for this scheme to work, we would still need very precise estimates of leverage scores.

To salvage the idea, we observe that the leverage scores $\vec{\sigma}(\vec{x})$ do not change too much between iterations. Moreover, we provide what we believe is an interesting technical result that an unbiased estimate to the changes in leverage scores can be computed using linear system solvers such that the *total error* of the estimate is bounded by the total change of the leverage scores (See Section 6.1). Using this result our scheme simply follows Vaidya's basic scheme in [248], however instead of minimizing the hybrid barrier function directly we alternate between taking Newton steps we can compute, changing the weights so that we can still compute Newton steps, and computing accurate unbiased estimates of the changes in the leverage scores so that the weights do not change adversarially by too much.

To make this scheme work, there are two additional details that need to be dealt with. First, we cannot let the weights vary too much as this might ultimately hurt the rate of progress of our algorithm. Therefore, in every iteration we compute a single leverage score to high precision to control the value of w_i and we show that by careful choice of the index we can ensure that no weight gets too large (See Section 6.2).

Second, we need to show that changing weights does not affect our progress by much more than the progress we make with respect to $\log\det$. To do this, we need to show the slacks are bounded above and below. We enforce this by adding regularization terms and consider the potential function

$$p_{\vec{e}}(\vec{x}) = -\sum_{i\in[m]} w_i \log s_i(\vec{x}) + \frac{1}{2}\log\det\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}\right) + \frac{\lambda}{2}\left\|x\right\|_2^2$$

This allows us to ensure that the entries of $\vec{s}(\vec{x})$ do not get too large or too small and therefore changing the weighting of the analytic center cannot affect the function value too much.

Third, we need to make sure our potential function is convex. If we simply take $\vec{w} = \vec{\tau} - \vec{\sigma}(\vec{x})$ with $\vec{\tau}$ as an estimator of $\vec{\sigma}(\vec{x})$, $\vec{w}$ can be negative and the potential function could be non-convex. To circumvent this issue, we use $\vec{w} = c_e + \vec{\tau} - \vec{\sigma}(\vec{x})$ and make sure $\left\|\vec{\tau} - \vec{\sigma}(\vec{x})\right\|_\infty < c_e$.

Combining these insights, using efficient algorithms for solving a sequence of slowly changing linear systems [249, 176, 177], and providing careful analysis ultimately allows us to achieve a running time of $O(n\text{SO}\log\kappa + n^3\log^{O(1)}\kappa)$ for the feasibility problem. Furthermore, in the case that K does not contain a ball of radius ϵ, our algorithm provides a proof/certificate that the polytope does not contain a ball of radius ϵ. This proof ultimately allows us to achieve running time improvements for strongly polynomial submodular minimization in Part III.

3.4 Organization

The rest of Part I is organized as follows. In Section 4 we provide some preliminary information and notation used throughout Part I. In Section 5 we then present and analyze our cutting plane method. In Section 6 we provide key technical tools for the analysis which may be of independent interest.

4 Preliminaries

Here we introduce some notation and concepts used throughout Part I.

4.1 Leverage Scores

Our algorithms make extensive use of *leverage scores*, a common measure of the importance of rows of a matrix. We denote the leverage scores of a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ by $\vec{\sigma} \in \mathbb{R}^n$ and the *leverage score of row* $i \in [n]$ by $\sigma_i \stackrel{\text{def}}{=} [\mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T]_{ii}$. For $\mathbf{A} \in \mathbb{R}^{n \times d}$, $\vec{d} \in \mathbb{R}^n_{>0}$, and $\mathbf{D} \stackrel{\text{def}}{=} \mathbf{diag}(\vec{d})$ we use the shorthand $\vec{\sigma}_{\mathbf{A}}(\vec{d})$ to denote the leverage scores of the matrix $\mathbf{D}^{1/2}\mathbf{A}$. We frequently use well known facts regarding leverage scores, such as $\sigma_i \in [0,1]$ and $\left\|\vec{\sigma}\right\|_1 \leq d$. (See [240, 189, 183, 47] for a more in-depth discussion of leverage scores, their properties, and their many applications.) In addition, we make use of the fact that given an efficient linear system solver of $\mathbf{A}^T\mathbf{A}$ we can efficiently compute multiplicative approximations to leverage scores (See Definition 4 and Lemma 5 below).

Definition 4 (Linear System Solver)**.** An algorithm $\mathtt{S}$ is a LO-time solver of a PD matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ if for all $\vec{b} \in \mathbb{R}^n$ and $\epsilon \in (0, 1/2]$, the algorithm outputs a vector $\mathtt{S}(\vec{b}, \epsilon) \in \mathbb{R}^n$ in time $O(\mathrm{LO} \cdot \log(\epsilon^{-1}))$ such that with high probability in n, $\left\|\mathtt{S}(\vec{b}, \epsilon) - \mathbf{M}^{-1}\vec{b}\right\|_{\mathbf{M}}^2 \leq \epsilon \left\|\mathbf{M}^{-1}\vec{b}\right\|_{\mathbf{M}}^2$.

Lemma 5 (Computing Leverage Scores [240])**.** *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$, let $\vec{\sigma}$ denote the leverage scores of $\mathbf{A}$, and let $\epsilon > 0$. If we have a* LO*-time solver for $\mathbf{A}^T\mathbf{A}$ then in time $\tilde{O}((\mathrm{nnz}(\mathbf{A}) + \mathrm{LO})\epsilon^{-2})$ we can compute $\vec{\tau} \in \mathbb{R}^n$ such that with high probability in d, $(1-\epsilon)\sigma_i \leq \tau_i \leq (1+\epsilon)\sigma_i$ for all $i \in [n]$.*

4.2 Hybrid Barrier Function

As explained in Section 3.3 our cutting plane method maintains a polytope $P = \{\vec{x} \in \mathbb{R}^n : \mathbf{A}\vec{x} \geq \vec{b}\}$ for $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\vec{b} \in \mathbb{R}^n$ that contains some target set K. We then maintain a minimizer of the following hybrid barrier function:

$$p_{\vec{e}}(\vec{x}) \stackrel{\text{def}}{=} -\sum_{i \in [m]} (c_e + e_i) \log\left(s_i(\vec{x})/R\right) + \frac{1}{2} \log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) + \frac{\lambda}{2}\left\|x\right\|_2^2$$

where $\vec{e} \in \mathbb{R}^m$ is a variable we maintain, $c_e \geq 0$ and $\lambda \geq 0$ are constants we fix later, $\vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b}$, $\mathbf{S}_x = \mathbf{diag}(\vec{s}(\vec{x}))$, and R is the radius of the box containing P. When the meaning is clear from context we often use the shorthand $\mathbf{A}_x \stackrel{\text{def}}{=} \mathbf{S}_x^{-1}\mathbf{A}$. The R terms are only used in the proof and can be ignored in the algorithm because it only adds a constant term to the barrier function.

Rather than maintaining $\vec{e}$ explicitly, we instead maintain a vector $\vec{\tau} \in \mathbb{R}^m$ that approximates the leverage score

$$\vec{\psi}(\vec{x}) \stackrel{\text{def}}{=} \operatorname{diag}\left(\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x + \lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\right)$$

and pick $\vec{e}$ using the function $\vec{e}_P(\vec{\tau},\vec{x}) \stackrel{\text{def}}{=} \vec{\tau} - \vec{\psi}(\vec{x})$. Note that $\vec{\psi}(\vec{x})$ is simply the leverage scores of the top rows of the matrix

$$\left[\begin{array}{c}\mathbf{A}_x\\ \sqrt{\lambda}\mathbf{I}\end{array}\right].$$

and therefore the usual properties of leverage scores hold, i.e. $\psi_i(\vec{x}) \in (0,1)$ and $\left\|\psi_i(\vec{x})\right\|_1 \leq n$. We write $\vec{\psi}(\vec{x})$ equivalently as $\vec{\psi}_x$ or $\vec{\psi}_P$ when we want the matrix to be clear. Furthermore, we let $\mathbf{\Psi}_x \stackrel{\text{def}}{=} \mathbf{diag}(\vec{\psi}(\vec{x}))$ and $\mu(\vec{x}) \stackrel{\text{def}}{=} \min_i \psi_i(\vec{x})$. Again, we use the subscripts of x and P interchangeably and often drop them when the meaning is clear from context.

We remark that the last term $\frac{\lambda}{2}\left\|x\right\|_2^2$ ensures that our point is always within a certain region (Lemma 23) and hence the term $(c_e + e_i)\log s_i(\vec{x})_i$ never gets too large. However, this ℓ^2 term changes the Hessian of the barrier function and hence we need to put a $\lambda\mathbf{I}$ term inside both the $\log\det$ and the leverage score to reflect this. This is the reason why we use $\vec{\psi}$ instead of the standard leverage score.

5 Our Cutting Plane Method

In this section we develop and prove the correctness of our cutting plane method. We use the notation introduced in Sections 2 and 4 as well as the technical tools introduced later in Section 6.

We break the presentation and proof of correctness of our cutting plane methods into multiple parts. First in Section 5.1 we describe how we maintain an approximate center/minimizer of the hybrid barrier function $p_{\vec{e}}$ and analyze this procedure. Then, in Section 5.2 we carefully analyze the effect of changing constraints on the hybrid barrier function and in Section 5.3 we prove properties of an approximate center of hybrid barrier function, which we call the hybrid center. In Section 5.4 we then provide our cutting plane method and in Section 5.5 we prove that the cutting plane method solves the feasibility problem as desired.

5.1 Centering

In this section we show how to compute approximate *centers* or minimizers of the hybrid barrier function for the current polytope $P = \{\vec{x} : \mathbf{A}\vec{x} \geq \vec{b}\}$. We split this proof up into multiple parts. First we simply bound the gradient and Hessian of the hybrid barrier function, $p_{\vec{e}}$, as follows.

Lemma 6. *For $f(\vec{x}) \stackrel{\text{def}}{=} \frac{1}{2}\log\det\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}\right)$, we have that*

$$\nabla f(\vec{x}) = -\mathbf{A}_x^T\vec{\psi}(\vec{x}) \quad \text{and} \quad \mathbf{A}_x^T\mathbf{\Psi}(\vec{x})\mathbf{A}_x \preceq \nabla^2 f(\vec{x}) \preceq 3\mathbf{A}_x^T\mathbf{\Psi}(\vec{x})\mathbf{A}_x \quad .$$

Proof. Our proof is similar to [7, Appendix] which proved the statement when $\lambda = 0$. This case does not change the derivation significantly, however for completeness we include the proof below.

We take derivatives on $\vec{s}$ first and then apply chain rule. Let $f(\vec{s}) = \frac{1}{2}\log\det\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)$. We use the notation $Df(\vec{x})[\vec{h}]$ to denote the directional derivative of f along the direction $\vec{h}$ at the point $\vec{x}$. Using the standard formula for the derivative of log det, i.e. $\frac{d}{dt}\log\det\mathbf{B}_t = \mathrm{Tr}((\mathbf{B}_t)^{-1}(\frac{d\mathbf{B}_t}{dt}))$, and the commutative property $\mathrm{Tr}(\mathbf{KJ}) = \mathrm{Tr}(\mathbf{JK})$, we have

$$\begin{aligned} Df(\vec{s})[\vec{h}] &= \frac{1}{2}\mathrm{Tr}((\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I})^{-1}(\mathbf{A}^T(-2)\mathbf{S}^{-3}\mathbf{H}\mathbf{A})) \\ &= -\sum_i \frac{h_i}{s_i}\vec{\mathbb{1}}_i^T\mathbf{S}^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{S}^{-1}\vec{\mathbb{1}}_i = -\sum_i \frac{\psi_i h_i}{s_i} \quad . \end{aligned} \tag{5.1}$$

Applying chain rules, we have $\nabla f(\vec{x}) = -\mathbf{A}_x^T\vec{\psi}$. Now let $\mathbf{P} \stackrel{\text{def}}{=} \mathbf{S}^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{S}^{-1}$. Taking the derivative of (5.1) again and using the commutative property of trace, we have

$$\begin{aligned} D^2 f(\vec{s})[\vec{h}_1, \vec{h}_2] &= \mathrm{Tr}\left(\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{A}^T(-2)\mathbf{S}^{-3}\mathbf{H}_2\mathbf{A}\right)\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{A}^T\mathbf{S}^{-3}\mathbf{H}_1\mathbf{A}\right)\right) \\ &\quad -\mathrm{Tr}\left(\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\left(\mathbf{A}^T(-3)\mathbf{S}^{-4}\mathbf{H}_2\mathbf{H}_1\mathbf{A}\right)\right) \\ &= 3\mathrm{Tr}\left(\mathbf{P}\mathbf{S}^{-2}\mathbf{H}_2\mathbf{H}_1\right) - 2\mathrm{Tr}\left(\mathbf{P}\mathbf{S}^{-1}\mathbf{H}_2\mathbf{P}\mathbf{S}^{-1}\mathbf{H}_1\right) \\ &= 3\sum_i P_{ii}\frac{\vec{h}_1(i)\vec{h}_2(i)}{s_i^2} - 2\sum_{ij} P_{ij}\frac{\vec{h}_2(j)}{s_j}P_{ji}\frac{\vec{h}_2(i)}{s_i} \\ &= 3\sum_i \psi_i\frac{\vec{h}_1(i)\vec{h}_2(i)}{s_i^2} - 2\sum_{ij} P_{ij}^2\frac{\vec{h}_2(j)}{s_j}\frac{\vec{h}_2(i)}{s_i} \quad . \end{aligned}$$

Consequently, $D^2 f(\vec{x})[\vec{\mathbb{1}}_i, \vec{\mathbb{1}}_j] = [\mathbf{S}^{-1}\left(3\mathbf{\Psi} - 2\mathbf{P}^{(2)}\right)\mathbf{S}^{-1}]_{ij}$ where $\mathbf{P}^{(2)}$ is the Schur product of $\mathbf{P}$ with itself.

Now note that

$$\begin{aligned} \sum_i P_{ij}^2 &= \vec{\mathbb{1}}_j\mathbf{S}^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{S}^{-1}\vec{\mathbb{1}}_j \\ &\leq \vec{\mathbb{1}}_j\mathbf{S}^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{S}^{-1}\vec{\mathbb{1}}_j = P_{jj} = \mathbf{\Psi}_{jj} \quad . \end{aligned}$$

Hence, the Gershgorin circle theorem shows that the eigenvalues of $\mathbf{\Psi} - \mathbf{P}^{(2)}$ are lies in union of the interval $[0, 2\psi_j]$ over all j. Hence, $\mathbf{\Psi} - \mathbf{P}^{(2)} \succeq \mathbf{0}$. On the other hand, Schur product theorem shows that $\mathbf{P}^{(2)} \succeq \mathbf{0}$ as $\mathbf{P} \succeq \mathbf{0}$. Hence, the result follows by chain rule. □

Lemma 6 immediately shows that under our choice of $\vec{e} = \vec{e}_P(\vec{x}, \vec{\tau})$ we can compute the gradient of the hybrid barrier function, $p_{\vec{e}}(\vec{x})$ efficiently. Formally, Lemma 6 immediately implies the following:

Lemma 7 (Gradient). *For $\vec{x} \in P = \{\vec{y} \in \mathbb{R}^n : \mathbf{A}\vec{y} \geq \vec{b}\}$ and $\vec{e} \in \mathbb{R}^m$ we have*

$$\bigtriangledown p_{\vec{e}}(\vec{x}) = -\mathbf{A}_x^T(c_e\vec{\mathbb{1}} + \vec{e} + \vec{\psi}_P(\vec{x})) + \lambda\vec{x}$$

and therefore for all $\vec{\tau} \in \mathbb{R}^m$, we have

$$\bigtriangledown p_{\vec{e}(\vec{\tau},\vec{x})}(\vec{x}) = -\mathbf{A}_x^T \left(c_e \vec{\mathbb{1}} + \vec{\tau} \right) + \lambda \vec{x}.$$

Remark 8. To be clear, the vector $\bigtriangledown p_{\vec{e}(\vec{\tau},\vec{x})}(\vec{x})$ is defined as the vector such that

$$[\bigtriangledown p_{\vec{e}(\vec{\tau},\vec{x})}(\vec{x})]_i = \lim_{t \to 0} \frac{1}{t} \left(p_{\vec{e}(\vec{\tau},\vec{x})}(\vec{x} + t\vec{\mathbb{1}}_i) - p_{\vec{e}(\vec{\tau},\vec{x})}(\vec{x}) \right) \quad .$$

In other words, we treat the parameter $\vec{e}(\vec{\tau},\vec{x})$ as fixed. This is the reason we denote it by subscript to emphasize that $p_{\vec{e}}(\vec{x})$ is a family of functions, $p_{\vec{e}(\vec{\tau},\vec{x})}$ is one particular function, and $\nabla p_{\vec{e}(\vec{\tau},\vec{x})}$ means taking gradient on that particular function.

Consequently, we can always compute $\bigtriangledown p_{\vec{e}(\vec{\tau},\vec{x})}(\vec{x})$ efficiently. Now, we measure *centrality* or how close we are to the hybrid center as follows.

Definition 9 (Centrality)**.** For $\vec{x} \in P = \left\{ \vec{y} \in \mathbb{R}^n \, : \, \mathbf{A}\vec{y} \geq \vec{b} \right\}$ and $\vec{e} \in \mathbb{R}^m$, we define the *centrality* of $\vec{x}$ by

$$\delta_{\vec{e}}(\vec{x}) \stackrel{\text{def}}{=} \left\| \bigtriangledown p_{\vec{e}}(\vec{x}) \right\|_{\mathbf{H}(\vec{x})^{-1}}$$

where $\mathbf{H}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}_x^T \left(c_e \mathbf{I} + \mathbf{\Psi}(\vec{x}) \right) \mathbf{A}_x + \lambda \mathbf{I}$. Often, we use *weights* $\vec{w} \in \mathbb{R}^m_{>0}$ to approximate this Hessian and consider $\mathbf{Q}(\vec{x},\vec{w}) \stackrel{\text{def}}{=} \mathbf{A}_x^T \left(c_e \mathbf{I} + \mathbf{W} \right) \mathbf{A}_x + \lambda \mathbf{I}$.

Next, we bound how much slacks can change in a region close to a nearly central point.

Lemma 10. *Let $\vec{x} \in P = \left\{ \vec{y} \in \mathbb{R}^n \, : \, \mathbf{A}\vec{y} \geq \vec{b} \right\}$ and $\vec{y} \in \mathbb{R}^n$ such that $\left\| \vec{x} - \vec{y} \right\|_{\mathbf{H}(\vec{x})} \leq \epsilon \sqrt{c_e + \mu(\vec{x})}$ for $\epsilon < 1$. Then $\vec{y} \in P$ and $(1-\epsilon)\mathbf{S}_x \preceq \mathbf{S}_y \preceq (1+\epsilon)\mathbf{S}_x$.*

Proof. Direct calculation reveals the following:

$$\begin{aligned} \left\| \mathbf{S}_x^{-1}(\vec{s}_{\vec{y}} - \vec{s}_x) \right\|_\infty &\leq \left\| \mathbf{A}_x(\vec{y} - \vec{x}) \right\|_2 \leq \frac{1}{\sqrt{c_e + \mu(\vec{x})}} \left\| \mathbf{A}_x(\vec{y} - \vec{x}) \right\|_{c_e\mathbf{I} + \mathbf{\Psi}(\vec{x})} \\ &\leq \frac{1}{\sqrt{c_e + \mu(\vec{x})}} \left\| \vec{y} - \vec{x} \right\|_{\mathbf{H}(\vec{x})} \leq \epsilon \quad . \end{aligned}$$

Consequently, $(1-\epsilon)\mathbf{S}_x \preceq \mathbf{S}_y \preceq (1+\epsilon)\mathbf{S}_x$. Since $y \in P$ if and only if $\mathbf{S}_y \succeq \mathbf{0}$ the result follows. □

Combining the previous lemmas we obtain the following.

Lemma 11. *Let $\vec{x} \in P = \{\vec{y} \in \mathbb{R}^n \, : \, \mathbf{A}\vec{y} \geq \vec{b}\}$ and $\vec{e}, \vec{w} \in \mathbb{R}^m$ such that $\left\|\vec{e}\right\|_\infty \leq \frac{1}{2}c_e \leq 1$ and $\mathbf{\Psi}(\vec{x}) \preceq \mathbf{W} \preceq \frac{4}{3}\mathbf{\Psi}(\vec{x})$. If $\vec{y} \in \mathbb{R}^n$ satisfies $\left\| \vec{x} - \vec{y} \right\|_{\mathbf{Q}(\vec{x},\vec{w})} \leq \frac{1}{10}\sqrt{c_e + \mu(\vec{x})}$, then*

$$\frac{1}{4}\mathbf{Q}(\vec{x},\vec{w}) \preceq \bigtriangledown^2 p_{\vec{e}}(\vec{y}) \preceq 8\mathbf{Q}(\vec{x},\vec{w}) \quad \text{and} \quad \frac{1}{2}\mathbf{H}(\vec{x}) \preceq \mathbf{H}(\vec{y}) \preceq 2\mathbf{H}(\vec{x}) \quad .$$

Also, we have that $\mathbf{H}(\vec{x}) \preceq \mathbf{Q}(\vec{x},\vec{w}) \preceq \frac{4}{3}\mathbf{H}(\vec{x})$.

Proof. Lemma 6 shows that

$$\mathbf{A}_y^T\left(c_e\mathbf{I}+\mathbf{E}+\mathbf{\Psi}(\vec{y})\right)\mathbf{A}_y+\lambda\mathbf{I}\preceq\bigtriangledown^2 p_{\vec{e}}(\vec{y})\preceq\mathbf{A}_y^T\left(c_e\mathbf{I}+\mathbf{E}+3\mathbf{\Psi}(\vec{y})\right)\mathbf{A}_y+\lambda\mathbf{I}\quad. \tag{5.2}$$

Since $\mathbf{W}\succeq\mathbf{\Psi}$, we have that $\mathbf{Q}(\vec{x},\vec{w})\succeq\mathbf{H}(\vec{x})$ and therefore $\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\leq\epsilon\sqrt{c_e+\mu(\vec{x})}$ with $\epsilon=0.1$. Consequently, by Lemma 10 we have $(1-\epsilon)\mathbf{S}_x\preceq\mathbf{S}_y\preceq(1+\epsilon)\mathbf{S}_x$ and therefore

$$\frac{(1-\epsilon)^2}{(1+\epsilon)^2}\mathbf{\Psi}(\vec{x})\preceq\mathbf{\Psi}(\vec{y})\preceq\frac{(1+\epsilon)^2}{(1-\epsilon)^2}\mathbf{\Psi}(\vec{x})$$

and

$$\frac{1}{2}\mathbf{H}(\vec{x})\preceq\frac{(1-\epsilon)^2}{(1+\epsilon)^4}\mathbf{H}(\vec{x})\preceq\mathbf{H}(\vec{y})\preceq\frac{(1+\epsilon)^2}{(1-\epsilon)^4}\mathbf{H}(\vec{x})\preceq2\mathbf{H}(\vec{x})$$

Furthermore, (5.2) shows that

$$\begin{aligned}\bigtriangledown^2 p_{\vec{e}}(\vec{y})&\preceq\mathbf{A}_y^T\left(c_e\mathbf{I}+\mathbf{E}+3\mathbf{\Psi}(\vec{y})\right)\mathbf{A}_y+\lambda\mathbf{I}\\&\preceq\frac{(1+\epsilon)^2}{(1-\epsilon)^4}\mathbf{A}_x^T\left(c_e\mathbf{I}+\mathbf{E}+3\mathbf{\Psi}(\vec{x})\right)\mathbf{A}_x+\lambda\mathbf{I}\\&\preceq2\mathbf{A}_x^T\left(\frac{3}{2}c_e\mathbf{I}+3\mathbf{W}\right)\mathbf{A}_x+\lambda\mathbf{I}\preceq8\mathbf{Q}(\vec{x},\vec{w})\end{aligned}$$

and

$$\begin{aligned}\bigtriangledown^2 p_{\vec{e}}(\vec{y})&\succeq\mathbf{A}_y^T\left(c_e\mathbf{I}+\mathbf{E}+\mathbf{\Psi}(\vec{y})\right)\mathbf{A}_y+\lambda\mathbf{I}\\&\succeq\frac{(1-\epsilon)^4}{(1+\epsilon)^2}\mathbf{A}_x^T\left(c_e\mathbf{I}+\mathbf{E}+\mathbf{\Psi}(\vec{x})\right)\mathbf{A}_x+\lambda\mathbf{I}\\&\succeq\frac{1}{2}\mathbf{A}_x^T\left(\frac{1}{2}c_e\mathbf{I}+\frac{3}{4}\mathbf{W}\right)\mathbf{A}_x+\lambda\mathbf{I}\succeq\frac{1}{4}\mathbf{Q}(\vec{x},\vec{w}).\end{aligned}$$

The last inequality follows from the definition of $\mathbf{H}(\vec{x})$ and $\mathbf{Q}(\vec{x},\vec{w})$ and the fact $\mathbf{\Psi}(\vec{x})\preceq\mathbf{W}\preceq\frac{4}{3}\mathbf{\Psi}(\vec{x})$. □

To analyze our centering scheme we use standard facts about gradient descent we prove in Lemma 12.

Lemma 12 (Gradient Descent)**.** *Let $f:\mathbb{R}^n\rightarrow\mathbb{R}$ be twice differentiable and $\mathbf{Q}\in\mathbb{R}^{n\times n}$ be positive definite. Let $\vec{x}_0\in\mathbb{R}^n$ and $\vec{x}_1\stackrel{\text{def}}{=}\vec{x}_0-\frac{1}{L}\mathbf{Q}^{-1}\bigtriangledown f(\vec{x}_0)$. Furthermore, let $\vec{x}_\alpha=\vec{x}_0+\alpha(\vec{x}_1-\vec{x})$ and suppose that $\mu\mathbf{Q}\preceq\bigtriangledown^2 f(\vec{x}_\alpha)\preceq L\mathbf{Q}$ for all $\alpha\in[0,1]$. Then,*

1. $\left\|\bigtriangledown f(\vec{x}_1)\right\|_{\mathbf{Q}^{-1}}\leq\left(1-\frac{\mu}{L}\right)\left\|\bigtriangledown f(\vec{x}_0)\right\|_{\mathbf{Q}^{-1}}$
2. $f(\vec{x}_1)\geq f(\vec{x}_0)-\frac{1}{L}\left\|\bigtriangledown f(\vec{x}_0)\right\|_{\mathbf{Q}^{-1}}^2$

Proof. Integrating we have that

$$\bigtriangledown f(\vec{x}_1)=\bigtriangledown f(\vec{x}_0)+\int_0^1\bigtriangledown^2 f(\vec{x}_\alpha)(\vec{x}_1-\vec{x}_0)d\alpha=\int_0^1\left(\mathbf{Q}-\frac{1}{L}\bigtriangledown^2 f(\vec{x}_\alpha)\right)\mathbf{Q}^{-1}\bigtriangledown f(\vec{x}_0)d\alpha$$

Consequently, by applying Jensen's inequality we have

$$\begin{aligned}\left\| \bigtriangledown f(\vec{x}_1)\right\|_{\mathbf{Q}^{-1}} &= \left\|\int_0^1 \left(\mathbf{Q} - \frac{1}{L}\bigtriangledown^2 f(\vec{x}_\alpha)\right)\mathbf{Q}^{-1}\bigtriangledown f(\vec{x}_0)d\alpha\right\|_{\mathbf{Q}^{-1}} \\ &\leq \int_0^1 \left\|\left(\mathbf{Q} - \frac{1}{L}\bigtriangledown^2 f(\vec{x}_\alpha)\right)\mathbf{Q}^{-1}\bigtriangledown f(\vec{x}_0)\right\|_{\mathbf{Q}^{-1}} d\alpha \\ &\leq \left\|\mathbf{Q}^{-1/2}\bigtriangledown f(\vec{x}_0)\right\|_{\left[\mathbf{Q}^{-1/2}\left(\mathbf{Q}-\frac{1}{L}\bigtriangledown^2 f(\vec{x}_\alpha)\right)\mathbf{Q}^{-1/2}\right]^2}\end{aligned}$$

Now we know that by assumption that

$$\mathbf{0} \preceq \mathbf{Q}^{-1/2}\left(\mathbf{Q} - \frac{1}{L}\bigtriangledown^2 f(\vec{x}_\alpha)\right)\mathbf{Q}^{-1/2} \preceq \left(1 - \frac{\mu}{L}\right)\mathbf{I}$$

and therefore combining these (1) holds.

Using the convexity of f, we have

$$\begin{aligned} f(\vec{x}_1) &\geq f(\vec{x}_0) + \langle \bigtriangledown f(\vec{x}_0), \vec{x}_1 - \vec{x}_0\rangle \\ &\geq f(\vec{x}_0) - \left\| \bigtriangledown f(\vec{x}_0)\right\|_{\mathbf{Q}^{-1}}\left\|\vec{x}_1 - \vec{x}_0\right\|_{\mathbf{Q}}\end{aligned}$$

and since $\left\|\vec{x}_1 - \vec{x}_0\right\|_{\mathbf{Q}} = \frac{1}{L}\left\| \bigtriangledown f(\vec{x}_0)\right\|_{\mathbf{Q}^{-1}}$, (2) holds as well. □

Next we bound the effect of changing $\vec{e}$ on the hybrid barrier function $p_{\vec{e}}(\vec{x})$.

Lemma 13. *For $\vec{x} \in P = \{\vec{y} \in \mathbb{R}^n : \mathbf{A}\vec{y} \geq \vec{b}\}$, $\vec{e}, \vec{f} \in \mathbb{R}^m$, and $\vec{w} \in \mathbb{R}^m_{>0}$ such that $\mathbf{W} \succeq \mathbf{\Psi}(\vec{x})$*

$$\left\| \bigtriangledown p_{\vec{f}}(\vec{x})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} \leq \left\| \bigtriangledown p_{\vec{e}}(\vec{x})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} + \frac{1}{\sqrt{c_e + \mu(\vec{x})}}\left\|\vec{f} - \vec{e}\right\|_2$$

Proof. Direct calculation shows the following

$$\begin{aligned}\left\| \bigtriangledown p_{\vec{f}}(\vec{x})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} &= \left\| -\mathbf{A}_x^T(c_e\vec{\mathbb{1}} + \vec{f} + \vec{\psi}_P(\vec{x})) + \lambda\vec{x}\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} && \text{(Formula for } \bigtriangledown p_{\vec{f}}(\vec{x})) \\ &\leq \left\| \bigtriangledown p_{\vec{e}}(\vec{x})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} + \left\|\mathbf{A}_x^T(\vec{f} - \vec{e})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} && \text{(Triangle inequality)} \\ &\leq \left\| \bigtriangledown p_{\vec{e}}(\vec{x})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} + \frac{1}{\sqrt{c_e + \mu(\vec{x})}}\left\|\mathbf{A}_x^T(\vec{f} - \vec{e})\right\|_{(\mathbf{A}_x^T\mathbf{A}_x)^{-1}} \\ & && \text{(Bound on } \mathbf{Q}(\vec{x},\vec{w})) \\ &\leq \left\| \bigtriangledown p_{\vec{e}}(\vec{x})\right\|_{\mathbf{Q}(\vec{x},\vec{w})^{-1}} + \frac{1}{\sqrt{c_e + \mu(\vec{x})}}\left\|\vec{f} - \vec{e}\right\|_2 \\ & && \text{(Property of projection matrix)}\end{aligned}$$

where in the second to third line we used $\mathbf{Q}(\vec{x},\vec{w}) \succeq \mathbf{H}(\vec{x}) \succeq (c_e + \mu(\vec{x}))\mathbf{A}_x^T\mathbf{A}_x$. □

We now have everything we need to analyze our centering algorithm.

Algorithm 1: $(\vec{x}^{(r)}, \vec{\tau}^{(r)}) = \mathtt{Centering}(\vec{x}^{(0)}, \vec{\tau}^{(0)}, r, c_\Delta)$

```
Input: Initial point $\vec{x}^{(0)} \in P = \{\vec{y} \in \mathbb{R}^n : \mathbf{A}\vec{y} \geq \vec{b}\}$, Estimator of leverage scores $\vec{\tau}^{(0)} \in \mathbb{R}^n$
Input: Number of iterations $r > 0$, Accuracy of the estimator $0 \leq c_\Delta \leq 0.01 c_e$.
Given: $\|\vec{e}^{(0)}\|_\infty \leq \frac{1}{3}c_e \leq \frac{1}{3}$ where $\vec{e}^{(0)} = \vec{e}(\vec{\tau}^{(0)}, \vec{x}^{(0)})$.
Given: $\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)}) = \|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\|_{\mathbf{H}(\vec{x}^{(0)})^{-1}} \leq \frac{1}{100}\sqrt{c_e + \mu(\vec{x}^{(0)})}$.
Compute $\vec{w}$ such that $\mathbf{\Psi}(\vec{x}^{(0)}) \preceq \mathbf{W} \preceq \frac{4}{3}\mathbf{\Psi}(\vec{x}^{(0)})$ (See Lemma 5)
Let $\mathbf{Q} \overset{\text{def}}{=} \mathbf{Q}(\vec{x}^{(0)}, \vec{w})$.
for $k = 1$ to $r$ do
    $\vec{x}^{(k)} := \vec{x}^{(k-1)} - \frac{1}{8}\mathbf{Q}^{-1} \bigtriangledown p_{\vec{e}^{(k-1)}}(\vec{x}^{(k-1)})$.
    Sample $\vec{\Delta}^{(k)} \in \mathbb{R}^n$ s.t.
        $\mathbf{E}[\vec{\Delta}^{(k)}] = \vec{\psi}(\vec{x}^{(k)}) - \vec{\psi}(\vec{x}^{(k-1)})$ and
        with high probability in $n$,
        $\|\vec{\Delta}^{(k)} - (\vec{\psi}(\vec{x}^{(k)}) - \vec{\psi}(\vec{x}^{(k-1)}))\|_2 \leq c_\Delta \|\mathbf{S}^{-1}_{\vec{x}^{(k-1)}}(\vec{s}_{\vec{x}^{(k)}} - \vec{s}_{\vec{x}^{(k-1)}})\|_2$ (See Section 6.1)
    $\vec{\tau}^{(k)} := \vec{\tau}^{(k-1)} + \vec{\Delta}^{(k)}$.
    $\vec{e}^{(k)} := \vec{e}(\vec{\tau}^{(k)}, \vec{x}^{(k)})$.
end
Output: $(\vec{x}^{(r)}, \vec{\tau}^{(r)})$
```

Lemma 14. *Let $\vec{x}^{(0)} \in P = \{\vec{y} \in \mathbb{R}^n : \mathbf{A}\vec{y} \geq \vec{b}\}$ and let $\vec{\tau}^{(0)} \in \mathbb{R}^m$ such that $\|\vec{e}(\vec{\tau}^{(0)}, \vec{x}^{(0)})\|_\infty \leq \frac{1}{3}c_e \leq \frac{1}{3}$. Assume that r is a positive integer, $0 \leq c_\Delta \leq 0.01 c_e$ and $\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)}) \leq \frac{1}{100}\sqrt{c_e + \mu(\vec{x}^{(0)})}$. With high probability in n, the algorithm* $\mathtt{Centering}(\vec{x}^{(0)}, \vec{\tau}^{(0)}, r, c_\Delta)$ *outputs $(\vec{x}^{(r)}, \vec{\tau}^{(r)})$ such that*

1. $\delta_{\vec{e}^{(r)}}(\vec{x}^{(r)}) \leq 2\left(1 - \frac{1}{64}\right)^r \delta_{\vec{e}^{(0)}}(\vec{x}^{(0)})$.
2. $\mathbf{E}[p_{\vec{e}^{(k)}}(\vec{x}^{(r)})] \geq p_{\vec{e}^{(0)}}(\vec{x}^{(0)}) - 8\left(\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right)^2$.
3. $\mathbf{E}\vec{e}^{(r)} = \vec{e}^{(0)}$ *and* $\|\vec{e}^{(r)} - \vec{e}^{(0)}\|_2 \leq \frac{1}{10}c_\Delta$.
4. $\|\mathbf{S}^{-1}_{\vec{x}^{(0)}}(\vec{s}(\vec{x}^{(r)}) - \vec{s}(\vec{x}^{(0)}))\|_2 \leq \frac{1}{10}$.

where $\vec{e}^{(r)} = \vec{e}(\vec{\tau}^{(r)}, \vec{x}^{(r)})$.

Proof. Let $\eta = \|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\|_{\mathbf{Q}^{-1}}$. First, we use induction to prove that $\|\vec{x}^{(r)} - \vec{x}^{(0)}\|_{\mathbf{Q}} \leq 8\eta$, $\|\bigtriangledown p_{\vec{e}^{(r)}}(\vec{x}^{(r)})\|_{\mathbf{Q}^{-1}} \leq \left(1 - \frac{1}{64}\right)^r \eta$ and $\|\vec{e}^{(r)} - \vec{e}^{(0)}\|_2 \leq \frac{1}{10}c_\Delta$ for all r.

Clearly the claims hold for $r = 0$. We now suppose they hold for all $r \leq t$ and show that they hold for $r = t+1$. Now, since $\|\vec{x}^{(t)} - \vec{x}^{(0)}\|_{\mathbf{Q}} \leq 8\eta$, $\vec{x}^{(t+1)} = \vec{x}^{(t)} - \frac{1}{8}\mathbf{Q}^{-1} \bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})$, and $\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\|_{\mathbf{Q}^{-1}} \leq \left(1 - \frac{1}{64}\right)^t \eta \leq \eta$, we have

$$\|\vec{x}^{(t+1)} - \vec{x}^{(0)}\|_{\mathbf{Q}} \leq \|\vec{x}^{(t)} - \vec{x}^{(0)}\|_{\mathbf{Q}} + \frac{1}{8}\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\|_{\mathbf{Q}^{-1}} \leq 9\eta.$$

We will improve this estimate later in the proof to finish the induction on $\left\|\vec{x}^{(t+1)}-\vec{x}^{(0)}\right\|_{\mathbf{Q}}$, but using this, $\eta \leq 0.01\sqrt{c_e+\mu(\vec{x}^{(0)})}$, and $\left\|\vec{e}^{(t)}\right\|_\infty \leq \left\|\vec{e}^{(t)}-\vec{e}^{(0)}\right\|_\infty + \left\|\vec{e}^{(0)}\right\|_\infty \leq \frac{c_e}{2}$, we can invoke Lemma 11 and Lemma 12 and therefore

$$\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t+1)})\right\|_{\mathbf{Q}^{-1}} \leq \left(1-\frac{1}{32}\right)\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}}.$$

By Lemma 13 we have

$$\left\|\bigtriangledown p_{\vec{e}^{(t+1)}}(\vec{x}^{(t+1)})\right\|_{\mathbf{Q}^{-1}} \leq \left(1-\frac{1}{32}\right)\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}} + \frac{1}{\sqrt{c_e+\mu(\vec{x}^{(0)})}}\left\|\vec{e}^{(t+1)}-\vec{e}^{(t)}\right\|_2. \quad (5.3)$$

To bound $\left\|\vec{e}^{(t+1)}-\vec{e}^{(t)}\right\|_2$, we use the definition of $\vec{\Delta}$ to shows that

$$\begin{aligned}\left\|\vec{e}^{(t+1)}-\vec{e}^{(t)}\right\|_2 &= \left\|\left(\vec{\tau}^{(t+1)}-\vec{\psi}(\vec{x}^{(t+1)})\right)-\left(\vec{\tau}^{(t)}-\vec{\psi}(\vec{x}^{(t)})\right)\right\|_2\\ &= \left\|\vec{\Delta}^{(t+1)}-\left(\vec{\psi}(\vec{x}^{(t+1)})-\vec{\psi}(\vec{x}^{(t)})\right)\right\|_2\\ &\leq c_\Delta\left\|\mathbf{S}_{x^{(t)}}^{-1}(\vec{s}_{x^{(t+1)}}-\vec{s}_{x^{(t)}})\right\|_2\end{aligned}$$

with high probability in n. Now, we note that Lemma 10 and the induction hypothesis $\left\|\vec{x}^{(t)}-\vec{x}^{(0)}\right\|_{\mathbf{H}(\vec{x}^{(0)})} \leq \left\|\vec{x}^{(t)}-\vec{x}^{(0)}\right\|_{\mathbf{Q}} \leq 8\eta$ shows that $(1-0.1)\mathbf{S}_{x^{(0)}} \preceq \mathbf{S}_{x^{(t)}} \preceq (1+0.1)\mathbf{S}_{x^{(0)}}$ and therefore

$$\begin{aligned}\left\|\vec{e}^{(t+1)}-\vec{e}^{(t)}\right\|_2 &\leq c_\Delta\left\|\mathbf{S}_{x^{(t)}}^{-1}(\vec{s}_{x^{(t)}}-\vec{s}_{x^{(t+1)}})\right\|_2\\ &\leq \frac{c_\Delta}{1-0.1}\left\|\mathbf{S}_{x^{(0)}}^{-1}\mathbf{A}\left(\vec{x}^{(t)}-\vec{x}^{(t+1)}\right)\right\|_2\\ &= \frac{c_\Delta}{1-0.1}\left\|\frac{1}{8}\mathbf{Q}^{-1}\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{A}^T\mathbf{S}_{x^{(0)}}^{-2}\mathbf{A}}\\ &\leq \frac{c_\Delta}{8\,(1-0.1)\sqrt{c_e+\mu(\vec{x}^{(0)})}}\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}} \quad (5.4)\end{aligned}$$

where in the last line we used $\min_{i\in[m]} w_i \geq \mu(\vec{x}^{(0)})$. Since $c_\Delta < 0.01c_e \leq 0.01$, by (5.3), we have

$$\begin{aligned}\left\|\bigtriangledown p_{\vec{e}^{(t+1)}}(\vec{x}^{(t+1)})\right\|_{\mathbf{Q}^{-1}} &\leq \left(1-\frac{1}{32}\right)\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}} + \frac{0.01}{8\,(1-0.1)}\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}}\\ &\leq \left(1-\frac{1}{64}\right)\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}}\quad.\end{aligned}$$

Furthermore, this implies that

$$\left\|\vec{x}^{(t+1)}-\vec{x}^{(0)}\right\|_{\mathbf{Q}} \leq \left\|\sum_{k=0}^{t}\frac{1}{8}\mathbf{Q}^{-1}\bigtriangledown p_{\vec{e}^{(k)}}(\vec{x}^{(k)})\right\|_{\mathbf{Q}^{-1}} \leq \frac{1}{8}\sum_{i=0}^{\infty}\left(1-\frac{1}{64}\right)^k\eta \leq \frac{64}{8}\eta = 8\eta\quad.$$

Similarly, we have that

$$\begin{aligned}\left\|\vec{e}^{(t+1)}-\vec{e}^{(0)}\right\|_2 &\leq \sum_{k=0}^{t}\frac{c_\Delta}{8\,(1-0.1)\sqrt{c_e+\mu(\vec{x}^{(0)})}}\left(1-\frac{1}{64}\right)^k \left\|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right\|_{\mathbf{Q}^{-1}}\\ &\leq \frac{8c_\Delta\eta}{(1-0.1)\sqrt{c_e+\mu(\vec{x}^{(0)})}} \leq \frac{8c_\Delta}{(1-0.1)\sqrt{c_e+\mu(\vec{x}^{(0)})}}\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)}) \leq \frac{1}{10}c_\Delta\end{aligned}$$

where we used $\eta=\left\|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right\|_{\mathbf{Q}^{-1}}\leq\left\|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right\|_{\mathbf{H}^{-1}}=\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)})$ and this finishes the induction on $\left\|\bigtriangledown p_{\vec{e}^{(t)}}(\vec{x}^{(t)})\right\|_{\mathbf{Q}^{-1}}$, $\left\|\vec{x}^{(t)}-\vec{x}^{(0)}\right\|_{\mathbf{Q}}$ and $\left\|\vec{e}^{(t)}-\vec{e}^{(0)}\right\|_2$.

Hence, for all r, Lemma 11 shows that

$$\begin{aligned}\delta_{\vec{e}^{(r)}}(\vec{x}^{(r)}) &= \left\|\bigtriangledown p_{\vec{e}^{(r)}}(\vec{x}^{(r)})\right\|_{\mathbf{H}(\vec{x}^{(r)})^{-1}} \leq \sqrt{2}\left\|\bigtriangledown p_{\vec{e}^{(r)}}(\vec{x}^{(r)})\right\|_{\mathbf{H}(\vec{x}^{(0)})^{-1}}\\ &\leq \sqrt{\frac{8}{3}}\left\|\bigtriangledown p_{\vec{e}^{(r)}}(\vec{x}^{(r)})\right\|_{\mathbf{Q}^{-1}} \leq \sqrt{\frac{8}{3}}\left(1-\frac{1}{64}\right)^r\left\|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right\|_{\mathbf{Q}^{-1}}\\ &\leq 2\left(1-\frac{1}{64}\right)^r\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)}).\end{aligned}$$

Using that $\mathbf{E}\vec{e}^{(t+1)}=\vec{e}^{(t)}$, we see that the expected change in function value is only due to the change while taking centering steps and therefore Lemma 12 shows that

$$\mathbf{E}[p_{\vec{e}^{(r)}}(\vec{x}^{(r)})] \geq p_{\vec{e}^{(0)}}(\vec{x}^{(0)})-\frac{1}{8}\sum_{k=0}^{\infty}\left(1-\frac{1}{64}\right)^{2k}\left\|\bigtriangledown p_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right\|_{\mathbf{Q}^{-1}}^2 \geq p_{\vec{e}^{(0)}}(\vec{x}^{(0)})-8\left(\delta_{\vec{e}^{(0)}}(\vec{x}^{(0)})\right)^2.$$

Finally, for (4), we note that

$$\left\|\frac{s(\vec{x}^{(r)})-s(\vec{x}^{(0)})}{s(\vec{x}^{(0)})}\right\|_2 = \left\|\vec{x}^{(r)}-\vec{x}^{(0)}\right\|_{\mathbf{A}^T\mathbf{S}_{x^{(0)}}^{-2}\mathbf{A}} \leq \frac{1}{\sqrt{\mu(\vec{x}^{(0)})+c_e}}\left\|\vec{x}^{(r)}-\vec{x}^{(0)}\right\|_{\mathbf{Q}^{-1}} \leq \frac{1}{10}.$$

□

5.2 Changing Constraints

Here we bound the effect that adding or a removing a constraint has on the hybrid barrier function. Much of the analysis in this section follows from the following lemma which follows easily from the Sherman Morrison Formula.

Lemma 15 (Sherman Morrison Formula Implications)**.** *Let $\mathbf{B}\in\mathbb{R}^{n\times n}$ be an invertible symmetric matrix and let $\vec{a}\in\mathbb{R}^n$ be arbitrary vector satisfying $\vec{a}^T\mathbf{B}^{-1}\vec{a}<1$. The following hold:*

1. $\left(\mathbf{B}\pm\vec{a}\vec{a}^T\right)^{-1}=\mathbf{B}^{-1}\mp\frac{\mathbf{B}^{-1}\vec{a}\vec{a}^T\mathbf{B}^{-1}}{1\pm\vec{a}^T\mathbf{B}^{-1}\vec{a}}$.

2. $\mathbf{0}\preceq\frac{\mathbf{B}^{-1}\vec{a}\vec{a}^T\mathbf{B}^{-1}}{1\pm\vec{a}^T\mathbf{B}^{-1}\vec{a}}\preceq\frac{\vec{a}^T\mathbf{B}^{-1}\vec{a}}{1\pm\vec{a}^T\mathbf{B}^{-1}\vec{a}}\mathbf{B}^{-1}$.

3. $\log\det\left(\mathbf{B}\pm\vec{a}\vec{a}^T\right)=\log\det\mathbf{B}+\log\left(1\pm\vec{a}^T\mathbf{B}^{-1}\vec{a}\right)$.

Proof. (1) follows immediately from Sherman Morrison [234]. (2) follows since $\vec{a}\vec{a}^T$ is PSD,

$$\frac{\mathbf{B}^{-1}\vec{a}\vec{a}^T\mathbf{B}^{-1}}{1 \pm \vec{a}^T\mathbf{B}^{-1}\vec{a}} = \mathbf{B}^{-1/2}\left[\frac{\mathbf{B}^{-1/2}\vec{a}\vec{a}^T\mathbf{B}^{-1/2}}{1 \pm \vec{a}^T\mathbf{B}^{-1}\vec{a}}\right]\mathbf{B}^{-1/2} \quad ,$$

and $\vec{y}\vec{y}^T \preceq \left\|\vec{y}\right\|_2^2\mathbf{I}$ for any vector $\vec{y}$. (3) follows immediately from the Matrix Determinant Lemma. □

We also make use of the following technical helper lemma.

Lemma 16. *For $\mathbf{A} \in \mathbb{R}^{n\times m}$ and all $\vec{a} \in \mathbb{R}^n$ we have*

$$\sum_{i\in[m]} \frac{1}{\psi_{\mathbf{A}}[i]}\left(\mathbf{A}\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)_i^4 \leq \left(\vec{a}^T\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)^2$$

Proof. We have by Cauchy Schwarz that

$$\left(\vec{\mathbb{1}}_i^T\mathbf{A}\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)^2 \leq \psi_{\mathbf{A}}[i]\cdot\vec{a}^T\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}$$

and consequently

$$\sum_{i\in[m]} \frac{\left(\vec{\mathbb{1}}_i^T\mathbf{A}\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)^4}{\psi_{\mathbf{A}}[i]} \leq \left(\vec{a}^T\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)\sum_{i\in[m]}\left(\vec{\mathbb{1}}_i\mathbf{A}\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)^2 .$$

Since

$$\begin{aligned}\sum_{i\in[m]}\left(\vec{\mathbb{1}}_i^T\mathbf{A}\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\right)^2 &= \vec{a}^T\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{A}\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a}\\ &\leq \vec{a}^T\left(\mathbf{A}^T\mathbf{A}+\lambda\mathbf{I}\right)^{-1}\vec{a},\end{aligned}$$

we have the desired result. □

We now bound the effect of adding a constraint.

Lemma 17. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\vec{b} \in \mathbb{R}^m$, $\vec{\tau} \in \mathbb{R}^m$, and $\vec{x} \in P \stackrel{\text{def}}{=} \left\{\vec{y} \in \mathbb{R}^n \,:\, \mathbf{A}\vec{y} \geq \vec{b}\right\}$. Let $\overline{\mathbf{A}} \in \mathbb{R}^{(m+1)\times n}$ be $\mathbf{A}$ with a row $\vec{a}_{m+1}$ added, let $\overline{b} \in \mathbb{R}^{m+1}$ be the vector $\vec{b}$ with an entry b_{m+1} added, and let $\overline{P} \stackrel{\text{def}}{=} \left\{\vec{y} \in \mathbb{R}^n \,:\, \overline{\mathbf{A}}\vec{y} \geq \overline{b}\right\}$. Let $s_{m+1} = \vec{a}_{m+1}^T\vec{x} - b_{m+1} > 0$, $\psi_a = \frac{\vec{a}_{m+1}^T(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I})^{-1}\vec{a}_{m+1}}{s_{m+1}^2}$.*

Now, let $\vec{\upsilon} \in \mathbb{R}^{m+1}$ be defined so that $\upsilon_{m+1} = \frac{\psi_a}{1+\psi_a}$ and for all $i \in [m]$

$$\upsilon_i = \tau_i - \frac{1}{1+\psi_a}\left[\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\frac{\vec{a}_{m+1}}{s_{m+1}}\right]_i^2 .$$

Then, the following hold

- [Leverage Score Estimation] $e_{\overline{P}}(\vec{\upsilon},\vec{x})_{m+1} = 0$ and $e_{\overline{P}}(\vec{\upsilon},\vec{x})_i = e_P(\vec{\tau},\vec{x})_i$ for all $i \in [m]$.

- [Function Value Increase] $p_{\vec{e}_{\overline{P}}(\vec{v},\vec{x})}(\vec{x}) = p_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) - c_e \log\left(s(\vec{x})_{m+1}/R\right) + \log(1+\psi_a)$.
- [Centrality Increase] $\delta_{\vec{e}_{\overline{P}}(\vec{v},\vec{x})}(\vec{x}) \leq \sqrt{1+\psi_a}\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) + \left(c_e\sqrt{1+\psi_a}+\psi_a\right)\sqrt{\frac{\psi_a}{\mu(\vec{x})}} + \psi_a$.

Proof. By (1) in Lemma 15, we have that for all $i \in [m]$

$$\psi_{\overline{P}}(\vec{x})_i = \psi_P(\vec{x})_i - \frac{1}{1+\psi_a}\left[\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\frac{\vec{a}_{m+1}}{s_{m+1}}\right]_i^2$$

and that

$$\psi_{\overline{P}}(\vec{x})_{m+1} = \psi_a - \frac{\psi_a^2}{1+\psi_a} = \frac{\psi_a}{1+\psi_a}.$$

Consequently [Leverage Score Estimation] holds.

By (3) in Lemma 15 we have that [Function Value Change] holds.

To bound the change in centrality, we first note that

$$\begin{aligned}\psi_{\overline{P}}(\vec{x})_i &\geq \psi_P(\vec{x})_i - \frac{1}{1+\psi_a}\left[\frac{\vec{a}_i^T}{s_i}\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\frac{\vec{a}_i}{s_i}\right]\left[\frac{\vec{a}_{m+1}^T}{s_{m+1}}\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\frac{\vec{a}_{m+1}}{s_{m+1}}\right]\\ &= \frac{\psi_P(\vec{x})_i}{1+\psi_a}.\end{aligned}$$

Therefore, we have that the approximate Hessian for $\overline{P}$, denoted $\overline{\mathbf{H}}(\vec{x})$, is bounded by $\overline{\mathbf{H}}^{-1} \preceq (1+\psi_a)\mathbf{H}^{-1}$.

To bound the change in centrality note that by (2) in Lemma 15 we have that $\overline{\mathbf{H}}^{-1} \preceq \mathbf{H}^{-1}$. Therefore if let $\vec{v}' \in \mathbb{R}^m$ be defined so that $\vec{v}'_i = \vec{v}_i$ for all $i \in [m]$ then by triangle inequality we have

$$\begin{aligned}\delta_{\vec{e}_{\overline{P}}(\vec{v},\vec{x})}(\vec{x}) &= \left\|\overline{\mathbf{A}}_x^T(c_e\vec{\mathbb{1}}+\vec{v})\right\|_{\overline{\mathbf{H}}^{-1}} \leq \sqrt{1+\psi_a}\left\|\overline{\mathbf{A}}_x^T(c_e\vec{\mathbb{1}}+\vec{v})\right\|_{\mathbf{H}^{-1}}\\ &\leq \sqrt{1+\psi_a}\left(\left\|\mathbf{A}_x^T(c_e\vec{\mathbb{1}}+\vec{\tau})\right\|_{\mathbf{H}^{-1}} + \left\|\frac{\vec{a}_{m+1}}{s_{m+1}}(c_e+v_{m+1})\right\|_{\mathbf{H}^{-1}} + \left\|\mathbf{A}_x^T(\vec{v}'-\vec{\tau})\right\|_{\mathbf{H}^{-1}}\right)\\ &= \sqrt{1+\psi_a}\left(\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) + \left(c_e+\frac{\psi_a}{1+\psi_a}\right)\left\|\frac{\vec{a}_{m+1}}{s_{m+1}}\right\|_{\mathbf{H}^{-1}} + \left\|\mathbf{A}_x^T(\vec{v}'-\vec{\tau})\right\|_{\mathbf{H}^{-1}}\right)\end{aligned}$$

Now, since $\mathbf{H}^{-1} \preceq \frac{1}{\mu(\vec{x})}\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}$, we have that

$$\left\|\frac{\vec{a}_{m+1}}{s_{m+1}}\right\|_{\mathbf{H}^{-1}} \leq \frac{1}{\sqrt{\mu(\vec{x})}}\left\|\frac{\vec{a}_{m+1}}{s_{m+1}}\right\|_{\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}} = \sqrt{\frac{\psi_a}{\mu(\vec{x})}}.$$

Since $\mathbf{\Psi}^{1/2}\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{\Psi}\mathbf{A}_x\right)^{-1}\mathbf{A}_x^T\mathbf{\Psi}^{1/2}$ is a projection matrix, we have $\mathbf{\Psi}^{-1} \succeq \mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{\Psi}\mathbf{A}_x\right)^{-1}\mathbf{A}_x^T \succeq$

$\mathbf{A}_x\mathbf{H}^{-1}\mathbf{A}_x^T$. By Lemma 16, we have

$$\begin{aligned}\left\|\mathbf{A}_x^T\left(\vec{\tau}'-\vec{\upsilon}\right)\right\|_{\mathbf{H}^{-1}}^2 &\leq \left\|\vec{\tau}'-\vec{\upsilon}\right\|_{\mathbf{\Psi}^{-1}}^2 \\ &= \sum_{i\in[m]}\frac{1}{\psi(\vec{x})_i}\left(\frac{1}{1+\psi_a}\left(\vec{\mathbb{1}}_i\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\frac{\vec{a}_{m+1}}{s_{m+1}}\right)^2\right)^2 \\ &\leq \left(\frac{1}{1+\psi_a}\right)^2\left(\frac{\vec{a}_{m+1}^T\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\vec{a}_{m+1}}{s_{m+1}^2}\right)^2=\left(\frac{\psi_a}{1+\psi_a}\right)^2\end{aligned}$$

Combining, we have that

$$\begin{aligned}\delta_{\vec{e}_{\overline{P}}(\vec{\upsilon},\vec{x})}(\vec{x}) &\leq \sqrt{1+\psi_a}\left(\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x})+\left(c_e+\frac{\psi_a}{1+\psi_a}\right)\sqrt{\frac{\psi_a}{\mu(\vec{x})}}+\frac{\psi_a}{1+\psi_a}\right) \\ &\leq \sqrt{1+\psi_a}\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x})+\left(c_e\sqrt{1+\psi_a}+\psi_a\right)\sqrt{\frac{\psi_a}{\mu(\vec{x})}}+\psi_a.\end{aligned}$$

□

We now bound the effect of removing a constraint.

Lemma 18 (Removing a Constraint)**.** *Let $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\vec{b}\in\mathbb{R}^m$, $\vec{\tau}\in\mathbb{R}^m$, and $\vec{x}\in P\stackrel{\text{def}}{=}\{\vec{y}\in\mathbb{R}^n : \mathbf{A}\vec{y}\geq\vec{b}\}$. Let $\overline{\mathbf{A}}\in\mathbb{R}^{(m-1)\times n}$ be $\mathbf{A}$ with row m removed, let $\overline{b}\in\mathbb{R}^{m-1}$ denote the first $m-1$ coordinates of $\vec{b}$, and let $\overline{P}\stackrel{\text{def}}{=}\{\vec{y}\in\mathbb{R}^n : \overline{\mathbf{A}}\vec{y}\geq\overline{b}\}$. Let $\psi_d=\psi_P(\vec{x})_m$.*

Now, let $\vec{\upsilon}\in\mathbb{R}^{m-1}$ be defined so that for all $i\in[m-1]$

$$\upsilon_i=\tau_i+\frac{1}{1-\psi_d}\left(\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\vec{\mathbb{1}}_m\right)_i^2.$$

Assume $\psi_d\leq 1.1\mu(\vec{x})\leq\frac{1}{10}$ and $\left\|\vec{e}_P(\vec{\tau},\vec{x})\right\|_\infty\leq c_e\leq\frac{1}{2}$, we have the following:

- [Leverage Score Estimation] $e_{\overline{P}}(\vec{\upsilon},\vec{x})_i=e_P(\vec{\tau},\vec{x})_i$ for all $i\in[m-1]$.
- [Function Value Decrease] $p_{\vec{e}_{\overline{P}}(\vec{\upsilon},\vec{x})}(\vec{x})=p_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x})+[c_e+e_P(\vec{\tau},\vec{x})_m]\log\left(s(\vec{x})_m/R\right)+\log(1-\psi_d)$
- [Centrality Increase] $\delta_{\vec{e}_{\overline{P}}(\vec{\upsilon},\vec{x})}(\vec{x})\leq\frac{1}{\sqrt{1-2\mu(\vec{x})}}\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x})+3(c_e+\mu(\vec{x}))$.

Proof. By (1) in Lemma 15, we have that for all $i\in[m-1]$

$$\psi_{\overline{P}}(\vec{x})_i=\psi_P(\vec{x})_i+\frac{1}{1-\psi_d}\left(\vec{\mathbb{1}}_i^T\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\vec{\mathbb{1}}_m\right)^2.$$

Consequently, [Leverage Score Estimation] holds. Furthermore, by (3) in Lemma 15, this then implies that [Function Value Change] holds.

To bound the change in centrality, we first note that by (1) and (2) in Lemma 15 and the fact $\psi_{\overline{P}}(\vec{x})_i \geq \psi_P(\vec{x})_i$, we have that the approximate Hessian for $\overline{P}$, denoted $\overline{\mathbf{H}}(\vec{x})$, is bounded by

$$\begin{aligned}\overline{\mathbf{H}}(\vec{x})^{-1} &\preceq \left(\mathbf{H}(\vec{x}) - \mathbf{A}_x^T\left(c_e\mathbf{I}+\mathbf{\Psi}_x\right)^{1/2}\vec{\mathbb{1}}_m\vec{\mathbb{1}}_m^T\left(c_e\mathbf{I}+\mathbf{\Psi}_x\right)^{1/2}\mathbf{A}_x\right)^{-1} \\ &\preceq \left(1+\frac{\alpha}{1-\alpha}\right)\mathbf{H}(\vec{x})^{-1} = \left(\frac{1}{1-\alpha}\right)\mathbf{H}(\vec{x})^{-1}\end{aligned}$$

where $\alpha \overset{\text{def}}{=} \vec{\mathbb{1}}_m^T\left(c_e\mathbf{I}+\mathbf{\Psi}_x\right)^{1/2}\mathbf{A}_x\mathbf{H}(\vec{x})^{-1}\mathbf{A}_x^T\left(c_e\mathbf{I}+\mathbf{\Psi}_x\right)^{1/2}\vec{\mathbb{1}}_m$. Using $c_e+\mu(\vec{x}) \leq \frac{1}{2}+\frac{1}{10} \leq 1$, we have

$$\mathbf{H}(\vec{x})^{-1} \preceq \left(\mathbf{A}_x^T(c_e+\mu(\vec{x}))\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1} \preceq \left(c_e+\mu(\vec{x})\right)^{-1}\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}. \tag{5.5}$$

Using this, we have

$$\alpha \leq \left(\frac{c_e+\psi_d}{c_e+\mu(\vec{x})}\right)\vec{\mathbb{1}}_m^T\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\vec{\mathbb{1}}_m = \left(\frac{c_e+\psi_d}{c_e+\mu(\vec{x})}\right)\psi_d\,. \tag{5.6}$$

Now let $\vec{\tau}' \in \mathbb{R}^{m-1}$ be defined so that $\tau_i' = \tau_i$ for all $i \in [m-1]$. We have by above that

$$\delta_{\vec{e}_{\overline{P}}(\vec{\upsilon},\vec{x})}(\vec{x}) = \left\|\overline{\mathbf{A}}_x^T(c_e\vec{\mathbb{1}}+\vec{\upsilon})\right\|_{\overline{\mathbf{H}}^{-1}} \leq \frac{1}{\sqrt{1-\alpha}}\left\|\overline{\mathbf{A}}_x^T(c_e\vec{\mathbb{1}}+\vec{\upsilon})\right\|_{\mathbf{H}^{-1}}$$

and therefore, by triangle inequality

$$\begin{aligned}\left\|\overline{\mathbf{A}}_x^T(c_e\vec{\mathbb{1}}+\vec{\upsilon})\right\|_{\mathbf{H}^{-1}} &\leq \left\|\mathbf{A}_x^T(c_e\vec{\mathbb{1}}+\vec{\tau})\right\|_{\mathbf{H}^{-1}} + \left\|\mathbf{A}_x^T\vec{\mathbb{1}}_m(c_e+\tau_m)\right\|_{\mathbf{H}^{-1}} + \left\|\mathbf{A}_x^T(\vec{\tau}'-\vec{\upsilon})\right\|_{\mathbf{H}^{-1}} \\ &= \delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) + (c_e+\tau_m)\left\|\mathbf{A}_x^T\vec{\mathbb{1}}_m\right\|_{\mathbf{H}^{-1}} + \left\|\mathbf{A}_x^T(\vec{\tau}'-\vec{\upsilon})\right\|_{\mathbf{H}^{-1}}.\end{aligned}$$

Now, (5.5) shows that

$$\left\|\mathbf{A}_x^T\vec{\mathbb{1}}_m\right\|_{\mathbf{H}^{-1}} \leq \frac{1}{\sqrt{c_e+\mu(\vec{x})}}\left\|\mathbf{A}_x^T\vec{\mathbb{1}}_m\right\|_{(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I})^{-1}} \leq \sqrt{\frac{\psi_d}{c_e+\mu(\vec{x})}}$$

Furthermore, since $\mathbf{\Psi}^{-1} \succeq \mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{\Psi}\mathbf{A}_x\right)^{-1}\mathbf{A}_x^T \succeq \mathbf{A}_x\mathbf{H}^{-1}\mathbf{A}_x^T$, by Lemma 16 we have

$$\begin{aligned}\left\|\mathbf{A}_x^T\left(\vec{\tau}'-\vec{\upsilon}\right)\right\|_{\mathbf{H}^{-1}}^2 &\leq \left\|\vec{\tau}'-\vec{\upsilon}\right\|_{\mathbf{\Psi}^{-1}}^2 \\ &= \sum_{i\in[m]}\frac{1}{\psi(\vec{x})_i}\left(\frac{1}{1-\psi_d}\left(\vec{\mathbb{1}}_i^T\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\vec{\mathbb{1}}_m\right)^2\right)^2 \\ &\leq \left(\frac{1}{1-\psi_d}\right)^2\left(\vec{\mathbb{1}}_m^T\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x+\lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\vec{\mathbb{1}}_m\right)^2 = \left(\frac{\psi_d}{1-\psi_d}\right)^2\end{aligned}$$

Combining, we have that

$$\delta_{\vec{e}_{\overline{P}}(\vec{\upsilon},\vec{x})}(\vec{x}) \leq \frac{1}{\sqrt{1-\alpha}}\left[\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) + (c_e+\tau_m)\sqrt{\frac{\psi_d}{c_e+\mu(\vec{x})}} + \frac{\psi_d}{1-\psi_d}\right].$$

Using the assumption $\psi_d \leq 1.1\mu(\vec{x}) \leq \frac{1}{10}$, $\left\|\vec{e}_P(\vec{\tau},\vec{x})\right\|_\infty \leq c_e$ and (5.6), we have $\alpha \leq 1.1\psi_d \leq 1.21\mu(\vec{x})$ and $\tau_m \leq \psi_d + c_e$, and

$$\begin{aligned}
\delta_{\vec{e}_{\overline{P}}(\vec{v},\vec{x})}(\vec{x}) &\leq \frac{1}{\sqrt{1-1.3\mu(\vec{x})}}\left[\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) + (c_e+\tau_m)\sqrt{1.1} + 1.2\psi_d\right] \\
&\leq \frac{1}{\sqrt{1-2\mu(\vec{x})}}\delta_{\vec{e}_P(\vec{\tau},\vec{x})}(\vec{x}) + \frac{1}{\sqrt{1-\frac{1.3}{10}}}\left(\sqrt{1.1}\cdot 2c_e + 1.1(\sqrt{1.1}+1.2)\mu(\vec{x})\right)
\end{aligned}$$

□

5.3 Hybrid Center Properties

Here we prove properties of points near the hybrid center. First we bound the distance between points in the $\mathbf{H}(\vec{x})$ norm in terms of the ℓ_2 norm of the points.

Lemma 19. *For $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\vec{b} \in \mathbb{R}^m$ suppose that $\vec{x} \in P = \{\vec{y} : \mathbf{A}\vec{y} \geq \vec{b}\}$ and $\vec{e} \in \mathbb{R}^m$ such that $\left\|\vec{e}\right\|_\infty \leq \frac{1}{2}c_e < \frac{1}{20}$ and $\delta_{\vec{e}} \leq 0.1\sqrt{c_e+\mu(\vec{x})}$. Then for all $\vec{y} \in P$ we have*

$$\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})} \leq \frac{12mc_e + 6n + 2\lambda\left\|\vec{y}\right\|_2^2}{\sqrt{c_e+\mu(\vec{x})}} \tag{5.7}$$

and

$$\left\|\vec{x}\right\|_2^2 \leq 4\lambda^{-1}(mc_e+n) + 2\left\|\vec{y}\right\|_2^2 \quad .$$

Proof. For notational simplicity let $\vec{t} \overset{\text{def}}{=} c_e\vec{\mathbb{1}}+\vec{e}+\vec{\psi}_x$, $\mathbf{T} \overset{\text{def}}{=} \mathbf{diag}(\vec{t})$, and $\mathbf{M} \overset{\text{def}}{=} \mathbf{A}_x^T(c_e\mathbf{I}+\mathbf{\Psi}_x)\mathbf{A}_x$. We have

$$\left\|\vec{x}-\vec{y}\right\|_{\mathbf{A}_x^T\mathbf{T}\mathbf{A}_x}^2 = \sum_{i\in[m]} t_i\frac{[\vec{s}_x-\vec{s}_y]_i^2}{[\vec{s}_x]_i^2} = \sum_{i\in[m]} t_i\left(1-2\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}+\frac{[\vec{s}_y]_i^2}{[\vec{s}_x]_i^2}\right) \tag{5.8}$$

and

$$\sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i^2}{[\vec{s}_x]_i^2} \leq \left(\sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}\right)\max_{i\in[m]}\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i} \leq \left(\sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}\right)\left(1+\left\|\mathbf{S}_x^{-1}(\vec{s}_y-\vec{s}_x)\right\|_\infty\right) \tag{5.9}$$

and

$$\begin{aligned}
\left\|\mathbf{S}_x^{-1}\left(\vec{s}_x-\vec{s}_y\right)\right\|_\infty &= \max_{i\in[m]}\left|\vec{\mathbb{1}}_i\mathbf{S}_x^{-1}\mathbf{A}\left(\vec{x}-\vec{y}\right)\right| \leq \left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\sqrt{\max_{i\in[m]}[\mathbf{S}_x^{-1}\mathbf{A}\mathbf{H}(\vec{x})^{-1}\mathbf{A}^T\mathbf{S}_x^{-1}]_{ii}} \\
&\leq (c_e+\mu(\vec{x}))^{-1/2}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})} \quad .
\end{aligned} \tag{5.10}$$

Now, clearly $\sum_{i\in[m]} t_i[\vec{s}_y]_i/[\vec{s}_x]_i$ is positive and since $\left\|\vec{e}\right\|_\infty \leq \frac{1}{2}c_e$ we know that $\frac{1}{2}\mathbf{M} \preceq \mathbf{A}_x^T\mathbf{T}\mathbf{A}_x$. Therefore, by combining, (5.8), (5.9), and (5.10) we have

$$\begin{aligned}
\frac{1}{2}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{M}}^2 &\leq \left\|\vec{t}\right\|_1 - \sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i} + \left(\sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}\right)\frac{\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}}{\sqrt{c_e+\mu(\vec{x})}} \\
&\leq \left\|\vec{t}\right\|_1 + \left(\sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}\right)\frac{\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}}{\sqrt{c_e+\mu(\vec{x})}}
\end{aligned} \tag{5.11}$$

Now since $\bigtriangledown p_{\vec{e}}(\vec{x}) = -\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{T}\vec{\mathbb{1}} + \lambda\vec{x}$ we have

$$\langle \vec{x}-\vec{y}, \bigtriangledown p_{\vec{e}}(\vec{x})\rangle = -\sum_{i\in[m]} t_i \frac{[\vec{s}_x - \vec{s}_y]_i}{[\vec{s}_x]_i} + \lambda\vec{x}^T(\vec{x}-\vec{y})$$

and therefore by Cauchy Schwarz and $\vec{x}^T\vec{y} \le \left\|\vec{x}\right\|_2^2 + \frac{1}{4}\left\|\vec{y}\right\|_2^2$,

$$\sum_{i\in[m]} t_i \frac{[\vec{s}_y]_i}{[\vec{s}_x]_i} = \left\|\vec{t}\right\|_1 - \lambda\left\|\vec{x}\right\|_2^2 + \lambda\vec{x}^T\vec{y} + \langle \vec{x}-\vec{y}, \bigtriangledown p_{\vec{e}}(\vec{x})\rangle \tag{5.12}$$

$$\le \left\|t\right\|_1 + \frac{\lambda}{4}\left\|\vec{y}\right\|_2^2 + \left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\delta_{\vec{e}}(\vec{x}) \quad . \tag{5.13}$$

Now, using (5.11), (5.13) and the definition of $\mathbf{H}(\vec{x})$, we have

$$\begin{aligned}
\frac{1}{2}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}^2 &= \frac{1}{2}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{M}}^2 + \frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2 \\
&\le \left\|\vec{t}\right\|_1 + \left(\sum_{i\in[m]} t_i \frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}\right) \frac{\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}}{\sqrt{c_e + \mu(\vec{x})}} + \frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2 \\
&\le \left\|\vec{t}\right\|_1 + \frac{\left\|t\right\|_1 + \frac{\lambda}{4}\left\|\vec{y}\right\|_2^2}{\sqrt{c_e+\mu(\vec{x})}}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})} + \delta_e(\vec{x})\frac{\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}^2}{\sqrt{c_e+\mu(\vec{x})}} + \frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2.
\end{aligned}$$

Using the fact that $\delta_e(\vec{x}) \le 0.1\sqrt{c_e+\mu(\vec{x})}$, we have

$$\frac{1}{4}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}^2 \le \left\|\vec{t}\right\|_1 + \frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2 + \frac{\left\|t\right\|_1 + \frac{\lambda}{4}\left\|\vec{y}\right\|_2^2}{\sqrt{c_e+\mu(\vec{x})}}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}. \tag{5.14}$$

Furthermore, since $\sum_{i\in[m]} t_i[\vec{s}_y]_i/[\vec{s}_x]_i$ is positive, (5.12) shows that

$$\lambda\vec{x}^T(\vec{x}-\vec{y}) = \lambda\left\|\vec{x}\right\|_2^2 - \lambda\vec{x}^T\vec{y} \le \left\|\vec{t}\right\|_1 + \langle \vec{x}-\vec{y}, \bigtriangledown p_{\vec{e}}(\vec{x})\rangle \le \left\|\vec{t}\right\|_1 + \left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\delta_e(\vec{x})$$

and hence

$$\begin{aligned}
\frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2 \le \frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2 + \frac{\lambda}{2}\left\|\vec{x}\right\|_2^2 &= \lambda\vec{x}^T(\vec{x}-\vec{y}) + \frac{\lambda}{2}\left\|\vec{y}\right\|_2^2 \\
&\le \left\|\vec{t}\right\|_1 + \frac{\lambda}{2}\left\|\vec{y}\right\|_2^2 + \left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\delta_e(\vec{x}) \quad .
\end{aligned} \tag{5.15}$$

Putting (5.15) into (5.14) and using the fact that $\delta_e(\vec{x}) \le 0.1\sqrt{c_e+\mu(\vec{x})}$, we have

$$\frac{1}{4}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}^2 \le 2\left\|\vec{t}\right\|_1 + \frac{\lambda}{2}\left\|\vec{y}\right\|_2^2 + \left(0.1 + \frac{\left\|\vec{t}\right\|_1 + \frac{\lambda}{4}\left\|\vec{y}\right\|_2^2}{\sqrt{c_e+\mu(\vec{x})}}\right)\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}.$$

Now, using $\left\|\vec{t}\right\|_1 \le 2mc_e + n$ and $\sqrt{c_e+\mu(\vec{x})} \le 1.05$, we have

$$\frac{1}{4}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}^2 \le 2.2\alpha + (0.1+\alpha)\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})} \quad \text{for} \quad \alpha = \frac{2mc_e + n + \frac{\lambda}{4}\left\|\vec{y}\right\|_2^2}{\sqrt{c_e+\mu(\vec{x})}} \quad .$$

Since $\alpha \geq 1/\sqrt{1.1}$, we have that

$$\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})} \leq \frac{0.1+\alpha+\sqrt{(\alpha+0.1)^2+2.2\alpha}}{2\cdot\frac{1}{4}} \leq 6\alpha$$

yielding (5.7).

We also have by (5.12) and the fact that $\delta_e(\vec{x}) \leq 0.1\sqrt{c_e+\mu(\vec{x})}$,

$$\begin{aligned}\lambda\left\|\vec{x}\right\|_2^2 &= \left\|t\right\|_1 + \lambda\vec{x}^T\vec{y} + \langle\vec{x}-\vec{y}, \bigtriangledown p_{\vec{e}}(\vec{x})\rangle - \sum_{i\in[m]} t_i\frac{[\vec{s}_y]_i}{[\vec{s}_x]_i}\\ &\leq \left\|t\right\|_1 + \frac{\lambda}{2}\left\|\vec{x}\right\|_2^2 + \frac{\lambda}{2}\left\|\vec{y}\right\|_2^2 + \left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\delta_e(\vec{x})\\ &\leq \left\|t\right\|_1 + \frac{\lambda}{2}\left\|\vec{x}\right\|_2^2 + \frac{\lambda}{2}\left\|\vec{y}\right\|_2^2 + 0.1\sqrt{c_e+\mu(\vec{x})}\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})}\end{aligned}$$

Hence, using $\left\|\vec{t}\right\|_1 \leq 2mc_e+n$ and $\left\|\vec{x}-\vec{y}\right\|_{\mathbf{H}(\vec{x})} \leq 6\alpha$, we have

$$\begin{aligned}\frac{\lambda}{2}\left\|\vec{x}\right\|_2^2 &\leq \left\|t\right\|_1 + \frac{\lambda}{2}\left\|\vec{y}\right\|_2^2 + 0.6\left(2mc_e+n+\frac{\lambda}{4}\left\|\vec{y}\right\|_2^2\right)\\ &\leq \lambda\left\|\vec{y}\right\|_2^2 + 2(mc_e+n).\end{aligned}$$

□

In the following lemma we show how we can write one hyperplane in terms of the others provided that we are nearly centered and show there is a constraint that the central point is close to.

Lemma 20. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\vec{b} \in \mathbb{R}^m$ such that $\left\|a_i\right\|_2 = 1$ for all i. Suppose that $\vec{x} \in P = \{\vec{y} : \mathbf{A}\vec{y} \geq \vec{b}\}$ and $\vec{e} \in \mathbb{R}^m$ such that $\left\|\vec{e}\right\|_\infty \leq \frac{1}{2}c_e \leq \frac{1}{2}$. Furthermore, let $\epsilon = \min_{j\in[m]} s_j(\vec{x})$ and suppose that $i = \operatorname{argmin}_{j\in[m]} s_j(\vec{x})$ then*

$$\left\|\vec{a}_i + \sum_{j\neq i}\left(\frac{s(x)_i}{s(x)_j}\right)\left(\frac{c_e+e_j+\psi_j(\vec{x})}{c_e+e_i+\psi_i(\vec{x})}\right)\vec{a}_j\right\|_2 \leq \frac{2\epsilon}{(c_e+\mu(\vec{x}))}\left[\lambda\left\|\vec{x}\right\|_2 + \delta_e(\vec{x})\sqrt{\frac{mc_e+n}{\epsilon^2}+\lambda}\right].$$

Proof. We know that

$$\begin{aligned}\bigtriangledown p_e(\vec{x}) &= -\mathbf{A}^T\mathbf{S}_x^{-1}(c_e\vec{\mathbb{1}}+\vec{e}+\vec{\psi}_x)+\lambda\vec{x}\\ &= \lambda\vec{x} - \sum_{i\in[m]}\frac{(c_e+e_i+\psi_i)}{s(\vec{x})_i}\vec{a}_i\end{aligned}$$

Consequently, by $\left\|\vec{e}\right\|_\infty \leq \frac{1}{2}c_e$, and $\psi_i(\vec{x}) \geq \mu(\vec{x})$, we have

$$\begin{aligned}\left\|\vec{a}_i + \sum_{j\neq i}\left(\frac{s(x)_i}{s(x)_j}\right)\left(\frac{c_e+e_j+\psi_j(\vec{x})}{c_e+e_i+\psi_i(\vec{x})}\right)\vec{a}_j\right\|_2 &= \frac{s_i(\vec{x})}{c_e+e_i+\psi_i(\vec{x})}\left\|\mathbf{A}^T\mathbf{S}_x^{-1}(c_e\vec{\mathbb{1}}+\vec{e}+\vec{\psi}_x)\right\|_2\\ &\leq \frac{2\epsilon}{c_e+\mu(\vec{x})}\left[\lambda\left\|\vec{x}\right\|_2 + \left\|\bigtriangledown p_e(\vec{x})\right\|_2\right].\end{aligned}$$

Using $\left\|\vec{a}_i\right\| = 1$, $\sum_i \psi_i \leq n$, and $s_i(\vec{x}) \geq \epsilon$, we have

$$\begin{aligned}\mathrm{Tr}(\mathbf{A}_x^T(c_e\mathbf{I}+\mathbf{\Psi}_x)\mathbf{A}_x) &= \mathrm{Tr}(\mathbf{A}_x\mathbf{A}_x^T(c_e\mathbf{I}+\mathbf{\Psi}_x)) \\ &= \sum_i (c_e+\psi_i)\frac{\left\|a_i\right\|_2^2}{s_i^2(\vec{x})} \leq \frac{mc_e+n}{\epsilon^2}. \end{aligned} \tag{5.16}$$

Hence, we have $\mathbf{H}(\vec{x}) \preceq \left(\frac{mc_e+n}{\epsilon^2}+\lambda\right)\mathbf{I}$ and $\left\|\bigtriangledown p_e(\vec{x})\right\|_2 \leq \delta_e(\vec{x})\sqrt{\frac{mc_e+n}{\epsilon^2}+\lambda}$ yielding the result. □

5.4 The Algorithm

Here, we put all the results in the previous sections to get our ellipsoid algorithm. Below is a sketch of the pseudocode; we use c_a, c_d, c_e, c_Δ to denote parameters we decide later.

Algorithm 2: Our Cutting Plane Method

```
Input: $\mathbf{A}^{(0)} \in \mathbb{R}^{m\times n}$, $\vec{b}^{(0)} \in \mathbb{R}^m$, $\epsilon > 0$, and radius $R > 0$.
Input: A separation oracle for a non-empty set $K \subset B_\infty(R)$.
Check: Throughout the algorithm, if $s_i(\vec{x}^{(k)}) < \epsilon$ output $P^{(k)}$.
Check: Throughout the algorithm, if $\vec{x}^{(k)} \in K$, output $\vec{x}^{(k)}$.
Set $P^{(0)} = B_\infty(R)$.
Set $\vec{x}^{(0)} := \vec{0}$ and compute $\tau_i^{(0)} = \psi_{P^{(0)}}(\vec{x}^{(0)})_i$ for all $i \in [m]$ exactly.
for $k = 0$ to $\infty$ do
    Let $m^{(k)}$ be the number of constraints in $P^{(k)}$.
    Compute $\vec{w}^{(k)}$ such that $\mathbf{\Psi}_{P^{(k)}}(\vec{x}^{(k)}) \preceq \mathbf{W}^{(k)} \preceq (1+c_\Delta)\mathbf{\Psi}_{P^{(k)}}(\vec{x}^{(k)})$.
    Let $i^{(k)} \in \arg\max_{i\in[m^{(k)}]} \left|w_i^{(k)} - \tau_i^{(k)}\right|$.
    Set $\tau_{i^{(k)}}^{(k+\frac{1}{3})} = \psi_{P^{(k)}}(\vec{x}^{(k)})_{i^{(k)}}$ and $\tau_j^{(k+\frac{1}{3})} = \tau_j^{(k)}$ for all $j \neq i^{(k)}$.
    if $\min_{i\in[m^{(k)}]} w_i^{(k)} \leq c_d$ then
        Remove constraint with minimum $w_i^{(k)}$ yielding polytope $P^{(k+1)}$.
        Update $\vec{\tau}$ according to Lemma 18 to get $\tau_j^{(k+\frac{2}{3})}$.
    else
        Use separation oracle at $\vec{x}^{(k)}$ to get a constraint $\{\vec{x} : \vec{a}^T\vec{x} \geq \vec{a}^T\vec{x}^{(k)}\}$ with
          $\left\|\vec{a}\right\|_2 = 1$.
        Add constraint $\{\vec{x} : \vec{a}^T\vec{x} \geq \vec{a}^T\vec{x}^{(k)} - c_a^{-1/2}\sqrt{\vec{a}^T(\mathbf{A}^T\mathbf{S}_{\vec{x}^{(k)}}^{-2}\mathbf{A}+\lambda\mathbf{I})^{-1}\vec{a}}\}$ yielding
          polytope $P^{(k+1)}$.
        Update $\vec{\tau}$ according to Lemma 17 to get $\tau_j^{(k+\frac{2}{3})}$.
    $(\vec{x}^{(k+1)}, \vec{\tau}^{(k+1)})$ = Centering$(\vec{x}^{(k)}, \vec{\tau}^{(k+\frac{2}{3})}, 200, c_\Delta)$.
end
```

In the algorithm, there are two main invariants we maintain. First, we maintain that the centrality $\delta_{P,\vec{e}}(\vec{x})$, which indicates how close $\vec{x}$ is to the minimum point of $p_{\vec{e}}$, is small. Second, we maintain that $\left\|\vec{e}(\vec{\tau},\vec{x})\right\|_{\infty}$, which indicates how accurate the leverage score estimate $\vec{\tau}$ is, is small. In the following lemma we show that we maintain both invariants throughout the algorithm.

Lemma 21. *Assume that $c_e \leq c_d \leq \frac{1}{10^6}$, $c_a\sqrt{c_a} \leq \frac{c_d}{10^3}$, $c_d \leq c_a$, and $c_\Delta \leq Cc_e/\log(n\log(R/\epsilon))$ for some small enough universal constant C. During our cutting plane method, for all k, with high probability in n, we have*

1. $\left\|\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})\right\|_{\infty} \leq \frac{1}{1000}c_e$, $\left\|\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})\right\|_{\infty} \leq \frac{1}{1000}c_e$, $\left\|\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})\right\|_{\infty} \leq \frac{1}{400}c_e$.
2. $\delta_{P^{(k)},\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) \leq \frac{1}{100}\sqrt{c_e+\min\left(\mu(\vec{x}^{(k)}),c_d\right)}$.
3. $\delta_{P^{(k+1)},\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})}(\vec{x}^{(k+1)}) \leq \frac{1}{400}\sqrt{c_e+\min\left(\mu(\vec{x}^{(k+1)}),c_d\right)}$.

Proof. Some statements of the proof hold only with high probability in n; we omit mentioning this for simplicity.

We prove by induction on k. Note that the claims are written in order consistent with the algorithm and proving the statement for k involves bounding centrality at the point $\vec{x}^{(k+1)}$. Trivially we define, $\vec{\tau}^{(-1)}=\vec{\tau}^{(-\frac{2}{3})}=\vec{\tau}^{(-\frac{1}{3})}=\vec{\tau}^{(0)}$, $\vec{x}^{(-1)}=\vec{x}^{(0)}$ and note that the claims then hold for $k=-1$ as we compute the initial leverage scores, $\vec{\tau}^{(0)}$, exactly and since the polytope is symmetric we have $\delta_{\vec{e}(\vec{\tau}^{(0)},\vec{x}^{(0)})}(\vec{0})=0$. We now suppose they hold for all $t<k$ and show that they hold for $t=k$.

We first bound δ. For notational simplicity, let $\eta_t \overset{\text{def}}{=} \sqrt{c_e+\min(\mu(\vec{x}^{(t)}),c_d)}$. By the induction hypothesis we know that $\delta_{P^{(t)},\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)}) \leq \frac{1}{400}\eta_t$. Now, when we update $\tau^{(t)}$ to $\tau^{(t+\frac{1}{3})}$, we set $\vec{e}_{i^{(t)}}$ to 0. Consequently, Lemma 13 and the induction hypothesis $\left\|\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})\right\|_{\infty} \leq \frac{1}{400}c_e$ show that

$$\begin{aligned}\delta_{P^{(t)},\vec{e}(\vec{\tau}^{(t+\frac{1}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)}) &\leq \delta_{P^{(t)},\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)})+\frac{1}{\sqrt{c_e+\mu(\vec{x}^{(t)})}}e_{i^{(t)}}(\vec{\tau}^{(t)},\vec{x}^{(t)})\\ &\leq \frac{1}{400}\eta_t+\frac{\sqrt{c_e}}{400} \leq \frac{\eta_t}{200} \qquad (5.17)\end{aligned}$$

Next, we estimate the δ changes when we remove or add a constraint.

For the case of removal, we note that it happens only if $\mu(\vec{x}^{(t)}) \leq \min_i w_i \leq c_d \leq \frac{1}{10^6}$. Also, the row we remove has leverage score at most $1.1\mu(\vec{x}^{(t)})$ because we pick the row with minimum w. Hence, Lemma 18 show that

$$\begin{aligned}\delta_{P^{(t+1)},\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)}) &\leq \frac{1}{\sqrt{1-2\mu(\vec{x}^{(t)})}}\delta_{P^{(t)},\vec{e}(\vec{\tau}^{(t+\frac{1}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)})+3(c_e+\mu(\vec{x}^{(t)}))\\ &\leq \frac{1}{\sqrt{1-2\cdot 10^{-6}}}\left(\frac{\eta_t}{200}\right)+3(c_e+\mu(\vec{x}^{(t)})) \leq \frac{\eta_t}{100}\end{aligned}$$

where we used the fact $\mu(\vec{x}^{(t)}) \leq c_d$ and hence $c_e+\mu(\vec{x}^{(t)}) \leq \sqrt{2c_d}\eta_t \leq \frac{\sqrt{2}}{1000}\eta_t$.

For the case of addition, we note that it happens only if $2\mu(\vec{x}^{(t)}) \geq \min_i w_i \geq c_d$. Furthermore, in this case the hyperplane we add is chosen precisely so that $\psi_a = c_a$. Furthermore, since $c_e \leq c_d \leq c_a \leq \frac{1}{100}$ by Lemma 17 we have that

$$\delta_{P^{(t+1)},\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)}) \leq \sqrt{1+c_a}\delta_{P^{(t)},\vec{e}(\vec{\tau}^{(t+\frac{1}{3})},\vec{x}^{(t)})} + \left(c_e\sqrt{1+c_a}+c_a\right)\sqrt{\frac{c_a}{\mu(\vec{x}^{(t)})}} + c_a$$
$$\leq \frac{\eta_t}{190} + 4c_a\sqrt{\frac{c_a}{c_d}} \quad .$$

Furthermore, since $c_a\sqrt{c_a} \leq \frac{c_d}{1000}$, $\mu(\vec{x}^{(t)}) \geq c_d/2$, and $c_d \leq 10^{-6}$ we know that $4c_a\sqrt{c_a/c_d} \leq \frac{1}{250}\eta_t$ and consequently in both cases we have $\delta_{P^{(t+1)},\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)}) \leq \frac{1}{100}\eta_t$.

Now, note that Lemmas 17 and 18 show that $\vec{e}$ does not change during the addition or removal of an constraint. Hence, we have $\left\|\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})\right\|_\infty \leq \left\|\vec{e}(\vec{\tau}^{(t+\frac{1}{3})},\vec{x}^{(t)})\right\|_\infty$. Furthermore, we know the step "$\vec{\tau}_{i^{(k)}}^{(k+\frac{1}{3})} = \psi_{P^{(k)}}(\vec{x}^{(k)})_{i^{(k)}}$" only decreases $\left\|\vec{e}\right\|_\infty$ and hence we have $\left\|\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})\right\|_\infty \leq \left\|\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})\right\|_\infty \leq \frac{c_e}{400}$. Thus, we have all the conditions needed for Lemma 14 and consequently

$$\delta_{P^{(t+1)},\vec{e}(\vec{\tau}^{(t+1)},\vec{x}^{(t+1)})}(\vec{x}^{(t+1)}) \leq 2\left(1-\frac{1}{64}\right)^{200}\delta_{P^{(t+1)},\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)}) \leq \frac{1}{1000}\eta_t \quad .$$

Lemma 14 also shows that that $\left\|\frac{s(\vec{x}^{(t+1)})-s(\vec{x}^{(t)})}{s(\vec{x}^{(t)})}\right\|_2 \leq \frac{1}{10}$ and hence $\psi_i(\vec{x}^{(t)}) \leq 2\psi_i(\vec{x}^{(t+1)})$ for all i. Therefore, $\eta_t \leq 2\eta_{t+1}$ and thus

$$\delta_{P^{(t+1)},\vec{e}(\vec{\tau}^{(t+1)},\vec{x}^{(t+1)})}(\vec{x}^{(t+1)}) \leq \frac{\sqrt{c_e+\min\left(c_d,\mu(\vec{x}^{(t+1)})\right)}}{400}.$$

completing the induction case for δ.

Now, we bound $\left\|\vec{e}\right\|_\infty$. Lemma 17 and 18 show that $\vec{e}$ does not change during the addition or removal of an constraint. Hence, $\vec{e}$ is affected by only the update step "$\tau_{i^{(k)}}^{(k+\frac{1}{2})} = \psi_{P^{(k)}}(\vec{x}^{(k)})_{i^{(k)}}$" and the centering step. Using the induction hypothesis $\delta_{P^{(t)},\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})}(\vec{x}^{(t)}) \leq \frac{1}{100}\eta_t$ and Lemma 14 shows that $\mathbf{E}\vec{e}(\vec{\tau}^{(t+1)},\vec{x}^{(t+1)}) = \vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})$ and $\left\|\vec{e}(\vec{\tau}^{(t+1)},\vec{x}^{(t+1)}) - \vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})\right\|_2 \leq \frac{1}{10}c_\Delta$. The goal for the update step is to decrease $\vec{e}$ by updating $\vec{\tau}$. In Section 6.2, we give a self-contained analysis of the effect of this step as a game. In each round, the vector $\vec{e}$ is corrupted by some mean 0 and bounded variance noise and the problem is how to update $\vec{e}$ such that $\left\|\vec{e}\right\|_\infty$ is bounded. Theorem 34 shows that we can do this by setting the $\vec{e}_i = 0$ for the almost maximum coordinate in each iteration. This is exactly what the update step is doing. Since our algorithm would run only $O(n\log(nR/\epsilon))$ many iterations, Theorem 34 shows that this strategy guarantees that after the update step, we have

$$\left\|\vec{e}(\tau^{(t+\frac{1}{3})},\vec{x}^{(t)})\right\|_\infty = O\left(c_\Delta \log\left(n\log(R/\epsilon)\right)\right).$$

Now, by our choice of c_Δ, we have $\left\|\vec{e}(\vec{\tau}^{(t+\frac{1}{3})},\vec{x}^{(t)})\right\|_\infty \leq \frac{1}{1000}c_e$. Lemma 17 and 18 show that $\vec{e}$ does not change during the addition or removal of an constraint. Hence, we have $\left\|\vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})\right\|_\infty \leq \frac{1}{1000}c_e$. Now, we note that again Lemma 14 shows $\left\|\vec{e}(\vec{\tau}^{(t+1)},\vec{x}^{(t+1)}) - \vec{e}(\vec{\tau}^{(t+\frac{2}{3})},\vec{x}^{(t)})\right\|_2 \leq \frac{1}{10}c_\Delta \leq \frac{1}{1000}c_e$, and we have $\left\|\vec{e}(\vec{\tau}^{(t+1)},\vec{x}^{(t+1)})\right\|_\infty \leq \frac{c_e}{400}$. This finishes the induction case for $\left\|\vec{e}\right\|_\infty$ and proves this lemma. $\square$

Next, we show the number of constraints is always linear to n.

Lemma 22. *Throughout our cutting plane method, there are at most* $1+\frac{2n}{c_d}$ *constraints.*

Proof. We only add a constraint if $\min_i w_i \geq c_d$. Since $2\psi_i \geq w_i$, we have $\psi_i \geq \frac{c_d}{2}$ for all i. Letting m denote the number of constraints after we add that row, we have $n \geq \sum_i \psi_i \geq (m-1)(c_d/2)$. □

Using $K \neq \varnothing$ and $K \subset B_\infty(R)$, here we show that the points are bounded.

Lemma 23. *During our Cutting Plane Method, for all* k, *we have* $\left\|\vec{x}^{(k)}\right\|_2 \leq 6\sqrt{n/\lambda}+2\sqrt{n}R$.

Proof. By Lemma 21 and Lemma 19 we know that $\left\|\vec{x}^{(k)}\right\|_2^2 \leq 4\lambda^{-1}(mc_e+n)+2\left\|\vec{y}\right\|_2^2$ for any $\vec{y} \in P^{(k)}$. Since our method never cuts out any point in K and since K is nonempty, there is some $\vec{y} \in K \subset P^{(k)}$. Since $K \subset B_\infty(R)$, we have $\left\|\vec{y}\right\|_2^2 \leq nR^2$. Furthermore, by Lemma 22 we have that $mc_e \leq c_e + 2n \leq 3n$ yielding the result. □

Lemma 24. $s_i\left(\vec{x}^{(k)}\right) \leq 12\sqrt{n/\lambda}+4\sqrt{n}R+\sqrt{\frac{1}{c_a\lambda}}$ *for all* i *and* k *in the our cutting plane method.*

Proof. Let $\vec{x}^{(j)}$ be the current point at the time that the constraint corresponding to s_i, denoted $\{\vec{x} : \vec{a}_i^T\vec{x} \geq a_i^T\vec{x}^{(j)} - s_i(\vec{x}^{(j)})\}$, was added. Clearly

$$s_i(\vec{x}^{(k)}) = \vec{a}_i^T\vec{x}^{(k)} - a_i^T\vec{x}^{(j)} + s_i(\vec{x}^{(j)}) \leq \left\|\vec{a}_i\right\| \cdot \left\|\vec{x}^{(k)}\right\| + \left|\vec{a}_i^T\vec{x}^{(j)} - s_i(\vec{x}^{(j)})\right| \quad .$$

On the one hand, if the constraint for s_i comes from the initial symmetric polytope $P^{(0)} = B_\infty(R)$, we know $\left|\vec{a}^T\vec{x}^{(j)} - \vec{s}_i(\vec{x}^{(j)})\right| \leq R$. On the other hand, if the constraint was added later then we know that

$$s_i(\vec{x}^{(j)}) = c_a^{-1/2}\sqrt{\vec{a}^T(\mathbf{A}^T\mathbf{S}_{\vec{x}^{(j)}}^{-2}\mathbf{A}+\lambda\mathbf{I})^{-1}\vec{a}} \leq (c_a\lambda)^{-1/2}$$

and $\left|\vec{a}^T\vec{x}^{(j)} - s_i(\vec{x}^{(j)})\right| \leq \left\|\vec{a}_i\right\| \cdot \left\|\vec{x}^{(j)}\right\| + \left|s_i(\vec{x}^{(j)})\right|$. Since $\left\|\vec{a}_i\right\|_2 = 1$ by design and $\left\|\vec{x}^{(j)}\right\|_2$ and $\left\|\vec{x}^{(k)}\right\|_2$ are upper bounded by $6\sqrt{n/\lambda}+2\sqrt{n}R$ by Lemma 23, in either case the result follows. □

Now, we have everything we need to prove that the potential function is increasing in expectation.

Lemma 25. *Under the assumptions of Lemma 21 if* $\lambda = \frac{1}{c_aR^2}$, $c_e = \frac{c_d}{4\log(6nR/\epsilon)}$, *and* $24c_d \leq c_a \leq \frac{1}{1000}$ *then for all* k *we have*

$$\mathbb{E}p_{\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})}(\vec{x}^{(k+1)}) \quad \geq \quad p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)}) - c_d + \log(1+\beta)$$

where $\beta = c_a$ *for the case of adding a constraint and* $\beta = -c_d$ *for the case of removal.*

Proof. Note that there are three places which affect the function value, namely the update step for $\tau^{(k+\frac{1}{3})}$, the addition/removal of constraints, and the centering step. We bound the effect of each separately.

First, for the update step, we have

$$p_{\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) = (c + e_{i^{(k)}}) \log(s_{i^{(k)}}(\vec{x}^{(k)})/R) + p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)}).$$

Lemma 24, the termination condition, $\lambda = \frac{1}{c_a R^2}$ and $c_a < \frac{1}{1000}$ ensure that

$$\epsilon \leq s_{i^{(k)}}(\vec{x}^{(k)}) \leq 12\sqrt{n/\lambda} + 4\sqrt{n}R + \sqrt{\frac{1}{c_a \lambda}} \leq 6\sqrt{n}R \tag{5.18}$$

and Lemma 21 shows that $|e_{i^{(k)}}| \leq c_e$. Hence, we have

$$p_{\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) \geq p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)}) + 2c_e \log(\epsilon/R).$$

For the addition step, Lemma 17 shows that

$$\begin{aligned} p_{\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) &= p_{\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) - c_e \log\left(s(\vec{x})_{m+1}/R\right) + \log(1 + c_a) \\ &\geq p_{\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) - c_e \log(6\sqrt{n}) + \log(1 + c_a) \end{aligned}$$

and for the removal step, Lemma 18 and $|e_i| \leq c_e$ shows that

$$\begin{aligned} p_{\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) &\geq p_{\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) + [c_e + e_P(\vec{\tau},\vec{x})_m] \log\left(s(\vec{x})_m/R\right) + \log(1 - c_d) \\ &\geq p_{\vec{e}(\vec{\tau}^{(k+\frac{1}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) - 2c_e \log(R/\epsilon) + \log(1 - c_d) \end{aligned}$$

After the addition or removal of a constraint, Lemma 21 shows that

$$\delta_{P^{(k)},\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) \leq \frac{1}{100}\sqrt{c_e + \min\left(\mu(\vec{x}^{(k)}), c_d\right)}$$

and therefore Lemma 14 and $c_e \leq c_d$ show that

$$\begin{aligned} \mathbb{E}p_{\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})}(\vec{x}^{(k+1)}) &\geq p_{\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) - 8\left(\frac{\sqrt{c_e + \min\left(\mu(\vec{x}^{(k)}), c_d\right)}}{100}\right)^2 \\ &\geq p_{\vec{e}(\vec{\tau}^{(k+\frac{2}{3})},\vec{x}^{(k)})}(\vec{x}^{(k)}) - \frac{c_d}{625}. \end{aligned}$$

Combining them with $c_e = \frac{c_d}{4\log(6nR/\epsilon)}$, we have

$$\begin{aligned} \mathbb{E}p_{\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})}(\vec{x}^{(k+1)}) &\geq p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)}) - 2c_e \log(6nR/\epsilon) - \frac{c_d}{625} + \log(1 + \beta) \\ &\geq p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)}) - c_d + \log(1 + \beta) \end{aligned}$$

where $\beta = c_a$ for the case of addition and $\beta = -c_d$ for the case of removal. □

Theorem 26. *For $c_a = \frac{1}{10^{10}}$, $c_d = \frac{1}{10^{12}}$, $c_e = \frac{c_d}{4\log(6nR/\epsilon)}$, $c_\Delta = \frac{Cc_e}{\log(n\log(R/\epsilon))}$ and $\lambda = \frac{1}{c_a R^2}$ for some small enough universal constant C, then we have*

$$\mathbb{E}p_{\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})}(\vec{x}^{(k+1)}) \geq p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)}) - \frac{1}{10^{11}} + \frac{9\beta}{10^{11}}$$

where $\beta = 1$ for the case of addition and $\beta = 0$ for the case of removal.

Proof. It is easy to see that these parameters satisfy the requirements of Lemma 25. □

5.5 Guarantees of the Algorithm

In this section we put everything together to prove Theorem 31, the main result of this section, providing the guarantees of our cutting plane method.

For the remainder of this section we assume that $c_a = \frac{1}{10^{10}}$, $c_d = \frac{1}{10^{12}}$, $c_e = \frac{c_d}{4\log(6nR/\epsilon)}$, $c_\Delta = \frac{Cc_e}{\log(n\log(R/\epsilon))}$ and $\lambda = \frac{1}{c_a R^2}$. Consequently, throughout the algorithm we have

$$\left\|\vec{x}\right\|_2 \leq 6\sqrt{n/\lambda} + 2\sqrt{n}R \leq 3\sqrt{n}R. \tag{5.19}$$

Lemma 27. *If $s_i(\vec{x}^{(k)}) < \epsilon$ for some i and k during our Cutting Plane Method then*

$$\max_{\vec{y}\in P^{(k)}\cap B_\infty(R)} \langle \vec{a}_i, \vec{y}\rangle - \min_{\vec{y}\in P^{(k)}\cap B_\infty(R)} \langle \vec{a}_i, \vec{y}\rangle \leq \frac{8n\epsilon}{c_a c_e}.$$

Proof. Let $\vec{y} \in P^{(k)} \cap B_\infty(R)$ be arbitrary. Since $\vec{y} \in B_\infty(R)$ clearly $\left\|\vec{y}\right\|_2^2 \leq nR^2$. Furthermore, by Lemma 22 and the choice of parameters $mc_e + n \leq 3n$. Consequently, by Lemma 19 and the fact that $\lambda = \frac{1}{c_a R^2}$ and $c_a < 1$ we have

$$\left\|\vec{x} - \vec{y}\right\|_{\mathbf{H}(\vec{x})} \leq \frac{12mc_e + 6n + 2\lambda\left\|\vec{y}\right\|_2^2}{\sqrt{c_e + \mu(\vec{x})}} \leq \frac{30n + 2\frac{n}{c_a}}{\sqrt{c_e + \mu(\vec{x})}} \leq \frac{4n}{c_a\sqrt{c_e}}$$

and therefore

$$\left\|\mathbf{S}_{x^{(k)}}^{-1}(s(\vec{x}^{(k)}) - s(\vec{y}))\right\|_\infty \leq \frac{1}{\sqrt{c_e}} \left\|\mathbf{S}_{x^{(k)}}^{-1}(s(\vec{x}^{(k)}) - s(\vec{y}))\right\|_{c_e\mathbf{I}+\mathbf{\Psi}} \leq \frac{4n}{c_a c_e} \quad .$$

Consequently, we have $(1 - \frac{4n}{c_a c_e})s_i(\vec{x}^{(k)}) \leq s_i(\vec{y}) \leq (1 + \frac{4n}{c_a c_e})s_i(\vec{x}^{(k)})$ for all $\vec{y} \in P^{(k)} \cap B_\infty(R)$. □

Now let us show how to compute a proof (or certificate) that the feasible region has small width on the direction $\vec{a}_i$.

Lemma 28. *Suppose that during some iteration k for $i = \mathrm{argmin}_j s_j(\vec{x}^{(k)})$ we have $s_i(\vec{x}^{(k)}) \leq \epsilon$. Let $(\vec{x}_*, \vec{\tau}_*) =$ **Centering**$(\vec{x}^{(k)}, \vec{\tau}^{(k)}, 64\log(2R/\epsilon), c_\Delta)$ where $\vec{\tau}^{(k)}$ is the τ at that point in the algorithm and let*

$$\vec{a}^* = \sum_{j\neq i} t_j\vec{a}_j \text{ where } t_j = \left(\frac{s(\vec{x}_*)_i}{s(\vec{x}_*)_j}\right)\left(\frac{c_e + e_j(\vec{x}_*, \vec{\tau}_*) + \psi_j(\vec{x}_*)}{c_e + e_i(\vec{x}_*, \vec{\tau}_*) + \psi_i(\vec{x}_*)}\right).$$

Then, we have that $\left\|\vec{a}_i + \vec{a}^*\right\|_2 \leq \frac{8\sqrt{n}\epsilon}{c_a c_e R}$ and $t_j \geq 0$ for all j. Furthermore, we have

$$\left(\sum_{j\neq i}^{O(n)} t_j a_j\right)^T \vec{x}_* - \sum_{j\neq i}^{O(n)} t_j b_j \leq \frac{4n\epsilon}{c_e} \quad .$$

Proof. By Lemma 14 and Lemma 21 we know that $\vec{e}(\vec{x}_*, \vec{\tau}_*) \le \frac{1}{2}c_e$ and $\delta_{\vec{e}(\vec{x}_*,\vec{\tau}_*)} \le \frac{\epsilon}{R}\sqrt{c_e + \mu(\vec{x}_*)}$. Since $\vec{e}(\vec{x}_*, \vec{\tau}_*) \le \frac{1}{2}c_e$, we have $t_j \ge 0$ for all j. Furthermore, by Lemma 20 and (5.19), we then have that with high probability in n

$$\begin{aligned}\left\|\vec{a}_i + \vec{a}^*\right\|_2 &\le \frac{2\epsilon}{(c_e + \mu(\vec{x}_*))}\left[\lambda\left\|\vec{x}_*\right\|_2 + \delta_e(\vec{x}_*)\sqrt{\frac{mc_e + n}{\epsilon^2} + \lambda}\right] \\ &\le \frac{2\epsilon}{c_e}\left[\frac{1}{c_a R^2}(3\sqrt{n}R) + \frac{2\epsilon}{R}\sqrt{\frac{3n}{\epsilon^2} + \frac{1}{c_a R^2}}\right] \\ &\le \frac{2\epsilon}{c_e}\left[\frac{3\sqrt{n}}{c_a R} + \frac{2\sqrt{3n}}{R} + \frac{2}{\sqrt{c_a}R}\right] \le \frac{8\sqrt{n}\epsilon}{c_a c_e R}.\end{aligned}$$

By Lemma 21 we know that $\vec{e}(\vec{x}_*, \vec{\tau}_*) \le \frac{1}{2}c_e$ and hence

$$\begin{aligned}\left(\sum_{j\ne i}^{O(n)} t_j a_j\right)^T \vec{x}_* - \sum_{j\ne i}^{O(n)} t_j b_j &= \sum_{j\ne i}^{O(n)} t_j s(\vec{x}_*)_j = s_i(\vec{x}_*)\sum_{j\ne i}^{O(n)}\left(\frac{c_e + e_j(\vec{x}_*, \vec{\tau}_*) + \psi_j(\vec{x}_*)}{c_e + e_i(\vec{x}_*, \vec{\tau}_*) + \psi_i(\vec{x}_*)}\right) \\ &\le s_i(\vec{x}_*)\sum_{j\ne i}^{O(n)}\left(\frac{\frac{3}{2}c_e + \psi_j(\vec{x}_*)}{\frac{1}{2}c_e + \psi_i(\vec{x}_*)}\right) \le s_i(\vec{x}_*)\frac{3mc_e + 2n}{c_e} \\ &\le \frac{3n}{c_e}s_i(\vec{x}_*) \le \frac{4n\epsilon}{c_e}\end{aligned}$$

where the last line follows the fact that $s_i(\vec{x}_*) \le 1.1 s_i(\vec{x}^{(k)}) \le 1.1\epsilon$ (Lemma 14). ◻

Lemma 29. *During our Cutting Plane Method, if $p_{\vec{e}}(\vec{x}^{(k)}) \ge 2n\log(\frac{nR}{c_a\epsilon}) + \frac{6n}{c_a}$, then we have $s_i(\vec{x}^{(k)}) \le \epsilon$ for some i.*

Proof. Recall that

$$p_{\vec{e}}(\vec{x}^{(k)}) = -\sum_{i\in[m]}(c_e + e_i)\log\left(s_i(\vec{x}^{(k)})_i/R\right) + \frac{1}{2}\log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_{x^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) + \frac{\lambda}{2}\left\|\vec{x}^{(k)}\right\|_2^2.$$

Using $\left\|\vec{x}^{(k)}\right\| \le 3\sqrt{n}R$ (5.19) and $\lambda = \frac{1}{c_a R^2}$, we have

$$p_{\vec{e}}(\vec{x}^{(k)}) \le -\sum_{i\in[m]}(c_e + e_i)\log(s(\vec{x}^{(k)})_i/R) + \frac{1}{2}\log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_{x^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) + \frac{5n}{c_a}.$$

Next, we note that $\left\|e_i\right\|_\infty \le c_e \le \frac{1}{4\log(6nR/\epsilon)}$ and $s_i\left(\vec{x}^{(k)}\right) \le 12\sqrt{n/\lambda} + 4\sqrt{n}R + \sqrt{\frac{1}{c_a\lambda}} \le 6\sqrt{n}R$ (Lemma 24). Hence, we have

$$p_{\vec{e}}(\vec{x}^{(k)}) \le \frac{1}{2}\log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_{x^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) + \frac{6n}{c_a} - \frac{n}{2}\log(\frac{\min_i s_i}{R}).$$

We prove $s_i(\vec{x}^{(k)}) \le \epsilon$ by contradiction. Since $p_{\vec{e}}(\vec{x}^{(k)}) \ge 2n\log(\frac{nR}{c_a\epsilon}) + \frac{6n}{c_a}$ and $s_i(\vec{x}^{(k)}) > \epsilon$ for all i, we have that $\frac{1}{2}\log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_{x^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) \ge n\log(\frac{nR}{c_a\epsilon})$. Using $\epsilon < R$, we have that

$$\sum_i \log\lambda_i\left(R^2\left(\mathbf{A}^T\mathbf{S}_{x^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) \ge n\log\left(\frac{n^2R^2}{c_a^2\epsilon^2}\right) \ge n\log\left(\frac{nR^2}{2c_a^2\epsilon^2} + R^2\lambda\right).$$

Therefore, we have $\log\lambda_{\max}\left(R^2\left(\mathbf{A}^T\mathbf{S}_{x^{(k)}}^{-2}\mathbf{A}+\lambda\mathbf{I}\right)\right)\geq\log\left(\frac{nR^2}{2c_a^2\epsilon^2}+R^2\lambda\right)$. Hence, we have some unit vector $\vec{v}$ such that $\vec{v}\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A}\vec{v}+\lambda\vec{v}^T\vec{v}\geq\frac{n}{2c_a^2\epsilon^2}+\lambda$. Thus,

$$\sum_i\frac{(\mathbf{A}\vec{v})_i^2}{s(\vec{x}^{(k)})_i^2}\geq\frac{n}{2c_a^2\epsilon^2}.$$

Lemma 22 shows that the number of constraints is bounded by $1+2n/c_d\leq n/(2c_a^2)$ and hence there is some i such that $\frac{(\mathbf{A}\vec{v})_i^2}{s(\vec{x}^{(k)})_i^2}\geq\frac{1}{\epsilon^2}$. Since $\vec{a}_i$ and $\vec{v}$ are unit vectors, we have $1\geq\langle\vec{a}_i,\vec{v}\rangle^2\geq s(\vec{x}^{(k)})_i^2/\epsilon^2$ and hence $s(\vec{x}^{(k)})_i\leq\epsilon$ (contradiction). ☐

Lemma 30. *With constant probability, the algorithm ends in* $10^{24}n\log(\frac{nR}{\epsilon})$ *iterations.* [2]

Proof. Theorem 26 shows that for all k

$$\mathbb{E}p_{\vec{e}(\vec{\tau}^{(k+1)},\vec{x}^{(k+1)})}(\vec{x}^{(k+1)})\quad\geq\quad p_{\vec{e}(\vec{\tau}^{(k)},\vec{x}^{(k)})}(\vec{x}^{(k)})-\frac{1}{10^{11}}+\frac{9\beta}{10^{11}}\tag{5.20}$$

where $\beta=1$ for the case of adding a constraint and $\beta=0$ for the case of removing a constraint. Now, for all t consider the random variable

$$\mathbf{X}_t=p_{\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)})-\frac{4.5m^{(t)}}{10^{11}}-\frac{3.5t}{10^{11}}$$

where $m^{(t)}$ is the number of constraints in iteration t of the algorithm. Then, since $m^{(t+1)}=m^{(t)}-1+2\beta$, (5.20) shows that

$$\begin{aligned}\mathbf{E}\mathbf{X}_{t+1}&\geq p_{\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)})-\frac{1}{10^{11}}+\frac{9\beta}{10^{11}}-\frac{4.5m^{(t+1)}}{10^{11}}-\frac{3.5(t+1)}{10^{11}}\\&=\mathbf{X}_t-\frac{1}{10^{11}}+\frac{9\beta}{10^{11}}-\frac{4.5(-1+2\beta)}{10^{11}}-\frac{3.5}{10^{11}}=\mathbf{X}_t.\end{aligned}$$

Hence, it is a sub-martingale. Let T be the iteration the algorithm outputs $\vec{x}^{(k)}$ or $P^{(k)}$. Optional stopping theorem shows that

$$\mathbb{E}\mathbf{X}_{\min(T,t)}\geq\mathbb{E}\mathbf{X}_0.\tag{5.21}$$

Since the polytope is $B_\infty(R)$, we have

$$\begin{aligned}p_{\vec{0}}(\vec{0})&=-\sum_{i\in[m^{(0)}]}c_e\log\left(s_i(\vec{0})/R\right)+\frac{1}{2}\log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_0^{-2}\mathbf{A}+\lambda\mathbf{I}\right)\right)+\frac{\lambda}{2}\left\|\vec{0}\right\|_2^2\\&\geq\frac{1}{2}\log\det\left(R^2\left(\frac{2}{R^2}\mathbf{I}+\frac{1}{c_aR^2}\mathbf{I}\right)\right)\\&\geq-15n\end{aligned}$$

[2]We have made no effort on improving this constant and we believe it can be improved to less than 300 using techniques in [8, 9].

where we used $c_a = 1/10^{10}$ on the last line. Hence, we have

$$\begin{aligned} \mathbf{X}_0 &\geq -15n - \frac{4.5m^{(0)}}{10^{11}} \\ &\geq -20n. \end{aligned}$$

Therefore, (5.21) shows that for all t we have

$$\begin{aligned} -20n &\leq \mathbb{E}\mathbf{X}_{\min(T,t)} \\ &= p\mathbb{E}\left[\mathbf{X}_{\min(T,t)}|T < t\right] + (1-p)\mathbb{E}\left[\mathbf{X}_{\min(T,t)}|T \geq t\right] \end{aligned} \tag{5.22}$$

where $p \overset{\text{def}}{=} \mathbb{P}(T < t)$.

Note that

$$\begin{aligned} \mathbb{E}\left[\mathbf{X}_{\min(T,t)}|T \geq t\right] &\leq \mathbb{E}\left[p_{\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)})|T \geq t\right] - \frac{4.5m^{(t)}}{10^{11}} - \frac{3.5t}{10^{11}}. \\ &\leq \mathbb{E}\left[p_{\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)})|T \geq t\right] - \frac{3.5t}{10^{11}}. \end{aligned}$$

Furthermore, by Lemma 29 we know that when $p_{\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)}) \geq 2n\log(\frac{nR}{c_a\epsilon}) + \frac{6n}{c_a}$, there is a slack that is too small and the algorithm terminates. Hence, we have

$$\mathbf{E}\left[\mathbf{X}_{\min(T,t)}|T \geq t\right] \leq 2n\log(\frac{nR}{c_a\epsilon}) + \frac{6n}{c_a} - \frac{3.5t}{10^{11}}.$$

The proof of Lemma 21 shows that the function value does not change by more than 1 in one iteration by changing $\vec{x}$ and can change by at most $mc_e\log(\frac{3nR}{\epsilon})$ by changing τ. Since by Lemma 22 we know that $m \leq 1+\frac{2n}{c_a}$ and $c_e = \frac{c_d}{4\log(6nR/\epsilon)}$, we have that $p_{\vec{e}}(\vec{x}) \leq 2n\log(\frac{nR}{c_a\epsilon})+\frac{7n}{c_a}$ throughout the execution of the algorithm. Therefore, we have

$$\mathbf{E}\left[\mathbf{X}_{\min(T,t)}|T \leq t\right] \leq \mathbb{E}_{T<t}p_{\vec{e}(\vec{\tau}^{(t)},\vec{x}^{(t)})}(\vec{x}^{(t)}) \leq 2n\log(\frac{nR}{c_a\epsilon}) + \frac{7n}{c_a}.$$

Therefore, (5.22) shows that

$$-20n \leq 2n\log\left(\frac{nR}{c_a\epsilon}\right) + \frac{7n}{c_a} - (1-p)\frac{3.5t}{10^{11}}.$$

Hence, we have

$$\begin{aligned} (1-p)\frac{3.5t}{10^{11}} &\leq 2n\log\left(\frac{nR}{c_a\epsilon}\right) + \frac{7n}{c_a} + 20n \\ &\leq 2n\log\left(\frac{nR}{c_a\epsilon}\right) + \frac{7n}{c_a} + 20n \\ &= 2n\log\left(\frac{Rn}{c_a\epsilon}\right) + 8\cdot 10^{10}n. \end{aligned}$$

Thus, we have

$$\mathbb{P}(T < t) = p \geq 1 - \frac{1}{t}\left(10^{11}n\log\left(\frac{nR}{\epsilon}\right) + 10^{22}n\right).$$

$\square$

Now, we gather all the result as follows:

Theorem 31 (Our Cutting Plane Method)**.** *Let $K \subseteq \mathbb{R}^n$ be a non-empty set contained in a box of radius R, i.e. $K \subseteq B_\infty(R)$. For any $\epsilon \in (0, R)$ in expected time $O(n\text{SO}_{\Omega(\epsilon/\sqrt{n})}(K)\log(nR/\epsilon)+n^3\log^{O(1)}(nR/\epsilon))$ our cutting plane method either outputs $\vec{x} \in K$ or finds a polytope $P = \{\vec{x} : \mathbf{A}\vec{x} \geq \vec{b}\} \supseteq K$ such that*

1. *P has $O(n)$ many constraints (i.e. $\mathbf{A} \in \mathbb{R}^{O(n)\times n}$ and $\vec{b} \in \mathbb{R}^{O(n)}$).*
2. *Each constraint of P is either an initial constraint from $B_\infty(R)$ or of the form $\langle \vec{a}, \vec{x}\rangle \geq b - \delta$ where $\langle \vec{a}, \vec{x}\rangle \geq b$ is a normalized hyperplane (i.e. $\left\|\vec{a}\right\|_2 = 1$) returned by the separation oracle and $\delta = \Omega\left(\frac{\epsilon}{\sqrt{n}}\right)$.*
3. *The polytope P has small width with respect to some direction $\vec{a}_1$ given by one of the constraints, i.e.*
$$\max_{\vec{y}\in P\cap B_\infty(R)} \langle \vec{a}_1, \vec{y}\rangle - \min_{\vec{y}\in P\cap B_\infty(R)} \langle \vec{a}_1, \vec{y}\rangle \leq O\left(n\epsilon \log(nR/\epsilon)\right)$$
4. *Furthermore, the algorithm produces a proof of the fact above involving convex combination of the constraints, namely, non-negatives $t_2, ..., t_{O(n)}$ and $\vec{x} \in P$ such that*
   (a) $\left\|\vec{x}\right\|_2 = O\left(\sqrt{n}R\right)$,
   (b) $\left\|\vec{a}_1 + \sum_{i=2}^{O(n)} t_i \vec{a}_i\right\|_2 = O\left(\frac{\epsilon}{R}\sqrt{n}\log(nR/\epsilon)\right)$,
   (c) $\vec{a}_1^T\vec{x} - \vec{b}_1 \leq \epsilon$,
   (d) $\left(\sum_{i=2}^{O(n)} t_i a_i\right)^T \vec{x} - \sum_{i=2}^{O(n)} t_i b_i \leq O(n\epsilon\log(nR/\epsilon))$.

Proof. Our algorithm either finds $\vec{x} \in K$ or we have $s_i(\vec{x}^{(k)}) < \epsilon$. When $s_i(\vec{x}^{(k)}) < \epsilon$, we apply Lemma 28 to construct the polytope P and the linear combination $\sum_{i=2}^{O(n)} t_i\vec{a}_i$.

Notice that each iteration of our algorithm needs to solve constant number of linear systems and implements the sampling step to find $\vec{\Delta}^{(k)} \in \mathbb{R}^n$ s.t. $\mathbf{E}[\vec{\Delta}^{(k)}] = \vec{\psi}(\vec{x}^{(k)}) - \vec{\psi}(\vec{x}^{(k-1)})$. Theorem 33 shows how to do the sampling in $\tilde{O}(1)$ many linear systems. Hence, in total, each iterations needs to solve $\tilde{O}(1)$ many linear systems plus nearly linear work. To output the proof for (4), we use Lemma 28.

Note that the linear systems the whole algorithm need to solve is of the form
$$(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I})^{-1}\vec{x} = \vec{y}.$$
where the matrix $\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}$ can be written as $\overline{\mathbf{A}}^T\mathbf{D}\overline{\mathbf{A}}$ for the matrix $\overline{\mathbf{A}} = [\mathbf{A}\ \mathbf{I}]$ and diagonal matrix
$$\mathbf{D} = \begin{bmatrix} \mathbf{S}^{-2} & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{I} \end{bmatrix}.$$
Note that Lemma 14 shows that $\left\| \left(\mathbf{S}^{(k)}\right)^{-1}(\vec{s}^{(k+1)} - \vec{s}^{(k)})\right\|_2 \leq \frac{1}{10}$ for the k^{th} and $(k+1)^{th}$ linear systems we solved in the algorithm. Hence, we have $\left\| \left(\mathbf{D}^{(k)}\right)^{-1}(\vec{d}^{(k+1)} - \vec{d}^{(k)})\right\|_2 \leq$

$\frac{1}{10}$. In [177], they showed how to solve such sequence of systems in $\tilde{O}(n^2)$ amortized cost. Moreover, since our algorithm always changes the constraints by δ amount where $\delta = \Omega(\frac{\epsilon}{\sqrt{n}})$ an inexact separation oracle $\mathrm{SO}_{\Omega(\epsilon/\sqrt{n})}$ suffices. (see Def 1). Consequently, the total work $O(n\mathrm{SO}_{\Omega(\epsilon/\sqrt{n})}(K)\log(nR/\epsilon) + n^3\log^{O(1)}(nR/\epsilon))$. Note that as the running time holds with only constant probability, we can restart the algorithm whenever the running time is too large.

To prove (2), we note that from the algorithm description, we know the constraints are either from $B_\infty(R)$ or of the form $\vec{a}^T\vec{x} \geq \vec{a}^T\vec{x}^{(k)} - \delta$ where

$$\delta = \sqrt{\frac{\vec{a}^T(\mathbf{A}^T\mathbf{S}_{\vec{x}^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I})^{-1}\vec{a}}{c_a}}.$$

From the proof of Lemma 29, we know that if $\lambda_{\max}(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}) \geq \frac{n}{c_a^2\epsilon^2}$, then there is $s_i < \epsilon$. Hence, we have $\lambda_{\min}((\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I})^{-1}) \geq \frac{c_a^2\epsilon^2}{n}$. Since $\vec{a}$ is a unit vector, we have

$$\sqrt{\frac{\vec{a}^T(\mathbf{A}^T\mathbf{S}_{\vec{x}^{(k)}}^{-2}\mathbf{A} + \lambda\mathbf{I})^{-1}\vec{a}}{c_a}} \geq \sqrt{\frac{c_a\epsilon^2}{n}}.$$

□

6 Technical Tools

In this section we provide stand-alone technical tools we use in our cutting plane method in Section 5. In Section 6.1 we show how to efficiently compute accurate estimates of changes in leverage scores using access to a linear system solver. In Section 6.2 we study what we call the "Stochastic Chasing $\vec{0}$ Game" and show how to maintain that a vector is small in ℓ_∞ norm by making small coordinate updates while the vector changes randomly in ℓ_2 norm.

6.1 Estimating Changes in Leverage Scores

In previous sections, we needed to compute leverage scores accurately and efficiently for use in our cutting plane method. Note that the leverage score definition we used was

$$\psi(\vec{w})_i = \vec{\mathbb{1}}_i^T\sqrt{\mathbf{W}}\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A} + \lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\sqrt{\mathbf{W}}\vec{\mathbb{1}}_i$$

for some $\lambda > 0$ which is different from the standard definition

$$\sigma(\vec{w})_i = \vec{\mathbb{1}}_i^T\sqrt{\mathbf{W}}\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}\mathbf{A}^T\sqrt{\mathbf{W}}\vec{\mathbb{1}}_i.$$

However, note that the matrix $\mathbf{A}^T\mathbf{W}\mathbf{A} + \lambda\mathbf{I}$ can be written as $\overline{\mathbf{A}}^T\mathbf{D}\overline{\mathbf{A}}$ for the matrix $\overline{\mathbf{A}}^T = [\mathbf{A}^T\ \mathbf{I}]$ and diagonal matrix

$$\mathbf{D} = \begin{bmatrix} \mathbf{W} & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{I} \end{bmatrix}.$$

and therefore computing ψ is essentially same as computing typical leverage scores. Consequently, we use the standard definition σ to simplify notation.

In [240], Spielman and Srivastava observed that leverage scores can be written as the norm of certain vectors

$$\sigma(\vec{w})_i = \left\| \sqrt{\mathbf{W}}\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}\mathbf{A}^T\sqrt{\mathbf{W}}\vec{\mathbb{1}}_i \right\|_2^2$$

and therefore leverage scores can be approximated efficiently using dimension reduction. Unfortunately, the error incurred by this approximation is too large to use inside the cutting point method. In this section, we show how to efficiently approximate the *change* of leverage score more accurately.

In particular, we show how to approximate $\sigma(\vec{w})-\sigma(\vec{v})$ for any given $\vec{w},\vec{v}$ with $\left\|\log(\vec{w})-\log(\vec{v})\right\|_2 \ll 1$. Our algorithm breaks $\sigma(\vec{w})_i-\sigma(\vec{v})_i$ into the sum of the norm of small vectors and then uses the Johnson-Lindenstrauss dimension reduction to approximate the norm of each vector separately. Our algorithm makes use of the following version of Johnson-Lindenstrauss.

Lemma 32 ([1])**.** *Let $0 \le \epsilon \le \frac{1}{2}$ and let $\vec{x}_1,...,\vec{x}_m \in \mathbb{R}^n$ be arbitrary m points. For $k = O(\epsilon^{-2}\log(m))$ let $\mathbf{Q}$ be a $k\times n$ random matrix with each entry sampled from $\{-\frac{1}{\sqrt{k}},\frac{1}{\sqrt{k}}\}$ uniformly and independently. Then, $\mathbb{E}\left\|\mathbf{Q}\vec{x}_i\right\|^2 = \left\|\vec{x}_i\right\|^2$ for all $i\in[m]$ and with high probability in m we have that for all $i\in[m]$*

$$(1-\epsilon)\left\|\vec{x}_i\right\|^2 \le \left\|\mathbf{Q}\vec{x}_i\right\|^2 \le (1+\epsilon)\left\|\vec{x}_i\right\|^2 \quad .$$

Algorithm 3: $\widehat{h} = \texttt{LeverageChange}(\mathbf{A},\vec{v},\vec{w},\epsilon)$

```
Input: $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\vec{v},\vec{w}\in\mathbb{R}^m_{>0}$, $\epsilon\in(0,0.5)$.
Given: $\left\|\mathbf{V}^{-1}(\vec{v}-\vec{w})\right\|_2 \le \frac{1}{10}$; $\mathbf{A}^T\mathbf{V}\mathbf{A}$ and $\mathbf{A}^T\mathbf{W}\mathbf{A}$ are invertible.
Sample $\mathbf{Q}_d\in\mathbb{R}^{O(\epsilon^{-2}\log(m))\times n}$ as in Lemma 32.
Let $\hat{d}_i = \left\|\mathbf{Q}_d\sqrt{\mathbf{W}}\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}\mathbf{A}^T\vec{\mathbb{1}}_i\right\|_2^2$ for all $i\in[n]$.
Let $t = O\left(\log(\epsilon^{-1})\right)$.
Sample $\mathbf{Q}_f\in\mathbb{R}^{O(\epsilon^{-2}\log(mt))\times n}$ as in Lemma 32.
Pick positive integer $u$ randomly such that $\Pr[u=i] = (\frac{1}{2})^i$.
for $j\in\{1,2,\cdots,t\}\cup\{t+u\}$ do
    if $j$ is even then
        Let $\hat{f}_i^{(j)} = \left\|\mathbf{Q}_f\sqrt{\mathbf{V}}\mathbf{A}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1}\left(\mathbf{A}^T\left(\mathbf{V}-\mathbf{W}\right)\mathbf{A}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1}\right)^{\frac{j}{2}}\mathbf{A}^T\vec{\mathbb{1}}_i\right\|_2^2$.
    else
        Let $\Delta^+ \stackrel{\text{def}}{=} (\mathbf{V}-\mathbf{W})^+$, i.e. the matrix $\mathbf{V}-\mathbf{W}$ with negative entries set to 0.
        Let $\Delta^- \stackrel{\text{def}}{=} (\mathbf{W}-\mathbf{V})^+$, i.e. the matrix $\mathbf{W}-\mathbf{V}$ with negative entries set to 0.
        Let $\hat{\alpha}_i^{(j)} = \left\|\mathbf{Q}_f\sqrt{\Delta^+}\mathbf{A}(\mathbf{A}^T\mathbf{V}\mathbf{A})^{-1}(\mathbf{A}^T\left(\mathbf{V}-\mathbf{W}\right)\mathbf{A}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1})^{\frac{j-1}{2}}\mathbf{A}^T\vec{\mathbb{1}}_i\right\|_2^2$.
        Let $\hat{\beta}_i^{(j)} = \left\|\mathbf{Q}_f\sqrt{\Delta^-}\mathbf{A}(\mathbf{A}^T\mathbf{V}\mathbf{A})^{-1}(\mathbf{A}^T(\mathbf{V}-\mathbf{W})\mathbf{A}(\mathbf{A}^T\mathbf{V}\mathbf{A})^{-1})^{\frac{j-1}{2}}\mathbf{A}^T\vec{\mathbb{1}}_i\right\|_2^2$.
        Let $\hat{f}_i^{(j)} = \hat{\alpha}_i^{(j)} - \hat{\beta}_i^{(j)}$.
    end
end
Let $\hat{f}_i = 2^u\hat{f}_i^{(t+u)} + \sum_{j=1}^t \hat{f}_i^{(j)}$.
Output: $\hat{h}_i = (w_i-v_i)\hat{d}_i + v_i\hat{f}_i$. for all $i\in[m]$
```

Theorem 33. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\vec{v}, \vec{w} \in \mathbb{R}^m_{>0}$ be such that $\alpha \stackrel{\text{def}}{=} \left\|\mathbf{V}^{-1}(\vec{v}-\vec{w})\right\|_2 \leq \frac{1}{10}$ and both $\mathbf{A}^T\mathbf{V}\mathbf{A}$ and $\mathbf{A}^T\mathbf{W}\mathbf{A}$ are invertible. For any $\epsilon \in (0, 0.5)$, Algorithm 3 generates a random variable $\widehat{h}$ such that $\mathbf{E}\hat{h} = \sigma(\vec{w}) - \sigma(\vec{v})$ and with high probability in m, we have $\|\widehat{h} - (\sigma(\vec{w}) - \sigma(\vec{v}))\|_2 \leq O(\alpha\epsilon)$. Furthermore, the expected running time is $\tilde{O}((\text{nnz}(\mathbf{A}) + \text{LO})/\epsilon^2)$ where LO is the amount of time needed to apply $\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1}$ and $\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}$ to a vector.*

Proof. First we bound the running time. To compute $\hat{d}_i, \hat{f}_i^{(j)}, \hat{\alpha}_i^{(j)}, \hat{\beta}_i^{(j)}$, we simply perform matrix multiplications from the left and then consider the dot products with each of the rows of $\mathbf{A}$. Naively this would take time $\tilde{O}((t+u)^2\log(mt)(\text{nnz}(\mathbf{A})+\text{LO}))$. However, we can reuse the computation in computing high powers of j to only take time $\tilde{O}((t+u)\log(mt)(\text{nnz}(\mathbf{A})+\text{LO}))$. Now since $\mathbf{E}[u]$ is constant we see that the total running time is as desired. It only remains to prove the desired properties of $\hat{h}$.

First we note that we can re-write leverage score differences using

$$\sigma(\vec{w})_i - \sigma(\vec{v})_i = (w_i - v_i)\left[\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}\mathbf{A}^T\right]_{ii} + v_i\left[\mathbf{A}\left(\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1} - \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1}\right)\mathbf{A}^T\right]_{ii} \quad .$$

Consequently, for all $i \in [m]$, if we let

$$\begin{aligned} d_i &\stackrel{\text{def}}{=} \vec{\mathbb{1}}_i^T\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}\mathbf{A}^T\vec{\mathbb{1}}_i, \\ f_i &\stackrel{\text{def}}{=} \vec{\mathbb{1}}_i^T\mathbf{A}\left[\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1} - \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1}\right]\mathbf{A}^T\vec{\mathbb{1}}_i. \end{aligned}$$

then

$$\sigma(\vec{w})_i - \sigma(\vec{v})_i = (w_i - v_i)d_i + (v_i)f_i \quad . \tag{6.1}$$

We show that $\hat{d}_i$ approximates d and $\hat{f}_i$ approximate $\hat{f}$ well enough to satisfy the statements in the Theorem.

First we bound the quality of $\hat{d}_i$. Note that $d_i = \left\|\sqrt{\mathbf{W}}\mathbf{A}\left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1}\mathbf{A}^T\vec{\mathbb{1}}_i\right\|_2^2$. Consequently, Lemma 32 shows that $\mathbf{E}[\hat{d}_i] = d_i$ and that with high probability in m we have $(1-\epsilon)d_i \leq \hat{d}_i \leq (1+\epsilon)d_i$ for all $i \in [m]$. Therefore, with high probability in m, we have

$$\begin{aligned} \left\|(\vec{w}-\vec{v})\,\widehat{d} - (\vec{w}-\vec{v})\,\vec{d}\right\|_2^2 &= \sum_{i\in[m]}(w_i - v_i)^2\left(\widehat{d}_i - d_i\right)^2 \leq \epsilon^2\sum_{i\in[m]}(w_i-v_i)^2 d_i^2 \\ &= \epsilon^2\sum_{i\in[m]}(w_i - v_i)^2\left(\frac{\sigma(\vec{w})_i}{\vec{w}_i}\right)^2 \leq 2\epsilon^2\sum_{i\in[m]}\left(\frac{w_i - v_i}{v_i}\right)^2 \quad . \end{aligned}$$

Next we show how to estimate f. Let $\mathbf{X} \stackrel{\text{def}}{=} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}\mathbf{A}^T\left(\mathbf{V}-\mathbf{W}\right)\mathbf{A}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}$. By the assumption on α we know $-\frac{1}{2}\mathbf{V} \prec \mathbf{V} - \mathbf{W} \prec \frac{1}{2}\mathbf{V}$ and therefore $-\frac{1}{2}\mathbf{I} \prec \mathbf{X} \prec \frac{1}{2}\mathbf{I}$. Consequently we have that

$$\begin{aligned} \left(\mathbf{A}^T\mathbf{W}\mathbf{A}\right)^{-1} &= \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}\left(\mathbf{I}-\mathbf{X}\right)^{-1}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \\ &= \sum_{j=0}^{\infty}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}\mathbf{X}^j\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}. \end{aligned}$$

and therefore

$$\begin{aligned} f_i &= \vec{\mathbb{1}}_i^T \mathbf{A} \left(\sum_{j=0}^{\infty} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{X}^j \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} - \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \right) \mathbf{A}^T \vec{\mathbb{1}}_i \\ &= \sum_{j=1}^{\infty} f_i^{(j)} \quad \text{where} \quad f_i^{(j)} \stackrel{\text{def}}{=} \vec{\mathbb{1}}_i^T \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{X}^j \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{A}^T \vec{\mathbb{1}}_i \quad . \end{aligned}$$

Furthermore, using the definition of $\mathbf{X}$ we have that for even j

$$\begin{aligned} f_i^{(j)} &= \left\| \mathbf{X}^{\frac{j}{2}} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{A}^T \vec{\mathbb{1}}_i \right\|_2^2 \\ &= \left\| \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \left(\mathbf{A}^T \left(\mathbf{V}-\mathbf{W}\right) \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \right)^{\frac{j}{2}} \mathbf{A}^T \vec{\mathbb{1}}_i \right\|_2^2 \\ &= \left\| \sqrt{\mathbf{V}} \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \left(\mathbf{A}^T \left(\mathbf{V}-\mathbf{W}\right) \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \right)^{\frac{j}{2}} \mathbf{A}^T \vec{\mathbb{1}}_i \right\|_2^2 \end{aligned}$$

For odd j, using our definition of $\mathbf{\Delta}^+$ and $\mathbf{\Delta}^-$ we have that

$$\begin{aligned} f_i^{(j)} &= \vec{\mathbb{1}}_i^T \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{X}^j \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{A}^T \vec{\mathbb{1}}_i \\ &= \vec{\mathbb{1}}_i^T \mathbf{A} \left(\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \mathbf{A}^T \left(\mathbf{V}-\mathbf{W}\right) \mathbf{A} \right)^{\frac{j-1}{2}} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \mathbf{A}^T \left(\mathbf{V}-\mathbf{W}\right) \\ &\quad \times \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \left(\mathbf{A}^T \left(\mathbf{V}-\mathbf{W}\right) \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \right)^{\frac{j-1}{2}} \mathbf{A}^T \vec{\mathbb{1}}_i \\ &= \alpha_i^{(j)} - \beta_i^{(j)} \end{aligned}$$

where

$$\begin{aligned} \alpha_i^{(j)} &\stackrel{\text{def}}{=} \left\| \sqrt{\Delta^+} \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \left(\mathbf{A}^T \left(\mathbf{W}-\mathbf{V}\right) \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \right)^{\frac{j-1}{2}} \mathbf{A}^T \vec{\mathbb{1}}_i \right\|_2^2 , \\ \beta_i^{(j)} &\stackrel{\text{def}}{=} \left\| \sqrt{\Delta^-} \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \left(\mathbf{A}^T \left(\mathbf{W}-\mathbf{V}\right) \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1} \right)^{\frac{j-1}{2}} \mathbf{A}^T \vec{\mathbb{1}}_i \right\|_2^2 \quad . \end{aligned}$$

Consequently, by Lemma 32 and the construction, we see that

$$\mathbf{E}\hat{f}_i = \sum_{j=1}^{\infty} f_i^{(j)} = f_i$$

and therefore $\mathbf{E}\hat{h} = \sigma(\vec{w}) - \sigma(\vec{v})$ as desired. All that remains is to bound the variance of $\hat{f}_i$.

To bound the variance of $\hat{f}$, let $|\mathbf{X}| = \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{A}^T \left|\mathbf{W}-\mathbf{V}\right| \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}$. Note that $-\frac{1}{4}\mathbf{I} \preceq -|\mathbf{X}| \preceq \mathbf{X} \preceq |\mathbf{X}| \preceq \frac{1}{4}\mathbf{I}$ and consequently for all j

$$\begin{aligned} g_i^{(j)} &\stackrel{\text{def}}{=} \vec{\mathbb{1}}_i^T \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} |\mathbf{X}|^j \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{A}^T \vec{\mathbb{1}}_i \\ &\leq \frac{1}{4^{j-1}} \vec{\mathbb{1}}_i^T \mathbf{A} \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} |\mathbf{X}| \left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2} \mathbf{A}^T \vec{\mathbb{1}}_i \\ &\stackrel{\text{def}}{=} \frac{1}{v_i 4^{j-1}} \vec{\mathbb{1}}_i^T \mathbf{P}_v \mathbf{\Delta} \mathbf{P}_v \vec{\mathbb{1}}_i \end{aligned}$$

where $\mathbf{P}_v = \sqrt{\mathbf{V}}\mathbf{A}\left(\mathbf{A}^T\mathbf{V}\mathbf{A}\right)^{-1/2}\mathbf{A}^T\sqrt{\mathbf{V}}$ and $\mathbf{\Delta}$ is a diagonal matrix with $\mathbf{\Delta}_{ii} = \left|\frac{w_i - v_i}{v_i}\right|$. Using that $\mathbf{0} \preceq \mathbf{P}_v \preceq \mathbf{I}$, we have that for all j

$$\begin{aligned}
(4^{j-1})^2 \sum_{i=1}^{m}\left(v_i g_i^{(j)}\right)^2 &= \sum_{i=1}^{m}\left(\vec{\mathbb{1}}_i^T \mathbf{P}_v \mathbf{\Delta} \mathbf{P}_v \vec{\mathbb{1}}_i\right)^2 = \mathrm{Tr}\left(\mathbf{P}_v\mathbf{\Delta}\mathbf{P}_v\mathbf{P}_v\mathbf{\Delta}\mathbf{P}_v\right) \\
&\leq \mathrm{Tr}\left(\mathbf{P}_v\mathbf{\Delta}\mathbf{\Delta}\mathbf{P}_v\right) = \mathrm{Tr}\left(\mathbf{\Delta}\mathbf{P}_v\mathbf{P}_v\mathbf{\Delta}\right) \\
&\leq \mathrm{Tr}\left(\mathbf{\Delta}^2\right) = \sum_{i=1}^{m}\left(\frac{w_i - v_i}{v_i}\right)^2 \leq \alpha^2
\end{aligned}$$

and thus $\left\|\mathbf{V}\vec{g}^{(j)}\right\|_2 \leq \frac{4\alpha}{4^j}$. Furthermore, since $\mathbf{\Delta}^+ \preceq |\mathbf{W}-\mathbf{V}|$ and $\mathbf{\Delta}^- \preceq |\mathbf{W}-\mathbf{V}|$ we have that $\left|\alpha_i^{(j)}\right| \leq g_i^{(j)}$ and $\left|\beta_i^{(j)}\right| \leq g_i^{(j)}$. Consequently, by Lemma 32 again, we have

$$\begin{aligned}
\left\|\mathbf{V}\hat{f}^{(j)} - \mathbf{V}\vec{f}^{(j)}\right\|_2^2 &= \sum_i v_i^2\left(\hat{f}_i^{(j)} - f_i^{(j)}\right)^2 \\
&\leq 2\sum_i v_i^2\left(\hat{\alpha}_i^{(j)} - \alpha_i^{(j)}\right)^2 + 2\sum_i v_i^2\left(\hat{\beta}_i^{(j)} - \beta_i^{(j)}\right)^2 \\
&\leq 2\epsilon^2\sum_i v_i^2\left(\left(\alpha_i^{(j)}\right)^2 + \left(\beta_i^{(j)}\right)^2\right) \\
&\leq 4\epsilon^2\sum_i\left(v_i g_i^{(j)}\right)^2 \leq \frac{4\alpha^2\epsilon^2}{(4^{j-1})^2}\,.
\end{aligned}$$

Putting this all together we have that

$$\begin{aligned}
\left\|\mathbf{V}\hat{f} - \mathbf{V}\vec{f}\right\|_2 &\leq \left\|2^u\mathbf{V}\hat{f}^{(t+u)} + \sum_{j=1}^{t}\mathbf{V}\hat{f}^{(j)} - \sum_{j=1}^{\infty}\mathbf{V}\vec{f}^{(j)}\right\|_2 \\
&\leq 2^u\left\|\mathbf{V}\hat{f}^{(t+u)}\right\|_2 + \sum_{j=1}^{t}\left\|\mathbf{V}\hat{f}^{(j)} - \mathbf{V}\vec{f}^{(j)}\right\|_2 + \sum_{j=t+1}^{\infty}\left\|\mathbf{V}\vec{f}^{(j)}\right\|_2 \\
&\leq 2^u\frac{4\alpha}{4^{t+u-1}} + \sum_{j=1}^{t}\frac{2\alpha\epsilon}{4^{j-1}} + \sum_{j=t+1}^{\infty}\frac{2\alpha}{4^{j-1}} \\
&= O\left(\alpha\epsilon + \frac{\alpha}{4^t}\right).
\end{aligned}$$

Consequently, since $t = O(\log(\epsilon^{-1}))$ we have the desired result. $\square$

6.2 The Stochastic Chasing $\vec{0}$ Game

To avoid computing leverage scores exactly, in Section 6.1 we showed how to estimate the difference of leverage scores and use these to update the leverage scores. However, if we only applied this technique, the error of leverage scores would accumulate in the algorithm and we need to fix it. Naturally, one may wish to use dimension reduction to compute a multiplicative approximation to the leverage scores and update our computed value if the error is too large. However, this strategy would fail if there are too many rows with inaccurate leverage scores in the same iteration. In this case, we would change the central point too much that we are not able to recover. In this section, we present this update problem in a general form that we call *Stochastic Chasing 0 game* and provide an effective strategy for playing this game.

The *Stochastic chasing 0 game* is as follows. There is a player, a stochastic adversary, and a point $\vec{x} \in \mathbb{R}^m$. The goal of the player is to keep the point close to $\vec{0} \in \mathbb{R}^m$ in ℓ_∞ norm and the goal of the stochastic adversary is to move $\vec{x}$ away from $\vec{0}$. The game proceeds for an infinite number of iterations where in each iteration the stochastic adversary moves the current point $\vec{x}^{(k)} \in \mathbb{R}^m$ to some new point $\vec{x}^{(k)} + \vec{\Delta}^{(k)} \in \mathbb{R}^m$ and the player needs to respond. The stochastic adversary cannot move the $\vec{\Delta}^{(k)}$ arbitrarily, instead he is only allowed to choose a probability distribution $\mathcal{D}^{(k)}$ and sample $\vec{\Delta}^{(k)}$ from it. Furthermore, it is required that $\mathbb{E}_{\mathcal{D}^{(k)}}\vec{\Delta} = \vec{0}$ and $\left\|\vec{\Delta}\right\|_2^2 \leq c$ for some fixed c and all $\vec{\Delta} \in \mathcal{D}^{(k)}$. The player does not know $\vec{x}^{(k)}$ or the distribution $\mathcal{D}^{(k)}$ or the move $\vec{\Delta}^{(k)}$ of the stochastic adversary. All the player knows is some $\vec{y}^{(k)} \in \mathbb{R}^n$ that is close to $\vec{x}^{(k)}$ in ℓ_∞ norm. With this information, the player is allowed to choose one coordinate i and set $x_i^{(k+1)}$ to be zero and for other j, we have $x_j^{(k+1)} = x_j^{(k)} + \Delta_j^{(k)}$.

The question we would like to address is, what strategy the player should choose to keep $\vec{x}^{(k)}$ close to $\vec{0}$ in ℓ_∞ norm? We show that there is a trivial strategy that performs well: simply pick the largest coordinate and set it to 0.

Algorithm 4: Stochastic chasing $\vec{0}$ game

```
**Constant:** $c > 0, R > 0$.
Let $\vec{x}^{(1)} = \vec{0} \in \mathbb{R}^m$.
**for** $k = 1$ *to* $\infty$ **do**
    **Stochastic Adversary**: Pick $\mathcal{D}^{(k)}$ such that $\mathbb{E}_{\mathcal{D}^{(k)}}\vec{\Delta} = \vec{0}$ and $\left\|\vec{\Delta}\right\|_2 \leq c$ all
      $\vec{\Delta} \in \mathcal{D}^{(k)}$.
    **Stochastic Adversary**: Pick $\vec{y}^{(k)} \in \mathbb{R}^m$ such that $\left\|\vec{y}^{(k)} - \vec{x}^{(k)}\right\|_\infty \leq R$.
    **Player**: Pick a coordinate $i^{(k)}$ using only $\vec{y}^{(k)}$.
    Sample $\vec{\Delta}^{(k)}$ from $\mathcal{D}^{(k)}$.
    Set $x_{i^{(k)}}^{(k+1)} = 0$ and $x_j^{(k+1)} = x_j^{(k)} + \Delta_j^{(k)}$ for all $j \neq i^{(k)}$.
**end**
```

Theorem 34. *Using the strategy $i^{(k)} = \arg\max_i \left|y_i^{(k)}\right|$, with probability at least $1-p$, we have*

$$\left\|\vec{x}^{(k)}\right\|_\infty \leq 2(c+R)\log\left(4mk^2/p\right)$$

for all k in the Stochastic Chasing $\vec{0}$ Game.

Proof. Consider the potential function $\Phi(\vec{x}) = \sum_i e^{\alpha x_i} + \sum_i e^{-\alpha x_i}$ where α is to be determined. Now for all x we know that $e^x \leq 1 + x + \frac{x^2}{2}e^{|x|}$ and therefore for all $|\delta| \leq c$, x and α, we have

$$e^{\alpha x + \alpha\delta} \leq e^{\alpha x} + \alpha\delta e^{\alpha x} + \frac{1}{2}\alpha^2\delta^2 e^{\alpha x + |\alpha| c} \quad .$$

Consequently,

$$\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\Phi(\vec{x}^{(k)}+\vec{\Delta}) \leq \Phi(\vec{x}^{(k)}) + \alpha\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\left(\sum_{i\in[m]} e^{\alpha x_i^{(k)}}\Delta_i - \sum_{i\in[m]} e^{-\alpha x_i^{(k)}}\Delta_i\right)$$
$$+\frac{\alpha^2}{2}e^{\alpha\left\|\vec{\Delta}\right\|_\infty}\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\left(\sum_{i\in[m]} e^{\alpha x_i^{(k)}}\Delta_i^2 + \sum_{i\in[m]} e^{-\alpha x_i^{(k)}}\Delta_i^2\right) \quad .$$

Since $\mathbb{E}_{\mathcal{D}^{(k)}}\vec{\Delta} = \vec{0}$ and $\left\|\vec{\Delta}\right\|_2 \leq c$, we have $\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\left(\sum_i e^{\alpha x_i^{(k)}}\Delta_i - \sum_i e^{-\alpha x_i^{(k)}}\Delta_i\right) = 0$ and

$$\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\left(\sum_i e^{\alpha x_i^{(k)}}\Delta_i^2 + \sum_i e^{-\alpha x_i^{(k)}}\Delta_i^2\right) \leq \mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\left(\sum_i \Delta_i^2\right)\left(\max_i e^{\alpha x_i^{(k)}} + \max_i e^{-\alpha x_i^{(k)}}\right)$$
$$\leq 2c^2 \max_i e^{\alpha\left|x_i^{(k)}\right|} \quad .$$

Letting $\eta^{(k)} = \max_i e^{\alpha\left|x_i^{(k)}\right|}$, we then have

$$\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\Phi(\vec{x}^{(k)}+\vec{\Delta}) \leq \Phi(\vec{x}^{(k)}) + \alpha^2 e^{\alpha c}c^2\eta^{(k)}.$$

Since $i^{(k)} = \arg\max_i \left|y_i^{(k)}\right|$ and $\left\|\vec{y}^{(k)} - \vec{x}^{(k)}\right\|_\infty \leq R$, the player setting $x_{i^{(k)}}^{(k+1)} = 0$ decreases Φ by at least $e^{-\alpha(R+c)}\eta^{(k)}$. Hence, we have

$$\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\Phi(\vec{x}^{(k+1)}) \quad\leq\quad \Phi(\vec{x}^{(k)}) + \alpha^2 e^{\alpha c}c^2\eta^{(k)} - e^{-\alpha(R+c)}\eta^{(k)}.$$

Picking $\alpha = \frac{1}{2(c+R)}$, we have $e^{2\alpha(c+R)}(\alpha(c+R))^2 \leq 1$ and hence $\alpha^2 e^{\alpha c}c^2 \leq e^{-\alpha(R+c)}$. Therefore, we have that

$$\mathbb{E}_{\vec{\Delta}\in\mathcal{D}^{(k)}}\Phi(\vec{x}^{(k+1)}) \leq \mathbf{E}\Phi(\vec{x}^{(k)}) \leq ... \leq \Phi(\vec{x}^{(1)}) = 2m\,.$$

Consequently, by Markov's inequality we have that $\Pr[\Phi(\vec{x}^{(k)}) \geq \lambda_k] \leq \frac{2m}{\lambda_k}$ for any λ_k. Furthermore, since clearly $\Phi(\vec{x}) \geq e^{\alpha\|\vec{x}\|_\infty}$ we have that $\Pr[\|\vec{x}^{(k)}\|_\infty \geq \log(\lambda_k)/\alpha] \leq \frac{2m}{\lambda_k}$ for all k. Choosing $\lambda_k = \frac{4mk^2}{p}$ and taking a union bound over all k, we have that

$$\left\|\vec{x}^{(k)}\right\|_\infty \leq 2(c+R)\log\left(4mk^2/p\right)$$

for all k with probability at least

$$1 - \sum_{i=1}^{\infty}\frac{2m}{\lambda_k} = 1 - \sum_{k=1}^{\infty}\frac{p}{2k^2} \geq 1-p \quad .$$

□

7 Glossary

Here we summarize problem specific notations used throughout this Part. For many quantities we included the typical order of magnitude as they appear in our algorithms.

- Our algorithm maintains a polytope $\{\mathbf{A}x \geq b\}$ which contains the set of solution K.
- Slacks $\vec{s}(\vec{x}) = \mathbf{A}\vec{x} - \vec{b}$, $\mathbf{S}(\vec{x})$ is the diagonal matrix corresponding to $\vec{s}(\vec{x})$. Rescaled constraint matrix $\mathbf{A}_s = \mathbf{S}^{-1}\mathbf{A}$.
- Leverage Score: $\vec{\psi}(\vec{x}) = \operatorname{diag}\left(\mathbf{A}_x\left(\mathbf{A}_x^T\mathbf{A}_x + \lambda\mathbf{I}\right)^{-1}\mathbf{A}_x^T\right)$, $\mu(\vec{x}) = \min \psi_i(\vec{x})$, $\lambda = \frac{1}{c_a R^2}$ and $c_a = \frac{1}{10^{10}}$.
- In each iteration constraints with leverage score less than $c_d = \frac{1}{10^{12}}$ are deleted. Otherwise, we add a constraint and put the leverage score c_a.
- In each iteration, our algorithm computes the change of leverage scores with accuracy $c_\Delta = \frac{Cc_e}{\log(n\log(R/\epsilon))}$ where $c_e = \frac{c_d}{4\log(6nR/\epsilon)}$ and R is the diameter of the box containing K.
- Hybrid barrier function

$$p_{\vec{e}}(\vec{x}) = -\sum_{i\in[m]}(c_e + e_i)\log\left(s_i(\vec{x})/R\right) + \frac{1}{2}\log\det\left(R^2\left(\mathbf{A}^T\mathbf{S}_x^{-2}\mathbf{A} + \lambda\mathbf{I}\right)\right) + \frac{\lambda}{2}\|x\|_2^2$$

- The algorithm starts with $p_{\vec{e}}(\vec{x}) \sim -\Theta(1)n$ and ends when $p_{\vec{e}}(\vec{x}) \sim \Theta(1)n\log(nR/\epsilon)$.
- Centrality $\delta_{\vec{e}}(\vec{x}) = \left\|\bigtriangledown p_{\vec{e}}(\vec{x})\right\|_{\mathbf{H}(\vec{x})^{-1}}$ where $\mathbf{H}(\vec{x}) = \mathbf{A}_x^T\left(c_e\mathbf{I} + \mathbf{\Psi}(\vec{x})\right)\mathbf{A}_x + \lambda\mathbf{I}$. $\delta_{\vec{e}}(\vec{x}) \approx \Theta(\sqrt{c_e + \mu(\vec{x})})$.
- Approximate Hessian $\mathbf{Q}(\vec{x},\vec{w}) = \mathbf{A}_x^T\left(c_e\mathbf{I} + \mathbf{W}\right)\mathbf{A}_x + \lambda\mathbf{I}$.

Part II
Convex Minimization & Intersection Problem via Cutting Plane Method

This Part is based on joint works with Yin Tat Lee and Aaron Sidford.

8 Introduction

Cutting plane methods have long been employed to obtain polynomial time algorithms for solving optimization problems. However, for many problems cutting plane methods are often regarded as inefficient both in theory and in practice. Here, in Part II we provide several techniques for applying cutting plane methods efficiently. Moreover, we illustrate the efficacy and versatility of these techniques by applying them to achieve improved running times for solving multiple problems including semidefinite programming, matroid intersection, and submodular flow.

We hope these results revive interest in ellipsoid and cutting plane methods. We believe these results demonstrate how cutting plan methods are often useful not just for showing that a problem is solvable in polynomial time, but in many yield substantial running time improvements. We stress that while some results in Part II are problem-specific, the techniques introduced here are quite general and are applicable to a wide range of problems.

In the remainder of this introduction we survey the key techniques we use to apply our cutting plane method (Section 8.1) and the key results we obtain on improving the running time for solving various optimization problems (Section 8.2). We conclude in Section 8.3 by providing an overview of where to find additional technical result in Part II.

8.1 Techniques

Although cutting plane methods are typically introduced as algorithms for finding a point in a convex set (as we did with the feasibility problem in Part I), this is often not the easiest way to apply the methods. Moreover, improperly applying results on the feasibility problem to solve convex optimization problems can lead to vastly sub-optimal running times. Our central goal, here, in Part II is to provide tools that allow cutting plane methods to be efficiently applied to solve complex optimization problems. Some of these tools are new and some are extensions of previously known techniques. Here we briefly survey the techniques we cover in Section 10 and Section 11.

Technique 0: From Feasibility to Optimization

In Section 10.1, we explain how to use our cutting plane method to solve convex optimization problems using an approximate subgradient oracle. Our result is based on a result of Nemirovski [207] in which he showed how to use a cutting plane method to solve convex optimization problems without smoothness assumptions on the function and with minimal assumptions on the size of the function's domain. We generalize his proof to accommodate for an approximate separation oracle, an extension which is essential for our applications. We use this result as the starting point for two new techniques we discuss below.

Technique 1: Dimension Reduction through Duality

In Section 10.2, we discuss how cutting plane methods can be applied to obtain both primal and dual solutions to convex optimization problems. Moreover, we show how this can be achieved while only applying the cutting plane method in the space, primal or dual, which has a fewer number of variables. Thus we show how to use duality to improve the convergence of cutting plane methods while still solving the original problem.

To illustrate this idea consider the following very simple linear program (LP)

$$\min_{x_i \geq 0, \sum x_i = 1} \sum_{i=1}^{n} w_i x_i$$

where $\vec{x} \in \mathbb{R}^n$ and $\vec{w} \in \mathbb{R}^n$. Although this LP has n variables, it should to be easy to solve purely on the grounds that it only has one equality constraint and thus dual linear program is simply

$$\max_{y \leq w_i \forall i} y\,,$$

i.e. a LP with only one variable. Consequently, we can apply our cutting plane method to solve it efficiently.

However, while this simple example demonstrates how we can use duality to decrease dimensions, it is not always obvious how to recover the optimal primal solution x variable given the optimal dual solution y. Indeed, for many problems their dual is significantly simpler than itself (primal), so some work is required to show that working in the space suffices to require a primal solution.

One such recent example of this approach proving successful is a recent linear programming result [176]. In this result, the authors show how to take advantage of this observation and get a faster LP solver and maximum flow algorithm. It is interesting to study how far this technique can extend, that is, in what settings can one recover the solution to a more difficult dual problem from the solution to its easier primal problem?

There is in fact another precedent for such an approach. Grï¿œtschel, Lovï¿œsz and Schrijver[113] showed how to obtain the primal solution for linear program by using a cutting plane method to solve the linear program exactly. This is based on the observation that cutting plane methods are able to find the active constraints of the optimal solution and hence one can take dual of the linear program to get the dual solution. This idea was further extended in [165] which also observed that cutting plane methods are incrementally building up a LP relaxation of the optimization problem. Hence, one can find a dual solution by taking the dual of that relaxation.

In Section 10.2, we provide a fairly general technique to recover a dual optimal solution from an approximately optimal primal solution. Unfortunately, the performance of this technique seems quite problem-dependent. We therefore only analyze this technique for semidefinite programming (SDP), a classic and popular convex optimization problem. As a result, we obtain a faster SDP solver in both the primal and dual formulations of the problem.

Technique 2: Using Optimization Oracles Directly

In the seminal works of Grï¿œtschel, Lovï¿œsz, Schrijver and independently Karp and Papadimitriou [111, 147], they showed the equivalence between optimization oracles and separation oracles, and gave a general method to construct a separation oracle for a convex set given an optimization oracle for that set, that is an oracle for minimizing linear functionals over the set. This seminal result led to the first weakly polynomial time algorithm for many algorithms such as submodular function minimization. Since then, this idea has been used extensively in various settings [141, 33, 34, 54].

Unfortunately, while this equivalence of separation and optimization is a beautiful and powerful tool for polynomial time solvability of problems, in many case it may lead to inefficient algorithms. In order to use this reduction to get a separation oracle, the optimization oracle may need to be called multiple times – essentially the number of times needed to run a cutting plane method and hence may be detrimental to obtaining small asymptotic running times. Therefore, it is an interesting question of whether there is a way of using an optimization oracle more directly.

In Section 11 we provide a partial answer to this question for the case of a broad class of problems, that we call the *intersection problem*. For these problems we demonstrate how to achieve running time improvements by using optimization oracles directly. The problem we consider is as follows. We wish to solve the problem for some cost vector $\vec{c} \in \mathbb{R}^n$ and convex set K. We assume that the convex set K can be decomposed as $K = K_1 \cap K_2$ such that $\max_{\vec{x} \in K_1} \langle \vec{c}, \vec{x} \rangle$ and $\max_{\vec{x} \in K_2} \langle \vec{c}, \vec{x} \rangle$ can each be solved efficiently. Our goal is to obtain a running time for this problem comparable to that of minimizing K given only a separation oracle for it.

We show that by considering a carefully regularized variant, we obtain a problem such that optimization oracles for K_1 and K_2 immediately yield a separation oracle for this regularized problem. By analyzing the regularizer and bounding the domains of the problem we are able to show that this allows us to efficiently compute highly accurate solutions to the intersection problem by applying our cutting plane method once. In other words, we do not need to use a complicated iterative scheme or directly invoke the equivalence between separation and optimization and thereby save $O(\mathsf{poly}(n))$ factors in our running times.

We note that this intersection problem can be viewed as a generalization of the matroid intersection problem and in Section 11.2, we show our reduction gives a faster algorithm in certain parameter regimes. As another example, in Section 11.3 we show our reduction gives a substantial polynomial improvement for the submodular flow problem. Furthermore, in Section 11.4 we show how our techniques allow us to minimize a linear function over the intersection of a convex set and an affine subspace in a number of iterations that depends only on the co-dimension of the affine space.

8.2 Applications

Our main goal in Part II is to provide general techniques for efficiently using cutting plane methods for various problems. Hence, in Part II we use minimally problem-specific techniques to achieve the best possible running time. However, we also demonstrate the efficacy of our approach by showing how techniques improve upon the previous best known running times for solve several classic problems in combinatorial and continuous optimization. Here we provide a brief overview of these applications, previous work on these problems, and our results.

In order to avoid deviating from our main discussion, our coverage of previous methods and techniques is brief. Given the large body of prior works on SDP, matroid intersection and submodular flow, it would be impossible to have an in-depth discussion on all of them. Therefore, this section focuses on running time comparisons and explanations of relevant preivous techniques.

Semidefinite Programming

In Section 10.2 we consider the classic semidefinite programming (SDP) problem:

$$\max_{\mathbf{X}\succeq\mathbf{0}} \mathbf{C}\bullet\mathbf{X} \text{ s.t. } \mathbf{A}_i\bullet\mathbf{X}=b_i \text{ (primal)} \qquad \min_{\vec{y}} \vec{b}^T\vec{y} \text{ s.t. } \sum_{i=1}^n y_i\mathbf{A}_i\succeq\mathbf{C} \text{ (dual)}$$

where $\mathbf{X}$, $\mathbf{C}$, $\mathbf{A}_i$ are $m\times m$ symmetric matrices, $\vec{b},\vec{y}\in\mathbb{R}^n$, and $\mathbf{A}\bullet\mathbf{B}\overset{\text{def}}{=}\text{Tr}(\mathbf{A}^T\mathbf{B})$. For many problems, $n\ll m^2$ and hence the dual problem has fewer variables than the primal. There are many results and applications of SDP; see [252, 246, 195] for a survey on this topic. Since our focus is on polynomial time algorithms, we do not discuss pseudo-polynomial algorithms such as the spectral bundle method [121], multiplicative weight update methods [11, 12, 140, 4], etc.

Currently, there are two competing approaches for solving SDP problems, namely interior point methods (IPM) and cutting plane methods. Typically, IPMs require fewer iterations than the cutting plane methods, however each iteration of these methods is more complicated and possibly more computationally expensive. For SDP problems, interior point methods require the computations of the Hessian of the function $-\log\det\left(\mathbf{C}-\sum_{i=1}^n y_i\mathbf{A}_i\right)$ whereas cutting plane methods usually only need to compute minimum eigenvectors of the slack matrix $\mathbf{C}-\sum_{i=1}^n y_i\mathbf{A}_i$.

In [10], Anstreicher provided the current fastest IPM for solving the dual SDP problem using a method based on the volumetric barrier function. This method takes $O((mn)^{1/4})$ iterations and each iteration is as cheap as usual IPMs. For general matrices $\mathbf{C},\mathbf{X},\mathbf{A}_i$, each iteration takes $O(n^2m^{\omega-1}+n^\omega+m^\omega)$ time where ω is the fast matrix multiplication exponent. If the constraint matrices $\mathbf{A}_i$ are rank one matrices, the iteration cost can be improved[167]. If the matrices are sparse, then [93, 206] show how to use matrix completion inside the IPM. However, the running time depends on the extended sparsity patterns which can be much larger than the total number of non-zeros.

In [166], Krishnan and Mitchell observed that the separation oracle for dual SDP takes only $O(m^\omega+S)$ time, where $S=\sum_{i=1}^n \text{nnz}(\mathbf{A}_i)$ be the total number of non-zeros in the

Authors	Years	Running times
Nesterov, Nemirovsky[213]	1992	$\tilde{O}(\sqrt{m}(n^2m^{\omega-1}+n^\omega+m^\omega))$
Anstreicher [10]	2000	$\tilde{O}((mn)^{1/4}(n^2m^{\omega-1}+n^\omega))$
Krishnan, Mitchell [166]	2003	$\tilde{O}(n(n^\omega+m^\omega+S))$ (dual SDP)
This work	2015	$\tilde{O}(n(n^2+m^\omega+S))$

Table 2: Previous algorithms for solving a $m \times m$ SDP with n constraints and S non-zeros entries

constant matrix. Hence, the cutting plane method by [250] gives a faster algorithm for SDP for many regimes. For $\omega = 2.38$, the cutting plane method is faster when $\mathbf{A}_i$ is not rank 1 and the problem is not too dense. While there are previous methods for using cutting plane methods to obtain primal solutions[165] , to the best of our knowledge, there are no worst case running time analysis for these techniques.

In Section 10.2, show how to alleviate this issue. We provide an improved algorithm for finding the dual solution and prove carefully how to obtain a comparable primal solution as well. See Figure 9.1 for a summary of the algorithms for SDP and their running times.

Matroid Intersection

In Section 11.2 we show how our optimization oracle technique can be used to improve upon the previous best known running times for matroid intersection. Matroid intersection is one of the most fundamental problems in combinatorial optimization. The first algorithm for matroid intersection is due to the seminal paper by Edmonds [66]. In Figures 9.2 and 9.3 we provide a summary of the previous algorithms for unweighted and weighted matroid intersection as well as the new running times we obtain in this work. While there is no total ordering on the running times of these algorithms due to the different dependence on various parameters, we would like to point out that our algorithms outperform the previous ones in regimes where r is close to n and/or the oracle query costs are relatively expensive. In particular, in terms of oracle query complexity our algorithms are the first to achieve the quadratic bounds of $\tilde{O}(n^2)$ and $\tilde{O}(nr)$ for independence and rank oracles. We hope our work will revive the interest in the problem of which progress has been mostly stagnated for the past 20-30 years.

Minimum-Cost Submodular Flow

In Section 11.3 we show how our optimization oracle technique can be used to improve upon the previous best known running times for (Minimum-cost) Submodular Flow. Submodular flow is a very general problem in combinatorial optimization which generalizes many problems such as minimum cost flow, the graph orientation, polymatroid intersection, directed cut covering [89]. In Figure 9.4 we provide an overview of the previous algorithms for submodular flow as well as the new running times we obtain in this work.

Many of the running times are in terms of a parameter h, which is the time required for computing an "exchange capacity". To the best of our knowledge, the most efficient way of

Authors	Years	Running times
Edmonds [66]	1968	not stated
Aigner, Dowling [3]	1971	$O(nr^2\mathcal{T}_{\text{ind}})$
Tomizawa, Iri [247]	1974	not stated
Lawler [173]	1975	$O(nr^2\mathcal{T}_{\text{ind}})$
Edmonds [69]	1979	not stated
Cunningham [51]	1986	$O(nr^{1.5}\mathcal{T}_{\text{ind}})$
This work	2015	$O(n^2 \log n\mathcal{T}_{\text{ind}} + n^3 \log^{O(1)} n)$ $O(nr \log^2 n\mathcal{T}_{\text{rank}} + n^3 \log^{O(1)} n)$

Table 3: Previous algorithms for (unweighted) matroid intersection. Here n is the size of the ground set, $r = \max\{r_1, r_2\}$ is the maximum rank of the two matroids, $\mathcal{T}_{\text{ind}}$ is the time needed to check if a set is independent (independence oracle), and $\mathcal{T}_{\text{rank}}$ is the time needed to compute the rank of a given set (rank oracle).

Authors	Years	Running times
Edmonds [66]	1968	not stated
Tomizawa, Iri [247]	1974	not stated
Lawler [173]	1975	$O(nr^2\mathcal{T}_{\text{ind}} + nr^3)$
Edmonds [69]	1979	not stated
Frank [80]	1981	$O(n^2 r(\mathcal{T}_{\text{circuit}} + n))$
Orlin, Ahuja [218]	1983	not stated
Brezovec, Cornuï¿œjols, Glover[28]	1986	$O(nr(\mathcal{T}_{\text{circuit}} + r + \log n))$
Fujishige, Zhang [91]	1995	$O(n^2 r^{0.5} \log rM \cdot \mathcal{T}_{\text{ind}})$
Shigeno, Iwata [236]	1995	$O((n + \mathcal{T}_{\text{circuit}})nr^{0.5} \log rM)$
This work	2015	$O((n^2 \log n\mathcal{T}_{\text{ind}} + n^3 \log^{O(1)} n) \log nM)$ $O((nr \log^2 n\mathcal{T}_{\text{rank}} + n^3 \log^{O(1)} n) \log nM)$

Table 4: Previous algorithms for weighted matroid intersection. In additions to the notations used in the unweighted table, $\mathcal{T}_{\text{circuit}}$ is the time needed to find a fundamental circuit and M is the bit complexity of the weights.

Authors	Years	Running times
Fujishige [83]	1978	not stated
Grï¿œtschel, Lovï¿œsz, Schrijver[111]	1981	weakly polynomial
Zimmermann [263]	1982	not stated
Barahona, Cunningham [18]	1984	not stated
Cunningham, Frank [52]	1985	$\rightarrow O(n^4 h \log C)$
Fujishige [85]	1987	not stated
Frank, Tardos [81]	1987	strongly polynomial
Cui, Fujishige [255]	1988	not stated
Fujishige, Rï¿œck, Zimmermann[90]	1989	$\rightarrow O(n^6 h \log n)$
Chung, Tcha [44]	1991	not stated
Zimmermann [264]	1992	not stated
McCormick, Ervolina [193]	1993	$O(n^7 h^* \log nCU)$
Wallacher, Zimmermann [256]	1994	$O(n^8 h \log nCU)$
Iwata [124]	1997	$O(n^7 h \log U)$
Iwata, McCormick, Shigeno [130]	1998	$O\left(n^4 h \min\left\{\log nC, n^2 \log n\right\}\right)$
Iwata, McCormick, Shigeno [131]	1999	$O\left(n^6 h \min\left\{\log nU, n^2 \log n\right\}\right)$
Fleischer, Iwata, McCormick[78]	1999	$O\left(n^4 h \min\left\{\log U, n^2 \log n\right\}\right)$
Iwata, McCormick, Shigeno [132]	1999	$O\left(n^4 h \min\left\{\log C, n^2 \log n\right\}\right)$
Fleischer, Iwata [76]	2000	$O(mn^5 \log nU \cdot \text{EO})$
This work	2015	$O(n^2 \log nCU \cdot \text{EO} + n^3 \log^{O(1)} nCU)$

Figure 8.1: Previous algorithms for Submodular Flow with n vertices, maximum cost C and maximum capacity U. The factor h is the time for an exchange capacity oracle, h^* is the time for a "more complicated exchange capacity oracle" and EO is the time for evaluation oracle of the submodular function. The arrow,$\rightarrow$, indicates that it used currently best maximum submodular flow algorithm as subroutine which was non-existent at the time of the publication.

computing an exchange capacity is to solve an instance of submodular minimization which previously took time $\tilde{O}(n^4\text{EO} + n^5)$ (and now takes $\tilde{O}(n^2\text{EO} + n^3)$ time using our result in Part III). Readers may wish to substitute $h = \tilde{O}(n^2\text{EO} + n^3)$ when reading the table.

The previous fastest weakly polynomial algorithms for submodular flow are by [132, 76, 78], which take time $\tilde{O}(n^6\text{EO} + n^7)$ and $O(mn^5 \log nU \cdot \text{EO})$, assuming $h = \tilde{O}(n^2\text{EO} + n^3)$. Our algorithm for submodular flow has a running time of $\tilde{O}(n^2\text{EO}+n^3)$, which is significantly faster by roughly a factor of $\tilde{O}(n^4)$.

For strongly polynomial algorithms, our results do not yield a speedup but we remark that our faster strongly polynomial algorithm for submodular minimization in Part III improves the previous algorithms by a factor of $\tilde{O}(n^2)$ as a corollary (because h requires solving an instance of submodular minimization).

8.3 Overview

After providing covering some preliminaries on convex analysis in Section 9 we split the remainder of Part II into Section 10 and Section 11. In Section 10 we cover our algorithm for convex optimization using an approximate subgradient oracle (Section 10.1) as well as our technique on using duality to decrease dimensions and improve the running time of semidefinite programming (Section 10.2). In Section 11 we provide our technique for using minimization oracles to minimize functions over the intersection of convex sets and provide several applications including matroid intersection (Section 11.2), submodular flow (Section 11.3), and minimizing a linear function over the intersection of an affine subspace and a convex set (Section 11.4).

9 Preliminaries

In this section we review basic facts about convex functions that we use throughout Part II. We also introduce two oracles that we use throughout Part II, i.e. subgradient and optimization oracles, and provide some basic reductions between them. Note that we have slightly extended some definitions and facts to accommodate for the noisy separation oracles used in this work.

First we recall the definition of strong convexity

Definition 35 (Strong Convexity). A real valued function f on a convex set Ω is α-strongly convex if for any $\vec{x},\vec{y}\in\Omega$ and $t\in[0,1]$, we have

$$f(t\vec{x}+(1-t)\vec{y})+\frac{1}{2}\alpha t(1-t)\left\|\vec{x}-\vec{y}\right\|^{2}\leq tf(\vec{x})+(1-t)f(\vec{y}).$$

Next we define an approximate subgradient.

Definition 36 (Subgradient). For any convex function f on a convex set Ω, the δ-subgradients of f at x are defined to be

$$\partial_{\delta}f(\vec{x})\stackrel{\mathrm{def}}{=}\{\vec{g}\in\Omega\,:\,f(\vec{y})+\delta\geq f(\vec{x})+\langle\vec{g},\vec{y}-\vec{x}\rangle\ \text{ for all }\vec{y}\in\Omega\}.$$

Here we provide some basic facts regarding convexity and subgradients. These statements are natural extensions of well known facts regarding convex functions and their proof can be found in any standard textbook on convex optimization.

Fact 37. *For any convex set Ω and $\vec{x}$ be a point in the interior of Ω, we have the following:*

1. *If f is convex on Ω, then $\partial_{0}f(\vec{x})\neq\emptyset$ and $\partial_{s}f(\vec{x})\subseteq\partial_{t}f(\vec{x})$ for all $0\leq s\leq t$.Otherwise, we have $\left\|\vec{g}\right\|_{2}>\frac{1}{2}\sqrt{\frac{\delta}{D}}$. For any $f(\vec{y})\leq f(\vec{x})$, we have $\delta\geq\langle\vec{g},\vec{y}-\vec{x}\rangle$ and hence*

2. *If f is a differential convex function on Ω, then $\nabla f(\vec{x})\in\partial_{0}f(\vec{x})$.*

3. *If f_1 and f_2 are convex function on Ω, $\vec{g}_1\in\partial_{\delta_1}f_1(\vec{x})$ and $\vec{g}_2\in\partial_{\delta_2}f_1(\vec{x})$, then $\alpha\vec{g}_1+\beta\vec{g}_2\in\partial_{\alpha\delta_1+\beta\delta_2}(\vec{g}_1+\vec{g}_2)(\vec{x})$.*

4. *If f is α-strongly convex on Ω with minimizer x^*, then for any $\vec{y}$ with $f(\vec{y}) \le f(\vec{x}^*)+\epsilon$, we have $\frac{1}{2}\alpha\left\Vert \vec{x}^* - \vec{y}\right\Vert^2 \le \epsilon$.*

Next we provide a reduction from subgradients to separation oracles. We will use this reduction several times in Part II to simplify our construction of separation oracles.

Lemma 38. *Let f be a convex function. Suppose we have $\vec{x}$ and $\vec{g} \in \partial_\delta f(\vec{x})$ with $\left\Vert \vec{x}\right\Vert_2 \le 1 \le D$ and $\delta \le 1$. If $\left\Vert \vec{g}\right\Vert_2 \le \frac{1}{2}\sqrt{\frac{\delta}{D}}$, then $f(\vec{x}) \le \min_{\Vert\vec{y}_2\Vert_2 \le D} f(\vec{y}) + 2\sqrt{\delta D}$ and if $\left\Vert \vec{g}\right\Vert_2 \le \frac{1}{2}\sqrt{\frac{\delta}{D}}$ then*

$$\{\left\Vert \vec{y}\right\Vert_2 \le D \,:\, f(\vec{y}) \le f(\vec{x})\} \subset \{\vec{y} : \vec{d}^T\vec{y} \le \vec{d}^T\vec{x} + 2\sqrt{\delta D}\}$$

with $\vec{d} = \vec{g}/\left\Vert \vec{g}\right\Vert_2$. Hence, this gives a $(2\sqrt{\delta D}, 2\sqrt{\delta D})$-separation oracle on the set $\{\left\Vert \vec{x}\right\Vert_2 \le D\}$.

Proof. Let $\vec{y}$ such that $\left\Vert \vec{y}\right\Vert_2 \le D$. By the definition of δ-subgradient, we have

$$f(\vec{y}) + \delta \ge f(\vec{x}) + \left\langle \vec{g}, \vec{y}-\vec{x}\right\rangle .$$

If $\left\Vert \vec{g}\right\Vert \le \frac{1}{2}\sqrt{\frac{\delta}{D}}$, then, we have $|\left\langle \vec{g}, \vec{y}-\vec{x}\right\rangle| \le \sqrt{\delta D}$ because $\left\Vert \vec{x}\right\Vert \le D$ and $\left\Vert \vec{y}\right\Vert_2 \le D$. Therefore,

$$\min_{\left\Vert \vec{y}\right\Vert_2 \le D} f(\vec{y}) + 2\sqrt{\delta D} \ge f(\vec{x}).$$

Otherwise, we have $\left\Vert \vec{g}\right\Vert_2 > \frac{1}{2}\sqrt{\frac{\delta}{D}}$. For any $f(\vec{y}) \le f(\vec{x})$, we have $\delta \ge \left\langle \vec{g}, \vec{y}-\vec{x}\right\rangle$ and hence

$$2\sqrt{\delta D} \ge \left\langle \frac{\vec{g}}{\left\Vert \vec{g}\right\Vert}, \vec{y}-\vec{x}\right\rangle .$$

□

At several times in Part II we will wish to construct subgradient oracles or separation oracles given only the ability to approximately maximize a linear function over a convex set. In the remainder of this section we formally define such a *optimization oracle* and prove this equivalence.

Definition 39 (Optimization Oracle)**.** Given a convex set K and $\delta > 0$ a δ-optimization oracle for K is a function on $\mathbb{R}^n$ such that for any input $\vec{c} \in \mathbb{R}^n$, it outputs $\vec{y}$ such that

$$\max_{\vec{x}\in K} \left\langle \vec{c}, \vec{x}\right\rangle \le \left\langle \vec{c}, \vec{y}\right\rangle + \delta.$$

We denote by $\mathrm{OO}_\delta(K)$ the time complexity of this oracle.

Lemma 40. *Given a convex set K, any ϵ-optimization oracle for K is a ϵ-subgradient oracle for $f(\vec{c}) \overset{\text{def}}{=} \max_{\vec{x}\in K} \left\langle \vec{c}, \vec{x}\right\rangle$.*

Proof. Let $\vec{x}_c$ be the output of ϵ-optimization oracle on the cost vector $\vec{c}$. We have

$$\max_{\vec{x}\in K} \langle \vec{c}, \vec{x} \rangle \leq \langle \vec{c}, \vec{x}_c \rangle + \epsilon.$$

Hence, for all $\vec{d}$, we have and therefore

$$\left\langle \vec{x}_c, \vec{d} - \vec{c} \right\rangle + f(\vec{c}) \leq f(\vec{d}) + \epsilon.$$

Hence, $\vec{x}_c \in \partial_\delta f(\vec{c})$. □

Combining these lemmas shows that having an ϵ-optimization oracle for a convex set K contained in a ball of radius D yields a $O(\sqrt{D\epsilon}, \sqrt{D\epsilon})$ separation oracle for $\max_{x\in K} \langle \vec{c}, \vec{x} \rangle$. We use these ideas to construction separation oracles throughout Part II.

10 Convex Optimization

In this section we show how to apply our cutting plane method to efficiently solve problems in convex optimization. First, in Section 10.1 we show how to use our result to minimize a convex function given an approximate subgradient oracle. Then, in Section 10.2 we illustrate how this result can be used to obtain both primal and dual solutions for a standard convex optimization problems. In particular, we show how our result can be used to obtain improved running times for semidefinite programming across a range of parameters.

10.1 From Feasibility to Optimization

In this section we consider the following standard optimization problem. We are given a convex function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ and we want to find a point $\vec{x}$ that approximately solves the minimization problem

$$\min_{\vec{x}\in\mathbb{R}^n} f(\vec{x})$$

given only a subgradient oracle for f.

Here we show how to apply the cutting plane method from Part I turning the small width guarantee of the output of that algorithm into a tool to find an approximate minimizer of f. Our result is applicable to any convex optimization problem armed with a separation or subgradient oracle. This result will serve as the foundation for many of our applications in Part II.

Our approach is an adaptation of Nemiroski's method [207] which applies the cutting plane method to solve convex optimiziation problems, with only minimal assumption on the cutting plane method. The proof here is a generalization that accommodates for the noisy separation oracle used in this work. In the remainder of this subsection we provide a key definition we will use in our algorithm (Defintion 41), provide our main result (Theorem 42), and conclude with a brief discussion of this result.

Definition 41. For any compact set K, we define the *minimum width* by $\mathrm{MinWidth}(K) \stackrel{\text{def}}{=} \min_{\|\vec{a}\|_2=1} \max_{\vec{x},\vec{y}\in K} \langle \vec{a}, \vec{x} - \vec{y} \rangle$.

Theorem 42. *Let f be a convex function on $\mathbb{R}^n$ and Ω be a convex set that contains a minimizer of f. Suppose we have a (η,δ)-separation oracle for f and Ω is contained inside $B_\infty(R)$. Using $B_\infty(R)$ as the initial polytope for our Cutting Plane Method, for any $0<\alpha<1$, we can compute $\vec{x}\in\mathbb{R}^n$ such that*

$$f(\vec{x})-\min_{\vec{y}\in\Omega}f(\vec{y})\leq\eta+\alpha\left(\max_{\vec{y}\in\Omega}f(\vec{y})-\min_{\vec{y}\in\Omega}f(\vec{y})\right)\quad . \tag{10.1}$$

with an expected running time of

$$O\left(n\mathrm{SO}_{\eta,\delta}(f)\log\left(\frac{n\kappa}{\alpha}\right)+n^3\log^{O(1)}\left(\frac{n\kappa}{\alpha}\right)\right),$$

where $\delta=\Theta\left(\frac{\alpha\mathrm{MinWidth}(\Omega)}{n^{3/2}\log(\kappa)}\right)$ and $\kappa=\frac{R}{\mathrm{MinWidth}(\Omega)}$. Furthermore, we only need the oracle defined on the set $B_\infty(R)$.

Remark. The algorithm requires no information about Ω (other than that $\Omega\subseteq B_\infty(R)$). The function f can have $+\infty$ value outside Ω.

Proof. Let $\vec{x}^*\in\arg\min_{\vec{x}\in\Omega}f(\vec{x})$. Since $B_\infty(R)\supset\Omega$ contains a minimizer of f, by the definition of (η,δ)-separation oracles, our Cutting Plane Method (Theorem 31) either returns a point $\vec{x}$ that is almost optimal or returns a polytope P of small width. In the former case we have a point $\vec{x}$ such that $f(\vec{x})\leq\min_{\vec{y}}f(\vec{y})+\eta$. Hence, the error is clearly at most $\eta+\alpha\left(\max_{\vec{z}\in\Omega}f(\vec{z})-\min_{\vec{x}\in\Omega}f(\vec{x})\right)$ as desired. Consequently, we assume the latter case.

Theorem 31 shows $\mathrm{MinWidth}(P)<Cn\epsilon\log(R/\epsilon)$ for some universal constant C. Picking

$$\epsilon=C'\frac{\alpha\mathrm{MinWidth}(\Omega)}{n\log\left(\frac{n\kappa}{\alpha}\right)} \tag{10.2}$$

for small enough constant C', we have $\mathrm{MinWidth}(P)<\alpha\mathrm{MinWidth}(\Omega)$. Let $\Omega^\alpha=\vec{x}^*+\alpha(\Omega-\vec{x}^*)$, namely, $\Omega^\alpha=\{\vec{x}^*+\alpha(\vec{z}-\vec{x}^*):\vec{z}\in\Omega\}$. Then, we have

$$\mathrm{MinWidth}(\Omega^\alpha)=\alpha\mathrm{MinWidth}(\Omega)>\mathrm{MinWidth}(P).$$

Therefore, Ω^α is not a subset of P and hence there is some point $\vec{y}\in\Omega^\alpha\backslash P$. Since $\Omega^\alpha\subseteq\Omega\subseteq B_\infty(R)$, we know that $\vec{y}$ does not violate any of the constraints of $B_\infty(R)$ and therefore must violate one of the constraints added by querying the separation oracle. Therefore, for some $j\leq i$, we have

$$\left\langle\vec{c}^{(j-1)},\vec{y}\right\rangle>\left\langle\vec{c}^{(j-1)},\vec{x}^{(j-1)}\right\rangle+c_s\epsilon/\sqrt{n}\quad .$$

By the definition of $(\eta,c_s\epsilon/\sqrt{n})$-separation oracle (Definition 2), we have $f(\vec{y})>f(\vec{x}^{(j-1)})$. Since $\vec{y}\in\Omega^\alpha$, we have $\vec{y}=(1-\alpha)\vec{x}^*+\alpha\vec{z}$ for some $\vec{z}\in\Omega$. Thus, the convexity of f implies that

$$f(\vec{y})\leq(1-\alpha)f(\vec{x}^*)+\alpha f(\vec{z}).$$

Therefore, we have

$$\min_{1\leq k\leq i}f(\vec{x}^{(k)})-\min_{\vec{x}\in\Omega}f(\vec{x})<f(\vec{y})-f(\vec{x}^*)\leq\alpha\left(\max_{\vec{z}\in\Omega}f(\vec{z})-\min_{\vec{x}\in\Omega}f(\vec{x})\right).$$

Hence, we can simply output the best $\vec{x}$ among all $\vec{x}^{(j)}$ and in either case $\vec{x}$ satisfies (10.1).

Note that we need to call (η,δ)-separation oracle with $\delta = \Omega(\epsilon/\sqrt{n})$ to ensure we do not cut out $\vec{x}^*$. Theorem 31 shows that the algorithm takes $O(n\text{SO}_{\eta,\delta}(f)\log(nR/\epsilon) + n^3\log^{O(1)}(nR/\epsilon))$ expected time, as promised. Furthermore, the oracle needs only be defined on $B_\infty(R)$ as an obvious separating hyperplane can be returned for a query point outside $B_\infty(R)$. $\square$

Observe that this algorithm requires no information about Ω (other than that $\Omega \subseteq B_\infty(R)$) and does not guarantee that the output is in Ω. Hence, even though Ω can be complicated to describe, the algorithm still gives a guarantee related to the gap $\max_{\vec{x}\in\Omega} f(\vec{x}) - \min_{\vec{x}\in\Omega} f(\vec{x})$. For specific applications, it is therefore advantageous to pick a Ω as large as possible while the bound on function value is as small as possible.

Before indulging into specific applications, we remark on the dependence on κ. Using John's ellipsoid, it can be shown that any convex set Ω can be transformed linearly such that (1) $B_\infty(1)$ contains Ω and, (2) $\text{MinWidth}(\Omega) = \Omega(n^{-3/2})$. In other words, κ can be effectively chosen as $O(n^{3/2})$. Therefore if we are able to find such a linear transformation, the running time is simply $O\left(n\text{SO}(f)\log(n/\alpha) + n^3\log^{O(1)}(n/\alpha)\right)$. Often this can be done easily using the structure of the particular problem and the running time does not depend on the size of domain at all.

10.2 Duality and Semidefinite Programming

In this section we illustrate how our result in Section 10.1 can be used to obtain both primal and dual solutions for standard problems in convex optimization. In particular we show how to obtain improved running times for semidefinite programming.

To explain our approach, consider the following minimax problem

$$\min_{\vec{y}\in Y}\max_{\vec{x}\in X} \langle \mathbf{A}\vec{x},\vec{y}\rangle + \langle \vec{c},\vec{x}\rangle + \left\langle \vec{d},\vec{y}\right\rangle \tag{10.3}$$

where $\vec{x}\in\mathbb{R}^m$ and $\vec{y}\in\mathbb{R}^n$. When $m \gg n$, solving this problem by directly using Part I could lead to an inefficient algorithm with running time at least m^3. In many situations, for any fixed $\vec{y}$, the problem $\max_{\vec{x}\in X}\langle\mathbf{A}\vec{x},\vec{y}\rangle$ is very easy and hence one can use it as a separation oracle and apply Part I and this would gives a running time almost independent of m. However, this would only give us the $\vec{y}$ variable and it is not clear how to recover $\vec{x}$ variable from it.

In this section we show how to alleviate this issue and give semidefinite programming (SDP) as a concrete example of how to apply this general technique. We do not write down the general version as the running time of the technique seems to be problem specific and faster SDP is already an interesting application.

For the remainder of this section we focus on the semidefinite programming (SDP) problem:

$$\max_{\mathbf{X}\succeq\mathbf{0}} \mathbf{C}\bullet\mathbf{X} \text{ s.t. } \mathbf{A}_i\bullet\mathbf{X} = b_i \tag{10.4}$$

and its dual

$$\min_{\vec{y}} \vec{b}^T\vec{y} \text{ s.t. } \sum_{i=1}^n y_i\mathbf{A}_i \succeq \mathbf{C} \tag{10.5}$$

where $\mathbf{X}$, $\mathbf{C}$, $\mathbf{A}_i$ are $m \times m$ symmetric matrices and $\vec{b}, \vec{y} \in \mathbb{R}^n$. Our approach is partially inspired by one of the key ideas of [121, 166]. These results write down the dual SDP in the form

$$\min_y \vec{b}^T \vec{y} - K \min(\lambda_{\min}(\sum_{i=1}^n y_i \mathbf{A}_i - \mathbf{C}), 0) \tag{10.6}$$

for some large number K and use non-smooth optimization techniques to solve the dual SDP problem. Here, we follow the same approach but instead write it as a max-min problem $\min_{\vec{y}} f_K(\vec{y})$ where

$$f_K(\vec{y}) = \max_{\mathrm{Tr}\mathbf{X} \leq K, \mathbf{X} \succeq \mathbf{0}} \left(\vec{b}^T \vec{y} + \left\langle \mathbf{X}, \mathbf{C} - \sum_{i=1}^n y_i \mathbf{A}_i \right\rangle \right). \tag{10.7}$$

Thus the SDP problem in fact assumes the form (10.3) and many ideas in this section can be generalized to the minimax problem (10.3).

To get a dual solution, we notice that the cutting plane method maintains a subset of the primal feasible solution $\mathrm{conv}(\mathbf{X}_i)$ such that

$$\min_{\vec{y}} \vec{b}^T \vec{y} + \max_{\mathrm{Tr}\mathbf{X} \leq K, \mathbf{X} \succeq \mathbf{0}} \left\langle \mathbf{X}, \mathbf{C} - \sum_{i=1}^n y_i \mathbf{A}_i \right\rangle \sim \min_{\vec{y}} \vec{b}^T \vec{y} + \max_{\mathbf{X} \in \mathrm{conv}(\mathbf{X}_i)} \left\langle \mathbf{X}, \mathbf{C} - \sum_{i=1}^n y_i \mathbf{A}_i \right\rangle.$$

Applying minimax theorem, this shows that there exists an approximation solution $\mathbf{X}$ in $\mathrm{conv}(\mathbf{X}_i)$ for the primal problem. Hence, we can restrict the primal SDP on the polytope $\mathrm{conv}(\mathbf{X}_i)$, this reduces the primal SDP into a linear program which can be solved very efficiently. This idea of getting primal/dual solution from the cutting plane method is quite general and is the main purpose of this example. As a by-product, we have a faster SDP solver in both primal and dual! We remark that this idea has been used as a heuristic to obtain [165] for getting the primal SDP solution and our contribution here is mainly the asymptotic time analysis.

We first show how to construct the separation oracle for SDP. For that we need to compute smallest eigenvector of a matrix. Below, for completeness we provide a folklore result showing we can do this using fast matrix multiplication.

Lemma 43. *Given a $n \times n$ symmetric matrix $\mathbf{Y}$ such that $-R\mathbf{I} \preceq \mathbf{Y} \preceq R\mathbf{I}$, for any $\epsilon > 0$, with high probability in n in time $O(n^{\omega+o(1)} \log^{O(1)}(R/\epsilon))$ we can find a unit vector $\vec{u}$ such that $\vec{u}^T \mathbf{Y} \vec{u} \geq \lambda_{\max}(\mathbf{Y}) - \epsilon$.*

Proof. Let $\mathbf{B} \overset{\mathrm{def}}{=} \frac{1}{R}\mathbf{Y} + \mathbf{I}$. Note that $\mathbf{B} \succeq \mathbf{0}$. Now, we consider the repeated squaring $\mathbf{B}_0 = \mathbf{B}$ and $\mathbf{B}_{k+1} = \frac{\mathbf{B}_k^2}{\mathrm{Tr}\mathbf{B}_k^2}$. Let $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of $\mathbf{B}$ and $\vec{v}_i$ be the corresponding eigenvectors. Then, it is easy to see the the eigenvalues of $\mathbf{B}_k$ are $\frac{\lambda_i^{2^k}}{\sum_{i=1}^n \lambda_i^{2^k}}$.

Let $\vec{q}$ be a random unit vector and $\vec{r} \overset{\mathrm{def}}{=} \mathbf{B}_k \vec{q}$. Now $\vec{q} = \sum \alpha_i \vec{v}_i$ for some α_i such that $\sum \alpha_i^2 = 1$. Letting

$$\vec{p} = \frac{\sum_{\lambda_i > (1-\delta)\lambda_n} \alpha_i \lambda_i^{2^k} \vec{v}_i}{\sum_{i=1}^n \lambda_i^{2^k}}$$

we have

$$\|\vec{r}-\vec{p}\|_2 = \left\| \frac{\sum_{\lambda_i \leq (1-\delta)\lambda_n} \alpha_i \lambda_i^{2^k} \vec{v}_i}{\sum_{i=1}^n \lambda_i^{2^k}} \right\|_2 \leq \frac{\sum_{\lambda_i \leq (1-\delta)\lambda_n} \lambda_i^{2^k}}{\sum_{i=1}^n \lambda_i^{2^k}} \leq (1-\delta)^{2^k} n.$$

Letting $k = \log_2\left(\frac{\log(n^{3/2}/\delta)}{\delta}\right)$, we have $\|\vec{r}-\vec{p}\|_2 \leq \delta/\sqrt{n}$. Since $\mathbf{0} \preceq \mathbf{B} \preceq 2\mathbf{I}$, we have

$$\begin{aligned} \sqrt{\vec{r}^T\mathbf{B}\vec{r}} &\geq \sqrt{\vec{p}^T\mathbf{B}\vec{p}} - \sqrt{(\vec{r}-\vec{p})^T\mathbf{B}(\vec{r}-\vec{p})} \\ &\geq \sqrt{\vec{p}^T\mathbf{B}\vec{p}} - 2\delta/\sqrt{n}. \end{aligned}$$

Note that $\vec{p}$ involves only eigenvectors between $(1-\delta)\lambda_n$ to λ_n. Hence, we have

$$\sqrt{\vec{r}^T\mathbf{B}\vec{r}} \geq \sqrt{(1-\delta)\lambda_n}\left\|\vec{p}\right\|_2 - 2\delta/\sqrt{n}.$$

With constant probability, we have $\alpha_n = \Omega(1/\sqrt{n})$. Hence, we have $\left\|\vec{p}\right\|_2 = \Omega(1/\sqrt{n})$. Using $\mathbf{B} \preceq 2\mathbf{I}$ and $\left\|\vec{p}\right\|_2 \geq \left\|\vec{r}\right\|_2 - \delta/\sqrt{n}$ we have that so long as δ is a small enough universal constant

$$\begin{aligned} \frac{\sqrt{\vec{r}^T\mathbf{B}\vec{r}}}{\left\|\vec{r}\right\|_2} &\geq \frac{\sqrt{(1-\delta)\lambda_n}\left\|\vec{p}\right\|_2 - 2\delta/\sqrt{n}}{\left\|\vec{p}\right\|_2 + \delta/\sqrt{n}} \\ &= (1-O(\delta))\sqrt{\lambda_n} - O(\delta) \\ &= \sqrt{\lambda_n} - O(\delta\sqrt{R}). \end{aligned}$$

Therefore, we have $\frac{\vec{r}^T\mathbf{Y}\vec{r}}{\left\|\vec{r}\right\|^2} \geq \lambda_{\max}(\mathbf{Y}) - O(R\delta)$. Hence, we can find vector $\vec{r}$ by computing k matrix multiplications. [56] showed that fast matrix multiplication is stable under Frobenius norm, i.e., for any $\eta > 0$, using $O(\log(n/b))$ bits, we can find $\mathbf{C}$ such that $\left\|\mathbf{C}-\mathbf{AB}\right\|_F \leq \frac{1}{b}\left\|\mathbf{A}\right\|\left\|\mathbf{B}\right\|$ in time $O(n^{\omega+\eta})$ where ω is the matrix multiplicative constant. Hence, this algorithm takes only $O(n^{\omega+o(1)}\log^{O(1)}(\delta^{-1}))$ time. The result follows from renormalizing the vector $\vec{r}$, repeating the algorithm $O(\log n)$ times to boost the probability and taking $\delta = \Omega(\epsilon/R)$. □

The following lemma shows how to compute a separation for f_K defined in (10.7).

Lemma 44. *Suppose that $\left\|\mathbf{A}_i\right\|_F \leq M$ and $\left\|\mathbf{C}\right\|_F \leq M$. For any $0 < \epsilon < 1$ and $\vec{y}$ with $\left\|\vec{y}\right\|_2 = O(L)$, with high probability in m, we can compute a (ϵ,ϵ)-separation of f_K on $\{\left\|\vec{x}\right\|_2 \leq L\}$ at $\vec{y}$ in time $O(S + m^{\omega+o(1)}\log^{O(1)}(nKML/\epsilon))$ where where S is the sparsity of the problem defined as* $\mathrm{nnz}(\mathbf{C}) + \sum_{i=1}^n \mathrm{nnz}(\mathbf{A}_i)$.

Proof. Note that $-O(nML)\mathbf{I} \preceq \mathbf{C} - \sum_{i=1}^n y_i\mathbf{A}_i \preceq O(nML)\mathbf{I}$. Using Lemma 43, we can find a vector $\vec{v}$ with $\left\|\vec{v}\right\|_2 = K$ in time $O(m^{\omega+o(1)}\log^{O(1)}(nKML/\delta))$ such that

$$\vec{v}^T\left(\mathbf{C} - \sum_{i=1}^n y_i\mathbf{A}_i\right)\vec{v} \geq \max_{\mathrm{Tr}\mathbf{X}\leq K, \mathbf{X}\succeq\mathbf{0}} \left\langle \mathbf{X}, \mathbf{C} - \sum_{i=1}^n y_i\mathbf{A}_i \right\rangle - \delta. \tag{10.8}$$

In other words, we have a δ-optimization oracle for the function f_K. Lemma 40 shows this yields a δ-subgradient oracle and Lemma 38 then shows this yields a $\left(O(\sqrt{\delta L}), O(\sqrt{\delta L})\right)$-separation oracle on the set $\{\left\|\vec{x}\right\|_2 \leq L\}$. By picking $\delta = \epsilon^2/L$, we have the promised oracle. $\square$

With the separation oracle in hand, we are ready to give the algorithm for SDP:

Theorem 45. *Given a primal-dual semidefinite programming problem in the form (10.4) and (10.5), suppose that for some $M \geq 1$ we have*

1. $\left\|b\right\|_2 \leq M$, $\left\|\mathbf{C}\right\|_F \leq M$ *and* $\left\|\mathbf{A}_i\right\|_F \leq M$ *for all* i.
2. *The primal feasible set lies inside the region* $\mathrm{Tr}\mathbf{X} \leq M$.
3. *The dual feasible set lies inside the region* $\left\|\vec{y}\right\|_\infty \leq M$.

Let opt *be the optimum solution of (10.4) and (10.5). Then, with high probability, we can find* $\mathbf{X}$ *and* $\vec{y}$ *such that*

1. $\mathbf{X} \succeq \mathbf{0}$, $\mathrm{Tr}\mathbf{X} = O(M)$, $\sum_i |b_i - \langle \mathbf{X}, \mathbf{A}_i \rangle| \leq \epsilon$ *for all* i *and* $\mathbf{C} \bullet \mathbf{X} \geq \mathrm{opt} - \epsilon$.
2. $\left\|\vec{y}\right\|_\infty = O(M)$, $\sum_{i=1}^n y_i \mathbf{A}_i \succeq \mathbf{C} - \epsilon \mathbf{I}$ *and* $\vec{b}^T \vec{y} \leq \mathrm{opt} + \epsilon$.

in expected time $O\left(\left(nS + n^3 + nm^{\omega + o(1)}\right) \log^{O(1)}\left(\frac{nM}{\epsilon}\right)\right)$ *where S is the sparsity of the problem defined as* $\mathrm{nnz}(\mathbf{C}) + \sum_{i=1}^n \mathrm{nnz}(\mathbf{A}_i)$ *and ω is the fast matrix multiplication constant.*

Proof. Let $K \geq M$ be some parameter to be determined. Since the primal feasible set is lies inside the region $\mathrm{Tr}\mathbf{X} \leq M \leq K$, we have

$$\begin{aligned}
\min_{\sum_{i=1}^n y_i \mathbf{A}_i \succeq \mathbf{C}} \vec{b}^T \vec{y} &= \max_{\mathbf{X} \succeq \mathbf{0}, \mathrm{Tr}\mathbf{X} \leq K, \mathbf{A}_i \bullet \mathbf{X} = b_i} \mathbf{C} \bullet \mathbf{X} \\
&= \max_{\mathbf{X} \succeq \mathbf{0}, \mathrm{Tr}\mathbf{X} \leq K} \min_{\vec{y}} \mathbf{C} \bullet \mathbf{X} - \sum_i y_i \left(\mathbf{A}_i \bullet \mathbf{X} - b_i\right) \\
&= \min_{\vec{y}} \max_{\mathbf{X} \succeq \mathbf{0}, \mathrm{Tr}\mathbf{X} \leq K} \left(\vec{b}^T \vec{y} + (\mathbf{C} - \sum_i y_i \mathbf{A}_i) \bullet \mathbf{X}\right) \\
&= \min_{\vec{y}} f_K(\vec{y}).
\end{aligned}$$

Lemma 44 shows that it takes $\mathrm{SO}_{\delta,\delta}(f_K) = O(S + m^{\omega + o(1)} \log(nKML/\delta))$ time to compute a (δ, δ)-separation oracle of f_K for any point $\vec{y}$ with $\|\vec{y}\|_\infty = O(L)$ where L is some parameter with $L \geq M$. Taking the radius $R = L$, Theorem 42 shows that it takes $O\left(n\mathrm{SO}_{\delta,\delta}(f_K) \log\left(\frac{n}{\alpha}\right) + n^3 \log^{O(1)}\left(\frac{n}{\alpha}\right)\right)$ expected time with $\delta = \Theta\left(\alpha n^{-3/2} L\right)$ to find $\vec{y}$ such that

$$f_K(\vec{y}) - \min_{\left\|\vec{y}\right\|_\infty \leq L} f_K(\vec{y}) \leq \delta + \alpha \left(\max_{\left\|\vec{y}\right\|_\infty \leq L} f_K(\vec{y}) - \min_{\left\|\vec{y}\right\|_\infty \leq L} f_K(\vec{y})\right) \leq \delta + 2\alpha \left(nML + 2nKML\right).$$

Picking $\alpha = \frac{\epsilon}{7nMKL}$, we have $f_K(\vec{y}) \leq \min_{\vec{y}} f_K(\vec{y}) + \epsilon$. Therefore,

$$\vec{b}^T\vec{y} + K\max(\lambda_{\max}(\mathbf{C} - \sum_{i=1}^{n} y_i\mathbf{A}_i), 0) \leq \text{opt} + \epsilon.$$

Let $\beta = \max(\lambda_{\max}(\mathbf{C} - \sum_{i=1}^{n} y_i\mathbf{A}_i), 0)$. Then, we have that $\sum_{i=1}^{n} y_i\mathbf{A}_i \succeq \mathbf{C} - \beta\mathbf{I}$ and

$$\begin{aligned}\vec{b}^T\vec{y} &\geq \min_{\sum_{i=1}^{n} y_i\mathbf{A}_i \succeq \mathbf{C}-\beta\mathbf{I}} \vec{b}^T\vec{y} \\ &= \max_{\mathbf{X}\succeq\mathbf{0}\mathbf{A}_i\bullet\mathbf{X}=b_i} (\mathbf{C} - \beta\mathbf{I}) \bullet \mathbf{X} \\ &\geq \text{opt} - \beta M\end{aligned}$$

because $\text{Tr}\mathbf{X} \leq M$. Hence, we have

$$\text{opt} - \beta M + \beta K \leq \vec{b}^T\vec{y} + K\max(\lambda_{\max}(\mathbf{C} - \sum_{i=1}^{n} y_i\mathbf{A}_i), 0) \leq \text{opt} + \epsilon$$

Putting $K = M + 1$, we have $\beta \leq \epsilon$. Thus,

$$\sum_{i=1}^{n} y_i\mathbf{A}_i \succeq \mathbf{C} - \epsilon\mathbf{I}.$$

This gives the result for the dual with the running time $O\left(\left(nS + n^3 + nm^{\omega+o(1)}\right)\log^{O(1)}\left(\frac{nML}{\epsilon}\right)\right)$.

Our Cutting Plane Method accesses the sub-problem

$$\max_{\mathbf{X}\succeq\mathbf{0},\text{Tr}\mathbf{X}\leq K} (\mathbf{C} - \sum_i y_i\mathbf{A}_i) \bullet \mathbf{X}$$

only through the separation oracle. Let $\vec{z}$ be the output of our Cutting Plane Method and $\{\vec{v}_i\vec{v}_i^T\}_{i=1}^{O(n)}$ be the matrices used to construct the separation for the $O(n)$ hyperplanes the algorithm maintains at the end. Let $\vec{u}$ be the maximum eigenvector of $\mathbf{C} - \sum_{i=1}^{n} z_i\mathbf{A}_i$. Now, we consider a realization of f_K

$$\tilde{f}_K(\vec{y}) = \vec{b}^T\vec{y} + \max_{\mathbf{X}\in\text{conv}(K\vec{u}\vec{u}^T, \vec{v}_i\vec{v}_i^T)} \left\langle \mathbf{X}, \mathbf{C} - \sum_{i=1}^{n} y_i\mathbf{A}_i \right\rangle.$$

Since applying our Cutting Plane Method to either f_K or $\tilde{f}_K$ gives the same result, the correctness of the our Cutting Plane Method shows that

$$\tilde{f}_K(\vec{z}) \leq \min_{\left\|\vec{y}\right\|_\infty \leq L} \tilde{f}_K(\vec{y}) + \epsilon.$$

Note that the function $\tilde{f}_K$ is defined such that $\tilde{f}_K(\vec{z}) = f_K(\vec{z})$. Hence, we have

$$\min_{\left\|\vec{y}\right\|_\infty \leq L} f_K(\vec{y}) \leq f_K(\vec{z}) \leq \tilde{f}_K(\vec{z}) \leq \min_{\left\|\vec{y}\right\|_\infty \leq L} \tilde{f}_K(\vec{y}) + \epsilon.$$

Also, note that $\tilde{f}_K(\vec{x}) \leq f_K(\vec{x})$ for all $\vec{x}$. Hence, we have

$$\min_{\left\|\vec{y}\right\|_\infty \leq L} f_K(\vec{y}) - \epsilon \leq \min_{\left\|\vec{y}\right\|_\infty \leq L} \tilde{f}(\vec{y}) \leq \min_{\left\|\vec{y}\right\|_\infty \leq L} f_K(\vec{y}).$$

Now, we consider the primal version of $\tilde{f}$, namely

$$g(\mathbf{X}) \stackrel{\text{def}}{=} \min_{\left\|\vec{y}\right\|_\infty \leq L} \vec{b}^T \vec{y} + \left\langle \mathbf{X}, \mathbf{C} - \sum_{i=1}^n y_i \mathbf{A}_i \right\rangle.$$

Sion's minimax theorem [239] shows that

$$\text{opt} \geq \max_{\mathbf{X} \in \text{conv}(K\vec{u}\vec{u}^T, \vec{v}_i\vec{v}_i^T)} g(\mathbf{X}) = \min_{\left\|\vec{y}\right\|_\infty \leq L} \tilde{f}(\vec{y}) \geq \text{opt} - \epsilon.$$

Therefore, to get the primal solution, we only need to find $\vec{u}$ by Lemma 43 and solve the maximization problem on g. Note that

$$\begin{aligned} g(\mathbf{X}) &= \min_{\left\|\vec{y}\right\|_\infty \leq L} \sum_{i=1}^n y_i \left(b_i - \langle \mathbf{X}, \mathbf{A}_i \rangle\right) + \langle \mathbf{X}, \mathbf{C} \rangle \\ &= -L \sum_i \left| b_i - \langle \mathbf{X}, \mathbf{A}_i \rangle \right| + \langle \mathbf{X}, \mathbf{C} \rangle . \end{aligned}$$

For notation simplicity, we write $K\vec{u}\vec{u}^T = \vec{v}_0\vec{v}_0^T$. Then, $\mathbf{X} = \sum_{j=0}^{O(n)} \alpha_j \vec{v}_j \vec{v}_j^T$ for some $\sum \alpha_j = 1$ and $\alpha_j \geq 0$. Substituting this into the function g, we have

$$g(\vec{\alpha}) = -L \sum_j \left| b_i - \sum_j \alpha_j \vec{v}_j^T \mathbf{A}_i \vec{v}_j \right| + \sum_j \alpha_j \vec{v}_j^T \mathbf{C} \vec{v}_j.$$

Hence, this can be easily written as a linear program with $O(n)$ variables and $O(n)$ constraints in time $O(nS)$. Now, we can apply interior point method to find $\vec{\alpha}$ such that

$$g(\vec{\alpha}) \geq \max_{\mathbf{X} \in \text{conv}(K\vec{u}\vec{u}^T, \vec{v}_i\vec{v}_i^T)} g(\mathbf{X}) - \epsilon \geq \text{opt} - 2\epsilon.$$

Let the corresponding approximate solution be $\widetilde{\mathbf{X}} = \sum \alpha_j \vec{v}_j \vec{v}_j^T$. Then, we have

$$\left\langle \widetilde{\mathbf{X}}, \mathbf{C} \right\rangle - L \sum_i \left| b_i - \langle \mathbf{X}, \mathbf{A}_i \rangle \right| \geq \text{opt} - 2\epsilon.$$

Now, we let $\tilde{b}_i = \left\langle \widetilde{\mathbf{X}}, \mathbf{A}_i \right\rangle$. Then, we note that

$$\begin{aligned} \left\langle \widetilde{\mathbf{X}}, \mathbf{C} \right\rangle &\leq \max_{\mathbf{X} \succeq \mathbf{0} \mathbf{A}_i \bullet \mathbf{X} = \tilde{b}_i} \mathbf{C} \bullet \mathbf{X} \\ &= \min_{\sum_{i=1}^n y_i \mathbf{A}_i \succeq \mathbf{C}} \tilde{b}_i^T \vec{y} \\ &\leq \text{opt} + M \sum_i \left| b_i - \left\langle \widetilde{\mathbf{X}}, \mathbf{A}_i \right\rangle \right| \end{aligned}$$

because $\left\|\vec{y}\right\|_\infty \le M$. Hence, we have

$$\mathrm{opt} + (M-L)\sum_i \left|b_i - \left\langle \widetilde{\mathbf{X}}, \mathbf{A}_i \right\rangle\right| \ge \left\langle \widetilde{\mathbf{X}}, \mathbf{C} \right\rangle - L\sum_i \left|b_i - \left\langle \widetilde{\mathbf{X}}, \mathbf{A}_i \right\rangle\right| \ge \mathrm{opt} - 2\epsilon.$$

Now, we put $L = M+2$, we have

$$\sum_i \left|b_i - \left\langle \widetilde{\mathbf{X}}, \mathbf{A}_i \right\rangle\right| \le \epsilon.$$

This gives the result for the primal. Note that it only takes $O(n^{5/2}\log^{O(1)}(nM/\epsilon))$ to solve a linear program with $O(n)$ variables and $O(n)$ constraints because we have an explicit interior point deep inside the feasible set, i.e. $\alpha_i = \frac{1}{m}$ for some parameter m [177].[3] Hence, the running time is dominated by the cost of cutting plane method which is

$$O\left(\left(nS + n^3 + nm^{\omega+o(1)}\right)\log^{O(1)}\left(\frac{nM}{\epsilon}\right)\right)$$

by putting $L = M+2$. □

We leave it as an open problem if it is possible to improve this result by reusing the computation in the separation oracle and achieve a runtime of $O\left(\left(nS + n^3 + nm^2\right)\log^{O(1)}\left(\frac{nM}{\epsilon}\right)\right)$.

11 Intersection of Convex Sets

In this section we introduce a general technique to optimize a linear function over the intersection of two convex sets, whenever the linear optimization problem on each of them can be done efficiently. At the very high level, this is accomplished by applying cutting plane to a suitably regularized version of the problem. In Section 11.1 we present the technique and in the remaining sections we provide several applications including, matroid intersection (Section 11.2), submodular flow (Section 11.3), and minimizing over the intersection of an affine subspace and a convex set (Section 11.4).

11.1 The Technique

Throughout this section we consider variants of the following general optimization problem

$$\max_{\vec{x}\in K_1\cap K_2} \langle \vec{c}, \vec{x}\rangle \tag{11.1}$$

where $\vec{x}, \vec{c} \in \mathbb{R}^n$, K_1 and K_2 are convex subsets of $\mathbb{R}^n$. We assume that

$$\max_{\vec{x}\in K_1} \|\vec{x}\|_2 < M,\ \max_{\vec{x}\in K_2} \|\vec{x}\|_2 < M,\ \|\vec{c}\|_2 \le M \tag{11.2}$$

[3]Without this, the running time of interior point method depends on the bit complexity of the linear programs.

for some constant $M \geq 1$ and we assume that

$$K_1 \cap K_2 \neq \varnothing. \tag{11.3}$$

Instead of a separation oracle, we assume that K_1 and K_2 each have optimization oracles (see Section 9).

To solve this problem we first introduce a relaxation for the problem (11.1) that we can optimize efficiently. Because we have only the optimization oracles for K_1 and K_2, we simply have variables $\vec{x}$ and $\vec{y}$ for each of them in the objective. Since the output should (approximately) be in the intersection of K_1 and K_2, a regularization term $-\frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2$ is added to force $\vec{x} \approx \vec{y}$ where λ is a large number to be determined later. Furthermore, we add terms to make the problem strong concave.

Lemma 46. *Assume (11.2) and (11.3). For $\lambda \geq 1$, let*

$$f_\lambda(\vec{x},\vec{y}) \overset{\text{def}}{=} \frac{1}{2}\langle \vec{c},\vec{x}\rangle + \frac{1}{2}\langle \vec{c},\vec{y}\rangle - \frac{\lambda}{2}\left\|\vec{x}-\vec{y}\right\|_2^2 - \frac{1}{2\lambda}\left\|\vec{x}\right\|_2^2 - \frac{1}{2\lambda}\left\|\vec{y}\right\|_2^2 \quad . \tag{11.4}$$

There is an unique maximizer $(\vec{x}_\lambda,\vec{y}_\lambda)$ for the problem $\max_{\vec{x}\in K_1,\vec{y}\in K_2} f_\lambda(\vec{x},\vec{y})$. The maximizer $(\vec{x}_\lambda,\vec{y}_\lambda)$ is a good approximation of the solution of (11.1), i.e. $\left\|\vec{x}_\lambda-\vec{y}_\lambda\right\|_2^2 \leq \frac{6M^2}{\lambda}$ and

$$\max_{\vec{x}\in K_1\cap K_2}\langle \vec{c},\vec{x}\rangle \leq f_\lambda(\vec{x}_\lambda,\vec{y}_\lambda) + \frac{M^2}{\lambda}. \tag{11.5}$$

Proof. Let $\vec{x}^*$ be a maximizer of $\max_{\vec{x}\in K_1\cap K_2}\langle \vec{c},\vec{x}\rangle$. By assumption (11.2), $\left\|\vec{x}^*\right\|_2 \leq M$, and therefore

$$f_\lambda(\vec{x}^*,\vec{x}^*) = \langle \vec{c},\vec{x}^*\rangle - \frac{\left\|\vec{x}^*\right\|_2^2}{\lambda} \geq \max_{\vec{x}\in K_1\cap K_2}\langle \vec{c},\vec{x}\rangle - \frac{M^2}{\lambda} \quad . \tag{11.6}$$

This shows (11.5). Since f_λ is strongly concave in $\vec{x}$ and $\vec{y}$, there is a unique maximizer $(\vec{x}_\lambda,\vec{y}_\lambda)$. Let $\text{opt}_\lambda = f_\lambda(\vec{x}_\lambda,\vec{y}_\lambda)$. Then, we have

$$\begin{aligned}
\text{opt}_\lambda &\leq \frac{1}{2}\left\|\vec{c}\right\|_2\left\|\vec{x}_\lambda\right\|_2 + \frac{1}{2}\left\|\vec{c}\right\|_2\left\|\vec{y}_\lambda\right\|_2 - \frac{\lambda}{2}\left\|\vec{x}_\lambda-\vec{y}_\lambda\right\|_2^2 \\
&\leq \frac{M^2}{2} + \frac{M^2}{2} - \frac{\lambda}{2}\left\|\vec{x}_\lambda-\vec{y}_\lambda\right\|_2^2.
\end{aligned}$$

On the other hand, using $\lambda \geq 1$, (11.6) shows that

$$\text{opt}_\lambda \geq f_\lambda(\vec{x}^*,\vec{x}^*) \geq \max_{\vec{x}\in K_1\cap K_2}\langle \vec{c},\vec{x}\rangle - \frac{M^2}{\lambda} \geq -2M^2.$$

Hence, we have

$$\left\|\vec{x}_\lambda-\vec{y}_\lambda\right\|_2^2 \leq \frac{2\left(M^2-\text{opt}_\lambda\right)}{\lambda} \leq \frac{6M^2}{\lambda}. \tag{11.7}$$

□

Now we write $\max f_\lambda(\vec{x},\vec{y})$ as a max-min problem. The reason for doing this is that the dual approximate solution is much easier to obtain and there is a way to read off a primal approximate solution from a dual approximate solution. This is analogous to the idea in [174] which showed how to convert a cut solution to a flow solution by adding regularization terms into the problem.

Lemma 47. *Assume (11.2) and (11.3). Let $\lambda \geq 2$. For any $\vec{x} \in K_1$ and $\vec{y} \in K_2$, the function f_λ can be represented as*

$$f_\lambda(\vec{x},\vec{y}) = \min_{(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)\in\Omega} g_\lambda(\vec{x},\vec{y},\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) \tag{11.8}$$

where $\Omega = \{(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) : \left\|\vec{\theta}_1\right\|_2 \leq 2M,\ \left\|\vec{\theta}_2\right\|_2 \leq M,\ \left\|\vec{\theta}_3\right\|_2 \leq M\}$ and

$$g_\lambda(\vec{x},\vec{y},\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) = \left\langle \frac{\vec{c}}{2} + \lambda\vec{\theta}_1 + \frac{\vec{\theta}_2}{\lambda}, \vec{x} \right\rangle + \left\langle \frac{\vec{c}}{2} - \lambda\vec{\theta}_1 + \frac{\vec{\theta}_3}{\lambda}, \vec{y} \right\rangle + \frac{\lambda}{2}\left\|\vec{\theta}_1\right\|_2^2 + \frac{1}{2\lambda}\left\|\vec{\theta}_2\right\|_2^2 + \frac{1}{2\lambda}\left\|\vec{\theta}_3\right\|_2^2. \tag{11.9}$$

Let $h_\lambda(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) = \max_{\vec{x}\in K_1,\vec{y}\in K_2} g_\lambda(\vec{x},\vec{y},\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)$. For any $(\vec{\theta}_1',\vec{\theta}_2',\vec{\theta}_3')$ such that $h_\lambda(\vec{\theta}_1',\vec{\theta}_2',\vec{\theta}_3') \leq \min_{(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)\in\Omega} h_\lambda(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) + \epsilon$, we know $\vec{z} = \frac{1}{2}(\vec{\theta}_2' + \vec{\theta}_3')$ satisfies

$$\max_{\vec{x}\in K_1\cap K_2} \langle \vec{c},\vec{x}\rangle \leq \langle \vec{c},\vec{z}\rangle + \frac{20M^2}{\lambda} + 20\lambda^3\epsilon.$$

and $\left\|\vec{z}-\vec{x}_\lambda\right\|_2 + \left\|\vec{z}-\vec{y}_\lambda\right\|_2 \leq 4\sqrt{2\lambda\epsilon} + \sqrt{\frac{6M^2}{\lambda}}$ where $(\vec{x}_\lambda,\vec{y}_\lambda)$ is the unique maximizer for the problem $\max_{\vec{x}\in K_1,\vec{y}\in K_2} f_\lambda(\vec{x},\vec{y})$.

Proof. Note that for any $\left\|\vec{\xi}\right\|_2 \leq \alpha$, we have

$$-\frac{1}{2}\left\|\vec{\xi}\right\|_2^2 = \min_{\left\|\vec{\theta}\right\|_2 \leq \alpha} \left\langle \vec{\theta},\vec{\xi} \right\rangle + \frac{1}{2}\left\|\vec{\theta}\right\|_2^2$$

Using this and (11.2), we have (11.8) for all $\vec{x} \in K_1$ and $\vec{y} \in K_2$ as desired. Since Ω is closed and bounded set and the function g_λ is concave in $(\vec{x},\vec{y})$ and convex in $(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)$, Sion's minimax theorem [239] shows that

$$\max_{\vec{x}\in K_1,\vec{y}\in K_2} f_\lambda(\vec{x},\vec{y}) = \min_{(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)\in\Omega} h_\lambda(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) \tag{11.10}$$

Since f_λ is strongly concave, there is an unique maximizer $(\vec{x}_\lambda,\vec{y}_\lambda)$ of f_λ. Since h_λ is strongly convex, there is a unique minimizer $(\vec{\theta}_1^*,\vec{\theta}_2^*,\vec{\theta}_3^*)$. By the definition of f_λ and h_λ, we have

$$h_\lambda(\vec{\theta}_1^*,\vec{\theta}_2^*,\vec{\theta}_3^*) \geq g_\lambda(\vec{x}_\lambda,\vec{y}_\lambda,\vec{\theta}_1^*,\vec{\theta}_2^*,\vec{\theta}_3^*) \geq f_\lambda(\vec{x}_\lambda,\vec{y}_\lambda) \quad .$$

Using (11.10), the equality above holds and hence $(\vec{\theta}_1^*,\vec{\theta}_2^*,\vec{\theta}_3^*)$ is the minimizer of $g_\lambda(\vec{x}_\lambda,\vec{y}_\lambda,\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)$ over $(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)$. Since the domain Ω is large enough that $(\vec{\theta}_1^*,\vec{\theta}_2^*,\vec{\theta}_3^*)$ is an interior point in Ω, the optimality condition of g_λ shows that we have $\vec{\theta}_2^* = \vec{x}_\lambda$ and $\vec{\theta}_3^* = \vec{y}_\lambda$.

Since h_λ is $\frac{1}{\lambda}$ strongly convex, we have $\left\|\vec{\theta}_1' - \vec{\theta}_1^*\right\|_2^2 + \left\|\vec{\theta}_2' - \vec{\theta}_2^*\right\|_2^2 + \left\|\vec{\theta}_3' - \vec{\theta}_3^*\right\|_2^2 \le 2\lambda\epsilon$ (Fact 37). Since $\vec{\theta}_2^* = \vec{x}_\lambda$ and $\vec{\theta}_3^* = \vec{y}_\lambda$, we have

$$\left\|\vec{\theta}_2' - \vec{x}_\lambda\right\|_2^2 + \left\|\vec{\theta}_3' - \vec{y}_\lambda\right\|_2^2 \le 2\lambda\epsilon. \tag{11.11}$$

Therefore, we have $\left\|\vec{x}_\lambda - \vec{y}_\lambda\right\|_2 \ge \left\|\vec{\theta}_2' - \vec{\theta}_3'\right\|_2 - 2\sqrt{2\lambda\epsilon}$, $\left\|\vec{x}_\lambda\right\|_2 \ge \left\|\vec{\theta}_2'\right\|_2 - \sqrt{2\lambda\epsilon}$ and $\left\|\vec{y}_\lambda\right\|_2 \ge \left\|\vec{\theta}_3'\right\|_2 - \sqrt{2\lambda\epsilon}$. Using these, $\left\|\vec{x}_\lambda\right\|_2 \le M$ and $\left\|\vec{y}_\lambda\right\|_2 \le M$, we have

$$\begin{aligned}
f_\lambda(\vec{\theta}_2', \vec{\theta}_3') &= \frac{1}{2}\left\langle \vec{c}, \vec{\theta}_2' \right\rangle + \frac{1}{2}\left\langle \vec{c}, \vec{\theta}_3' \right\rangle - \frac{\lambda}{2}\|\vec{\theta}_2' - \vec{\theta}_3'\|_2^2 - \frac{1}{2\lambda}\|\vec{\theta}_2'\|_2^2 - \frac{1}{2\lambda}\|\vec{\theta}_3'\|_2^2 \\
&\ge \frac{1}{2}\left\langle \vec{c}, \vec{x}_\lambda \right\rangle + \frac{1}{2}\left\langle \vec{c}, \vec{y}_\lambda \right\rangle - M\sqrt{2\lambda\epsilon} \\
&\quad -\frac{\lambda}{2}\left(\left\|\vec{x}_\lambda - \vec{y}_\lambda\right\|_2 + 2\sqrt{2\lambda\epsilon}\right)^2 \\
&\quad -\frac{1}{2\lambda}\left(\left\|\vec{x}_\lambda\right\|_2 + \sqrt{2\lambda\epsilon}\right)^2 - \frac{1}{2\lambda}\left(\left\|\vec{y}_\lambda\right\|_2 + \sqrt{2\lambda\epsilon}\right)^2 \\
&= \frac{1}{2}\left\langle \vec{c}, \vec{x}_\lambda \right\rangle + \frac{1}{2}\left\langle \vec{c}, \vec{y}_\lambda \right\rangle - \frac{\lambda}{2}\|\vec{x}_\lambda - \vec{y}_\lambda\|_2^2 - \frac{1}{2\lambda}\|\vec{x}_\lambda\|_2^2 - \frac{1}{2\lambda}\|\vec{y}_\lambda\|_2^2 \\
&\quad -M\sqrt{2\lambda\epsilon} - 2\lambda\sqrt{2\lambda\epsilon}\left\|\vec{x}_\lambda - \vec{y}_\lambda\right\|_2 - 4\lambda^2\epsilon \\
&\quad -\frac{1}{\lambda}\left\|\vec{x}_\lambda\right\|_2\sqrt{2\lambda\epsilon} - \epsilon - \frac{1}{\lambda}\left\|\vec{y}_\lambda\right\|_2\sqrt{2\lambda\epsilon} - \epsilon.
\end{aligned}$$

Using $\left\|\vec{x}_\lambda - \vec{y}_\lambda\right\|_2 \le \sqrt{\frac{6M^2}{\lambda}}$ (Lemma 46), $\left\|\vec{x}_\lambda\right\|_2 < M$ and $\left\|\vec{y}_\lambda\right\|_2 < M$, we have

$$\begin{aligned}
f_\lambda(\vec{\theta}_2', \vec{\theta}_3') &\ge f_\lambda(\vec{x}_\lambda, \vec{y}_\lambda) \\
&\quad -M\sqrt{2\lambda\epsilon} - 2\lambda\sqrt{2\lambda\epsilon}\left\|\vec{x}_\lambda - \vec{y}_\lambda\right\|_2 - 4\lambda^2\epsilon \\
&\quad -\frac{1}{\lambda}\left\|\vec{x}_\lambda\right\|_2\sqrt{2\lambda\epsilon} - \epsilon - \frac{1}{\lambda}\left\|\vec{y}_\lambda\right\|_2\sqrt{2\lambda\epsilon} - \epsilon. \\
&\ge f_\lambda(\vec{x}_\lambda, \vec{y}_\lambda) \\
&\quad -M\sqrt{2\lambda\epsilon} - 2\lambda\sqrt{12\epsilon}M - 4\lambda^2\epsilon \\
&\quad -2M\sqrt{2\frac{\epsilon}{\lambda}} - 2\epsilon.
\end{aligned}$$

Since $\lambda \ge 2$, we have

$$f_\lambda(\vec{\theta}_2', \vec{\theta}_3') \ge f_\lambda(\vec{x}_\lambda, \vec{y}_\lambda) - 20M\lambda\sqrt{\epsilon} - 10\lambda^2\epsilon.$$

Let $\vec{z} = \frac{\vec{\theta}_2' + \vec{\theta}_3'}{2}$. Lemma 46 shows that

$$\begin{aligned}
\max_{\vec{x}\in K_1\cap K_2} \left\langle \vec{c}, \vec{x} \right\rangle &\le \max_{\vec{x}\in K_1, \vec{y}\in K_2} f_\lambda(\vec{x}, \vec{y}) + \frac{M^2}{\lambda} \\
&\le f_\lambda(\vec{\theta}_2', \vec{\theta}_3') + \frac{M^2}{\lambda} + 20M\lambda\sqrt{\epsilon} + 10\lambda^2\epsilon \\
&\le \left\langle \vec{c}, \vec{z} \right\rangle + \frac{20M^2}{\lambda} + 20\lambda^3\epsilon
\end{aligned}$$

because $20M\lambda\sqrt{\epsilon} \le 10\frac{M^2}{\lambda} + 10\lambda^3\epsilon$. Furthermore, we have

$$\begin{aligned}\left\|\vec{z}-\vec{x}_\lambda\right\|_2 + \left\|\vec{z}-\vec{y}_\lambda\right\|_2 &\le \left\|\vec{\theta}_2' - \vec{x}_\lambda\right\|_2 + \left\|\vec{\theta}_3' - \vec{y}_\lambda\right\|_2 + \left\|\vec{\theta}_2' - \vec{\theta}_3'\right\|_2 \\ &\le 4\sqrt{2\lambda\epsilon} + \sqrt{\frac{6M^2}{\lambda}}.\end{aligned}$$

□

We now apply our cutting plane method to solve the optimization problem (11.1). First we show how to transform the optimization oracles for K_1 and K_2 to get a separation oracle for h_λ, with the appropriate parameters.

Lemma 48. *Suppose we have a ϵ-optimization oracle for K_1 and K_2 for some $0 < \epsilon < 1$. Then on the set $\{\left\|\vec{\theta}\right\|_2 \le D\}$, we have a $(O(\sqrt{\epsilon\lambda D}), O(\sqrt{\epsilon\lambda D}))$-separation oracle for h_λ with time complexity $\mathrm{OO}_\epsilon(K_1) + \mathrm{OO}_\epsilon(K_2)$.*

Proof. Recall that the function h_λ is defined by

$$\begin{aligned}& h_\lambda(\vec{\theta}_1, \vec{\theta}_2, \vec{\theta}_3) \\ = & \max_{\vec{x}\in K_1, \vec{y}\in K_2}\left(\left\langle \frac{\vec{c}}{2} + \lambda\vec{\theta}_1 + \frac{\vec{\theta}_2}{\lambda}, \vec{x}\right\rangle + \left\langle \frac{\vec{c}}{2} - \lambda\vec{\theta}_1 + \frac{\vec{\theta}_3}{\lambda}, \vec{y}\right\rangle + \frac{\lambda}{2}\left\|\vec{\theta}_1\right\|_2^2 + \frac{1}{2\lambda}\left\|\vec{\theta}_2\right\|_2^2 + \frac{1}{2\lambda}\left\|\vec{\theta}_3\right\|_2^2\right) \\ = & \max_{\vec{x}\in K_1}\left\langle \frac{\vec{c}}{2} + \lambda\vec{\theta}_1 + \frac{\vec{\theta}_2}{\lambda}, \vec{x}\right\rangle + \max_{\vec{y}\in K_2}\left\langle \frac{\vec{c}}{2} - \lambda\vec{\theta}_1 + \frac{\vec{\theta}_3}{\lambda}, \vec{y}\right\rangle + \frac{\lambda}{2}\left\|\vec{\theta}_1\right\|_2^2 + \frac{1}{2\lambda}\left\|\vec{\theta}_2\right\|_2^2 + \frac{1}{2\lambda}\left\|\vec{\theta}_3\right\|_2^2.\end{aligned}$$

Lemma 40 shows how to compute the subgradient of functions of the form $f(\vec{c}) = \max_{\vec{x}\in K}\langle\vec{c}, \vec{x}\rangle$ using the optimization oracle for K. The rest of the term are differentiable so its subgradient is just the gradient. Hence, by addition rule for subgradients (Fact 37), we have a $O(\epsilon\lambda)$-subgradient oracle for f_λ using a $O(\epsilon)$-optimization oracle for K_1 and K_2. The result then follows from Lemma 38. □

Theorem 49. *Assume (11.2) and (11.3). Suppose that we have ϵ-optimization oracle for every $\epsilon > 0$. For $0 < \delta < 1$, we can find $\vec{z} \in \mathbb{R}^n$ such that*

$$\max_{\vec{x}\in K_1\cap K_2}\langle\vec{c}, \vec{x}\rangle \le \delta + \langle\vec{c}, \vec{z}\rangle$$

and $\left\|\vec{z}-\vec{x}\right\|_2 + \left\|\vec{z}-\vec{y}\right\|_2 \le \delta$ for some $\vec{x}\in K_1$ and $\vec{y}\in K_2$ in time

$$O\left(n\left(\mathrm{OO}_\eta(K_1) + \mathrm{OO}_\eta(K_2)\right)\log\left(\frac{nM}{\delta}\right) + n^3\log^{O(1)}\left(\frac{nM}{\delta}\right)\right)$$

where $\eta = \Omega\left(\left(\frac{\delta}{nM}\right)^{O(1)}\right)$.

Proof. Setting $\lambda = \frac{40M^2}{\delta^2}$ and $\epsilon = \frac{\delta^7}{10^7M^6}$ in Lemma 47 we see that so long as we obtain any approximate solution $(\vec{\theta}_1', \vec{\theta}_2', \vec{\theta}_3')$ such that

$$h_\lambda(\vec{\theta}_1', \vec{\theta}_2', \vec{\theta}_3') \le \min_{(\vec{\theta}_1, \vec{\theta}_2, \vec{\theta}_3)\in\Omega} h_\lambda(\vec{\theta}_1, \vec{\theta}_2, \vec{\theta}_3) + \epsilon,$$

then we obtain the point we want. To apply Theorem 42, we use

$$\tilde{h}(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) = \begin{cases} h_\lambda(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) & \text{if } (\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) \in \Omega \\ +\infty & \text{else} \end{cases}.$$

Lemma 48 shows that for any $\gamma > 0$ we can obtain a (γ,γ)-separation oracle of $h_\lambda(\vec{\theta})$ by using sufficiently accurate optimization oracles. Since Ω is just a product of ℓ^2 balls, we can produce a separating hyperplane easily when $(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3) \notin \Omega$. Hence, we can obtain a (γ,γ)-separation oracle of $\tilde{h}(\vec{\theta})$. For simplicity, we use $\vec{\theta}$ to represent $(\vec{\theta}_1,\vec{\theta}_2,\vec{\theta}_3)$. Note that $B_\infty(2M) \supseteq \Omega$ and therefore we can apply Theorem 42 with $R = 2M$ to compute $\vec{\theta}'$ such

$$\tilde{h}(\vec{\theta}') - \min_{\vec{\theta}\in\Omega} \tilde{h}(\vec{\theta}) \leq \gamma + \alpha \left(\max_{\vec{\theta}\in\Omega} \tilde{h}(\vec{\theta}) - \min_{\vec{\theta}\in\Omega} \tilde{h}(\vec{\theta}) \right)$$

in time $O\left(n\text{SO}_{\gamma,\gamma} \log\left(\frac{n\kappa}{\alpha}\right) + n^3 \log^{O(1)}\left(\frac{n\kappa}{\alpha}\right)\right)$ where $\gamma = \Omega\left(\alpha \text{MinWidth}(\Omega)/n^{O(1)}\right)$ $= \Omega\left(\alpha M/n^{O(1)}\right)$ and $\kappa = \frac{2M}{\text{MinWidth}(\Omega)} = O(1)$. Using $\lambda \geq 1$ and $M \geq 1$, we have

$$\max_{\vec{\theta}\in\Omega} \tilde{h}(\vec{\theta}) - \min_{\vec{\theta}\in\Omega} \tilde{h}(\vec{\theta}) \leq O\left(\lambda M^2\right) \leq O\left(\frac{M^4}{\delta^2}\right).$$

Setting $\alpha = \Theta\left(\frac{\delta^9}{M^{10}}\right)$ with some small enough constant, we have that we can find $\vec{\theta}'$ such that

$$\begin{aligned} h_\lambda(\vec{\theta}') &\leq \min_{\vec{\theta}\in P} h_\lambda(\vec{\theta}) + \gamma + \alpha O\left(\frac{M^4}{\delta^2}\right) \\ &= \min_{\vec{\theta}\in P} h_\lambda(\vec{\theta}) + O\left(\frac{\delta^7}{M^6}\right) \\ &= \min_{\vec{\theta}\in P} h_\lambda(\vec{\theta}) + \epsilon \end{aligned}$$

in time $O\left(n\text{SO}_{\gamma,\gamma} \log\left(\frac{nM}{\delta}\right) + n^3 \log^{O(1)}\left(\frac{nM}{\delta}\right)\right)$ where $\gamma = \Omega\left(\left(\frac{\delta}{nM}\right)^{O(1)}\right)$. Lemma 48 shows that the cost of (γ,γ)-separation oracle is just $O(\text{OO}_\eta(K_1)+\text{OO}_\eta(K_2))$ where $\eta = \Omega\left(\left(\frac{\delta}{nM}\right)^{O(1)}\right)$. □

Remark 50. Note that the algorithm does not promise that we obtain a point close to $K_1 \cap K_2$. It only promises to give a point that is close to both some point in K_1 and some point in K_2. It appears to the authors that a further assumption is needed to get a point close to $K_1 \cap K_2$. For example, if K_1 and K_2 are two almost parallel lines, it would be difficult to get an algorithm that does not depend on the angle. However, as far as we know, most algorithms tackling this problem are pseudo-polynomial and have polynomial dependence on the angle. Our algorithm depends on the logarithmic of the angle which is useful for combinatorial problems.

This reduction is very useful for problems in many areas including linear programming, semi-definite programming and algorithmic game theory. In the remainder of this section we demonstrate its power by applying it to classical combinatorial problems.

There is however one issue with applying our cutting plane algorithm to these problems. As with other convex optimization methods, only an approximately optimal solution is found. On the other hand, typically an exact solution is insisted in combinatorial optimization. To overcome this gap, we introduce the following lemma which (1) transforms the objective function so that there is only one optimal solution and (2) shows that an approximate solution is close to the optimal solution whenever it is unique. As we shall see in the next two subsections, this allows us to round an approximate solution to an optimal one.

Lemma 51. *Given a linear program* $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{c}^T\vec{x}$ *where* $\vec{x},\vec{c}\in\mathbb{Z}^n$, $\vec{b}\in\mathbb{Z}^m$ *and* $\mathbf{A}\in\mathbb{Z}^{m\times n}$. *Suppose* $\{\mathbf{A}\vec{x}\geq\vec{b}\}$ *is an integral polytope (i.e. all extreme points are integral) contained in the set* $\{\left\Vert\vec{x}\right\Vert_\infty \leq M\}$. *Then we can find a random cost vector* $\vec{z}\in\mathbb{Z}^n$ *with* $\left\Vert\vec{z}\right\Vert_\infty \leq O(n^2M^2\left\Vert\vec{c}\right\Vert_\infty)$ *such that with constant probability,* $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{z}^T\vec{x}$ *has an unique minimizer* $\vec{x}^*$ *and this minimizer is one of the minimizer(s) of* $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{c}^T\vec{x}$. *Furthermore, if there is an interior point* $\vec{y}$ *such that* $\vec{z}^T\vec{y} < \min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{z}^T\vec{x}+\delta$, *then* $\left\Vert\vec{y}-\vec{x}^*\right\Vert_\infty \leq 2nM\delta$.

Proof. The first part of the lemma follows by randomly perturbing the cost vector $\vec{c}$. We consider a new cost vector $\vec{z} = 100n^2M^2\vec{c}+\vec{r}$ where each coordinate of $\vec{r}$ is sampled randomly from $\{0,1,\cdots,10nM\}$. [156, Lem 4] shows that the linear program $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{z}^T\vec{x}$ has a unique minimizer with constant probability. Furthermore, it is clear that the minimizer of $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{z}^T\vec{x}$ is a minimizer of $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{c}^T\vec{x}$ (as $\vec{r}_i \ll 100n^2M^2|\vec{c}_i|$).

Now we show the second part of the lemma. Given an interior point $\vec{y}$ of the polytope $\{\mathbf{A}\vec{x}\geq\vec{b}\}$, we can write $\vec{y}$ as a convex combination of the vertices of $\{\mathbf{A}\vec{x}\geq\vec{b}\}$, i.e. $\vec{y}=\sum t_i\vec{v}_i$. Note that $\vec{z}^T\vec{y}=\sum t_i\vec{z}^T\vec{v}_i$. If all $\vec{v}_i$ are not the minimizer, then $\vec{z}^T\vec{v}_i \geq \text{opt}+1$ and hence $\vec{z}^T\vec{y}\geq\text{opt}+1$ which is impossible. Hence, we can assume that $\vec{v}_1$ is the minimizer. Hence, $\vec{z}^T\vec{v}_i=\text{opt}$ if $i=1$ and $\vec{z}^T\vec{v}_i\geq\text{opt}+1$ otherwise. We then have $\vec{z}^T\vec{y}\geq\text{opt}+(1-t_1)$ which gives $1-t_1<\delta$. Finally, the claim follows from $\left\Vert\vec{y}-\vec{v}_1\right\Vert_\infty \leq \sum_{i\neq1} t_i\left\Vert\vec{v}_i-\vec{v}_1\right\Vert_\infty \leq 2nM\delta$. □

11.2 Matroid Intersection

Let $M_1=(E,\mathcal{I}_1)$ and $M_2=(E,\mathcal{I}_2)$ be two matroids sharing the same ground set. In this section we consider the weighted matroid intersection problem

$$\min_{S\in\mathcal{I}_1\cap\mathcal{I}_2} \vec{w}(S).$$

where $\vec{w}\in\mathbb{R}^E$ and $w(S)\overset{\text{def}}{=}\sum_{e\in S} w_e$.

For any matroid $M=(E,\mathcal{I})$, it is well known that the polytope of all independent sets has the following description [69]:

$$\text{conv}(\mathcal{I}_1)=\{\vec{x}\in\mathbb{R}^E \text{ s.t. } 0\leq x(S)\leq r(S) \text{ for all } S\subseteq E\} \tag{11.12}$$

where r is the rank function for M, i.e. $r(S)$ is the size of the largest independent set that is a subset of S. Furthermore, the polytope of the matroid intersection satisfies $\text{conv}(\mathcal{I}_1\cap\mathcal{I}_2)=\text{conv}(\mathcal{I}_1)\cap\text{conv}(\mathcal{I}_2)$.

It is well known that the optimization problem

$$\min_{S\in\mathcal{I}_1} w(S) \text{ and } \min_{S\in\mathcal{I}_2} w(S)$$

can be solved efficiently by the greedy method. Given a matroid (polytope), the greedy method finds a maximum weight independent subset by maintaining a candidate independent subset S and iteratively attempts to add new element to S in descending weight. A element i is added to S if $S\cup\{i\}$ is still independent. A proof of this algorithm is well-known and can be found in any standard textbook on combinatorial optimization.

Clearly, the greedy method can be implemented by $O(n)$ calls to the independence oracle (also called membership oracle). For rank oracle, it requires $O(r\log n)$ calls by finding the next element to add via binary search. Therefore, we can apply Theorem 49 to get the following result (note that this algorithm is the fastest if r is close to n for the independence oracle).

Theorem 52. *Suppose that the weights $\vec{w}$ are integer with $\|w\|_\infty \leq M$. Then, we can find*

$$S \in \operatorname{argmin}_{S\in\mathcal{I}_1\cap\mathcal{I}_2} w(S)$$

in time $O\left(n\mathrm{GO}\log(nM) + n^3\log^{O(1)}(nM)\right)$ where GO *is the cost of greedy method for $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Proof. Applying Lemma 51, we can find a new cost $\vec{z}$ such that

$$\min_{\vec{x}\in\mathrm{conv}(\mathcal{I}_1)\cap\mathrm{conv}(\mathcal{I}_2)} \vec{z}^T\vec{x}$$

has an unique solution. Note that for any $\vec{x}\in\mathrm{conv}(\mathcal{I}_1)$, we have $\|\vec{x}\|_\infty \leq 1$. Hence, applying theorem 49, we can find $\vec{q}$ such that $\vec{q}^T\vec{z} \leq \mathrm{opt}+\epsilon$ and $\|\vec{q}-\vec{x}\|_2 + \|\vec{q}-\vec{y}\|_2 \leq \epsilon$ for some $\vec{x}\in\mathrm{conv}(\mathcal{I}_1)$ and $\vec{y}\in\mathrm{conv}(\mathcal{I}_2)$. Using (11.12), we have the coordinate wise minimum of $\vec{x},\vec{y}$, i.e. $\min\{\vec{x},\vec{y}\}$, is in $\mathrm{conv}(\mathcal{I}_1)\cap\mathrm{conv}(\mathcal{I}_2)$. Since $\|\vec{q}-\min\{\vec{x},\vec{y}\}\|_2 \leq \|\vec{q}-\vec{x}\|_2 + \|\vec{q}-\vec{y}\|_2 \leq \epsilon$, we have

$$\left(\min\{\vec{x},\vec{y}\}\right)^T\vec{z} \leq \mathrm{opt} + nM\epsilon.$$

Hence, we have a feasible point $\min\{\vec{x},\vec{y}\}$ which has value close to optimal and Lemma 51 shows that $\|\min(\vec{x},\vec{y})-\vec{s}\|_\infty \leq 2n^2M^2\epsilon$ where $\vec{s}$ is the optimal solution. Hence, we have $\|\vec{q}-\vec{s}\|_\infty \leq 2n^2M^2\epsilon+\epsilon$. Picking $\epsilon = \frac{1}{6n^2M^2}$, we have $\|\vec{q}-\vec{s}\|_\infty < \frac{1}{2}$ and hence, we can get the optimal solution by rounding to the nearest integer.

Since optimization over $\mathcal{I}_1$ and $\mathcal{I}_2$ involves applying greedy method on certain vectors, it takes only $O(\mathrm{GO})$ time. Theorem 49 shows it only takes $O\left(n\mathrm{GO}\log(nM) + n^3\log^{O(1)}(nM)\right)$ in finding such $\vec{q}$. $\square$

This gives the following corollary.

Corollary 53. *We have $O(n^2\mathcal{T}_{ind}\log(nM) + n^3\log^{O(1)} nM)$ and $O(nr\mathcal{T}_{rank}\log n\log(nM) + n^3\log^{O(1)} nM)$ time algorithms for weighted matroid intersection. Here $\mathcal{T}_{ind}$ is the time needed to check if a subset is independent, and $\mathcal{T}_{rank}$ is the time needed to compute the rank of a given subset.*

Proof. By Theorem 52, it suffices to show that the optimization oracle for the matroid polytope can be implemented in $O(n\mathcal{T}_{\text{ind}})$ and $O(r\mathcal{T}_{\text{rank}}\log n)$ time. This is simply attained by the well-known greedy algorithm which iterates through all the positively-weighted elements in decreasing order, and adds an element to our candidate independent set whenever possible.

For the independence oracle, this involves one oracle call for each element. On the other hand, for the rank oracle, we can find the next element to add by binary search which takes time $O(\mathcal{T}_{\text{rank}}\log n)$. Since there are at most r elements to add, we have the desired running time. $\square$

11.3 Submodular Flow

Let $G = (V, E)$ be a directed graphwith $|E| = m$, let f be a submodular function on $\mathbb{R}^V$ with $|V| = n$, $f(\varnothing) = 0$ and $f(V) = 0$, and let A be the incidence matrix of G. In this section we consider the submodular flow problem

$$\begin{aligned} \text{Minimize} \quad & \langle c, \varphi \rangle \\ \text{subject to} \quad & l(e) \leq \varphi(e) \leq u(e) \quad \forall e \in E \\ & x(v) = (A\varphi)(v) \quad \forall v \in V \\ & \sum_{v \in S} x(v) \leq f(S) \quad \forall S \subseteq V \end{aligned} \tag{11.13}$$

where $c \in \mathbb{Z}^E$, $l \in \mathbb{Z}^E$, $u \in \mathbb{Z}^E$ where $C = \left\|\vec{c}\right\|_\infty$ and $U = \max\left(\left\|u\right\|_\infty, \left\|l\right\|_\infty, \max_{S \subset V}|f(S)|\right)$. Here c is the cost on edges, φ is the flow on edges, l and u are lower and upper bounds on the amount of flow on the edges, and $x(v)$ is the net flow out of vertex v. The submodular function f upper bounds the total net flow out of any subset S of vertices by $f(S)$.

Theorem 54. *Suppose that the cost vector $\vec{c}$ is integer weight with $\left\|\vec{c}\right\|_\infty \leq C$ and the capacity vector and the submodular function satisfy $U = \max\left(\left\|u\right\|_\infty, \left\|l\right\|_\infty, \max_{S \subset V}|f(S)|\right)$. Then, we can solve the submodular flow problem (11.13) in time $O\left(n^2 \mathrm{EO}\log(mCU) + n^3\log^{O(1)}(mCU)\right)$ where EO is the cost of function evaluation.*

Proof. First, we can assume $l(e) \leq u(e)$ for every edge e, otherwise, the problem is infeasible. Now, we apply a similar transformation in [111] to modify the graph. We create a new vertex v_0. For every vertex v in V, we create a edge from v_0 to v with capacity lower bounded by 0, upper bounded by $4nU$, and with cost $2mCU$. Edmonds and Giles showed that the submodular flow polytope is integral [70]. Hence, there is an integral optimal flow on this new graph. If the optimal flow passes through the newly created edge, then it has cost at least $2mCU - mCU$ because the cost of all other edges in total has at least $-mCU$. That means the optimal flow has the cost larger than mCU which is impossible. So the optimal flow does not use the newly created edges and vertex and hence the optimal flow in the new problem gives the optimal solution of the original problem. Next, we note that for any φ on the original graph such that $l(e) \leq \varphi(e) \leq u(e)$, we can send suitable amount of flow from v_0 to v to make φ feasible. Hence, this modification makes the feasibility problem trivial.

Lemma 51 shows that we can assume the new problem has an unique solution and it only blows up C by a $(mU)^{O(1)}$ factors.

Note that the optimal value is an integer and its absolute value at most mCU. By binary search, we can assume we know the optimal value opt. Now, we reduce the problem to finding a feasible φ with $\{\langle d,\varphi\rangle \leq \text{opt}+\epsilon\}$ with ϵ determined later. Let P_ϵ be the set of such φ. Note that $P_\epsilon = K_{1,\epsilon}\cap K_{2,\epsilon}$ where

$$\begin{aligned} K_{1,\epsilon} &= \left\{ x\in\mathbb{R}^V \text{ such that } \begin{array}{c} l(e)\leq\varphi(e)\leq u(e) \quad \forall e\in E \\ x(v) = (A\varphi)(v) \quad \forall v\in V \\ \langle d,\varphi\rangle \leq \text{opt}+\epsilon \end{array} \text{ for some } \varphi \right\}, \\ K_{2,\epsilon} &= \left\{ y\in\mathbb{R}^V \text{ such that } \sum_{v\in S} y(v)\leq f(S) \quad \forall S\subseteq V, \sum_{v\in V} y(v) = f(V) \right\}. \end{aligned}$$

Note that the extra condition $\sum_v y(v) = f(V)$ is valid because $\sum_v y(v) = \sum_v (A\varphi)(v) = 0$ and $f(V)=0$, and $K_{1,\epsilon}$ has radius bounded by $O((mCU)^{O(1)})$ and $K_{2,\epsilon}$ has radius bounded by $O(nU)$. Furthermore, for any vector $\vec{c}\in\mathbb{R}^V$, we note that

$$\begin{aligned} \max_{x\in K_{1,\epsilon}} \langle c,x\rangle &= \max_{l\leq\varphi\leq u, \langle d,\varphi\rangle\leq \text{opt}+\epsilon, x=A\varphi} \langle c,x\rangle \\ &= \max_{l\leq\varphi\leq u, \langle d,\varphi\rangle\leq \text{opt}+\epsilon} \langle c,A\varphi\rangle \\ &= \max_{l\leq\varphi\leq u, \langle d,\varphi\rangle\leq \text{opt}+\epsilon} \left\langle A^T c,\varphi\right\rangle. \end{aligned}$$

To solve this problem, again we can do a binary search on $\langle d,\varphi\rangle$ and reduce the problem to

$$\max_{l\leq\varphi\leq u, \langle d,\varphi\rangle = K} \left\langle A^T c,\varphi\right\rangle$$

for some value of K. Since $A^T c$ is fixed, this is a linear program with only the box constraints and an extra equality constraint. Hence, it can be solved in nearly linear time [177, Thm 17, ArXiv v1]. As the optimization oracle for $K_{1,\epsilon}$ involves only computing $A^T c$ and solving this simple linear program, it takes only $O(n^2\log^{O(1)}(mCU/\epsilon))$ time. On the other hand, since $K_{2,\epsilon}$ is just a base polyhedron, the optimization oracle for $K_{2,\epsilon}$ can be done by greedy method and only takes $O(n\text{EO})$ time.

Applying Theorem 49, we can find q such that $\left\|q-x\right\|_2 + \left\|q-y\right\|_2 \leq \delta$ for some $x\in K_{1,\epsilon}$, $y\in K_{2,\epsilon}$ and δ to be chosen later. According to the definition of $K_{1,\epsilon}$, there is φ such that $l(e)\leq\varphi(e)\leq u(e)$ and $x(v) = (A\varphi)(v)$ for all v and $\langle d,\varphi\rangle\leq\text{opt}+\epsilon$. Since $\left\|y-x\right\|_2\leq 2\delta$, that means $|y(v)-(A\varphi)(v)|\leq 2\delta$ for all v.

- Case 1) If $y(v)\geq(A\varphi)(v)$, then we can replace $y(v)$ by $(A\varphi)(v)$, note that y is still in $K_{2,\epsilon}$ because of the submodular constraints.
- Case 2) If $y(v)\leq(A\varphi)(v)$, then we can send a suitable amount of flow from v_0 to v to make φ feasible $y(v)\leq(A\varphi)(v)$.

Note that under this modification, we increased the objective value by $(\delta n)(2mCU)$ because the new edge cost $2mCU$ per unit of flow. Hence, we find a flow φ which is feasible in new graph with objective value $\epsilon+(\delta n)(2mCU)$ far from optimum value. By picking $\delta = \frac{1}{2mnCU}$, we have the value 2ϵ far from opt. Now, we use Lemma 51 to shows that when ϵ is small enough, i.e, $\frac{1}{(mCU)^c}$ for some constant c, then we can guarantee that $\left\|y - x^*\right\|_\infty \le \frac{1}{4}$ where x^* is the optimal demand. Now, we note that $\left\|q - y\right\|_2 \le \delta$ and we note that we only modify y by a small amount, we in fact have $\left\|q - x^*\right\|_\infty < \frac{1}{2}$. Hence, we can read off the solution x^* by rounding q to the nearest integer. Note that we only need to solve the problem $K_{1,\epsilon} \cap K_{2,\epsilon}$ to $\frac{1}{(mCU)^{\Theta(1)}}$ accuracy and the optimization oracle for $K_{1,\epsilon}$ and $K_{2,\epsilon}$ takes time $O(n^2 \log^{O(1)}(mCU))$ and $O(n\text{EO})$ respectively. Hence, Theorem 49 shows that it takes $O\left(n^2\text{EO}\log(mCU) + n^3\log^{O(1)}(mCU)\right)$ time to find x^* exactly.

After getting x^*, one can find φ^* by solving a min cost flow problem using interior point method [175], which takes $O(m\sqrt{n}\log^{O(1)}(mCU))$ time. □

11.4 Affine Subspace of Convex Set

In this section, we give another example about using optimization oracle directly via regularization. We consider the following optimization problem

$$\max_{\vec{x}\in K \text{ and } \mathbf{A}\vec{x}=\vec{b}} \langle \vec{c}, \vec{x}\rangle \tag{11.14}$$

where $\vec{x}, \vec{c} \in \mathbb{R}^n$, K is a convex subset of $\mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{r\times n}$ and $\vec{b} \in \mathbb{R}^m$. We suppose that $r \ll n$ and thus, the goal of this subsection is to show how to obtain an algorithm takes only $\tilde{O}(r)$ many iterations. To do this, we assume a slightly stronger optimization oracle for K:

Definition 55. Given a convex set K and $\delta > 0$. A δ-2nd-order-optimization oracle for K is a function on $\mathbb{R}^n$ such that for any input $\vec{c} \in \mathbb{R}^n$ and $\lambda > 0$, it outputs $\vec{y}$ such that

$$\max_{\vec{x}\in K}\left(\langle \vec{c}, \vec{x}\rangle - \lambda\left\|\vec{x}\right\|^2\right) \le \delta + \langle \vec{c}, \vec{y}\rangle - \lambda\left\|\vec{y}\right\|^2.$$

We denote by $\text{OO}^{(2)}_{\delta,\lambda}(K)$ the time complexity of this oracle.

The strategy for solving this problem is very similar to the intersection problem and hence some details are omitted.

Theorem 56. *Assume that $\max_{\vec{x}\in K}\left\|\vec{x}\right\|_2 < M$, $\left\|\vec{b}\right\|_2 < M$, $\left\|\vec{c}\right\|_2 < M$, $\left\|\mathbf{A}\right\|_2 < M$ and $\lambda_{\min}(\mathbf{A}) > 1/M$. Assume that $K \cap \{\mathbf{A}\vec{x} = \vec{b}\} \neq \varnothing$ and we have ϵ-2nd-order-optimization oracle for every $\epsilon > 0$. For $0 < \delta < 1$, we can find $\vec{z} \in K$ such that*

$$\max_{\vec{x}\in K \text{ and } \mathbf{A}\vec{x}=\vec{b}} \langle \vec{c}, \vec{x}\rangle \le \delta + \langle \vec{c}, \vec{z}\rangle$$

and $\left\|\mathbf{A}\vec{z} - \vec{b}\right\|_2 \le \delta$. This algorithm takes time

$$O\left(r\text{OO}^{(2)}_{\eta,\lambda}(K)\log\left(\frac{nM}{\delta}\right) + r^3\log^{O(1)}\left(\frac{nM}{\delta}\right)\right)$$

where r is the number of rows in $\mathbf{A}$, $\eta = \left(\frac{\delta}{nM}\right)^{\Theta(1)}$ and $\lambda = \left(\frac{\delta}{nM}\right)^{\Theta(1)}$.

Proof. The proof is based on the minimax problem

$$\mathrm{OPT}_\lambda \overset{\text{def}}{=} \min_{\|\vec{\eta}\|_2 \le \lambda} \max_{\vec{x}\in K} \langle \vec{c}, \vec{x}\rangle + \left\langle \vec{\eta}, \mathbf{A}\vec{x} - \vec{b} \right\rangle - \frac{1}{\lambda}\|\vec{x}\|_2^2$$

where $\lambda = \left(\frac{\delta}{nM}\right)^c$ for some large constant c. We note that

$$\begin{aligned} \mathrm{OPT}_\lambda &= \max_{\vec{x}\in K} \min_{\|\vec{\eta}\|_2 \le \lambda} \langle \vec{c}, \vec{x}\rangle + \left\langle \vec{\eta}, \mathbf{A}\vec{x} - \vec{b} \right\rangle - \frac{1}{\lambda}\|\vec{x}\|_2^2 \\ &= \max_{\vec{x}\in K} \langle \vec{c}, \vec{x}\rangle - \lambda\|\mathbf{A}\vec{x} - \vec{b}\|_2 - \frac{1}{\lambda}\|\vec{x}\|_2^2. \end{aligned}$$

Since $\lambda_{\min}(\mathbf{A}) > 1/M$ and the set K is bounded by M, one can show that the saddle point $(\vec{x}^*, \vec{\eta}^*)$ of the minimax problem gives a good enough solution $\vec{x}$ for the original problem for large enough constant c.

For any $\vec{\eta}$, we define

$$\vec{x}_{\vec{\eta}} = \arg\max_{\vec{x}\in K} \langle \vec{c}, \vec{x}\rangle + \left\langle \vec{\eta}, \mathbf{A}\vec{x} - \vec{b} \right\rangle - \frac{1}{\lambda}\|\vec{x}\|_2^2.$$

Since the problem is strongly concave in $\vec{x}$, one can prove that

$$\|\vec{x}_{\vec{\eta}} - \vec{x}^*\|_2 \le \left(\frac{nM}{\delta}\right)^{O(c)} \|\vec{\eta} - \vec{\eta}^*\|_2.$$

Hence, we can first find an approximate minimizer of the function $f(\vec{\eta}) = \max_{\vec{x}\in K} \langle \vec{c}, \vec{x}\rangle + \left\langle \vec{\eta}, \mathbf{A}\vec{x} - \vec{b} \right\rangle - \frac{1}{\lambda}\|\vec{x}\|_2^2$ and use the oracle to find $\vec{x}_{\vec{\eta}}$.

To find an approximate minimizer of f, we note that the subgradient of f can be found using the optimization oracle similar to Theorem 49. Hence, the result follows from our cutting plane method and the fact that $\vec{\eta} \in \mathbb{R}^r$. $\square$

Remark 57. In [175], they considered the special case $K = \{\vec{x} : 0 \le x_i \le 1\}$ and showed that it can be solved in $\tilde{O}(\sqrt{r})$ iterations using interior point methods. This gives the current fastest algorithm for the maximum flow problem on directed weighted graphs. Our result generalizes their result to any convex set K but with $\tilde{O}(r)$ iterations. This suggests the following open problem: under what condition on K can one optimize linear functions over affine subspaces of K with r constraints in $\tilde{O}(\sqrt{r})$ iterations?

Part III
Submodular Function Minimization via Cutting Plane Method

This Part is based on joint works with Yin Tat Lee and Aaron Sidford.

12 Introduction

Submodularity is a fundamental concept central in the field of combinatorial optimization. Examples of submodular functions include graph cut functions, set coverage function, and utility functions from economics. Since the seminal work by Edmonds in 1970 [67], submodular functions and the problem of minimizing such functions (i.e. submodular function minimization (SFM)) have served as a popular modeling and optimization tools in various fields such as theoretical computer science, operations research, game theory, and most recently, machine learning. Given its prevalence, fast algorithms for SFM are of immense interest both in theory and in practice.

Throughout Part III our goal is to provide faster algorithms for SFM. We consider the standard formulation of SFM: we are given a submodular function f defined on subsets of a n-element ground set and we wish to find the set of minimum value. We assume f is integer valued with absolute value at most M and we assume f can be evaluated on a set in time EO. Our goal is to produce an algorithm that solves the SFM problem for f, i.e. finds a minimizer of f, while minimizing both the number of oracle calls made and the total running time.

Our main results of Part III are an $O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$ time algorithm and an $O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$ time algorithm for SFM. These algorithms improve upon the previous fastest weakly and strongly polynomial time algorithms for SFM which had a running times of $O((n^4 \cdot \text{EO} + n^5) \log M)$ [126] and $O(n^5 \cdot \text{EO} + n^6)$ [217] respectively. Consequently, we improve the running time for SFM in both regimes by roughly a factor of $O(n^2)$.

Both of our algorithms bear resemblance to the classic approach of Grotschel, Lovasz and Schrijver [111, 113] using the Lovasz extension. In fact, our weakly polynomial time algorithm directly applies the results of Part II to the Lovasz Extension. Our strongly polynomial time algorithm does the same as a first step, but then uses further structure of our cutting plan method as well as more modern tools for submodular funciton analysis to proceed. In particular, we apply our cutting plane method for enough iterations to prove that the minimizer is contained in some narrow convex set and then apply techniques originally developed by Iwata, Fleischer, and Fujishige (IFF) [128] to deduce useful structural information about the minimizers. To achieve our fastest algorithms we provide an extension of these IFF techniques that may be of independent interest.

Over the past few decades, SFM has drawn considerable attention from various research communities, most recently in machine learning [16, 164]. Given this abundant interest in SFM, we hope that our ideas will be of value in various practical applications. Indeed, one of the critiques against existing theoretical algorithms is that their running time is too slow to be practical. We hope that given the magnitude of our theoretical running time improvements, our results may ultimately be used to inform the design of faster algorithms in practice.

12.1 Previous Work

Here we provide a brief survey of the history of algorithms for SFM. For a more comprehensive account of the rich history of SFM, we refer the readers to recent surveys [191, 127].

The first weakly and strongly polynomial time algorithms for SFM were based on the ellipsoid method [152] and were established in the foundational work of Grotschel, Lovasz and Schrijver in 1980's [111, 113]. Their work was complemented by a landmark paper by Cunningham in 1985 which provided a pseudopolynomial algorithm that followed a flow-style algorithmic framework [50]. His tools foreshadowed many SFM advances that would take place 15 years later; many modern algorithms synthesize his framework with techniques inspireed by maximum flow algorithms.

The first such "flow style" strongly polynomial algorithms for SFM were discovered independently in breakthrough papers by Schrijver [231] and Iwata, Fleischer, and Fujishige (IFF) [128]. Schrijver's algorithm has a running of $O(n^8 \cdot \text{EO} + n^9)$ and borrows ideas from the push-relabel algorithms [109, 57] for maximum flow. On the other hand, IFF's algorithm runs in time $O(n^7 \log n \cdot \text{EO})$ and $O(n^5 \cdot \text{EO} \log M)$, and applies a flow-scaling scheme with the aid of certain proximity-type lemmas as in work of Tardos [245]. Their method has roots in flow algorithms such as [124, 110].

Subsequent work on SFM provided algorithms with considerably faster running time by extending the ideas in these two "genesis" papers [231, 128] in various novel directions [254, 77, 126, 217, 133]. Currently, the fastest weakly and strongly polynomial time algorithms for SFM have running times of $O((n^4 \cdot \text{EO} + n^5) \log M)$ [126] and $O(n^5 \cdot \text{EO} + n^6)$ [217] respectively. Despite this impressive track record, the running time has not been improved further in the last eight years.

We remark that all of the previous algorithms for SFM proceed by maintaining a convex combination of $O(n)$ BFS's of the base polyhedron, and incrementally improving it in a relatively local manner. As we shall discuss in Section 12.2, our algorithms do not explicitly maintain a convex combination in the same manner and instead operate a bit more globally. This may be one of the fundamental reasons why our algorithms achieve a faster running time.

Finally, beyond the distinction between weakly and strongly polynomial time algorithms for SFM, there has been interest in another type of SFM algorithm, known as fully combinatorial algorithms in which only additions and subtractions are permitted. Previous such algorithms include [133, 126, 125]. We do not consider such algorithms in the remainder of

Authors	Years	Running times	Remarks
Grotschel, Lovasz, Schrijver [111, 113]	1981,1988	$\widetilde{O}(n^5 \cdot \mathrm{EO} + n^7)$[191]	first weakly and strongly
Cunningham [50]	1985	$O(Mn^6 \log nM \cdot \mathrm{EO})$	first pseudopoly
Schrijver [231]	2000	$O(n^8 \cdot \mathrm{EO} + n^9)$	first combin. strongly
Iwata, Fleischer, Fujishige[128]	2000	$O(n^5 \cdot \mathrm{EO} \log M)$ $O(n^7 \log n \cdot \mathrm{EO})$	first combin. strongly
Iwata, Fleischer [77]	2000	$O(n^7 \cdot \mathrm{EO} + n^8)$	
Iwata [126]	2003	$O((n^4 \cdot \mathrm{EO} + n^5) \log M)$ $O((n^6 \cdot \mathrm{EO} + n^7) \log n)$	current best weakly
Vygen [254]	2003	$O(n^7 \cdot \mathrm{EO} + n^8)$	
Orlin [217]	2007	$O(n^5 \cdot \mathrm{EO} + n^6)$	current best strongly
Iwata, Orlin [133]	2009	$O((n^4 \cdot \mathrm{EO} + n^5) \log nM)$ $O((n^5 \cdot \mathrm{EO} + n^6) \log n)$	
Our algorithms	2015	$O(n^2 \log nM \cdot \mathrm{EO} + n^3 \log^{O(1)} nM)$ $O(n^3 \log^2 n \cdot \mathrm{EO} + n^4 \log^{O(1)} n)$	

Table 5: Algorithms for submodular function minimization. Note that some of these algorithms were published in both conferences and journals, in which case the year provided is the earlier one.

the thesis and leave it as an open question if it is possible to turn our algorithms into fully combinatorial ones.

12.2 Our Results and Techniques

In Part III we show how to improve upon the previous best known running times for SFM by a factor of $O(n^2)$ in both the strongly and weakly polynomial regimes. Table 5 summarizes both the previous known running times as well as those presented in this work.

Both our weakly and strongly polynomial algorithms for SFM utilize a convex relaxation of the submodular function, called the *Lovasz extension*. Our algorithms apply our cutting plane method from Part I using as a separation oracle the subgradient of the Lovasz extension. This convex optimization framework for SFM is a well known classic approach that to the best of the author's knowledge originated in the seminal work of Grotschel, Lovasz and Schrijver [111, 113].

Our weakly polynomial algorithms follow from two rather direct observations. First, we show that cutting plane methods such as Vaidya's [250] can be applied to SFM to yield faster algorithms. Second, as our cutting plane method, Theorem 42, improves upon previous cutting plane algorithms we show that it further improves the running time of SFM. This yields a running time of $O(n^2 \log nM \cdot \mathrm{EO} + n^3 \log^{O(1)} nM)$ improving upon the previous best algorithm by Iwata [126] by a factor of almost $O(n^2)$.

Our strongly polynomial algorithms require substantially more innovation. We first providea a simple geometric argument that SFM can be solved in $O(n^3 \log n \cdot \mathrm{EO})$ oracle calls (but in exponential time). This proof only uses Grunbaum's Theorem from convex geometry and is completely independent from the rest of the thesis. It was the starting point of our method and suggests that a running time of $\widetilde{O}(n^3 \cdot \text{EO} + n^{O(1)})$ for SFM is achievable in principle.

To turn this oracle call bound into an effecient algorithm, instead of appealing to Grubaum's Theorem we use our cutting plane method from Part I. We first run our cutting plane method, i.e. Theorem 31, for enough iterations to either comput a minimizer or a set P containing the minimizers that fits within a narrow strip. This narrow strip consists of the intersection of two approximately parallel hyperplanes. If our narrow strip separates P from one of the faces $x_i = 0$, $x_i = 1$, we can eliminate the element i from consideration and reduce the dimension of our problem by 1. Otherwise a pair of elements p, q can be identified for which q is guaranteed to be in any minimizer containing p (but p may not be contained in a minimizer). Our first algorithm deduces only one such pair at a time and thereby achieves a $\widetilde{O}(n^4 \cdot \text{EO} + n^5)$ time algorithm for SFM (See Section 15.3). We then improve the running time to $\widetilde{O}(n^3 \cdot \text{EO} + n^4)$ by showing how to deduce many such pairs simultaneously. Similar to past algorithms, this structural information is deduced from a point in the so-called base polyhedron (See Section 13).

Readers well-versed in SFM literature may recognize that our strongly polynomial algorithms are reminiscent of the scaling-based approach first used by IFF [128] and later in [126, 133]. While both approaches share the same skeleton, there are differences in how structural information about minimizers is deduced. A comparison of these algorithms is presented in Section 16.

There is one more crucial difference between these algorithms which we believe is responsible for much of our speedup. One common feature of the previous algorithms is that they maintain a convex combination of $O(n)$ BFS's of the base polyhedron, and incrementally improve by introducing new BFS's by making local changes to existing ones. Our algorithms, on the other hand, choose new BFS's by our cutting plane method. Because of this, our algorithm considers the geometry of a larger set of BFS's and threby essentially chooses the next BFS is in a more "global" manner.

12.3 Organization

The rest of Part III is organized as follows. We begin with a gentle introduction to submodular functions in Section 13. In Section 14, we apply our cutting plane method to SFM to obtain a faster weakly polynomial algorithms. In Section 15 we then present our results for achieving better strongly polynomial algorithms, where a warm-up $\widetilde{O}(n^4 \cdot \text{EO} + n^5)$ algorithm is given before the full-fledged $\widetilde{O}(n^3 \cdot \text{EO} + n^4)$ algorithm. We conclude Part III with a discussion and comparison between our algorithms and previous ones in Section 16.

There are a few results in Part III that can be read fairly independently of the rest of the thesis. In Theorem 67 we show how Vaidya's algorithm can be applied to SFM to obtain a faster weakly polynomial running time for SFM and in Theorem 71 we present a simple geometric argument that SFM can be solved with $O(n^3 \log n \cdot \text{EO})$ oracle calls (but exponential time). These results can be read with only a working knowledge of the Lovasz extension of submodular functions as presented in Section 13.

13 Preliminaries

Here we introduce background on submodular function minimization (SFM) and notation that we use throughout Part III. Our exposition is kept to a minimal amount sufficient for our purposes. We refer interested readers to the extensive survey by McCormick [191] for further intuition.

13.1 Submodular Function Minimization

Throughout the rest of this Part, let $V = \{1, ..., n\} = [n]$ denote a ground set and let $f : 2^V \longrightarrow \mathbb{Z}$ denote a *submodular function* defined on subsets of this ground set. We use V and $[n]$ interchangeably and let $[0] \stackrel{\text{def}}{=} \varnothing$. We abuse notation and let $S + i \stackrel{\text{def}}{=} S \cup \{i\}$ and $S - i \stackrel{\text{def}}{=} S \backslash \{i\}$ for an element $i \in V$ and a set $S \subseteq 2^V$. Formally, we call a function submodular if it obeys the following property of *diminishing marginal returns*:

Definition 58 (Submodularity)**.** A function $f : 2^V \longrightarrow \mathbb{Z}$ is *submodular* if $f(T+i) - f(T) \leq f(S+i) - f(S)$ for all $S \subseteq T$ and $i \in V \backslash T$.

For convenience we assume without loss of generality that $f(\varnothing) = 0$ by replacing $f(S)$ by $f(S) - f(\varnothing)$ for all S. We also let $M \stackrel{\text{def}}{=} \max_{S \in 2^V} |f(S)|$.

The central goal of Part III is to design algorithms for SFM, i.e. computing a minimizer of f. We call such an algorithm *strongly polynomial* if its running time depends only polynomially on n and EO, the time needed to compute $f(S)$ for a set S, and we call such an algorithm *weakly polynomial* if it also depends polylogarithmically on M.

13.2 Lovasz Extension

Our new algorithms for SFM all consider a convex relaxation of a submodular function, known as the Lovasz extension, and then carefully apply our cutting plane methods to it. Here we formally introduce the Lovasz extension and present basic facts that we use throughout Part III.

The Lovasz extension of $\hat{f} : [0,1]^n \longrightarrow \mathbb{R}$ of our submodular function f is defined for all $\vec{x}$ by

$$\hat{f}(\vec{x}) \stackrel{\text{def}}{=} \mathbb{E}_{t \sim [0,1]}[f(\{i : x_i \geq t\})],$$

where $t \sim [0,1]$ is drawn uniformly at random from $[0,1]$. The Lovasz extension allows us to reduce SFM to minimizing a convex function defined over the interior of the hypercube. Below we state that the Lovasz extension is a convex relaxation of f and that it can be evaluated efficiently.

Theorem 59. *The Lovasz extension $\hat{f}$ satisfies the following properties:*

1. *$\hat{f}$ is convex;*

2. *$f(S) = \hat{f}(I_S)$, where I_S is the characteristic vector for S, i.e. $I_S(i) = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{if } i \notin S \end{cases}$;*

3. *S is a minimizer of f if and only if I_S is a minimizer of $\hat{f}$;*

4. *Suppose $x_1 \geq \cdots \geq x_n \geq x_{n+1} \stackrel{\text{def}}{=} 0$, then*
$$\hat{f}(\vec{x}) = \sum_{i=1}^{n} f([i])(x_i - x_{i+1}) = \sum_{i=1}^{n} (f([i]) - f([i-1]))x_i \,.$$

Proof. See [113] or any standard textbook on combinatorial optimization, e.g. [233]. □

Next we provide the well known fact that we can efficiently compute a subgradient of the Lovasz extension, which in turn is a separating hyperplane for the set of minimizers of our submodular function f. First we remind the reader of the definition of a separation oracle, and then we prove the result, Theorem 61.

Definition 60 (separation oracle, Defintion 1 restated for Lovasz extension)**.** Given a point $\bar{x}$ and a convex function $\hat{f}$ over a convex set P, $\vec{a}^T\vec{x} \leq b$ is a separating hyperplane if $\vec{a}^T\bar{x} \geq b$ and any minimizer x^* of $\hat{f}$ over P satisfies $\vec{a}^T x^* \leq b$.

Theorem 61. *Given a point $\bar{x} \in [0,1]^n$ assume without loss of generality (by re-indexing the coordinates) that $\bar{x}_1 \geq \cdots \geq \bar{x}_n$. The following inequality is a valid separating hyperplane for $\vec{x}$ and f:*
$$\sum_{i=1}^{n} (f([i]) - f([i-1]))x_i \leq \hat{f}(\bar{x})$$
i.e., it satisfies the following:

1. *(separating) $\bar{x}$ lies on $\sum_{i=1}^{n} (f([i]) - f([i-1]))x_i \leq \hat{f}(\bar{x})$.*

2. *(valid) For any $\vec{x}$, we have $\sum_{i=1}^{n} (f([i]) - f([i-1]))x_i \leq \hat{f}(\vec{x})$. In particular, $\sum_{i=1}^{n} (f([i]) - f([i-1]))x_i^* \leq \hat{f}(\bar{x})$ for any minimizer $\vec{x}^*$, i.e. the separating hyperplane does not cut out any minimizer.*

Moreover, such a hyperplane can be computed with n oracle calls to f and in time $O(n \cdot EO + n^2)$.

Proof. Note that by Theorem 59 we have that $\sum_{i\in[n]} (f([i]) - f([i-1]))x_i = \hat{f}(\bar{x})$ and thus the hyperplane satisfies the separating condition. Moreover, clearly computing it only takes time $O(n \cdot \text{EO} + n^2)$ as we simply need to sort the coordinates and evaluate f at n points, i.e. each of the $[i]$. All that remains is to show that the hyperplane satisfies the valid condition.

Let $L^{(t)} \stackrel{\text{def}}{=} \{i : x_i \geq t\}$. Recall that $\hat{f}(\vec{x}) = \mathbb{E}_{t\sim[0,1]}[f(L_t)]$. Thus $\hat{f}(\vec{x})$ can be written as a convex combination $\hat{f}(\vec{x}) = \sum_t \alpha_t f(L^{(t)})$, where $\alpha_t \geq 0$ and $\sum_t \alpha_t = 1$. However, by diminishing marginal differences we see that for all t
$$\begin{aligned}
\sum_{i\in[n]} (f([i]) - f([i-1]))\left(I_{L^{(t)}}\right)_i &= \sum_{i\in L^{(t)}} (f([i]) - f([i-1])) \\
&\leq \sum_{i\in L^{(t)}} \left(f([i]\cap L^{(t)}) - f([i-1]\cap L^{(t)})\right) \\
&= f(L^{(t)}) - f(\varnothing) = f(L^{(t)})
\end{aligned}$$

and therefore since $\sum_t \alpha_t I_{L^{(t)}} = \vec{x}$ we have

$$\sum_{i\in[n]} (f[i] - f([i-1])x_i = \sum_t \alpha_t \sum_{i=1}^{n} (f([i]) - f([i-1]))\left(I_{L^{(t)}}\right)_i \leq \sum_t \alpha_t f(L^{(t)}) = \hat{f}(\vec{x}).$$

□

13.3 Polyhedral Aspects of SFM

Here we provide a natural primal dual view of SFM that we use throughout the analysis. We provide a dual convex optimization program to minimizing the Lovasz extension and provide several properties of these programs. We believe the material in this section helps crystallize some of the intuition behind our algorithm and we make heavy use of the notation presented in this section. However, we will not need to appeal to the strong duality of these programs in our proofs.

Consider the following primal and dual programs, where we use the shorthands $y(S) = \sum_{i\in S} y_i$ and $y_i^- = \min\{0, y_i\}$. Here the primal constraints are often called the *base polyhedron* $\mathcal{B}(f) \overset{\text{def}}{=} \{\vec{y} \in \mathbb{R}^n : y(S) \leq f(S) \forall S \not\subseteq V, y(V) = f(V)\}$ *and the dual program directly corresponds to minimizing the Lovasz extension and thus f.*

Primal	Dual
$\max y^-(V)$ $y(S) \leq f(S) \forall S \not\subseteq V$ $y(V) = f(V)$	$\min \hat{f}(\vec{x})$ $0 \leq \vec{x} \leq 1$

Theorem 62. *$\vec{h}$ is a basic feasible solution (BFS) of the base polyhedron $\mathcal{B}(f)$ if and only if*

$$h_i = f(\{v_1, ..., v_i\}) - f(\{v_1, ..., v_{i-1}\})$$

for some permutation $v_1, ..., v_n$ of the ground set V. We call $v_1, ..., v_n$ the defining permutation of $\vec{h}$. We call v_i precedes v_j for $i < j$.

This theorem gives a nice characterization of the BFS's of $\mathcal{B}(f)$. It also gives the key observation underpinning our approach: **the coefficients of each separating hyperplane in Theorem 61 precisely corresponds to a primal BFS (Theorem 62)**. Our analysis relies heavily on this connection. We re-state Theorem 61 in the language of BFS.

Lemma 63. *We have $\vec{h}^T\vec{x} \leq \hat{f}(\vec{x})$ for any $\vec{x} \in [0,1]^n$ and BFS $\vec{h}$.*

Proof. Any BFS is given by some permutation. Thus this is just Theorem 61 in disguise. □

We also note that since the objective function of the primal program is non-linear, we cannot say that the optimal solution to the primal program is a BFS. Instead we only know that it is a convex combination of the BFS's that satisfy the following property. A proof can be found in any standard textbook on combinatorial optimization.

Theorem 64. *The above primal and dual programs have no duality gap. Moreover, there always exists a primal optimal solution $\vec{y} = \sum_k \lambda^{(k)} \vec{h}^{(k)}$ with $\sum_k \lambda^{(k)} = 1$ (a convex combination of BFS $\vec{h}^{(k)}$) s.t. any i with $y_i < 0$ precedes any j with $y_j > 0$ in the defining permutation for each BFS $\vec{h}^{(k)}$.*

Our algorithms will maintain collections of BFS and use properties of $\vec{h} \in \mathcal{B}(f)$, i.e. convex combination of BFS. To simplify our analysis at several points we will want to assume that such a vector $\vec{h} \in \mathcal{B}(f)$ is *non-degenerate*, meaning it has both positive and negative entries. Below, we prove that such degenerate points in the base polytope immediately allow us to trivially solve the SFM problem.

Lemma 65 (Degenerate Hyperplanes)**.** *If $\vec{h} \in \mathcal{B}(f)$ is non-negative then $\varnothing$ is a minimizer of f and if $\vec{h}$ is non-positive then V is a minimizer of f.*

Proof. While this follows immediately from Theorem 64, for completeness we prove this directly. Let $S \in 2^V$ be arbitrary. If $\vec{h} \in \mathcal{B}(f)$ is non-negative then by the we have

$$f(S) \geq \vec{h}(S) = \sum_{i \in S} h_i \geq 0 = f(\varnothing)\,.$$

On the other hand if $\vec{h}$ is non-positive then by definition we have

$$f(S) \geq \vec{h}(S) = \sum_{i \in S} h_i \geq \sum_{i \in V} h_i = h(V) = f(V)\,.$$

□

14 Our Weakly Polynomial Algorithm

In this section we show how our cutting plane method can be used to obtain a $O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$ time algorithm for SFM. Our main result in this section is the following theorem, which shows how directly applying our results from earlier parts to minimize the Lovasz extension yields the desired running time.

Theorem 66. *We have an $O(n^2 \log nM \cdot EO + n^3 \log^{O(1)} nM)$ time algorithm for submodular function minimization.*

Proof. We apply Theorem 42 to the Lovasz extension $\hat{f} : [0,1]^n \longrightarrow \mathbb{R}$ with the separation oracle given by Theorem 61. $\hat{f}$ fulfills the requirement on the domain as its domain $\Omega = [0,1]^n$ is symmetric about the point $(1/2, \ldots, 1/2)$ and has exactly $2n$ constraints.

In the language of Theorem 42, our separation oracle is a $(0,0)$-separation oracle with $\eta = 0$ and $\delta = 0$.

We first show that $\delta = 0$. Firstly, our separating hyperplane can be written as

$$\sum_{i=1}^{n} (f([i]) - f([i-1]))x_i \leq \hat{f}(\bar{x}) = \sum_{i=1}^{n} (f([i]) - f([i-1]))\bar{x}_i,$$

where the equality follows from Theorem 59. Secondly, for any $\vec{x}$ with $\hat{f}(\vec{x}) \leq \hat{f}(\bar{x})$ we have by Theorem 61 that

$$\sum_{i=1}^{n}(f([i]) - f([i-1]))x_i \leq \hat{f}(\vec{x}) \leq \hat{f}(\bar{x})$$

which implies that $\vec{x}$ is not cut away by the hyperplane.

Next we show that $\eta = 0$. Our separating hyperplane induces a valid halfspace whenever it is not nonzero, i.e. $f([i]) \neq f([i-1])$ for some i. In the case that it is zero $f([i]) = f([i-1]) \forall i$, by the same argument above, we have $\hat{f}(\bar{x}) = \sum_{i=1}^{n}(f([i]) - f([i-1]))\bar{x}_i = 0$ and

$$\hat{f}(\vec{x}) \geq \sum_{i=1}^{n}(f([i]) - f([i-1]))x_i = 0 = \hat{f}(\bar{x}).$$

In other words, $\bar{x}$ is an exact minimizer, i.e. $\eta = 0$.

Note that $\left|\hat{f}(\vec{x})\right| = \left|\mathbb{E}_{t\sim[0,1]}[f(\{i : x_i \geq t\})]\right| \leq M$ as $M = \max_S |f(S)|$. Now plugging in $\alpha = \frac{1}{4M}$ in the guarantee of Theorem 31, we can find a point x^* such that

$$\begin{aligned}
\hat{f}(x^*) - \min_{\vec{x}\in[0,1]^n} \hat{f}(\vec{x}) &\leq \frac{1}{4M}\left(\max_{\vec{x}\in[0,1]^n} \hat{f}(\vec{x}) - \min_{\vec{x}\in[0,1]^n} \hat{f}(\vec{x})\right) \\
&\leq \frac{1}{4M}(2M) \\
&< 1
\end{aligned}$$

We claim that $\min_{t\in[0,1]} f(\{i : x_i^* \geq t\})$ is minimum. To see this, recall from 59 that $\hat{f}$ has an integer minimizer and hence $\min_{\vec{x}\in[0,1]^n} \hat{f}(\vec{x}) = \min_S f(S)$. Moreover, $\hat{f}(x^*)$ is a convex combination of $f(\{i : x_i^* \geq t\})$ which gives

$$1 > \hat{f}(x^*) - \min_{\vec{x}\in[0,1]^n} \hat{f}(\vec{x}) = \hat{f}(x^*) - \min_S f(S) \geq \min_{t\in[0,1]} f(\{i : x_i^* \geq t\}) - \min_S f(S).$$

Since f is integer-valued, we must then have $\min_{t\in[0,1]} f(\{i : x_i^* \geq t\}) = \min_S f(S)$ as desired. Since our separation oracle can be computed by n oracle calls and runs in time $O(n \cdot \text{EO} + n^2)$, by Theorem 42 the overall running time is then $O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$ as claimed. □

Needless to say the proof above completely depends on Theorem 42. We remark that one can use the Vaidya's cutting plane instead of ours to get a time complexity $O(n^2 \log nM \cdot \text{EO} + n^{\omega+1} \log^{O(1)} n \cdot \log M)$. There is actually an alternate argument that gives a time complexity of $O(n^2 \log M \cdot \text{EO} + n^{O(1)} \cdot \log M)$. Thus it requires slightly fewer oracle calls at the expense of slower running time. A proof is offered in this section, which can be skipped without any risk of discontinuation. This proof relies the following cutting plane method.

Theorem 67 ([20]). *Given any convex set $K \subset [0,1]^n$ with a separation oracle of cost SO, in time $O(kSO + kn^{O(1)})$ one can find either find a point $\vec{x} \in K$ or find a polytope P such that $K \subset P$ and the volume of K is at most $\left(\frac{2}{3}\right)^k$.*

The Theorem allows us to decrease the volume of the feasible region by a factor of $\left(\frac{2}{3}\right)^k$ after k iterations. Similar to above, we apply cutting plane to minimize $\hat{f}$ over the hypercube $[0,1]^n$ for $O(n \log M)$ iterations, and outputs *any* integral point in the remaining feasible region P.

Lemma 68. *Let x^* achieve the minimum function value $\hat{f}(x^*)$ among the points used to query the separation oracle. Then*

1. *$x^* \in P^{(k)}$, the current feasible region.*
2. *Any $\vec{x}$ with $\hat{f}(\vec{x}) \leq \hat{f}(x^*)$ belongs to $P^{(k)}$.*
3. *suppose $x^*_{i_1} \geq \cdots \geq x^*_{i_n}$ and let $S_j = \{i_1, \ldots, i_j\}$. Then $S_l \in \arg\min_{S_j} f(S_j)$ also belongs to $P^{(k)}$.*

Proof. For any separating hyperplane $\vec{h}^T x \leq \hat{f}(\bar{x})$ given by $\bar{x}$, we have by Lemma 63 that $\vec{h}^T x^* \leq \hat{f}(x^*)$. Since $\hat{f}(x^*)$ is the minimum among all $\hat{f}(\bar{x})$, $\vec{h}^T x^* \leq \hat{f}(\bar{x})$ and hence x^* is not removed by any new separating hyperplane. In other words, $x^* \in P^{(k)}$. The argument for (2) is analogous.

For (3), recall that by the definition of Lovasz extension $\hat{f}(x^*)$ is a convex combination of $f(S_j)$ and thus the indicator variable I_{S_l} for S_l satisfies $f(I_{S_l}) \leq \hat{f}(x^*)$. By Lemma 63 again, this implies $\vec{h}^T I_{S_l} \leq f(I_{S_l}) \leq \hat{f}(x^*) \leq \hat{f}(\bar{x})$ for any separating hyperplane $\vec{h}^T x \leq \hat{f}(\bar{x})$. □

Theorem 69. *Suppose that we run Cutting Plane in Theorem 67 for $O(n \log M)$ iterations. Then S_l from the last lemma also minimizes f.*

Proof. We use the notations from the last lemma. After $k = Kn \log_{2/3} M$ iterations, the volume of the feasible region $P^{(k)}$ is at most $1/M^{Kn}$. By the last lemma, $I_{S_l} \in P^{(k)}$.

Suppose for the sake of contradiction that S minimizes f but $f(S) < f(S_l)$. Since f is integer-valued, $f(S) + 1 \leq f(S_l)$. Let $r \stackrel{\text{def}}{=} 1/6M$. Consider the set $B \stackrel{\text{def}}{=} \{\vec{x} : 0 \leq x_i \leq r \,\forall \text{i} \notin S,\ 1 - r \leq x_i \leq 1 \,\forall i \in S\}$. We claim that for $\vec{x} \in B$,

$$\hat{f}(\vec{x}) \leq f(S) + 1.$$

To show this, note that $f(\{i : x_i \geq t\}) = f(S)$ for $r < t \leq 1 - r$ as $x_i \leq r$ for $i \notin S$ and $x_i \geq 1 - r$ for $i \in S$. Now using conditional probability and $|f(T)| \leq M$ for any T,

$$\begin{aligned}
\hat{f}(\vec{x}) &= \mathbb{E}_{t \sim [0,1]}[f(\{i : x_i \geq t\})] \\
&= (1-2r)\,\mathbb{E}[f(\{i : x_i \geq t\})|r < t \leq 1 - r] + \\
&\quad r\,(\mathbb{E}[f(\{i : x_i \geq t\})|0 \leq t \leq r] + \mathbb{E}[f(\{i : x_i \geq t\})|1 - r \leq t \leq 1]]) \\
&= (1-r)\,f(S) + r\,(\mathbb{E}[f(\{i : x_i \geq t\})|0 \leq t \leq r + \mathbb{E}[f(\{i : x_i \geq t\})|1 - r \leq t \leq 1]]) \\
&\leq (1-2r)\,f(S) + 2rM \\
&\leq f(S) + 4rM \\
&\leq f(S) + 1
\end{aligned}$$

But now $B \subseteq P^{(k)}$ as $\hat{f}(\vec{x}) \leq f(S) + 1 \leq f(S_l)$ and by (2) of the last lemma. This would lead to a contradiction since

$$\mathrm{vol}(B) = \frac{1}{(6M)^n} > \frac{1}{M^{Kn}} \geq \mathrm{vol}(P^{(k)})$$

for sufficiently large K. $\square$

Corollary 70. *There is an $O(n^2 \log M \cdot EO + n^{O(1)} \log M)$ time algorithm for submodular function minimization.*

Proof. This simply follows from the last lemma, Theorem 67, and the fact that our separation oracle runs in time $O(n \cdot \mathrm{EO} + n^2)$. $\square$

Curiously, we obtained $O(\log M)$ rather than $O(\log nM)$ as in our algorithm. We leave it as an open problem whether one can slightly improve our running time to $O(n^2 \log M \cdot \mathrm{EO} + n^3 \log^{O(1)} n \cdot \log M)$. The rest of this Part is devoted to obtaining better strongly polynomial running time.

15 Our Strongly Polynomial Algorithm

In this section we show how our cutting plane method can be used to obtain a $\widetilde{O}(n^3 \cdot \mathrm{EO} + n^4)$ time algorithm for SFM improving upon the previous best strongly polynomial running time of $O(n^5 \cdot \mathrm{EO} + n^6)$ due to Orlin [217]. We provide this result in several steps. First in Section 15.1 we show that f can be minimized with $O(n^3 \log n)$ oracle calls. The running time for this algorithm is large but the proof is simple and motivates our approach. In Section 15.2 we then introduce various technical tools which we will use to improve the running time of our algorithms. In Section 15.3 we then present a $\widetilde{O}(n^4 \cdot \mathrm{EO} + n^5)$ time algorithm that uses many of these technical tools. Finally, in Section 15.4 we show how to improve this algorithm to achieve our $\widetilde{O}(n^3 \cdot \mathrm{EO} + n^4)$ running time for SFM.

15.1 Improved Oracle Complexity

Here we present a simple geometric argument that f can be minimized with $O(n^3 \log n)$ oracle calls. This improves upon the previous best known bound by a factor of $O(n^2)$ and is the strongest bound we present in this thesis, but unfortunately the algorithm associated with the proof runs in exponential time. Nevertheless, this analysis provides insight into how our more efficient algorithms should proceed. In the rest of this Part, we combine this insight with existing SFM tools developed over the last decade (as well as some improvements) to get our fastest strongly polynomial time algorithms.

Theorem 71. *Submodular functions can be minimized with $O(n^3 \log n \cdot EO)$ oracle calls.*

Proof. We use the cutting plane method in Theorem 67 with the separation oracle given by Theorem 61. This method reduce the volume of the feasible region by a factor of $(\frac{2}{3})^k$ after k iterations if the optimal has not found yet.

Now, we argue that after $O(n \log n)$ iterations of this procedure we have either found a minimizer of f or we have enough information to reduce the dimension of the problem by 1. To see this, first note that if the separation oracle ever returns a degenerate hyperplane, then by Lemma 65 then either $\varnothing$ or V is the minimizer, which we can determine in time $O(\text{EO} + n)$. Otherwise, after $100n \log n$ iterations, our feasible region P must have a volume of at most $1/n^{10n}$. In this case, we claim that the remaining integer points in P all lie on a hyperplane. This holds, as if this was not the case, then there is a simplex $\triangle$, with integral vertices $v_0, v_1, \ldots, v_n$, contained in P. But then

$$\text{vol}(P) \geq \text{vol}(\triangle) = \frac{1}{n!} \left| \det \left(v_1 - v_0 \; v_2 - v_0 \; \ldots \; v_n - v_0 \right) \right| \geq \frac{1}{n!}$$

where the last inequality holds since the determinant of an integral matrix is integral, yielding a contradiction.

In other words after $O(n \log n)$ iterations, we have reduced the dimension of all viable solutions by at least 1. Thus, we can recurse by applying the cutting plane method to the lower dimensional feasible region, i.e. P is (replaced by) the convex combination of all the remaining integer points. There is a minor technical issue we need to address as our space is now lower dimensional and the starting region is not necessarily the hypercube anymore and the starting volume is not necessarily equal to 1.

We argue that the starting volume is bounded by $n^{O(n)}$. If this is indeed the case, then our previous argument still works as the volume goes down by a factor of $1/n^{O(n)}$ in $O(n \log n)$ iterations.

Let $v \in P$ be an integer point. Now the $\dim(P)$-dimensional ball of radius $\sqrt{n}$ centered at v must contain all the other integer points in P as any two points of $\{0, 1\}^n$ are at most $\sqrt{n}$ apart. Thus the volume of P is bounded by the volume of the ball which is $n^{O(n)}$. Now to get the volume down to $1/n^{10n}$, the number of iterations is still $O(n \log n)$.

In summary, we have reduced our dimension by 1 using $O(n \log n)$ iterations which requires $O(n^2 \log n \cdot \text{EO})$ oracle calls (as each separating hyperplane is computed with $n \cdot \text{EO}$ oracle calls). This can happen at most n times. The overall query complexity is then $O(n^3 \log n \cdot \text{EO})$.

Note that the minimizer $\vec{x}$ obtained may not be integral. This is not a problem as the definition of Lovasz extension implies that if $\hat{f}(\vec{x})$ is minimal, then $f(\{i : x_i \geq t\})$ is minimal for any $t \in [0, 1]$.

We remark that this algorithm does not have a polynomial runtime. Even though all the integral vertices of P lie on a hyperplane, the best way we know of that identifies it takes exponential time by checking for all the integer points $\{0, 1\}^n$. $\square$

Remark 72. Note that this algorithm works for minimizing any convex function over the hypercube that obtains its optimal value at a vertex of the hypercube. Formally, our proof of Theorem 71 holds whenever a function $f : 2^V \longrightarrow \mathbb{R}^n$ admits a convex relaxation $\hat{f}$ with the following properties:

1. For every $S \subseteq V$, $\hat{f}(I_S) = f(S)$.

2. Every $\hat{f}(\vec{x})$ can be written as a convex combination $\sum_{S \in \mathcal{S}} \alpha_S f(S)$, where $\sum \alpha_S = 1$, $|\mathcal{S}| = O(n)$, and $\mathcal{S}$ can be computed without any oracle call.

3. A subgradient $\partial\hat{f}(\vec{x})$ of $\hat{f}$ at any point $\vec{x}\in[0,1]^n$ can be computed with $O(n\cdot\text{EO})$ oracle calls.

In this case, the proof of Theorem 71, implies that $\hat{f}$ and f can be minimized with $O(n^3\log n\cdot \text{EO})$ oracle calls by using the separating hyperplane $\partial\hat{f}(\bar{x})^T(\vec{x}-\bar{x})\leq 0$.

15.2 Technical Tools

To improve upon the running time of the algorithm in the previous section, we use more structure of our submodular function f. Rather than merely showing that we can decrease the dimension of our SFM problem by 1 we show how we can reduce the degrees of freedom of our problem in a more principled way. In Section 15.2.1 we formally define the abstraction we use for this and discuss how to change our separation oracle to accommodate this abstraction, and in Section 15.2.2 we show how we can deduce these constraints. These tools serve as the foundation for the faster strongly polynomial time SFM algorithms we present in Section 15.3 and Section 15.4.

15.2.1 SFM over Ring Family

For the remainder of this Part we consider a more general problem than SFM in which we wish to compute a minimizer of our submodular function f over a ring family of the ground set $V=[n]$. A ring family $\mathcal{F}$ is a set of subsets of V such that for any $S_1,S_2\in\mathcal{F}$, we have $S_1\cup S_2, S_1\cap S_2\in\mathcal{F}$. Thus SFM corresponds to the special case where $\mathcal{F}$ consists of every subset of V. This generalization has been considered before in the literature and was essential to the IFF algorithm.

It is well known that any ring family $\mathcal{F}$ over V can be represented by a directed graph $D=(V,A)$ where $S\in\mathcal{F}$ if and only if S contains all of the descendants of any $i\in S$. An equivalent definition is that for any arc $(i,j)\in A$, $i\in S$ implies $j\in S$. It is customary to assume that A is acyclic as any (directed) cycle of A can be contracted (see Section 15.3.1).

We let $R(i)$ denote the set of descendants of i (including i itself) in D and let $Q(i)$ denote the set of ancestors of i (including i itself). To perform SFM restricted to a ring family we can encode an arc $(i,j)\in A$ by the constraint $x_i\leq x_j$ as shown by the next lemma.

Lemma 73. *Let $\mathcal{F}$ be a ring family over V and $D=(V,A)$ be its directed acyclic graph representation. Suppose $f:V\longrightarrow\mathbb{R}$ is submodular with Lovasz extension $\hat{f}$. Then the characteristic vector I_S of any minimizer $S=\arg\min_{S\in\mathcal{F}}f(S)$ over $\mathcal{F}$ is also the solution to*

$$\begin{aligned}&\min\hat{f}(\vec{x})\\&\quad x_i\leq x_j\,\forall(i,j)\in A\\&\quad 0\leq\vec{x}\leq 1\end{aligned}\tag{15.1}$$

Proof. Let x^* be a minimizer, and $L^{(t)}=\{i:x_i^*\geq t\}$. It is easy to check that the indicator variable $I_{L^{(t)}}$ satisfies (15.1) since x^* does. Moreover, recall that $\hat{f}(x^*)=\mathbb{E}_{t\sim[0,1]}[f(L_t)]$. Thus $\hat{f}(x^*)$ can be written as a convex combination $\hat{f}(x^*)=\sum_t\alpha_t f(L^{(t)})=\sum_t\alpha_t\hat{f}(I_{L^{(t)}})$, where $\alpha_t>0$ and $\sum_t\alpha_t=1$. Thus all such $\hat{f}(I_{L^{(t)}})$ are minimal, i.e. (15.1) has no "integrality gap". □

We also modify our separation oracle to accommodate for this generalization as follows. Before doing so we need a definition which relates our BFS to the ring family formalism.

Definition 74. A permutation $(v_1, \ldots, v_n)$ of V is said to be *consistent* with an arc (i,j) if j precedes i in $(v_1, \ldots, v_n)$. Similarly, a BFS of the base polyhedron is consistent with (i,j) if j precedes i in its defining permutation. $(v_1, \ldots, v_n)$ (or a BFS) is consistent with A if it is consistent with every $(i,j) \in A$.

Readers may find it helpful to keep in mind the following picture which depicts the relative positions between $R(i), i, Q(i)$ in the defining permutation of $\vec{h}$ that is consistent with A:

$$\cdots\cdots\ R(i)\backslash\{i\}\ \cdots\cdots\ i\ \cdots\cdots\ Q(i)\backslash\{i\}\ \cdots\cdots$$

In Theorem 61, given $\bar{x} \in [0,1]^n$ our separating hyperplane is constructed by sorting the entries of $\bar{x}$. This hyperplane is associated with some BFS $\vec{h}$ of the base polyhedron. As we shall see towards the end of the section, we would like $\vec{h}$ to be consistent with every arc $(i,j) \in A$.

This task is easy initially as $\bar{x}$ satisfies $x_i \leq x_j$ for $(i,j) \in A$ for the starting polytope of (15.1). If $x_i < x_j$, nothing special has to be done as j must precede i in the ordering. On the other hand, whenever $x_i = x_j$, we can always break ties by ranking j ahead of i.

However, a technical issue arises due to the fact that our cutting plane algorithm may drop constraints from the current feasible region P. In other words, $\bar{x}$ may violate $x_i \geq 0$, $x_j \leq 1$ or $x_i \leq x_j$ if it is ever dropped. Fortunately this can be fixed by reintroducing the constraint. We summarize the modification needed in the pseudocode below and formally show that it fulfills our requirement.

Algorithm 5: Modified Separation Oracle

```
Input: $\bar{x} \in \mathbb{R}^n$ and the set of arcs $A$
if $\bar{x}_i < 0$ for some $i$ then
    Output: $x_i \geq 0$
else if $\bar{x}_j > 1$ for some $j$ then
    Output: $x_j \leq 1$
else if $\bar{x}_i > \bar{x}_j$ for some $(i,j) \in A$ then
    Output: $x_i \leq x_j$
else
    Let $i_1, \ldots, i_n$ be a permutation of $V$ such that $\bar{x}_{i_1} \geq \ldots \geq \bar{x}_{i_n}$and for all
      $(i,j) \in A$, $j$ precedes $i$ in $i_1, \ldots, i_n$.
    Output: $\vec{h}^T \vec{x} \leq \hat{f}(\bar{x})$, where $\vec{h}$ is the BFS defined by the permutation $i_1, \ldots, i_n$.
```

Lemma 75. *Our modified separation oracle returns either some BFS $\vec{h} = 0$ or a valid separating hyperplane, i.e.*

1. *$\bar{x}$ either lies on the separating hyperplane or is cut away by it.*

2. *Any minimizer of (15.1) is not cut away by the separating hyperplane.*

Such a hyperplane can be computed with n oracle calls to f and in time $O(n \cdot EO + n^2)$.

Proof. If we get $x_i \geq 0$, $x_j \leq 1$ or $x_i \leq x_j$ (if loop or the first two else loops), then clearly $\bar{x}$ is cut away by it and any minimizer must of course satisfy $x_i \geq 0$, $x_j \leq 1$ and $x_i \leq x_j$ as they are the constraints in (15.1). This proves (1) and (2) for the case of getting $x_i \geq 0$, $x_j \leq 1$ or $x_i \leq x_j$.

Thus it remains to consider the case $\vec{h}^T\vec{x} \leq \hat{f}(\bar{x})$ (last else loop). First of all, $\bar{x}$ lies on it as $\hat{f}(\bar{x}) = \vec{h}^T\bar{x}$. This proves (1). For (2), we have from Lemma 63 that $\vec{h}^T\vec{x} \leq \hat{f}(\vec{x})$. If x^* is a minimizer of (15.1), we must then have $\vec{h}^T x^* \leq \hat{f}(x^*) \leq \hat{f}(\bar{x})$ as $\bar{x}$ is also feasible for (15.1).

Finally we note that the running time is self-evident. □

We stress again that the main purpose of modifying our separation oracle is to ensure that any BFS $\vec{h}$ used to define a new separating hyperplane must be consistent with every $(i, j) \in A$.

15.2.2 Identifying New Valid Arcs

The reason for considering the ring family generalization of SFM is that our algorithms (and some previous algorithms too) work by adding new arcs to our digraph D. This operation yields a strongly polynomial algorithm since there are only $2 \cdot \binom{n}{2}$ possible arcs to add. Of course, a new arc (i, j) is valid only if $i \in S_{\min} \implies j \in S_{\min}$ for some minimizer $S_{\min}$. Here we show how to identify such valid arcs by extracting information from certain nice elements of the base polyhedron.

This is guaranteed by the next four lemmas, which are stated in a way different from previous works e.g. our version is extended to the ring family setting. This is necessary as our algorithms require a more general formulation. We also give a new polyhedral proof, which is mildly simpler than the previous combinatorial proof. On the other hand, Lemma 80 is new and unique to our work. It is an important ingredient of our $\widetilde{O}(n^3 \cdot \mathrm{EO} + n^4)$ time algorithm.

Recall that each BFS of the base polyhedron is defined by some permutation of the ground set elements.

First, we prove the following two lemmas which show that should we ever encounter a non-degenerate point in the base polytope with a coordinate of very large value, then we can immediately conclude that that coordinate must be or must not be in solution to SFM over the ring family.

Lemma 76. *If $\vec{y} \in \mathcal{B}(f)$ is non-degenerate and satisfies $y_i > -(n-1)\min_j y_j$, then i is not in any minimizer of f (over the ring family A).*

Proof. We proceed by contradiction and suppose that S is a minimizer of f that contains i. Now since $\vec{y}$ is non-degenerate we know that $\min_j y_j \leq 0$ and by the definition of $\vec{y}$ we have the following contradiction

$$0 < y_i + (n-1)\min_j y_j \leq \sum_{j \in S} y_j = \vec{y}(S) \leq f(S) \leq f(\varnothing) = 0\,.$$

□

Lemma 77. *If $\vec{y} \in \mathcal{B}(f)$ is non-degenerate and satisfies $y_i < -(n-1)\max_j y_j$, then i is in every minimizer of f (over the ring family A).*

Proof. We proceed by contradiction and suppose that S is a minimizer of f that does not contain i. Now since $\vec{y}$ is non-degenerate we know that $\max_j y_j \geq 0$ and therefore

$$\sum_{j\in[n]} y_j = y_i + \sum_{j\in S} y_j + \sum_{j\in V-(S+i)} y_j < -(n-1)\max_j y_j + \sum_{j\in S} y_j + (|V|-|S|-1)\max_j y_j \leq \sum_{j\in S} y_j \,.$$

However by the definition of $\vec{y}$ we have

$$\sum_{j\in S} y_j = \vec{y}(S) \leq f(S) \leq f(V) = \sum_{j\in[n]} y_j \,.$$

Thus we have a contradiction and the result follows. $\square$

Now we are ready to present conditions under which a new valid arc can be added. We begin with a simple observation. Let $\texttt{upper}(i) \stackrel{\text{def}}{=} f(R(i)) - f(R(i)-i)$ and $\texttt{lower}(i) \stackrel{\text{def}}{=} f(V\backslash Q(i)+i) - f(V\backslash Q(i))$. As the names suggest, they bound the value of h_i for any BFS used.

Lemma 78. *For any BFS $\vec{h}$ used to construct a separating hyperplane given by our modified separation oracle, we have $\texttt{lower}(i) \leq h_i \leq \texttt{upper}(i)$.*

Proof. Note that by Lemma 75, $\vec{h}$ is consistent with every $(j_1, j_2) \in A$ and hence i must precede $Q(i)$ and be preceded by $R(i)$. Let S be the set of elements preceding i in the defining permutation of $\vec{h}$. Then $h_i = f(S+i) - f(S) \leq f(R(i)) - f(R(i)-i)$ because of diminishing return and $R(i) - i \subseteq S$. The lower bound follows from the same argument as $Q(i) - i$ comes after i, and so $Q(i) \subseteq V\backslash S$. $\square$

In the following two lemmas, we show that if $\texttt{upper}(i)$ is ever sufficiently positive or $\texttt{lower}(i)$ is sufficiently negative, then we find a new arc.

While these lemmas may appear somewhat technical but actually has an intuitive interpretation. Suppose an element p is in a minimizer $S_{\min}$ of f over the ring family D. Then $R(p)$ must also be part of $S_{\min}$. Now if $f(R(p))$ is very large relative to $f(R(p)-p)$, there should be some element $q \in S_{\min}\backslash R(p)$ compensating for the discrepancy. The lemma says that such an element q can in fact be found efficiently.

Lemma 79 (new arc)**.** *Let $\vec{y} = \sum_k \lambda^{(k)}\vec{y}^{(k)}$ be a non-degenerate convex combination of $O(n)$ base polyhedron BFS's $\vec{y}^{(k)}$ which are consistent with every arc $(i,j) \in A$. If some element p satisfies $\texttt{upper}(p) > n^4 \max y_j$, then we can find, using $O(n \cdot EO)$ oracle calls and $O(n^2)$ time, some $q \notin R(p)$ such that the arc (p,q) is valid, i.e. if p is in a minimizer, then so is q.*

Proof. If $\max y_j < 0$ then we are immediately done by Lemma 65. We assume $\max y_j \geq 0$ in the proof. For all k let $\vec{y}'^{(k)}$ be the BFS obtained by taking the defining permutation of $\vec{y}^{(k)}$ and moving $R(p)$ to the front while preserving the relative ordering of $R(p)$ within each permutation). Furthermore, let $\vec{y}' \stackrel{\text{def}}{=} \sum_k \lambda^{(k)}\vec{y}'^{(k)}$. Then since $y_p'^{(k)} = f(R(p)) - f(R(p)-p) = \texttt{upper}(p)$ we have $\texttt{upper}(p) = y_p' = f(R(p)) - f(R(p)-p)$. Moreover,

$$y_j' \geq y_j \quad \forall j \in R(p) \text{ and } y_j' \leq y_j \quad \forall j \notin R(p) \tag{15.2}$$

by diminishing marginal return.

Now, suppose p is in a minimizer $S_{\min}$. Then $R(p) \subseteq S_{\min}$ by definition. We then define $f'(S) = f(S \cup R(p))$ for $S \subseteq V \backslash R(p)$. It can be checked readily that f' is submodular and $S_{\min} \backslash R(p)$ is a minimizer of f' (over the corresponding ring family). Note that now $\vec{y}'_{V \backslash R(p)}$ (the restriction of $\vec{y}'$ to $V \backslash R(p)$) is a convex combination of the BFS's of the base polyhedron $\mathcal{B}(f')$ of f'. We shall show that $\vec{y}'_{V \backslash R(p)}$ has the desired property in Lemma 77.

Note that $y'(V \backslash R(p) + p) \leq y(V \backslash R(p) + p)$ since

$$y'(V \backslash R(p)+p) = y'(V) - y'(R(p)-p) = y(V) - y'(R(p)-p) \leq y(V) - y(R(p)-p) = y(V \backslash R(p)+p).$$

But now since $\vec{y}$ is non-degenerate $\max_j y_j \geq 0$ and therefore

$$\begin{aligned} y'(V \backslash R(p)) &\leq y(V \backslash R(p) + p) - y'_p \\ &= y(V \backslash R(p) + p) - (f(R(p)) - f(R(p) - p)) \\ &\leq n \max y_j - (f(R(p)) - f(R(p) - p)) \\ &< (n - n^4) \max y_j \end{aligned} \tag{15.3}$$

Therefore by the Pigeonhole Principle some $q \notin R(p)$ must satisfy

$$\begin{aligned} y'_q &< \left((n - n^4) \max y_j\right) / (n-1) \\ &= -(n^3 + n^2 + n) \max y_j \\ &\leq -(n^3 + n^2 + n) \max_{j \notin R(p)} y_j \\ &\leq -(n^3 + n^2 + n) \max_{j \notin R(p)} y'_j \quad \text{by (15.2)} \end{aligned}$$

By Lemma 77, this q must be in any minimizer of f'. In other words, whenever p is in a minimizer of f, then so is q.

Note however that computing all $\vec{y}'$ would take $O(n^2)$ oracle calls in the worst case as there are $O(n)$ $\vec{y}'^{(k)}$'s. We use the following trick to identify some q with $y'_q < -(n-1) \max y_j$ using just $O(n)$ calls. The idea is that we actually only want to have sufficient decreases in $y'(V \backslash R(p))$ which can be accomplished by having a large corresponding decrease in some $\vec{y}'^{(k)}$.

For each k, by the same argument above (see (15.3))

$$y'^{(k)}(V \backslash R(p)) - y^{(k)}(V \backslash R(p)) \leq y_p^{(k)} - (f(R(p)) - f(R(p) - p)) \tag{15.4}$$

The "weighted decrease" $\lambda^{(k)} \left(y_p^{(k)} - (f(R(p)) - f(R(p) - p))\right)$ for $\vec{y}'^{(k)}$ sum up to

$$\sum \lambda^{(k)} \left(y_p^{(k)} - (f(R(p)) - f(R(p) - p))\right) = y_p - (f(R(p)) - f(R(p) - p)) < (1 - n^4) \max y_j$$

Thus by the Pigeonhole Principle, some l will have

$$\lambda^{(l)} \left(y_p^{(l)} - (f(R(p)) - f(R(p) - p))\right) < \left((1 - n^4) \max y_j\right) / O(n) < -n^2 \max y_j.$$

For this $\vec{y}^{(l)}$ we compute $\vec{y}'^{(l)}$. We show that $\vec{y}'' = \lambda^{(l)}\vec{y}'^{(l)} + \sum_{k\neq l}\lambda^{(k)}\vec{y}^{(k)}$ has the same property as $\vec{y}'$ above.

$$\begin{aligned}
y''(V\backslash R(p)) &= \lambda^{(l)}y'^{(l)}(V\backslash R(p)) + \sum_{k\neq l}\lambda^{(k)}y^{(k)}(V\backslash R(p)) \\
&= y(V\backslash R(p)) + \lambda^{(l)}\left(y'^{(l)}(V\backslash R(p)) - y^{(l)}(V\backslash R(p))\right) \\
&\leq y(V\backslash R(p)) + \lambda^{(l)}\left(y_p^{(l)} - (f(R(p)) - f(R(p)-p))\right) \quad \text{by (15.4)} \\
&< (n-1)\max y_j - n^2 \max y_j \\
&< (n-n^2)\max y_j
\end{aligned}$$

Then some $q \in V\backslash R(p)$ must satisfy

$$y''_q < \frac{n-n^2}{n-1}\max y_j = -n\max y_j$$

That is, the arc (p,q) is valid. This takes $O(n)$ oracle calls as given $\vec{y} = \sum_k \lambda^{(k)}\vec{y}^{(k)}$, computing $\vec{y}''$ requires knowing only $f(R(p))$, $f(R(p)-p)$, and $\vec{y}'^{(l)}$ which can be computed from $\vec{y}^{(l)}$ with n oracle calls. The runtime is $O(n^2)$ which is needed for computing $\vec{y}''$. □

Lemma 80. *Let $\vec{y} = \sum_k \lambda^{(k)}\vec{y}^{(k)}$ be a non-degenerate convex combination of base polyhedron BFS $\vec{y}^{(k)}$ which is consistent with every arc $(i,j) \in A$. If* `lower`$(p) < n^4 \min y_j$*, then we can find, using $O(n\cdot EO)$ oracle calls and $O(n^2)$ time, some $q \notin Q(p)$ such that the arc (q,p) is valid, i.e. if p is not in a minimizer, then q is not either.*

Proof. It is possible to follow the same recipe in the proof of Lemma 79 but using Lemma 76 instead of Lemma 77. Here we offer a proof which directly invokes Lemma 77 on a different submodular function.

Let g be defined by $g(S) \overset{\text{def}}{=} f(V\backslash S)$ for any S, and A_g be the set of arcs obtained by reversing the directions of the arcs of A. Consider the problem of minimizing g over the ring family A_g. Using subscripts to avoid confusion with f and g, e.g. $R_g(i)$ is the set of descendants of i w.r.t. A_g, it is not hard to verify the following:

- g is submodular
- $R_g(i) = Q_f(i)$
- $g(R_g(p)) - g(R_g(p)-p) = -\left(f(V\backslash Q_f(p)+p) - f(V\backslash Q_f(p))\right)$
- $-\vec{y}^{(k)}$ is a BFS of $\mathcal{B}(g)$ if and only if $\vec{y}^{(k)}$ is a BFS of $\mathcal{B}(f)$
- $\max(-y_j) = -\min y_j$

By using the above correspondence and applying Lemma 79 to g and A_g, we can find, using $O(n)$ oracle calls and $O(n^2)$ time, some $q \notin R_g(p) = Q(p)$ such that the arc (p,q) is valid for g and A_g. In other words, the reverse (q,p) will be valid for f and A. □

These lemmas lay the foundation of our algorithm. They suggests that if the positive entries of a point in the base polyhedron are small relative to some $\texttt{upper}(p) = f(R(p)) - f(R(p) - p)$, a new arc (p, q) can be added to A. This can be seen as a robust version of Lemma 65.

Finally, we end the section with a technical lemma that will be used crucially for both of our algorithms. The importance of it would become obvious when it is invoked in our analyses.

Lemma 81. *Let $\vec{h}''$ denote a convex combination of two vectors $\vec{h}$ and $\vec{h}'$ in the base polyhedron, i.e. $\vec{h}'' = \lambda\vec{h} + (1-\lambda)\vec{h}'$ for some $\lambda \in [0,1]$. Further suppose that*

$$\left\|\vec{h}''\right\|_2 \le \alpha \min\left\{\lambda\left\|\vec{h}\right\|_2, (1-\lambda)\left\|\vec{h}'\right\|_2\right\}$$

for some $\alpha \le \frac{1}{2\sqrt{n}}$. Then for $p = \arg\max_j(\max\{\lambda|h_j|, (1-\lambda)|h'_j|\})$ we have

$$\texttt{lower}(p) \le -\frac{1}{2\alpha\sqrt{n}} \cdot \left\|\vec{h}''\right\|_\infty \quad \text{and} \quad \texttt{upper}(p) \ge \frac{1}{2\alpha\sqrt{n}} \cdot \left\|\vec{h}''\right\|_\infty \quad .$$

Proof. Suppose without loss of generality that $\lambda|h_p| \ge (1-\lambda)|h'_p|$. Then by assumptions we have

$$\left\|\vec{h}''\right\|_\infty \le \left\|\vec{h}''\right\|_2 \le \alpha \cdot \min\left\{\lambda\left\|\vec{h}\right\|_2, (1-\lambda)\left\|\vec{h}'\right\|_2\right\} \le \alpha\sqrt{n}\,|\lambda h_p| \quad .$$

However, since $\alpha \le \frac{1}{2\sqrt{n}}$ we see that

$$\left|\lambda h_p + (1-\lambda)h'_p\right| \le \left\|h''\right\|_\infty \le \alpha\sqrt{n}\,|\lambda h_p| \le \frac{1}{2}\,|\lambda h_p| \quad .$$

Consequently, λh_p and $(1-\lambda)h'_p$ have opposite signs and $\left|(1-\lambda)h'_p\right| \ge \frac{1}{2}\left|\lambda h'_p\right|$. We then have,

$$\texttt{lower}(p) \le \min\left\{h_p, h'_p\right\} \le \min\left\{\lambda h_p, (1-\lambda)h'_p\right\} \le -\frac{1}{2}\,|\lambda h_p| \le -\frac{1}{2\alpha\sqrt{n}}\left\|h''\right\|_\infty$$

and

$$\texttt{upper}(p) \ge \max\left\{h_p, h'_p\right\} \ge \max\left\{\lambda h_p, (1-\lambda)h'_p\right\} \ge \frac{1}{2}\,|\lambda h_p| \ge \frac{1}{2\alpha\sqrt{n}}\left\|h''\right\|_\infty .$$

□

15.3 $\widetilde{O}(n^4 \cdot \mathbf{EO} + n^5)$ Time Algorithm

Here we present a $\widetilde{O}(n^4 \cdot \text{EO} + n^5)$ time, i.e. strongly polynomial time algorithm, for SFM. We build upon this algorithm to achieve a faster running time in Section 15.4.

Our new algorithm combines existing tools for SFM developed over the last decade with our cutting plane method and some new extensions of the tools. While there are certain similarities with previous algorithms (especially [126, 133, 128]), our approach significantly departs from all the old approaches in one important aspect.

All of the previous algorithms actively maintain a point in the base polyhedron and represent it as a *convex combination* of BFS's. At each step, a new BFS may enter the convex combination and an old BFS may exit. Our algorithm, on the other hand, maintains only a *collection* of BFS's (corresponding to our separating hyperplanes), rather than an explicit convex combination. A "good" convex combination is computed from the *collection* of BFS's only after running Cutting Plane for enough iterations. We believe that this crucial difference is the fundamental reason which offers the speedup. This is achieved by the Cutting Plane method which considers the *geometry* of the collection of BFS's. On the other hand, considering only a convex combination of BFS's effectively narrows our sight to only one *point* in the base polyhedron.

Overview

Now we are ready to describe our strongly polynomial time algorithm. Similar to the weakly polynomial algorithm, we first run our cutting plane for enough iterations on the initial feasible region $\{\vec{x} \in [0,1]^n : x_i \leq x_j \,\forall (i,j) \in A\}$, after which a pair of approximately parallel supporting hyperplanes F_1, F_2 of width $1/n^{\Theta(1)}$ can be found. Our strategy is to write F_1 and F_2 as a nonnegative combination of the facets of remaining feasible region P. This combination is made up of newly added separating hyperplanes as well as the inequalities $x_i \geq 0$, $x_j \leq 1$ and $x_i \leq x_j$. We then argue that one of the following updates can be done:

- Collapsing: $x_i = 0$, $x_j = 1$ or $x_i = x_j$
- Adding a new arc (i,j): $x_i \leq x_j$ for some $(i,j) \notin A$

The former case is easy to handle by elimination or contraction. If $x_i = 0$, we simply eliminate i from the ground set V; and if $x_i = 1$, we redefine f so that $f(S) = f(S+i)$ for any $S \subseteq V - i$. $x_i = x_j$ can be handled in a similar fashion. In the latter case, we simply add the arc (i,j) to A. We then repeat the same procedure on the new problem.

Roughly speaking, our strongly polynomial time guarantee follows as eliminations and contractions can happen at most n times and at most $2 \cdot \binom{n}{2}$ new arcs can be added. While the whole picture is simple, numerous technical details come into play in the execution. We advise readers to keep this overview in mind when reading the subsequent sections.

Algorithm

Our algorithm is summarized below. Again, we remark that our algorithm simply uses Theorem 82 regarding our cutting plane and is agnostic as to how the cutting plane works, thus it could be replaced with other methods, albeit at the expense of slower runtime.

1. Run cutting plane on (15.1) (Theorem 82 with $\tau = \Theta(1)$) using our modified separation oracle (Section 15.2.1).
2. Identify a pair of "narrow" approximately parallel supporting hyperplanes or get some BFS $\vec{h} = 0$ (in which case both $\varnothing$ and V are minimizers).
3. Deduce from the hyperplanes some new constraint of the forms $x_i = 0, x_j = 1, x_i = x_j$ or $x_i \leq x_j$ (Section 15.3.2).

4. Consolidate A and f (Section 15.3.1).

5. Repeat by running our cutting plane method on (15.1) with updated A and f. (Note that Any previously found separating hyperplanes are discarded.)

We call step (1) a *phase* of cutting plane. The minimizer can be constructed by unraveling the recursion.

15.3.1 Consolidating A and f

Here we detail how the set of valid arcs A and submodular function f should be updated once we deduce new information $x_i = 0, x_i = 1, x_i = x_j$ or $x_i \le x_j$. Recall that $R(i)$ and $Q(i)$ are the sets of descendants and ancestors of i respectively (including i itself). The changes below are somewhat self-evident, and are actually used in some of the previous algorithms so we only sketch how they are done without a detailed justification.

Changes to the digraph representation D of our ring family include:

- $x_i = 0$: remove $Q(i)$ from the ground set and all the arcs incident to $Q(i)$
- $x_i = 1$: remove $R(i)$ from the ground set and all the arcs incident to $R(i)$
- $x_i = x_j$: contract i and j in D and remove any duplicate arcs
- $x_i \le x_j$: insert the arc (i, j) to A
- For the last two cases, we also contract the vertices on a directed cycle of A until there is no more. Remove any duplicate arcs.

Here we can contract any cycle $(i_1, \ldots, i_k)$ because the inequalities $x_{i_1} \le x_{i_2}, \ldots, x_{i_{k-1}} \le x_{i_k}, x_{i_k} \le x_{i_1}$ imply $x_{i_1} = \ldots = x_{i_k}$.

Changes to f:

- $x_i = 0$: replace f by $f' : 2^{V \backslash Q(i)} \longrightarrow \mathbb{R}$, $f'(S) = f(S)$ for $S \subseteq V \backslash Q(i)$
- $x_i = 1$: replace f by $f' : 2^{V \backslash R(i)} \longrightarrow \mathbb{R}$, $f'(S) = f(S \cup R(i))$ for $S \subseteq V \backslash R(i)$
- $x_i = x_j$: see below
- $x_i \le x_j$: no changes to f needed if it does not create a cycle in A; otherwise see below
- Contraction of $C = \{i_1, \ldots, i_k\}$: replace f by $f' : 2^{V \backslash C + l} \longrightarrow \mathbb{R}$, $f'(S) = f(S)$ for $S \subseteq V \backslash C$ and $f'(S) = f((S - l) \cup C)$ for $S \ni l$

Strictly speaking, these changes are in fact *not* needed as they will automatically be taken care of by our cutting plane method. Nevertheless, performing them lends a more natural formulation of the algorithm and simplifies its description.

15.3.2 Deducing New Constraints $x_i = 0$, $x_j = 1$, $x_i = x_j$ or $x_i \leq x_j$

Here we show how to deduce new constraints through the result of our cutting plane method. This is the most important ingredient of our algorithm. As mentioned before, similar arguments were used first by IFF [128] and later in [126, 133]. There are however two important differences for our method:

- We maintain a collection of BFS's rather a convex combination; a convex combination is computed and needed only after each phase of cutting plane.
- As a result, our results are proved mostly *geometrically* whereas the previous ones were proved mostly *combinatorially.*

Our ability to deduce such information hinges on the power of the cutting plane method in Part I. We re-state our main result Theorem 31 in the language of SFM. Note that Theorem 82 is formulated in a fairly general manner in order to accommodate for the next section. Readers may wish to think $\tau = \Theta(1)$ for now.

Theorem 82 (Theorem 31 restated for SFM)**.** *For any $\tau \geq 100$, applying our cutting plane method, Theorem 82, to (15.1) with our modified separation oracle (or its variant in Section 15.4) with high probability in n either*

1. *Finds a degenerate BFS $\vec{h} \geq \vec{0}$ or $\vec{h} \leq \vec{0}$.*
2. *Finds a polytope P consisting of $O(n)$ constraints which are our separating hyperplanes or the constraints in (15.1). Moreover, P satisfies the following inequalities*
$$\vec{c}^T\vec{x} \leq M \quad \text{and} \quad \vec{c}'^T\vec{x} \leq M',$$
both of which are nonnegative combinations of the constraints of P, where $||\vec{c}+\vec{c}'||_2 \leq \min\{||\vec{c}||_2, ||\vec{c}'||_2\}/n^{\Theta(\tau)}$ and $|M+M'| \leq \min\{||\vec{c}||_2, ||\vec{c}'||_2\}/n^{\Theta(\tau)}$.

Furthermore, the algorithm runs in expected time $O(n^2\tau\log n \cdot EO + n^3\tau^{O(1)}\log^{O(1)} n)$.

Proof. In applying Theorem 82 we let K be the set of minimizers of f over the ring family and the box is the hypercube with $R = 1$. We run cutting plane with our modified separation oracle (Lemma 75). The initial polytope $P^{(0)}$ can be chosen to be, say, the hypercube. If some separating hyperplane is degenerate, then we have the desired result (and know that either $\varnothing$ or V is optimal). Otherwise let P be the current feasible region. Note that $P \neq \varnothing$, because our minimizers of $\hat{f}$ are all in $P^{(0)}$ and $P^{(k)}$ as they are never cut away by the separating hyperplanes.

Let $\mathcal{S}$ be the collection of inequalities (15.1) as well as the separating hyperplanes $\vec{h}^T\vec{x} \leq \hat{f}(\bar{x}_h) = \vec{h}^T\bar{x}_h$ used. By Theorem 31, all of our minimizers will be contained in P, consisting of $O(n)$ constraints $\mathbf{A}\vec{x} \geq \vec{b}$. Each such constraint $\vec{a}_i^T\vec{x} \geq b_i$ is a scaling and shifting of some inequality $\vec{p}_i^T\vec{x} \geq q_i$ in $\mathcal{S}$, i.e. $\vec{a}_i = \vec{p}_i/||\vec{p}_i||_2$ and $b_i \leq q_i/||\vec{p}_i||_2$.

By taking $\epsilon = 1/n^{\Theta(\tau)}$ with sufficiently large constant in Θ, our theorem certifies that P has a narrow width by $\vec{a}_1$, some nonnegative combination $\sum_{i=2}^{O(n)} t_i \vec{a}_i$ and point $\vec{x}_o \in P$ with $||\vec{x}_o||_\infty \leq 3\sqrt{n}R = 3\sqrt{n}$ satisying the following:

$$\left\| \vec{a}_1 + \sum_{i=2}^{O(n)} t_i \vec{a}_i \right\|_2 \leq 1/n^{\Theta(\tau)}$$

$$0 \leq \vec{a}_1^T \vec{x}_o - \vec{b}_1 \leq 1/n^{\Theta(\tau)}$$

$$0 \leq \left(\sum_{i=2}^{O(n)} t_i a_i \right)^T \vec{x}_o - \sum_{i=2}^{O(n)} t_i b_i \leq 1/n^{\Theta(\tau)}$$

We convert these inequalities to $\vec{p}$ and q. Let $t_i' \stackrel{\text{def}}{=} t_i \cdot ||\vec{p}_1||_2 / ||\vec{p}_i||_2 \geq 0$.

$$\left\| \vec{p}_1 + \sum_{i=2}^{O(n)} t_i' \vec{p}_i \right\|_2 \leq ||\vec{p}_1||_2 / n^{\Theta(\tau)}$$

$$0 \leq \vec{p}_1^T \vec{x}_o - q_1 \leq ||\vec{p}_1||_2 / n^{\Theta(\tau)}$$

$$0 \leq \left(\sum_{i=2}^{O(n)} t_i' \vec{p}_i \right)^T \vec{x}_o - \sum_{i=2}^{O(n)} t_i' q_i \leq ||\vec{p}_1||_2 / n^{\Theta(\tau)}$$

We claim that[4] $\vec{c} = -\vec{p}_1$, $M = -q_1$, $\vec{c}' = -\sum_{i=2}^{O(n)} t_i' \vec{p}_i$, $M' = -\sum_{i=2}^{O(n)} t_i' q_i$ satisfy our requirement.

We first show that $||\vec{c} + \vec{c}'||_2 \leq \min\{||\vec{c}||_2, ||\vec{c}'||_2\}/n^{\Theta(\tau)}$. We have $||\vec{c} + \vec{c}'||_2 \leq ||\vec{c}||_2 / n^{\Theta(\tau)}$ from the first inequality. If $||\vec{c}||_2 \leq ||\vec{c}'||_2$ we are done. Otherwise, by triangle inequality

$$||\vec{c}'||_2 - ||\vec{c}||_2 \leq ||\vec{c} + \vec{c}'||_2 \leq ||\vec{c}||_2 / n^{\Theta(\tau)} \implies 2||\vec{c}||_2 \geq ||\vec{c}'||_2$$

and hence $||\vec{c} + \vec{c}'||_2 \leq ||\vec{c}||_2 / n^{\Theta(\tau)} \leq ||\vec{c}'||_2 / 2n^{\Theta(\tau)} = ||\vec{c}'||_2 / n^{\Theta(\tau)}$.

We also need to prove $|M + M'| \leq \min\{||\vec{c}||_2, ||\vec{c}'||_2\}/n^{\Theta(\tau)}$. Summing the second and third inequalities,

$$-||\vec{c}||_2 / n^{\Theta(\tau)} \leq (\vec{c} + \vec{c}')^T \vec{x}_o - (M + M') \leq 0$$

Recall that we have $||\vec{x}_o||_\infty \leq 3\sqrt{n}$. Then

$$\begin{aligned} |M + M'| &\leq |(\vec{c} + \vec{c}')^T \vec{x}_o - (M + M')| + |(\vec{c} + \vec{c}')^T \vec{x}_o| \\ &\leq ||\vec{c}||_2 / n^{\Theta(\tau)} + 3\sqrt{n}||\vec{c} + \vec{c}'||_2 \\ &\leq ||\vec{c}||_2 / n^{\Theta(\tau)} + 3\sqrt{n}||\vec{c}||_2 / n^{\Theta(\tau)} \\ &= ||\vec{c}||_2 / n^{\Theta(\tau)} \end{aligned}$$

as desired. Our result then follows as we proved $2||\vec{c}'||_2 \geq ||\vec{c}||_2$.

Finally, we have the desired runtime as our modified separation oracle runs in time $O(n \cdot \text{EO} + n^2 \log^{O(1)} n)$. $\square$

[4]Minus signs is needed because we express our inequalities as e.g. $\vec{h}^T \vec{x} \leq \vec{h}^T \vec{x}_h$ whereas in Theorem 31, $\vec{a}_i^T \vec{x} \geq b_i$ is used. We apologize for the inconvenience.

Informally, the theorem above simply states that after $O(n\tau \log n)$ iterations of cutting plane, the remaining feasible region P can be sandwiched between two approximately parallel supporting hyperplanes of width $1/n^{O(\tau)}$. A good intuition to keep in mind is that every $O(n)$ iterations of cutting plane reduces the minimum width by a constant factor.

Remark 83. As shown in the proof of Theorem 82, one of the two approximately parallel hyperplanes can actually be chosen to be a constraint of our feasible region P. However we do not exploit this property as it does not seem to help us and would break the notational symmetry in $\vec{c}$ and $\vec{c}'$.

Setup

In each phase, we run cutting plane using Theorem 82 with $\tau = \Theta(1)$. If some separating hyperplane used is degenerate, we have found the minimizer by Lemma 65.

Now assume none of the separating hyperplanes are degenerate. By Theorem 82, P is sandwiched by a pair of approximately parallel supporting hyperplanes F, F' which are of width $1/10n^{10}$ apart. The width here can actually be $1/n^c$ for any constant c by taking a sufficiently large constant in Theta.

Here, we show how to deduce from F and F' some $x_i = 0$, $x_j = 1$, $x_i = x_j$, or $x_i \leq x_j$ constraint on the minimizers of f over the ring family. Let

$$\vec{c}^T\vec{x} = \sum c_i x_i \leq M \quad \text{and} \quad \vec{c}'^T\vec{x} = \sum c'_i x_i \leq M'$$

be the inequality for F and F' such that

$$|M + M'|,\ ||\vec{c} + \vec{c}'||_2 \leq \mathsf{gap}, \quad \text{where } \mathsf{gap} \overset{\text{def}}{=} \frac{1}{10n^{10}} \min\{||\vec{c}||_2, ||\vec{c}'||_2\}.$$

By the same theorem we can write $\vec{c}^T\vec{x} \leq M$ as a nonnegative combination of the constraints for P. Recall that the constraints for P take on four different forms: (1) $-x_i \leq 0$; (2) $x_j \leq 1$; (3) $-(x_j - x_i) \leq 0$; (4) $\vec{h}^T\vec{x} = \sum h_i x_i \leq \hat{f}(\bar{x}_h)$. Here the first three types are present initially whereas the last type is the separating hyperplane added. As alleged previously, the coefficient vector $\vec{h}$ corresponds to a BFS of the base polyhedron for f. Our analysis crucially exploits this property.

Thus suppose $\vec{c}^T\vec{x} = \sum_i c_i x_i \leq M$ is a nonnegative combination of our constraints with weights $\alpha_i, \beta_j, \gamma_{ij}, \lambda_h \geq 0$. The number of (positive) $\alpha_i, \beta_j, \gamma_{ij}, \lambda_h$ is at most $O(n)$. Here we denote separating hyperplanes by $\vec{h}^T\vec{x} \leq \hat{f}(\bar{x}_h)$. Let H be the set of BFS's used to construct separating hyperplanes.

$$\vec{c}^T\vec{x} = -\sum_i \alpha_i x_i + \sum_j \beta_j x_j + \sum_{(i,j)\in A} \gamma_{ij}(x_i - x_j) + \sum_{h\in H} \lambda_h \vec{h}^T\vec{x} \quad \text{and} \quad M = \sum_j \beta_j + \sum_{h\in H} \lambda_h \hat{f}(\bar{x}_h). \tag{15.5}$$

Similarly, we write the inequality for F' as a nonnegative combination of the constraints for P and the number of (positive) $\alpha'_i, \beta'_j, \gamma'_{ij}, \lambda'_h$ is $O(n)$:

$$\vec{c}'^T\vec{x} = -\sum \alpha'_i x_i + \sum \beta'_j x_j + \sum_{(i,j)\in A} \gamma'_{ij}(x_i - x_j) + \sum_{h\in H} \lambda'_h \vec{h}^T\vec{x} \quad \text{and} \quad M' = \sum \beta'_j + \sum_{h\in H} \lambda'_h \hat{f}(\bar{x}_h). \tag{15.6}$$

We also scale $\vec{c}, \vec{c}', \alpha, \alpha', \beta, \beta', \gamma, \gamma', \lambda, \lambda'$ so that

$$\sum_{h\in H}(\lambda_h + \lambda'_h) = 1$$

as this does not change any of our preceding inequalities regarding F and F'.

Now that F, F' have been written as combinations of our constraints, we have gathered the necessary ingredients to derive our new arc. We first give a geometric intuition why we would expect to be able to derive a new constraint. Consider the nonnegative combination making up F. We think of the coefficient β_j as the contribution of $x_j \le 1$ to F. Now if β_j is very large, F is "very parallel" to $x_j \le 1$ and consequently F' would miss $x_j = 0$ as the gap between F and F' is small. P would then miss $x_j = 0$ too as it is sandwiched between F and F'. Similarly, a large α_i and a large γ_{ij} would respectively imply that $x_i = 1$ and $(x_i = 0, x_j = 1)$ would be missed. The same argument works for F' as well.

But on the other hand, if the contributions from $x_i \ge 0, x_j \le 1, x_i \le x_j$ to both F and F' are small, then the supporting hyperplanes $\vec{c}^T\vec{x} \le ...$ and $\vec{c}'^T\vec{x} \le ...$ would be mostly made up of separating hyperplanes $\vec{h}^T\vec{x} \le \hat{f}(\bar{x}_h)$. By summing up these separating hyperplanes (whose coefficients form BFS's), we would then get a point in the base polyhedron which is very close to the origin 0. Moreover, by Lemma 81 and Lemma 79 we should then be able to deduce some interesting information about the minimizer of f over D.

The rest of this section is devoted to realizing the vision sketched above. We stress that while the algebraic manipulations may be long, they are simply the execution of this elegant geometric picture.

Now, consider the following weighted sum of $\vec{h}^T\vec{x} \le \hat{f}(\bar{x}_h)$:

$$\left(\sum_{h\in H}\lambda_h\vec{h}^T + \sum_{h\in H}\lambda'_h\vec{h}^T\right)\vec{x} = \sum_{h\in H}\lambda_h\vec{h}^T\vec{x} + \sum_{h\in H}\lambda'_h\vec{h}^T\vec{x} \le \sum_{h\in H}\lambda_h\hat{f}(\bar{x}_h) + \sum_{h\in H}\lambda'_h\hat{f}(\bar{x}_h).$$

Observe that $\sum_{h\in H}\lambda_h\vec{h}^T + \sum_{h\in H}\lambda'_h\vec{h}^T$ is in the base polyhedron since it is a convex combination of BFS $\vec{h}$. Furthermore, using (15.5) and (15.6) this can also be written as

$$\begin{aligned}\left(\sum_{h\in H}\lambda_h\vec{h}^T + \sum_{h\in H}\lambda'_h\vec{h}^T\right)\vec{x} &= \left(\vec{c}^T\vec{x} + \sum\alpha_i x_i - \sum\beta_j x_j + \sum_{(i,j)\in A}\gamma_{ij}(x_j - x_i)\right)\\ &\quad + \left(\vec{c}'^T\vec{x} + \sum\alpha'_i x_i - \sum\beta'_j x_j + \sum_{(i,j)\in A}\gamma'_{ij}(x_j - x_i)\right)\end{aligned} \tag{15.7}$$

and

$$\begin{aligned}\sum_{h\in H}\lambda_h\hat{f}(\bar{x}_h) + \sum_{h\in H}\lambda'_h\hat{f}(\bar{x}_h) &= \left(M - \sum\beta_j\right) + \left(M' - \sum\beta'_j\right)\\ &= (M + M') - \sum\beta_j - \sum\beta'_j\end{aligned}$$

Furthermore, we can bound $\vec{c}^T\vec{x} + \vec{c}'^T\vec{x}$ by $\vec{c}^T\vec{x} + \vec{c}'^T\vec{x} \ge -||\vec{c} + \vec{c}'||_1 \ge -\sqrt{n}||\vec{c} + \vec{c}'||_2 \ge -\sqrt{n}\texttt{gap}$ as $\vec{x} \le 1$. Since $M + M' \le \texttt{gap}$, we obtain

$$LHS \overset{\text{def}}{=} \sum\alpha_i x_i + \sum\alpha'_i x_i - \sum\beta_j x_j - \sum\beta'_j x_j + \sum_{(i,j)\in A}\gamma_{ij}(x_j - x_i) + \sum_{(i,j)\in A}\gamma'_{ij}(x_j - x_i)$$

$$\leq 2\sqrt{n}\mathtt{gap} - \sum \beta_j - \sum \beta'_j$$

Geometrically, the next lemma states that if the contribution from, say $x_i \geq 0$, to F is too large, then F' would be forced to miss $x_i = 1$ because they are close to one another.

Lemma 84. *Suppose $\vec{x}$ satisfies (15.1) and $LHS \leq 2\sqrt{n}\mathtt{gap} - \sum \beta_j - \sum \beta'_j$ with $\alpha_i, \beta_j, \gamma_{ij}, \alpha'_i, \beta'_j, \gamma'_{ij} \geq 0$.*

1. *If $\alpha_i > 2\sqrt{n}\mathtt{gap}$ or $\alpha'_i > 2\sqrt{n}\mathtt{gap}$, then $x_i < 1$.*
2. *If $\beta_j > 2\sqrt{n}\mathtt{gap}$ or $\beta'_j > 2\sqrt{n}\mathtt{gap}$, then $x_j > 0$.*
3. *If $\gamma_{ij} > 2\sqrt{n}\mathtt{gap}$ or $\gamma'_{ij} > 2\sqrt{n}\mathtt{gap}$, then $0 \leq x_j - x_i < 1$.*

Proof. We only prove it for $\alpha_i, \beta_j, \gamma_{ij}$ as the other case follows by symmetry.

Using $0 \leq x \leq 1$ and $x_i \leq x_j$ for $(i,j) \in A$, we have $LHS \geq \alpha_i x_i - \sum \beta_j - \sum \beta'_j$. Hence $\alpha_i x_i \leq 2\sqrt{n}\mathtt{gap}$ and we get $x_i < 1$ if $\alpha_i > 2\sqrt{n}\mathtt{gap}$.

Similarly, $LHS \geq -\beta_k x_k - \sum_{j \neq k} \beta_j - \sum \beta'_j$ which gives $-\beta_k x_k \leq 2\sqrt{n}\mathtt{gap} - \beta_k$. Then $x_k > 0$ if $\beta_k > 2\sqrt{n}\mathtt{gap}$.

Finally, $LHS \geq \gamma_{ij}(x_j - x_i) - \sum \beta_j - \sum \beta'_j$ which gives $\gamma_{ij}(x_j - x_i) \leq 2\sqrt{n}\mathtt{gap}$. Then $x_j - x_i < 1$ if $\gamma_{ij} > 2\sqrt{n}\mathtt{gap}$. We have $x_i \leq x_j$ since $(i,j) \in A$. □

So if either condition of Lemma 84 holds, we can set $x_i = 0$ or $x_j = 1$ or $x_i = x_j$ since our problem (15.1) has an integral minimizer and any minimizer of $\hat{f}$ is never cut away by Lemma 75. Consequently, in this case we can reduce the dimension by at least 1. From now on we may assume that

$$\max\{\alpha_i, \alpha'_i, \beta_j, \beta'_j, \gamma_{ij}, \gamma'_{ij}\} \leq 2\sqrt{n}\mathtt{gap}. \tag{15.8}$$

Geometrically, (15.8) says that if the supporting hyperplanes are both mostly made up of the separating hyperplanes, then their aggregate contributions to F and F' should be small in absolute value.

The next lemma identifies some $p \in V$ for which $f(R(p)) - f(R(p) - p)$ is "big". This prepares for the final step of our approach which invokes Lemma 79.

Lemma 85. *Let $\vec{y} \overset{\text{def}}{=} \sum_{h \in H} \lambda_h \vec{h}$ and $\vec{y}' \overset{\text{def}}{=} \sum_{h \in H} \lambda'_h \vec{h}$ and let $p \in \arg\max_l\{\max\{|y_l|, |y'_l|\}\}$ then*

$$\mathtt{upper}(p) \geq n^7 \left\|\vec{y} + \vec{y}'\right\|_\infty$$

assuming (15.8).

Proof. Recall that $\left\|\vec{c} + \vec{c}'\right\|_2 \leq \mathtt{gap}$ where $\mathtt{gap} = \frac{1}{10n^{10}} \min\{||\vec{c}||_2, ||\vec{c}'||_2\}$,

$$\vec{c} = \vec{y} - \sum_i \alpha_i \vec{\mathbb{1}}_i + \sum_j \beta_j \vec{\mathbb{1}}_j + \sum_{(i,j)} \gamma_{ij}(\vec{\mathbb{1}}_i - \vec{\mathbb{1}}_j) \quad \text{and } \vec{c}' = \vec{y}' - \sum_i \alpha'_i \vec{\mathbb{1}}_i + \sum_j \beta'_j \vec{\mathbb{1}}_j + \sum_{(i,j)} \gamma'_{ij}(\vec{\mathbb{1}}_i - \vec{\mathbb{1}}_j).$$

By (15.8) we know that $\left\|\vec{c} - \vec{y}\right\|_2 \leq 4n^2\mathtt{gap} \leq \frac{4}{10n^8}\left\|\vec{c}\right\|_2$ and $\left\|\vec{c}' - \vec{y}'\right\|_2 \leq 4n^2\mathtt{gap} \leq \frac{4}{10n^8}\left\|\vec{c}'\right\|_2$. Consequently, by the triangle inequality we have that

$$\left\|\vec{y} + \vec{y}'\right\|_2 \leq \left\|\vec{c} + \vec{c}'\right\|_2 + \left\|\vec{c} - \vec{y}\right\|_2 + \left\|\vec{c}' - \vec{y}'\right\|_2 \leq 9n^2\mathtt{gap}$$

and

$$\|\vec{c}\|_2 \le \|\vec{c}-\vec{y}\|_2 + \|\vec{y}\|_2 \le \frac{4}{10n^8}\|\vec{c}\|_2 + \|\vec{y}\|_2 \quad \Rightarrow \quad \|\vec{c}\|_2 \le 2\|\vec{y}\|_2$$

Similarly, we have that $\|\vec{c}'\|_2 \le 2\|\vec{y}'\|_2$. Consequently since $\texttt{gap} \le \frac{1}{10n^{10}} \min\{||\vec{c}||_2, ||\vec{c}'||_2\}$, we have that

$$\|\vec{y}+\vec{y}'\|_2 \le \frac{2}{n^8} \min\left\{\|\vec{y}\|_2, \|\vec{y}'\|_2\right\}$$

and thus, invoking Lemma 81 yields the result. $\square$

We summarize the results in the lemma below.

Corollary 86. *Let P be the feasible region after running cutting plane on (15.1). Then one of the following holds:*

1. *We found a degenerate BFS and hence either $\varnothing$ or V is a minimizer.*
2. *The integral points of P all lie on some hyperplane $x_i = 0$, $x_j = 1$ or $x_i = x_j$ which we can find.*
3. *Let H be the collection of BFS's $\vec{h}$ used to construct our separating hyperplanes for P. Then there is a convex combination $\vec{y}$ of H such that $n^4|y_i| < \max_p \texttt{upper}(p)$ for all i.*

Proof. As mentioned before, (1) happens if some separating hyperplane is degenerate. We have (2) if one of the conditions in Lemma 84 holds. Otherwise, $y = \sum_{h\in H} \lambda_h \vec{h} + \sum_{h\in H} \lambda'_h \vec{h}$ is a candidate for Case 3 by Lemma 85. $\square$

Let us revisit the conditions of Lemma 79 and explain that they are satisfied by Case 3 of the last lemma.

- $\vec{y}$ is a convex combination of at most $O(n)$ BFS's. This holds in Case 3 since our current feasible region consists of only $O(n)$ constraints thanks to the Cutting Plane method.
- Those BFS's must be consistent with every arc of A. This holds because Case 3 uses the BFS's for constructing our separating hyperplane. Our modified separation oracle guarantees that they are consistent with A.

Thus in Case 3 of the last corollary, Lemma 79 allows us to deduce a new constraint $x_p \le x_q$ for some $q \notin R(p)$.

15.3.3 Running Time

Here we bound the total running time of our algorithm and prove the following.

Theorem 87. *Our algorithm runs in expected time $O(n^4 \log n \cdot EO + n^5 \log^{O(1)} n)$.*

Proof. To avoid being repetitive, we appeal to Corollary 86. Each phase of cutting plane takes time $O(n^2 \log n \cdot \text{EO} + n^3 \log^{O(1)} n)$ (Theorem 82 with τ being a big constant. Given F and F' represented as a nonnegative combination of facets, we can check for the conditions in Lemma 84 in $O(n)$ time as there are only this many facets of P. This settles Case 2 of Corollary 86. Finally, Lemma 79 tells us that we can find a new arc in $O(n \cdot \text{EO} + n^2)$ time for Case 3 of Corollary 86. Our conclusion follows from the fact that we can get $x_i = 0$, $x_i = 1$, $x_i = x_j$ at most n times and $x_i \le x_j$ at most $O(n^2)$ times. $\square$

15.4 $\widetilde{O}(n^3 \cdot \mathbf{EO} + n^4)$ Time Algorithm

Here we show how to improve our running time for strongly polynomial SFM to $\widetilde{O}(n^3 \cdot \text{EO} + n^4)$. Our algorithm can be viewed as an extension of the algorithm we presented in the previous Section 15.3. The main bottleneck of our previous algorithm was the time needed to identify a new arc, which cost us $\widetilde{O}(n^2 \cdot \text{EO} + n^3)$. Here we show how to reduce our amortized cost for identifying a valid arc down to $\widetilde{O}(n \cdot \text{EO} + n^2)$ and thereby achieve our result.

The key observation we make to improve this running time is that our choice of p for adding an arc in the previous lemma can be relaxed. p actually need not be $\arg\max_i \texttt{upper}(i)$; instead it is enough to have $\texttt{upper}(p) > n^4 \max\{\alpha_i, \alpha'_i, \beta_j, \beta'_j, \gamma_{ij}, \gamma'_{ij}\}$. For each such p a new constraint $x_p \leq x_q$ can be identified via Lemma 79. So if there are many p's satisfying this we will be able to obtain many new constraints and hence new valid arcs (p, q).

On the other hand, the bound in Lemma 85 says that our point in the base polyhedron is small in *absolute* value. This is actually stronger than what we need in Lemma 79 which requires only its positive entries to be "small". However as we saw in Lemma 80 we can generate a constraint of the form $x_q \leq x_p$ whenever $\texttt{lower}(p)$ is sufficiently negative.

Using this idea, we divide V into different buckets according to $\texttt{upper}(p)$ and $\texttt{lower}(p)$. This will allow us to get a speedup for two reasons.

First, bucketing allows us to disregard unimportant elements of V during certain executions of our cutting plane method. If both $\texttt{upper}(i)$ and $\texttt{lower}(i)$ are small in absolute value, then i is essentially negligible because for a separating hyperplane $\vec{h}^T\vec{x} \leq \hat{f}(\bar{x})$, any $h_i \in [\texttt{lower}(i), \texttt{upper}(i)]$ small in absolute value would not really make a difference. We can then run our cutting plane algorithm only on those non-negligible i's, thereby reducing our time complexity. Of course, whether h_i is small is something relative. This suggests that partitioning the ground set by the relative size of $\texttt{upper}(i)$ and $\texttt{lower}(i)$ is a good idea.

Second, bucketing allows us to ensure that we can always add an arc for many edges simultaneously. Recall that we remarked that all we want is $n^{O(1)}|y_i| \leq \texttt{upper}(p)$ for some $\vec{y}$ in the base polyhedron. This would be sufficient to identify a new valid arc (p, q). Now if the marginal differences $\texttt{upper}(p)$ and $\texttt{upper}(p')$ are close in value, knowing $n^{O(1)}|y_i| \leq \texttt{upper}(p)$ would effectively give us the same for p' for free. This suggests that elements with similar marginal differences should be grouped together.

The remainder of this section simply formalizes these ideas. In Section 15.4.1 we discuss how we partition the ground set V. In Section 15.4.2, we present our cutting plane method on a subset of the coordinates. Then in Section 15.4.3 we show how we find new arcs. Finally, in Section 15.4.4 we put all of this together to achieve our desired running time.

15.4.1 Partitioning Ground Set into Buckets

We partition the ground set V into different buckets according to the values of $\texttt{upper}(i)$ and $\texttt{lower}(i)$. This is reminiscent to Iwata-Orlin's algorithm [133] which considers elements with big $\texttt{upper}(i)$. However they did not need to do bucketing by size or to consider $\texttt{lower}(i)$, whereas these seem necessary for our algorithm.

Let $N = \max_i\{\max\{\texttt{upper}(i), -\texttt{lower}(i)\}\}$ be the largest marginal difference in absolute value. By Lemma (78), $N \geq 0$. We partition our ground set V as follows:

$$B_1 = \{i : \texttt{upper}(i) \geq N/n^{10} \text{ or } \texttt{lower}(i) \leq -N/n^{10}\}$$

$$\begin{aligned} B_k \quad &= \{i \notin B_1 \cup \ldots \cup B_{k-1} : \quad N/n^{10k} \leq \texttt{upper}(i) < N/n^{10(k-1)} \\ &\qquad \text{or } -N/n^{10(k-1)} < \texttt{lower}(i) \leq -N/n^{10k}\}, \quad k \geq 2 \end{aligned}$$

We call B_k *buckets*. Our buckets group elements by the values of $\texttt{upper}(i)$ and $\texttt{lower}(i)$ at $1/n^{10}$ "precision". There are two cases.

- Case 1: the number of buckets is at most $\log n$[5], in which case $\texttt{upper}(i) > N/n^{O(\log n)}$ or $\texttt{lower}(i) < -N/n^{O(\log n)}$ for all i.

- Case 2: there is some k for which $|B_1 \cup \ldots \cup B_k| \geq |B_{k+1}|$.

This is because if there is no such k in Case 2, then by induction each bucket B_{k+1} has at least $2^k|B_1| \geq 2^k$ elements and hence $k \leq \log n$.

Case 1 is easier to handle, and is in fact a special case of Case 2. We first informally sketch the treatment for Case 1 which should shed some light into how we deal with Case 2.

We run Cutting Plane for $O(n \log^2 n)$ iterations (i.e. $\tau = \Theta(\log n)$). By Theorem 82, our feasible region P would be sandwiched by a pair of approximately parallel supporting hyperplanes of width at most $1/n^{\Theta(\log n)}$. Now proceeding as in the last section, we would be able to find some $\vec{y}$ in the base polyhedron and some element p such that $n^{\Theta(\log n)}|y_i| \leq \texttt{upper}(p)$. This gives

$$n^{\Theta(\log n)}|y_i| \leq \frac{\texttt{upper}(p)}{n^{\Theta(\log n)}} \leq \frac{N}{n^{\Theta(\log n)}}.$$

Since $\texttt{upper}(i) > N/n^{\Theta(\log n)}$ or $\texttt{lower}(i) < -N/n^{\Theta(\log n)}$ for all i in Case 1, we can then conclude that some valid arc (i,q) or (q,i) can be added for every i. Thus we add $n/2$ arcs simultaneously in one phase of the algorithm at the expense of blowing up the runtime by $O(\log n)$. This saves a factor of $n/\log n$ from our runtime in the last section, and the amortized cost for an arc would then be $\widetilde{O}(n \cdot \text{EO} + n^2)$.

On the other hand, in Case 2 we have a "trough" at B_{k+1}. Roughly speaking, this trough is useful for acting as a soft boundary between $B_1 \cup \ldots \cup B_k$ and $\bigcup_{l \geq k+2} B_l$. Recall that we are able to "ignore" $\bigcup_{l \geq k+2} B_l$ because their h_i is relatively small in absolute value. In particular, we know that for any $p \in B_1 \cup \ldots \cup B_k$ and $i \in B_l$, where $l \geq k+2$,

$$\max\{\texttt{upper}(p), -\texttt{lower}(p)\} \geq n^{10} \max\{\texttt{upper}(i), -\texttt{lower}(i)\}.$$

This is possible because B_{k+1}, which is sandwiched in between, acts like a shield preventing B_l to "mess with" $B_1 \cup \ldots \cup B_k$. This property comes at the expense of sacrificing B_{k+1} which must confront B_l.

Furthermore, we require that $|B_1 \cup \ldots \cup B_k| \geq |B_{k+1}|$, and run Cutting Plane on $B = (B_1 \cup \ldots \cup B_k) \cup B_{k+1}$. If $|B_{k+1}| \gg |B_1 \cup \ldots \cup B_k|$, our effort would mostly be wasted on B_{k+1} which is sacrificed, and the amortized time complexity for $B_1 \cup \ldots \cup B_k$ would then be large.

Before discussing the algorithm for Case 2, we need some preparatory work.

[5]More precisely, $B_k = \varnothing$ for $k > \lceil \log n \rceil$.

15.4.2 Separating Hyperplane: Project and Lift

Our speedup is achieved by running our cutting plane method on the projection of our feasible region onto $B := (B_1 \cup \cdots \cup B_k) \cup B_{k+1}$. More precisely, we start by running our cutting plane on $P^B = \{\vec{x} \in \mathbb{R}^B : \exists \vec{x}' \in \mathbb{R}^{\bar{B}} \text{ s.t. } (\vec{x}, \vec{x}') \text{ satisfies (15.1)}\}$, which has a lower dimension. However, to do this, we need to specify a separation oracle for P^B. Here we make one of the most natural choices.

We begin by making an apparently immaterial change to our set of arcs A. Let us take the *transitive closure* of A by adding the arc (i,j) whenever there is a path from i to j. Clearly this would not change our ring family as a path from i to j implies $j \in R(i)$. Roughly speaking, we do this to handle pathological cases such as $(i,k),(k,j) \in A, (i,j) \notin A$ and $i,j \in B, k \notin B$. Without introducing the arc (i,j), we risk confusing a solution containing i but not j as feasible since we are restricting our attention to B and ignoring $k \notin B$.

Definition 88. Given a digraph $D = (V, A)$, the transitive closure of A is the set of arcs (i,j) for which there is a directed path from i to j. We say that A is *complete* if it is equal to its transitive closure.

Given $\bar{x} \in [0,1]^B$, we define the completion of $\bar{x}$ with respect to A as follows.

Definition 89. Given $\bar{x} \in [0,1]^B$ and a set of arcs A, $x^{\mathcal{C}} \in [0,1]^n$ is a completion of $\bar{x}$ if $x^{\mathcal{C}}_B = \bar{x}$ and $x^{\mathcal{C}}_i \leq x^{\mathcal{C}}_j$ for every $(i,j) \in A$. Here $x^{\mathcal{C}}_B$ denotes the restriction of $x^{\mathcal{C}}$ to B.

Lemma 90. *Given $\bar{x} \in [0,1]^B$ and a complete set of arcs A, there is a completion of $\bar{x}$ if* $\bar{x}_i \leq \bar{x}_j$ for every $(i,j) \in A \cap (B \times B)$. Moreover, it can be computed in $O(n^2)$ time.

Proof. We set $x^{\mathcal{C}}_B = \bar{x}$. For $i \notin B$, we set

$$x^{\mathcal{C}}_i = \begin{cases} 1 & \text{if } \nexists j \in B \text{ s.t. } (i,j) \in A \\ \min_{(i,j)\in A, j \in B} x^{\mathcal{C}}_j & \text{otherwise} \end{cases}$$

One may verify that $x^{\mathcal{C}}$ satisfies our requirement as A is complete. Computing each $x^{\mathcal{C}}_i$ takes $O(n)$ time. Since $|V \backslash B| = |\bar{B}| \leq n$, computing the whole $x^{\mathcal{C}}$ takes $O(n^2)$ time. □

This notion of completion is needed since our original separation oracle requires a full dimensional input $\bar{x}$. Now that $\bar{x} \in \mathbb{R}^B$, we need a way of extending it to $\mathbb{R}^n$ while retaining the crucial property that $\vec{h}$ is consistent with every arc in A.

Note that the runtime is still $O(n \cdot \text{EO} + n^2 \log^{O(1)} n)$ as $x^{\mathcal{C}}$ can be computed in $O(n^2)$ time by the last lemma.

We reckon that the hyperplane $\vec{h}_B^T \vec{x}_B \leq \sum_{i \in B} h_i \bar{x}_i$ returned by the oracle is *not* a valid separating hyperplane (i.e. it may cut out the minimizers). Nevertheless, we will show that it is a decent "proxy" to the true separating hyperplane $\vec{h}^T \vec{x} \leq \hat{f}(x^{\mathcal{C}}) = \sum_{i \in V} h_i x^{\mathcal{C}}_i$ and is good enough to serve our purpose of sandwiching the remaining feasible region in a small strip. To get a glimpse, note that the terms missing $\vec{h}_B^T \vec{x}_B \leq \sum_{i \in B} h_i \bar{x}_i$ all involve h_i for $i \notin B$, which is "negligible" compared to $B_1 \cup \cdots \cup B_k$.

One may try to make $\vec{h}_B^T \vec{x}_B \leq \sum_{i \in B} h_i \bar{x}_i$ valid, say, by $\vec{h}_B^T \vec{x}_B \leq \sum_{i \in B} h_i \bar{x}_i + \sum_{i \notin B} |h_i|$. The problem is that such hyperplanes would not be separating for $\bar{x}$ anymore as $\vec{h}_B^T \bar{x} =$

Algorithm 6: Projected Separation Oracle

```
Input: $\bar{x} \in \mathbb{R}^B$ and a complete set of arcs $A$
if  $\bar{x}_i < 0$ for some $i \in B$ then
  | Output: $x_i \geq 0$
else if  $\bar{x}_j > 1$ for some $j \in B$ then
  | Output: $x_j \leq 1$
else if  $\bar{x}_i > \bar{x}_j$ for some $(i,j) \in A \cap B^2$ then
  | Output: $x_i \leq x_j$
else
  | Let $x^{\mathcal{C}} \in \mathbb{R}^n$ be a completion of $\bar{x}$
  | Let $i_1, \ldots, i_n$ be a permutation of $V$ such that $x^{\mathcal{C}}_{i_1} \geq \ldots \geq x^{\mathcal{C}}_{i_n}$ and for all
  |   $(i,j) \in A$, $j$ precedes $i$ in $i_1, \ldots, i_n$.
  | Output: $\vec{h}_B^T \vec{x}_B = \sum_{i \in B} h_i x_i \leq \sum_{i \in B} h_i \bar{x}_i$, where $\vec{h}$ is the BFS defined by the
  |   permutation $i_1, \ldots, i_n$.
```

$\sum_{i \in B} h_i \bar{x}_i < \sum_{i \in B} h_i \bar{x}_i + \sum_{i \notin B} |h_i|$. Consequently, we lose the width (or volume) guarantee of our cutting plane algorithm. Although this seems problematic, it is actually still possible to show a guarantee sufficient for our purpose as $\sum_{i \notin B} |h_i|$ is relatively small. We leave it as a nontrivial exercise to interested readers.

In conclusion, it seems that one cannot have the best of both worlds: the hyperplane returned by the oracle cannot be simultaneously valid and separating.

Algorithm

We take k to be the first for which $|B_1 \cup \ldots \cup B_k| \geq |B_{k+1}|$, i.e. $|B_1 \cup \ldots \cup B_l| < |B_{l+1}|$ for $l \leq k-1$. Thus $k \leq \log n$. Let $b = |B|$, and so $|B_1 \cup \cdots \cup B_k| \geq b/2$. Case 1 is a special case by taking $B = V$.

Our algorithm is summarized below. Here A is always complete as A is replaced its transitive closure whenever a new valid arc is added.

1. Run Cutting Plane on $P^B = \{x \in \mathbb{R}^B : \exists x' \in \mathbb{R}^{\bar{B}} \text{ s.t. } (x, x') \text{ satisfies (15.1)}\}$ with the new projected separation oracle.

2. Identify a pair of "narrow" approximately parallel supporting hyperplanes.

3. Deduce from the hyperplanes certain new constraints of the forms $x_i = 0, x_j = 1, x_i = x_j$ or $x_i \leq x_j$ by lifting separating hyperplanes back to $\mathbb{R}^n$

4. Consolidate A and f. If some $x_i \leq x_j$ added, replace A by its transitive closure.

5. Repeat Step 1 with updated A and f. (Any previously found separating hyperplanes are discarded.)

The minimizer can be constructed by unraveling the recursion.

First of all, to be able to run Cutting Plane on P^B we must come up with a polyhedral description of P^B which consists of just the constraints involving B. This is shown in the next lemma.

Lemma 91. *Let $P^B = \{\vec{x} \in \mathbb{R}^B : \exists \vec{x}' \in \mathbb{R}^{\bar{B}}$ s.t. $(\vec{x}, \vec{x}')$ satisfies (15.1)$\}$. Then*

$$P^B = \{\vec{x} \in \mathbb{R}^B : 0 \leq \vec{x} \leq 1, x_i \leq x_j \forall (i,j) \in A \cap (B \times B)\}$$

Proof. It is clear that $P^B \subseteq \{\vec{x} \in \mathbb{R}^B : 0 \leq \vec{x} \leq 1, x_i \leq x_j \forall (i,j) \in A \cap (B \times B)\}$ as the constraints $0 \leq x \leq 1, x_i \leq x_j \forall (i,j) \in A \cap (B \times B)$ all appear in (15.1).

Conversely, for any $\vec{x} \in \mathbb{R}^B$ satisfying $0 \leq \vec{x} \leq 1, x_i \leq x_j \forall (i,j) \in A \cap (B \times B)$, we know there is some completion $x^{\mathcal{C}}$ of $\vec{x}$ by Lemma 90 as A is complete. Now $x^{\mathcal{C}}$ satisfies (15.1) by definition, and hence $\vec{x} \in P^B$. □

The only place where we have really changed the algorithm is Step (3).

15.4.3 Deducing New Constraints $x_i = 0$, $x_j = 1$, $x_i = x_j$ or $x_i \leq x_j$

Our method will deduce one of the following:

- $x_i = 0$, $x_j = 1$ or $x_i = x_j$
- for each $p \in B_1 \cup \cdots \cup B_k$, $x_p \leq x_q$ for some $q \notin R(p)$ or $x_p \geq x_q$ for some $q \notin Q(p)$

Our argument is very similar to the last section's. Roughly speaking, it is the same argument but with "noise" introduced by $i \notin B$. We use extensively the notations from the last section.

Our main tool is again Theorem 82. Note that n should be replaced by b in the Theorem statement. We invoke it with $\tau = k \log_b n = O(\log^2 n)$ (using $k \leq \log n$) to get a width of $1/b^{\Theta(\tau)} = 1/n^{\Theta(k)}$. This takes time at most $O(bn \log^2 n \cdot \text{EO} + bn^2 \log^{O(1)} n)$. Again, this is intuitively clear as we run it for $O(kb \log n)$ iterations, each of which takes time $O(n \cdot \text{EO} + n^2 \log^{O(1)} n)$.

After each phase of (roughly $O(kb \log n)$ iterations) of Cutting Plane, P^B is sandwiched between a pair of approximately parallel supporting hyperplanes F and F' which have width $1/n^{20k}$. Let F and F' be

$$\vec{c}^T \vec{x}_B = \sum_{i \in B} c_i x_i \leq M, \quad \vec{c}'^T \vec{x}_B = \sum_{i \in B} c'_i x_i \leq M',$$

such that

$$|M + M'|, \ ||\vec{c} + \vec{c}'||_2 \leq \mathsf{gap}, \quad \text{where } \mathsf{gap} = \frac{1}{n^{20k}} \min\{||\vec{c}||_2, ||\vec{c}'||_2\}.$$

The rest of this section presents an execution of the ideas discussed above. All of our work is basically geared towards bringing the amortized cost for identifying a valid arc down to $\widetilde{O}(n \cdot \text{EO} + n^2)$. Again, we can write these two constraints as a nonnegative combination. Here $\bar{x}_h^{\mathcal{C}}$ is the completion of the point $\bar{x}_h$ used to construct $\vec{h}_B^T \vec{x}_B \leq \vec{h}_B^T \left(\bar{x}_h^{\mathcal{C}}\right)_B$. (Recall that $\left(\bar{x}_h^{\mathcal{C}}\right)_B$ is the restriction of $\bar{x}_h^{\mathcal{C}}$ to B.)

$$\vec{c}^T \vec{x}_B = -\sum_{i \in B} \alpha_i x_i + \sum_{j \in B} \beta_j x_j + \sum_{(i,j) \in A \cap B^2} \gamma_{ij}(x_i - x_j) + \sum_{h \in H} \lambda_h \vec{h}_B^T \vec{x}_B \quad \text{and}$$

$$M = \sum_{j \in B} \beta_j + \sum_{h \in H} \lambda_h \vec{h}_B^T \left(\bar{x}_h^{\mathcal{C}}\right)_B.$$

$$\vec{c}'^T \vec{x}_B = -\sum_{i\in B} \alpha'_i x_i + \sum_{j\in B} \beta'_j x_j + \sum_{(i,j)\in A\cap B^2} \gamma'_{ij}(x_i - x_j) + \sum_{h\in H} \lambda'_h \vec{h}_B^T \vec{x}_B \quad \text{and}$$

$$M' = \sum_{j\in B} \beta'_j + \sum_{h\in H} \lambda'_h \vec{h}_B^T \left(\bar{x}_h^{\mathcal{C}}\right)_B.$$

As we have discussed, the problem is that the separating hyperplanes $\vec{h}_B^T \vec{x}_B \leq \vec{h}_B^T \left(\bar{x}_h^{\mathcal{C}}\right)_B$ are not actually valid. We can, however, recover their valid counterpart by lifting them back to $\vec{h}^T \vec{x} \leq \vec{h}^T \bar{x}_h^{\mathcal{C}}$. The hope is that $\vec{h}_B^T \vec{x}_B \leq \vec{h}_B^T \left(\bar{x}_h^{\mathcal{C}}\right)_B$ and $\vec{h}^T \vec{x} \leq \vec{h}^T \bar{x}_h^{\mathcal{C}}$ are not too different so that the arguments will still go through. We show that this is indeed the case.

Again, we scale $c, c', \alpha, \alpha', \beta, \beta', \gamma, \gamma', \lambda, \lambda'$ so that

$$\sum_{h\in H} (\lambda_h + \lambda'_h) = 1.$$

By adding all the constituent separating hyperplane inequalities, we get

$$\sum_{h\in H} \lambda_h \vec{h}^T \vec{x} + \sum_{h\in H} \lambda'_h \vec{h}^T \vec{x} \leq \sum_{h\in H} \lambda_h \vec{h}^T \bar{x}_h^{\mathcal{C}} + \sum_{h\in H} \lambda'_h \vec{h}^T \bar{x}_h^{\mathcal{C}}$$

Let

$$LHS \stackrel{\text{def}}{=} \sum \alpha_i x_i + \sum \alpha'_i x_i - \sum \beta_j x_j - \sum \beta'_j x_j + \sum \gamma_{ij}(x_j - x_i) + \sum \gamma'_{ij}(x_j - x_i).$$

Here we know that

$$\sum_{h\in H} \lambda_h \vec{h}^T \vec{x} + \sum_{h\in H} \lambda'_h \vec{h}^T \vec{x} = LHS + (\vec{c} + \vec{c}')^T \vec{x}_B + \sum_{h\in H} \lambda_h \vec{h}_{\bar{B}}^T \vec{x}_{\bar{B}} + \sum_{h\in H} \lambda'_h \vec{h}_{\bar{B}}^T \vec{x}_{\bar{B}}$$

$$\sum_{h\in H} \lambda_h \vec{h}^T \bar{x}_h^{\mathcal{C}} + \sum_{h\in H} \lambda'_h \vec{h}^T \bar{x}_h^{\mathcal{C}} = (M + M') + \sum_{h\in H} \lambda_h \vec{h}_{\bar{B}}^T \left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}} + \sum_{h\in H} \lambda'_h \vec{h}_{\bar{B}}^T \left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}} - \sum \beta_j - \sum \beta'_j$$

Combining all yields

$$LHS + (\vec{c} + \vec{c}')^T \vec{x}_B + \sum_{h\in H} \lambda_h \vec{h}_{\bar{B}}^T \vec{x}_{\bar{B}} + \sum_{h\in H} \lambda'_h \vec{h}_{\bar{B}}^T \vec{x}_{\bar{B}}$$

$$\leq (M + M') + \sum_{h\in H} \lambda_h \vec{h}_{\bar{B}}^T \left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}} + \sum_{h\in H} \lambda'_h \vec{h}_{\bar{B}}^T \left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}} - \sum \beta_j - \sum \beta'_j$$

Here $(\vec{c} + \vec{c}')^T \vec{x}_B$ can be bounded as before: $(\vec{c} + \vec{c}')^T \vec{x}_B \geq -\sqrt{n}||\vec{c} + \vec{c}'||_2 \geq -\sqrt{n}\mathsf{gap}$. Since $M + M' \leq \mathsf{gap}$, We then obtain

$$LHS + \sum_{h\in H} \lambda_h \vec{h}_{\bar{B}}^T \vec{x}_{\bar{B}} + \sum_{h\in H} \lambda'_h \vec{h}_{\bar{B}}^T \vec{x}_{\bar{B}} \leq 2\sqrt{n}\mathsf{gap} + \sum_{h\in H} \lambda_h \vec{h}_{\bar{B}}^T \left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}} + \sum_{h\in H} \lambda'_h \vec{h}_{\bar{B}}^T \left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}} - \sum \beta_j - \sum \beta'_j$$

We should expect the contribution from $\vec{h}_{\bar{B}}$ to be small as h_i for $i \notin B$ is small compared to $B_1 \cup \ldots \cup B_k$. We formalize our argument in the next two lemmas.

Lemma 92. *We have* $\sum_{h\in H}\lambda_h\vec{h}_{\bar{B}}^T\left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}}+\sum_{h\in H}\lambda'_h\vec{h}_{\bar{B}}^T\left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}}\le N/n^{10(k+1)-1}$.

Proof. We bound each component of $\sum_{h\in H}\lambda_h\vec{h}_{\bar{B}}^T\left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}}+\sum_{h\in H}\lambda'_h\vec{h}_{\bar{B}}^T\left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}}$. For $i\in\bar{B}$, we have $\texttt{upper}(i)\le N/n^{10(k+1)}$. By Lemma 78 $h_i\le\texttt{upper}(i)$. Therefore,

$$\sum_{h\in H}\lambda_h\vec{h}_i^T\left(\bar{x}_h^{\mathcal{C}}\right)_i+\sum_{h\in H}\lambda'_h\vec{h}_i^T\left(\bar{x}_h^{\mathcal{C}}\right)_i \;\le\; \left(\sum_{h\in H}\lambda_h+\sum_{h\in H}\lambda'_h\right)N/n^{10(k+1)}=N/n^{10(k+1)}.$$

Our result then follows since

$$\sum_{h\in H}\lambda_h\vec{h}_{\bar{B}}^T\left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}}+\sum_{h\in H}\lambda'_h\vec{h}_{\bar{B}}^T\left(\bar{x}_h^{\mathcal{C}}\right)_{\bar{B}}=\sum_{i\in\bar{B}}\left(\sum_{h\in H}\lambda_h\vec{h}_i^T\left(\bar{x}_h^{\mathcal{C}}\right)_i+\sum_{h\in H}\lambda'_h\vec{h}_i^T\left(\bar{x}_h^{\mathcal{C}}\right)_i\right).$$

□

Lemma 93. *We have* $\sum_{h\in H}\lambda_h\vec{h}_{\bar{B}}^T\vec{x}_{\bar{B}}+\sum_{h\in H}\lambda'_h\vec{h}_{\bar{B}}^T\vec{x}_{\bar{B}}\ge -N/n^{10(k+1)-1}$.

Proof. The proof is almost identical to the last lemma except that we use $h_i\ge\texttt{lower}(i)$ instead of $h_i\le\texttt{upper}(i)$, and $\texttt{lower}(i)\ge -N/n^{10(k+1)}$. □

The two lemmas above imply that

$$LHS\le 2\sqrt{n}\texttt{gap}-\sum\beta_j-\sum\beta_j+2N/n^{10(k+1)-1}=\texttt{gap}'-\sum\beta_j-\sum\beta_j$$

where $\texttt{gap}'=2\sqrt{n}\texttt{gap}+2N/n^{10(k+1)-1}$.

Lemma 94. *Suppose x satisfies (15.1) and $LHS\le\texttt{gap}'-\sum\beta_j-\sum\beta'_j$ with $\alpha_i,\beta_j,\gamma_{ij},\alpha'_i,\beta'_j,\gamma'_{ij}\ge 0$.*

1. *If $\alpha_i>\texttt{gap}'$ or $\alpha'_i>\texttt{gap}'$, then $x_i<1$.*
2. *If $\beta_j>\texttt{gap}'$ or $\beta'_j>\texttt{gap}'$, then $x_j>0$.*
3. *If $\gamma_{ij}>\texttt{gap}'$ or $\gamma'_{ij}>\texttt{gap}'$, then $0\le x_j-x_i<1$.*

Proof. The proof is exactly the same as Lemma 84 with $2\sqrt{n}\texttt{gap}$ replaced by $\texttt{gap}'$. □

From now on we may assume that

$$\max\{\alpha_i,\alpha'_i,\beta_j,\beta'_j,\gamma_{ij},\gamma'_{ij}\}\le\texttt{gap}'. \tag{15.9}$$

Lemma 95. *Let $\vec{y}\overset{\text{def}}{=}\sum_{h\in H}\lambda_h\vec{h}$ and $\vec{y}'\overset{\text{def}}{=}\sum_{h\in H}\lambda'_h\vec{h}$ and let $p\in\arg\max_{l\in B}\{\max\{|y_l|,|y'_l|\}$ then*

$$N\ge n^{10k+6}\left\|\vec{y}_B+\vec{y}'_B\right\|_\infty$$

assuming (15.9).

Proof. Recall that $\left\|\vec{c}+\vec{c}'\right\|_2 \leq \texttt{gap} < \texttt{gap}'$ where $\texttt{gap} = \frac{1}{n^{20k}}\min\{||\vec{c}||_2, ||\vec{c}'||_2\}$ and $\texttt{gap}' = 2\sqrt{n}\texttt{gap} + 2N/n^{10(k+1)-1}$. Now there are two cases.

Case 1: $2\sqrt{n}\texttt{gap} \geq 2N/n^{10(k+1)-1}$. Then $\texttt{gap}' \leq 4\sqrt{n}\texttt{gap}$ and we follow the same proof of Lemma 85. We have

$$\vec{c} = \vec{y}_B - \sum_i \alpha_i \vec{\mathbb{1}}_i + \sum_j \beta_j \vec{\mathbb{1}}_j + \sum_{(i,j)} \gamma_{ij}(\vec{\mathbb{1}}_i - \vec{\mathbb{1}}_j) \quad \text{and } \vec{c}' = \vec{y}'_B - \sum_i \alpha'_i \vec{\mathbb{1}}_i + \sum_j \beta'_j \vec{\mathbb{1}}_j + \sum_{(i,j)} \gamma'_{ij}(\vec{\mathbb{1}}_i - \vec{\mathbb{1}}_j)\,.$$

By (15.9) we know that $\left\|\vec{c}-\vec{y}_B\right\|_2 \leq 4n^2\texttt{gap}' \leq \frac{1}{n^{17k}}\left\|\vec{c}\right\|_2$ and $\left\|\vec{c}'-\vec{y}'_B\right\|_2 \leq 4n^2\texttt{gap}' \leq \frac{1}{n^{17k}}\left\|\vec{c}\right\|_2$. Consequently, by the triangle inequality we have that

$$\left\|\vec{y}_B + \vec{y}'_B\right\|_2 \leq \left\|\vec{c}+\vec{c}'\right\|_2 + \left\|\vec{c}-\vec{y}_B\right\|_2 + \left\|\vec{c}'-\vec{y}'_B\right\|_2 \leq 9n^2\texttt{gap}'$$

and

$$\left\|\vec{c}\right\|_2 \leq \left\|\vec{c}-\vec{y}_B\right\|_2 + \left\|\vec{y}_B\right\|_2 \leq \frac{1}{n^{17k}}\left\|\vec{c}\right\|_2 + \left\|\vec{y}_B\right\|_2 \quad \Rightarrow \quad \left\|\vec{c}\right\|_2 \leq 2\left\|\vec{y}_B\right\|_2$$

Similarly, we have that $\left\|\vec{c}'\right\|_2 \leq 2\left\|\vec{y}'_B\right\|_2$. Consequently since $\texttt{gap}' \leq \frac{1}{n^{19k}}\min\{||\vec{c}||_2, ||\vec{c}'||_2\}$, we have that

$$\left\|\vec{y}_B + \vec{y}'_B\right\|_2 \leq \frac{18}{n^{17k}}\min\left\{\left\|\vec{y}_B\right\|_2, \left\|\vec{y}'_B\right\|_2\right\}$$

and thus, invoking Lemma 81 yields $N \geq \texttt{upper}(p) \geq n^{16k}\left\|\vec{y}_B + \vec{y}'_B\right\|_\infty$, as desired.

Case 2: $2\sqrt{n}\texttt{gap} < 2N/n^{10(k+1)-1}$. Then for any $i \in B$, $|c_i + c'_i| \leq ||\vec{c}+\vec{c}'||_2 \leq \texttt{gap} < 2N/n^{10(k+1)-1}$. Since

$$\vec{y}_B + \vec{y}'_B = (\vec{c}+\vec{c}') + \sum_i \alpha_i \vec{\mathbb{1}}_i - \sum_j \beta_j \vec{\mathbb{1}}_j - \sum_{(i,j)} \gamma_{ij}(\vec{\mathbb{1}}_i - \vec{\mathbb{1}}_j) + \sum_i \alpha'_i \vec{\mathbb{1}}_i - \sum_j \beta'_j \vec{\mathbb{1}}_j - \sum_{(i,j)} \gamma'_{ij}(\vec{\mathbb{1}}_i - \vec{\mathbb{1}}_j)$$

we have

$$\left\|\vec{y}_B + \vec{y}'_B\right\|_\infty \leq 2N/n^{10(k+1)-1} + 2n^{1.5}\texttt{gap}' \leq N/n^{10k+7}.$$

□

Corollary 96. *Let P be the feasible region after running Cutting Plane on (15.1) with the projected separation oracle. Then one of the following holds:*

1. *We found a BFS $\vec{h}$ with $\vec{h}_B = 0$.*
2. *The integral points of P all lie on some hyperplane $x_i = 0, x_j = 1$ or $x_i = x_j$.*
3. *Let H be the collection of BFS's $\vec{h}$ used to construct our separating hyperplanes for P. Then there is a convex combination $\vec{y}$ of H such that for $p \in B_1 \cup \cdots \cup B_k$, we have $n^4|y_i| < \texttt{upper}(p)$ or $\texttt{lower}(p) < -n^4|y_i|$ for all i.*

Proof. As mentioned before, (1) happens if some separating hyperplane satisfies $\vec{h}_B = 0$ when running cutting plane on the non-negligible coordinates. We have (2) if some condition in

Lemma 94 holds. Otherwise, we claim $y = \sum_{\boldsymbol{h}} \lambda_{\boldsymbol{h}} \boldsymbol{h} + \sum_{\boldsymbol{h}} \lambda'_{\boldsymbol{h}} \boldsymbol{h}$ is a candidate for Case 3. y is a convex combination of BFS and by Lemma 95, for the big elements $i \in B$ we have

$$|y_i| \leq N/n^{10k+6} \leq \frac{1}{n^4} \max\{\texttt{upper}(p), -\texttt{lower}(p)\}.$$

where the last inequality holds since for $p \in B_1 \cup \cdots \cup B_k$, $\max\{\texttt{upper}(p), -\texttt{lower}(p)\} \geq N/n^{10k}$.

On the other hand, for the small elements $i \notin B$,

$$|y_i| \leq N/n^{10(k+1)} \leq \frac{1}{n^4} \max\{\texttt{upper}(p), -\texttt{lower}(p)\}$$

as desired. □

The gap is then smaller enough to add an arc for each $p \in B_1 \cup \cdots \cup B_k$ by Lemmas 79 and 80. Therefore we can add a total of $|B_1 \cup \cdots \cup B_k|/2 \geq b/4$ arcs with roughly $O(kb \log n) = \widetilde{O}(b)$ iterations of Cutting Plane, each of which takes $\widetilde{O}(n \cdot \text{EO} + n^2)$. That is, the amortized cost for each arc is $\widetilde{O}(n \cdot \text{EO} + n^2)$. We give a more formal time analysis in below but it should be somewhat clear why we have the desired time complexity.

Lemma 97. *Suppose there is a convex combination $\vec{y}$ of H such that for $p \in B_1 \cup \cdots \cup B_k$, we have $n^4|y_i| < \texttt{upper}(p)$ or $\texttt{lower}(p) < -n^4|y_i|$ for all i. Then we can identify at least $b/4$ new valid arcs.*

Proof. We have $|H| = O(n)$ since H is the set of BFS's used for the constraints of P which has $O(n)$ constraints. By Lemmas 79 and 80, for $p \in B_1 \cup \cdots \cup B_k$ we can add a new valid arc (p, q) or (q, p). However note that a new arc (p_1, p_2) may added twice by both p_1 and p_2. Therefore the total number of new arcs is only at least $|B_1 \cup \cdots \cup B_k|/2 \geq b/4$. □

15.4.4 Running Time

Not much changes to the previous runtime analysis are needed. To avoid repetition, various details already present in the corresponding part of the last section are omitted. Recall $k \leq \log n$, and of course, $b \leq n$.

For each (roughly) $O(kb \log n)$ iterations of Cutting Plane we either get $x_i = 0$,$x_i = 1$,$x_i = x_j$ or $b/4$ $x_i \leq x_j$'s. The former can happen at most n times while in the latter case, the amortized cost of each arc is $O(k \log n)$ iterations of Cutting Plane. In the worst case the overall number of iterations required is $\widetilde{O}(n^2)$. Thus our algorithm has a runtime of $\widetilde{O}(n^3 \cdot \text{EO} + n^4)$ since each iteration is $\widetilde{O}(n \cdot \text{EO} + n^2)$ as shown below.

Theorem 98. *Our algorithm runs in time $O(n^3 \log^2 n \cdot EO + n^4 \log^{O(1)} n)$.*

Proof. We use Corollary 96. First we note that Case 1 can actually be integrated into Case 3 since $\max\{\texttt{upper}(p), -\texttt{lower}(p)\} \geq N/n^{10k} = n^{10}N/n^{10(k+1)} \geq h_i$ for $i \notin B$.

As we have argued in the beginning of the last section, Theorem 82 with $\tau = k \log_b n$ implies that the runtime for each phase is $O(bn \log^2 n \cdot \text{EO} + bn^2 \log^{O(1)} n)$. In each phase we either get $x_i = 0$, $x_i = 1$, $x_i = x_j$ (Case 2) or $b/4$ $x_i \leq x_j$'s (Case 3), the latter of which follows from Corollary 96 and Lemma 97.

Case 2 can only happen n times. Thus the total cost is at most $O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$. The overhead cost is also small. Similar to before, given F and F' represented as a nonnegative combination of facets, we can check for the conditions in Lemma 94 in $O(n)$ time as there are only this many facets of P. This settles Case 2.

For case 3 the amortized cost for each arc is $O(n \log^2 n \cdot \text{EO} + n^2 \log^{O(1)} n)$. Our desired runtime follows since there are only $O(n^2)$ arcs to add. Unlike Case 2 some extra care is needed to handle the overhead cost. The time needed to deduce a new arc (applying Lemmas 79 and 80 to $\vec{y}$ and $p \in B_1 \cup \cdots \cup B_k$) is still $O(n \cdot \text{EO} + n^2)$. But as soon as we get a new arc, we must update A to be its transitive closure so that it is still complete. Given A complete and a new arc $(p, q) \notin A$, we can simply add the arcs from the ancestors of p to q and from p to the descendants of q. There are at most $O(n)$ arcs to add so this takes time $O(n^2)$ per arc, which is okay. □

16 Discussion and Comparison with Previous Algorithms

We compare and contrast our algorithms with the previous ones. We focus primarily on strongly polynomial time algorithms.

Convex combination of BFS's

All of the previous algorithms maintain a convex combination of BFS's and iteratively improve over it to get a better primal solution. In particular, the new BFS's used are typically obtained by making local changes to existing ones. Our algorithms, on the other hand, considers the geometry of the existing BFS's. The weighted "influences"[6] then aggregately govern the choice of the next BFS. We believe that this is the main driving force for the speedup of our algorithms.

Scaling schemes

Many algorithms for combinatorial problems are explicitly or implicitly scaling a potential function or a parameter. In this Part, our algorithms in some sense aim to minimize the volume of the feasible region. Scaling schemes for different potential functions and parameters were also designed in previous works [128, 126, 133, 125]. All of these functions and parameters have an explict form. On the contrary, our potential function is somewhat unusual in the sense that it has no closed form.

Deducing new constraints

As mentioned in the main text, our algorithms share the same skeleton and tools for deducing new constraints with [128, 126, 133, 125]. Nevertheless, there are differences in the way these tools are employed. Our algorithms proceed by invoking them in a geometric manner, whereas previous algorithms were mostly combinatorial.

Big elements and bucketing

Our bucketing idea has roots in Iwata-Orlin's algorithm [133] but is much more sophisticated. For instance, it is sufficient for their algorithm to consider only big elements, i.e.

[6]In the terminology of Part I, these weighted influences are the leverage scores.

$\texttt{upper}(i) \geq N/n^{O(1)}$. Our algorithm, on the other hand, must carefully group elements by the size of both $\texttt{upper}(i)$ and $\texttt{lower}(i)$. The speedup appears impossible without these new ideas. We do however note that it is unfair to expect such a sophisticated scheme in Iwata-Orlin's algorithm as it would not lead to a speedup. In other words, their method is fully sufficient for their purposes, and the simplicity in their case is a virtue rather than a shortcoming.

16.1 Open Problems

One natural open problem is improving our weakly polynomial algorithm to $O(n^2 \log M \cdot \text{EO} + n^3 \log^{O(1)} n \cdot \log M)$ time. Our application of center of mass to SFM demonstrates that it should be possible.

For strongly polynomial algorithms, the existential result of Theorem 71 shows that SFM can be solved with $O(n^3 \log n \cdot \text{EO})$ oracle calls. Unfortunately, our algorithm incurs an overhead of $\log n$ as there can be as many as $\log n$ buckets each time. One may try to remove this $\log n$ overhead by designing a better bucketing scheme or arguing that more arcs can be added.

The other $\log n$ overhead seem much trickier to remove. Our method currently makes crucial use of the tools developed by [128], where the $\log n$ factors in the runtime seem inevitable. We suspect that our algorithm may have an analogue similar to [231, 217], which do not carry any $\log n$ overhead in the running time.

Perhaps an even more interesting open problem is whether our algorithm is optimal (up to polylogarithmic factors). There are grounds for optimism. So far the best way of *certifying* the optimality of a given solution $S \subseteq V$ is to employ duality and express some optimal solution to the base polyhedron as a convex combination of $n+1$ BFS's. This already takes n^2 oracle calls as each BFS requires n. Thus one would expect the optimal number of oracle calls needed for SFM to be at least n^2. Our bound is not too far off from it, and anything strictly between n^2 and n^3 seems instinctively unnatural.

Part IV

Computing Walrasian Equilibria: Fast Algorithms and Structural Properties

This Part is based on joint works with Renato Paes Leme.

17 Introduction

17.1 A macroscopic view of the market

As part of our everyday experience, prices reach equilibria in a wide range of economics settings. Yet, markets are complicated and consist of heterogeneous goods and a huge population of buyers can have very diverse preferences that are hard to model analytically. With the sheer amount of information needed to describe the economy, how can the market possibly reach an equilibrium? *Well, perhaps not all this information is needed.*

In this Part, we provide evidence supporting this belief through the lens of algorithms. Specifically, we propose algorithms for computing market equilibrium using very limited amount of information. Our result suggests that information theoretically, it is not necessary to make too many measurements or observations of the market to compute an equilibrium. This may also shed light into how markets operate.

As the first step, we must design a realistic model to represent the economy. The standard approach in Theoretical Computer Science would require the entire input be specified but for a market, it is simply too computationally expensive to model its individual agents in full details. So what should we turn to? If equilibrium represents the collective behavior of the agents, perhaps some kind of aggregate information would be enough. Such information can be average salaries, interest rate, population, fashion trend and so on. An algorithm would ideally process these *macroscopic-scale* information in an efficient manner to compute equilibrium price that allows the market to clear.

We show that it is possible to compute market equilibrium by exploiting the very rudimentary information of *aggregate demand*, i.e. the quantity demanded for each item at a given price aggregated over the entire population of buyers. This result implies, among other things, that a market can be viewed as an aggregate entity. For the sake of reaching equilibrium, detailed knowledge about its individual buyers at the microscopic level may not really be needed. Rather, it should be their collective behavior that dictates the outcome of the market.

The use of aggregate demand by our algorithm also resonates with a common perception of the role played by excess demand/supply. A highly sought-after good would usually see its price soar whereas an unpopular good would be inexpensive. This is similar to our algorithms which, in some sense, operate by increasing the price of overdemanded good and vice versa in an iterative fashion. We note however that by no means are we suggesting that our algorithms closely mirror how a market actually works. While the holy grail of this research direction is to understand how a market reaches equilibrium in practice, perhaps a humble first step is to show that this can be done algorithmically with as little information and assumption as possible.

Our starting point is the Gul and Stachetti's model [115] of an economy of (multi-unit) indivisible goods, but we make no further assumptions on the structure of the valuation functions. The goal is to compute market equilibrium: a set of item prices and allocations of items to buyers such that the market clears and each buyer gets his or her favorite bundle of goods under the current prices.

The market can only be accessed via an *aggregate demand oracle*: given prices for each item, what is the demand for each item aggregated over the entire population. Clearly in this model, it is not possible to compute an allocation of items to buyers, since the oracle access model allows no access to buyer-specific information. Curiously, equilibrium prices are still computable and in a very efficient manner:

Theorem 99 (informal)**.** *In a consumer market with n goods and m buyers, we can find a vector of equilibrium prices, whenever it exists, using $\widetilde{O}(n^2)$ calls to the aggregate demand oracle and $\widetilde{O}(n^5)$ time. If valuations are gross substitutes, $\widetilde{O}(n)$ calls to the aggregate demand oracle and $\widetilde{O}(n^3)$ time suffice.*

Notably, the number of buyers plays no role. Our algorithm has query and time complexity essentially independent of the number of buyers. This feature is especially relevant in practice as markets are usually characterized by a large population and relatively few number of goods. The city of Berkeley, for example, has about 350 restaurants but 120,000+ people!

17.2 From telescopes to augmenting lenses

Aggregate demand oracles are like looking at the economy from a telescope. Having a telescope has its advantages: it is possible to get a very global view of the economy with a few queries. On the other hand, extracting details is hard.

Our second question is how fast equilibrium can be computed with only a *local view of the economy*? Our analogue for augmenting lenses will be the *value oracle model*, in which one can query the value of each buyer for each bundle of items. This again has its advantages: it provides very fine-grained information about the market, but has the shortcoming that many queries are needed to extract any sort of global information.

Can equilibrium prices be computed with small amount of information even at the microscopic level? This quest is clearly hopeless for general valuation functions. But for one of the most important classes of valuation functions in economics, *gross substitute valuations*, there are enough structures to allow us to construct equilibrium prices using microscopic information.

The history of *gross substitutes* is intertwined with the development of theory of Walrasian equilibrium (also called market equilibrium in this Part). Indeed, Kelso and Crawford [149] show that Walrasian equilibrium always exists for gross substitute valuations. Hatfield and Milgrom [119] argue that most important examples of valuation functions arising in matching markets belong to the class of gross substitutes. Gross substitutes were used to guarantee the convergence of salary adjustment processes in the job market [149], to guarantee the existence of stable matchings in a variety of settings [227, 159], to show the stability of trading networks [118, 123], to design combinatorial auctions [14, 204] and even to analyze settings with complementarities [243, 117].

Since the oracle access is very local, we clearly need to query each agent at least once, so the dependence on the number of buyers must be at least linear. We show that indeed it is possible to solve this problem with a linear dependence on the number of buyers and cubic dependence in the number of items:

Theorem 100 (informal)**.** *In a consumer market with n goods and m buyers whose valuation functions satisfy the gross substitute condition, we can find an equilibrium (or Walrasian) price and allocation using $mn + \widetilde{O}(n^3)$ calls to the value oracle and $\widetilde{O}(n^3)$ time*[7]*.*

With buyer-specific information, we can also compute the optimal allocation in the same runtime.

Proving this result requires novel insights about the structure of gross substitute valuations. In particular, one of our main structural lemmas answers a question posed by Hsu et al [122]: *when do prices coordinate markets?* In general, Walrasian prices mean that there is a choice of favorite bundles for each buyer that clears the market. It is far from trivial how to choose those bundles, since each agent can have various favorite bundles (for example, consider the case where all items are identical and all agents have the same valuation for the items). We say that a price vector form *robust Walrasian prices* if each agent demands a unique bundle under those prices and the bundles clear the market. Such vectors would allow prices alone to clear the market without any extra coordination. We show that:

Theorem 101 (informal)**.** *In a consumer market with n goods and m buyers whose valuation functions satisfy the gross substitute condition, robust Walrasian prices exist if and only if there is a unique Walrasian allocation (i.e. buyers get the same bundle in any Walrasian equilibrium). Whenever they exist, they can be computed in $\widetilde{O}(mn + n^3)$ time from the Walrasian allocation.*

17.3 Our algorithms and techniques

We study the Walrasian equilibrium problem in three different settings: (i) general valuations in the aggregate demand oracle model; (ii) gross substitute valuations in the aggregate demand oracle model and (iii) gross substitutes in the value oracle model. In all three settings, the starting point is the linear programming formulation of Bikhchandani and Mamer [21].

General valuations in the aggregate demand oracle model (Section 3). The main difficulty in working with the LP in Bikhchandani and Mamer [21] is that the constraints depend on the value of buyers for each bundle, to which we have no access to. In particular, we are not able to test for any given setting of variables of the LP if it is feasible or not. We are in a strange situation that if we knew that a point was infeasible, we could find an appropriate separating hyperplane, and if we knew it was feasible, we could use the objective

[7]The notation $g = \widetilde{O}(f)$ means that $g \leq \alpha f \log^{\beta}(f)$ for some constants $\alpha, \beta > 0$

function as a separating hyperplane, but since we can't test feasibility, we can't decide which to use. Our solution is to move the constraints to the objective function and turn the problem into an unconstrained convex minimization problem. The problem in hand has the feature that we can't evaluate the function but we can compute its subgradients.

Traditional cutting plane algorithms such as the Ellipsoid Method need access to both a separation oracle and functional values. To overcome this issue we use the fact that the cutting plane of Lee, Sidford and Wong [178] provides strong dual guarantees and that our separation oracle is given by subgradients. Originally, [178] show how to adapt their cutting plane method to convex optimization, however, their algorithm and proof still rely on being able to evaluate the function. We show in our Theorem 114 that their algorithm can be slightly modified to achieve the same guarantee using only subgradients, (i.e., without using functional values).

A second obstacle we face is that algorithms to minimize convex functions only provide approximate guarantees and to find a Walrasian equilibrium we need the exact minimum. In general minimizing a convex function exactly is impossible, but in our case, this can be done by exploiting the connection to the LP. Note that given the very restricted way that we can access the problem (only via the aggregate demand oracle), we can't apply the techniques to perturb and round the linear programming in a black-box fashion. Khachiyan's approach [152] can be adapted to our setting, but can't be applied directly since we don't have a proper LP. We show how to perturb and round the objective function to achieve the desired running time.

Gross substitutes in the aggregate demand model (Section 4). If valuation functions satisfy gross substitutes, then we can exploit the fact that the set of Walrasian prices form an integral polytope with a lattice structure to simplify the algorithm and obtain an improved running time and oracle call complexity. The improvement comes from using structural properties of gross substitutes to show that a simpler and milder perturbation to the objective is enough and that rounding can be done in a simpler manner. This highlights the importance of looking at perturbation and rounding in a non-black-box manner.

Gross substitutes in the value oracle model (Section 5). An aggregate demand oracle call can be simulated from $O(mn^2)$ value oracles calls. This can be plugged into the previous algorithm to obtain a running time of $\tilde{O}(mn^3 T_V)$ where T_V is the time required by the value oracle. We use two ideas to improve the running time to $\tilde{O}((mn+n^3)T_V)$. The first one is to regularize the objective function. As with the use of regularizers in other context in optimization, this is to penalize the algorithm for being too aggressive. The bound of $O(mn^2)$ value oracle calls per iteration of the cutting plane algorithm is so costly precisely because we are trying to take an aggressively large step. A second idea is to re-use one of the stages of the subgradient computation in multiple iterations, amortizing its cost per iteration.

Robust Walrasian Prices and Market Coordination (Section 6). Still in the value oracle model, we show how to obtain the efficient allocation from the subgradients observed in the optimization procedure. An important by-product of our analysis is that we give necessary and sufficient conditions for the existence of robust Walrasian prices, i.e., Walrasian prices under which each buyer has a unique bundle in their demand set. Whenever such prices exist, we give an $\tilde{O}(mn + n^3)$ algorithm to compute them. This answers an open question in Hsu et al [122], who ask when it is possible to completely coordinate markets by solely using prices.

Combinatorial Algorithms for Walrasian equilibrium (Section 7). Murota and Tamura [205] gave combinatorial algorithms for the problem of computing Walrasian equilibria via a reduction to the M-convex submodular flow problem. It is also possible to obtain combinatorial algorithms for the welfare problem by reducing it to the valuated matroid intersection problem and applying the algorithms in Murota [199, 200]. The running time is not explicitly analyzed in [205, 199, 200]. Here we describe those algorithms for the reader unfamiliar with M-convex submodular flows in terms of a sequence of elementary shortest path computations and analyze its running time. We show that they have running time of $\tilde{O}(mn^3)$ in the value oracle model. We use the same ideas used to speed up the computation of Walrasian prices by regularizing the market potential to speed up those algorithms and improve its running time to $\tilde{O}(mn + n^3)$.

17.4 Comparison to related work

Iterative auctions and subgradient algorithms. The first algorithm for computing Walrasian equilibria in an economy of indivisible goods is due to Kelso and Crawford [149] and it is inspired by Walras' tâtonnement procedure [257], which means "trial-and-error". Despite the name, it constitutes a very ingenious greedy algorithm: goods start with any price, then we compute the aggregate demand of the agents, increase the price by one for all goods that were over-demanded and decrease by one the price of all goods that are under-demanded. This gives a very natural and simple algorithm in the aggregate demand oracle model. This algorithm, however, is not polynomial time since it runs in time proportional to the magnitude of the valuations.

The seminal work of [149] originated two lines of attack of the problem of computing Walrasian equilibria: the first line is by applying subgradient descent methods [219, 220, 14]. Such methods typically either only guarantee convergence to an approximate solution or converge in pseudo-polynomial time to an exact solution. This is unavoidable if subgradient descent methods are used, since their convergence guarantee is polynomial in M/ϵ where M is the maximum valuation of a buyer for a bundle and ϵ is the accuracy of the desired solution. To be able to round to an *exact* optimal solution the running time must depend on the magnitude of the valuations. Another family of methods is based on primal-dual algorithms. We refer to de Vries, Schummer and Vohra [55] for a systematic treatment of the topic. For primal-dual methods to converge exactly, they need to update the prices slowly – in fact, in [55] prices change by one unit in each iteration – causing the running time to be pseudo-polynomial time.

Polynomial time approaches via the Ellipsoid Method. The Welfare Problem for gross substitutes was independently shown to be solvable in polynomial by Murota [200] and

Nisan and Segal [215]. Remarkably, this was done using completely different methods.

Nisan and Segal's approach is based on a linear programming formulation of Walrasian equilibrium due to Bikhchandani and Mamer [21]. The authors show that the dual of this formulation can be solved using both the value and demand oracles for gross substitutes as a separation oracle for the LP. This can be combined with the fact that demand oracles for gross substitutes can be constructed from value oracles in $O(n^2)$ time [65] to obtain a polynomial-time algorithm in value oracle model.

This is the method that is closer to ours in spirit: since we both approach the problem via a mathematical programming formulation and apply interior point methods. In terms of oracle access, Nisan and Segal crucially rely on value oracles to implement the separation oracle in their LP – so their solution wouldn't generalize to the aggregate demand oracle model, since neither per-agent demand oracles nor value oracles can be recovered from aggregate demand oracle[8]. The running time in their paper is never formally analyzed, but since their formulation has $m+n$ variables, it would lead to a superlinear dependence in the number of agents.

Nisan and Segal employ the LP to compute a set of Walrasian prices and the value of the Walrasian allocation. In order to compute the allocation itself, they employ a clever technique called *self-reducibility*, commonly used in complexity theory. While it allows for an elegant argument, it is a very inefficient technique, since it requires solving nm linear programs. In total, this would lead to a running time of $O(mn^2(m+n)^3)$ using currently fastest cutting plane algorithms as the LP solver.

Combinatorial approaches. A second technique was developed by Murota [199, 200] and leads to very efficient combinatorial algorithms. Murota's original paper never mentions the term "gross substitutes". They were developed having a different object in mind, called *valuated matroids*, introduced by Dress and Wenzel [63, 62] as a generalization of the Grassmann-Plücker relations in p-adic analysis. Murota developed a strongly-polynomial time algorithm based on network flows for a problem called the *valuated matroids assignment problem*. There is a tortuous path connecting gross substitutes to valuated matroids. Valuated matroid turned out to be one aspect of a larger theory, Discrete Convex Analysis, developed by Murota (see his book [197] for a comprehensive discussion). One central object of this theory is the concept of $M^\natural$-concave functions, introduced by Murota and Shioura [203]. It came to many as a surprise when Fujishige and Yang [92] showed that $M^\natural$-concave functions and gross substitutes are the same thing. Their equivalence is highly non-trivial and their definitions are very different to the point it took at least a decade for those concepts to be connected. Murota and Tamura [205] later apply the ideas in discrete convex analysis to give polynomial time algorithms to various equilibrium problems in economics. The running time is never explicitly analyzed in their paper. Here we show that their running time is $\tilde{O}(mn^3)$ and improve it to $\tilde{O}(mn+n^3)$

[8]Note that the construction of Blumrosen and Nisan [24] to construct value oracles from demand oracles crucially requires *per-buyer* demand oracles. The same construction doesn't carry over to aggregate demand oracles.

Market Coordination Related to Section 22 in our work is the line of research on Market Coordination. This line of inquiry was initiated by Hsu, Morgenstern, Rogers, Roth and Vohra [122] who pose the question of when prices are enough to coordinate markets. They argue that the minimum vector of Walrasian prices always causes ties and bound how much overdemand can be caused by any given vector of Walrasian prices for unit-demand valuations and matroid based valuations. They ask such questions in both the traditional market model of Gul and Stachetti and in a stochastic setting where valuations are drawn from distributions. In the traditional market model, we answer the question in the title of their paper for the general class of gross substitutes, by giving necessary and sufficient conditions for markets to be coordinated by only using prices (Theorem 138). We also give a simple algorithm for computing those prices whenever they exist. Such necessary and sufficient conditions were given simultaneously and independently by Cohen-Addad, Eden, Feldman and Fiat [48]

18 Preliminaries

Let $\mathbb{Z}$ be the set of integers, $\mathbb{Z}_{\geq 0}$ be the set of non-negative integers. For any $k \in \mathbb{Z}_{\geq 0}$, we define $[k] := \{1, 2, \ldots, k\}$ and $[\![k]\!] = \{0, 1, \ldots, k\}$. Given $\vec{cs} = (s_1, \ldots, s_n) \in \mathbb{Z}^n_{\geq 0}$, we define $[\![\vec{cs}]\!] = \prod_{j=1}^n [\![s_j]\!]$.

18.1 Market equilibrium: prices, allocations and welfare

Following the classic model of Gul and Stachetti [115], we define a *market of indivisible goods* as a set $[m]$ of buyers, a set $[n]$ of items and a supply $s_j \in \mathbb{Z}_{\geq 0}$ of each item j. Each buyer i has a valuation function $v_i : \mathbb{Z}^n_{\geq 0} \to \mathbb{Z}$ over the multisets of items with $v_i(0) = 0$. For the first part of our work, we make no further assumptions about the valuation function or the supply.

Given a price vector $\vec{cp} \in \mathbb{R}^n$, we define the *utility* of agent i for a bundle $x \in [\![\vec{cs}]\!]$ under price vector $\vec{cp}$ as: $u_i(x; \vec{cp}) := v_i(x) - \vec{cp} \cdot x$, where $\vec{cp} \cdot x$ refers to the standard dot product $\sum_{j=1}^n p_j x_j$. Notice also that we make no assumptions about the signs of v_i and $\vec{cp}$.

An *allocation* $\vec{cx} = (x^{(1)}, x^{(2)}, \ldots, x^{(m)})$ is simply an assignment of items to each agent where i receives $x^{(i)} \in [\![\vec{cs}]\!]$. An allocation $\vec{cx}$ is *valid* if it forms a partition of the items, i.e. $\sum_i x_j^{(i)} = s_j$ for every j. The *social welfare* of a valid allocation $\vec{cx}$ is defined as $\mathrm{Sw}(\vec{cx}) = \sum_{i \in [m]} v_i(x^{(i)})$. Finally, the optimal social welfare is simply the largest possible social welfare among all valid allocations.

Given prices $\vec{cp} \in \mathbb{R}^n$, we would expect a rational agent to buy x such that his utility $v_i(x) - \vec{cp} \cdot x$ is maximized. We call x the demand of i under $\vec{cp}$, as defined formally below. Note that there may be multiple utility-maximizing subsets.

Definition 102 (Demand set)**.** Given prices $\vec{cp} \in \mathbb{R}^n$ on each item, the demand set $D(v, \vec{cp})$ for a valuation function v is the collection of vectors (bundles) for which the utility is maximized:

$$D(v, \vec{cp}) := \arg\max_{x \in [\![\vec{cs}]\!]} v(x) - \vec{cp} \cdot x.$$

If v is the valuation function v_i of agent i, we also use the shorthand $D(i, \vec{cp})$ as the demand set.

Definition 103 (Equilibrium)**.** A *Walrasian equilibrium* (also called *competitive equilibrium*) consists of a price vector $\vec{cp} \in \mathbb{R}^n$ and a valid allocation $\vec{cx} = (x^{(1)}, x^{(2)}, \ldots, x^{(m)})$ s.t. $x^{(i)} \in D(i, \vec{cp}) \forall i$. We call $\vec{cp}$ a *Walrasian/equilibrium price* and $\vec{cx}$ a *Walrasian/equilibrium allocation* induced by $\vec{cp}$.

In other words, a competitive equilibrium describes a state where items are sold in such a way that the total demand $\sum_i x_j^{(i)}$ for each item precisely meets its supply s_j, i.e. the market *clears*. The reason for the name *competitive* equilibrium is that it achieves the optimal social welfare. This is known as the first and second welfare theorems in economics.

Lemma 104 (First & Second Welfare Theorems)**.** *Let $\vec{cx}$ be an equilibrium allocation induced by an equilibrium prices $\vec{cp}$. Then $\vec{cx}$ achieves the optimal social welfare. Moreover, if $\vec{cp}$ is any Walrasian price and $\vec{cx}$ is any optimal allocation, then the pair $(\vec{cp}, \vec{cx})$ form a Walrasian equilibrium.*

This nicely reduces social welfare maximization to finding competitive equilibrium whenever it exists. Note that the definition of social welfare has nothing to do with prices. In a way, this lemma shows that equilibrium prices act as a certificate for optimality of equilibrium allocation.

18.2 Oracles

To study market equilibrium computationally, we must clarify how we can extract information about the market. In this Part we consider three models that access the market in different scales:

Microscopic Scale: Value Oracle Model Here the algorithm has access to the value that each agent has for each bundle. This gives the algorithm very fine-grained information, but may require many calls for the algorithm to access any sort of macroscopic information about the market.

Definition 105 (Value oracle)**.** The value oracle for agent $i \in [m]$ takes $x \in [\![\vec{cs}]\!]$ as input and outputs $v_i(x)$. We denote by T_V the time spent by the value oracle to answer a query.

Agent Scale: Demand Oracle Model Here the algorithm presents a price vector $\vec{cp}$ to an agent and obtains how many units are demanded at $\vec{cp}$. At any given price, the agent could be indifferent between various bundles. The demand oracle can return an arbitrary bundle.

Definition 106 (Demand oracle)**.** The demand oracle for agent i takes as input a price vector $\vec{cp} \in \mathbb{R}^n$ and outputs a demand vector $d_i(\vec{cp}) \in D(i, \vec{cp})$. We denote by T_D the time spent by the demand oracle to answer a query.

Macroscopic (Market) Scale: Aggregate Demand Oracle Model The *aggregate demand oracle model* presents a macroscopic view of the market where algorithms cannot observe individual agents but only their aggregate response to any given price $\vec{cp}$.

Definition 107 (Aggregate Demand oracle)**.** The aggregate demand oracle takes as input a price vector $\vec{cp} \in \mathbb{R}^n$ and outputs a demand vector $d(\vec{cp}) \in \mathbb{Z}_{\geq 0}$ such that there exist bundles $x^{(i)} \in D(i, \vec{cp})$ satisfying $d(\vec{cp}) = \sum_i x^{(i)}$. We denote by T_{AD} the time spent to answer a query.

19 Walrasian equilibrium for General Valuations in $\tilde{O}(n^2 \cdot T_{AD} + n^5)$

We show that in the aggregate demand oracle model, whenever a Walrasian equilibrium exists, it can be computed using $\tilde{O}(n^2)$ aggregate demand oracles calls and $\tilde{O}(n^2 \cdot T_{AD} + n^5)$ time. We emphasize that our result is in the *aggregate* demand oracle model – which is the typical information available to markets: *which goods are under- and over-demanded by the population of buyers?* Previous polynomial-time algorithms for computing Walrasian equilibria [21, 215, 200, 205] require buyer-level demand oracles or value oracles. Previous algorithms that use only aggregate demand oracles (such as [149, 115, 14, 220, 55] and related methods based on ascending auctions) are pseudopolynomial since they depend linearly on the magnitude of the valuation functions.

First we discuss a linear programming formulation for this problem and a corresponding convex programming formulation. The formulation itself is fairly standard and appears in various previous work. When we try to find an *exact* minimizer of this formulation in polynomial time using only aggregate demand oracles, we encounter a series of obstacles. The main ones are: (i) how to optimize a function we cannot evaluate; and (ii) how to find an exact minimizer using convex optimization (which typically gives only an approximate guarantee). In both cases, we must apply optimization algorithms in a *non-black-box* manner and exploit special structures of the problem.

Linear Programming Formulation We start from the formulation of Bikhchandani and Mamer [21] (also studied by Nisan and Segal [215]) of the Walrasian equilibrium problem as a linear program. Consider the following primal-dual pair:

$$\begin{aligned} \max_z \sum_{i\in[m], x\in[\![\vec{cs}]\!]} & v_i(x)\cdot z_{i,x} \text{ s.t.} \\ \sum_x z_{i,x} &= 1, \quad \forall i \quad (u_i) \\ \sum_{i,x} x_j \cdot z_{i,x} &= s_j, \quad \forall j \quad (p_j) \\ z_{i,x} &\geq 0, \quad \forall i, x \end{aligned}$$

$$\begin{aligned} \min_{(\vec{cp},\vec{cu})} \sum_i u_i + \vec{cp}\cdot\vec{cs} & \text{ s.t.} \\ u_i \geq v_i(x) - \vec{cp}\cdot x, & \quad \forall i, x \quad (z_{i,x}) \end{aligned}$$

Lemma 108 ([21]). *A market equilibrium $(\vec{cp}^{eq}, \vec{cx})$ exists iff the primal program has an integral solution. In such case, the set of equilibrium prices is the set of solutions to the dual LP projected to the $\vec{cp}$-coordinate.*

We provide a proof of the previous lemma in Section 24 for completeness. This lemma reduces the problem of finding a Walrasian equilibrium, whenever it exists, to the problem of solving the dual program. The approach in Nisan and Segal [215] is to use a (per buyer) demand oracle as a separation oracle for the dual program. Given that we care only about the $\vec{cp}$ variables, we can reformulate the dual as:

$$\begin{aligned} &\min_{(\vec{cp},u)} u + \vec{cp} \cdot \vec{cs} \text{ s.t.} \\ &\quad u \geq \sum_i v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)}, \quad \forall x^{(i)} \in [\![\vec{cs}]\!] \end{aligned} \tag{D}$$

It is simple to see that for every feasible vector $(\vec{cp}, \vec{cu})$ of the original dual, we can find a corresponding point $(\vec{cp}, \sum_i u_i)$ of the transformed dual with the same value. Conversely, given a feasible point $(\vec{cp}, u)$ of the transformed dual, we can come up with a point $(\vec{cp}, \vec{cu})$ of the original dual with equal or better value by setting $u_i = \max_x v_i(x) - \vec{cp} \cdot x$.

Thus it suffices to find an optimal solution to the transformed dual program. The separation problem is now simpler: consider a point $(\vec{cp}_0, u_0)$ that is infeasible. If some constraint is violated, then it must be the constraint for $x^{(i)}$ maximizing $\sum_i v_i(x^{(i)}) - \vec{cp}_0 \cdot x^{(i)}$, so $u_0 < \sum_i v_i(x^{(i)}) - \vec{cp}_0 \cdot x^{(i)}$. For all feasible $(\vec{cp}, u)$ we know that $u \geq \sum_i v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)}$. Therefore:

$$u - u_0 + (\vec{cp} - \vec{cp}_0) \cdot \vec{cd}(\vec{cp}_0) \geq 0$$

is a valid separating hyperplane, where $\vec{cd}(\vec{cp}_0) = \sum_i x^{(i)}$ is the output of the aggregate demand oracle. If on the other hand $(\vec{cp}_0, u_0)$ is feasible, we can use the objective function to find a separating hyperplane, since for any optimal solution $(\vec{cp}, u)$ we know that $u + \vec{cp} \cdot \vec{cs} \leq u_0 + \vec{cp}_0 \cdot \vec{cs}$. So we can separate it using:

$$u - u_0 + (\vec{cp} - \vec{cp}_0) \cdot \vec{cs} \leq 0$$

An Obstacle The main obstacle to this approach is that since the aggregate demand oracle has no access to the value of $\sum_i v_i(x^{(i)})$, one cannot construct a separation oracle from the aggregate demand oracle.

Convex Programming Formulation One way to get around this obstacle is to further transform the dual program to get rid of utility variable u altogether. We do so by moving the constraints to the objective function in the form of penalties. The LP then becomes the problem of minimizing a convex function. The same function has been used as a potential function to analyze price adjustment procedures [15] and combinatorial algorithms [204] and is the basis for the family of algorithms known as subgradient algorithms (e.g. [219, 220, 14, 41]).

Given a market with supply $\vec{cs}$ and agents with valuations $v_1, \ldots, v_m$, we define the market potential function $f : \mathbb{R}^n \to \mathbb{R}$ as:

$$f(\vec{c}p) = \sum_{i=1}^{m} \left(\max_{x \in [\![\vec{c}s]\!]} v_i(x) - \vec{c}p \cdot x \right) + \vec{c}p \cdot \vec{c}s. \tag{C}$$

Since f is nothing more than the dual linear program with the constraints moved to the objective function, the set of minimizers of f is exactly the set of optimal solutions of the dual linear program (or more precisely, their projections to the $\vec{c}p$-variable). Each term $\max_{x \in [\![\vec{c}s]\!]} v_i(x) - \vec{c}p \cdot x$ is a convex function in $\vec{c}p$ since it is a maximum over linear functions. Hence f is convex since it is a sum of convex functions.

One remarkable fact about f is that we can obtain a subgradient from the aggregate demand oracle:

Lemma 109 (Subgradient oracle)**.** *Let $\vec{c}d(\vec{c}p)$ be the aggregate demand, then $\vec{c}s - \vec{c}d(\vec{c}p)$ is a subgradient of f in $\vec{c}p$.*

Proof. From the Envelope Theorem, a subgradient of $\max_{x \in [\![\vec{c}s]\!]} v_i(x) - \vec{c}p \cdot x$ is given by the subgradient of $v_i(x^{(i)}) - \vec{c}p \cdot x^{(i)}$ for $x^{(i)} \in \arg\max_x v_i(x) - \vec{c}p \cdot x$. Since $v_i(x^{(i)}) - \vec{c}p \cdot x^{(i)}$ is a linear function in $\vec{c}p$, its gradient is simply $-x^{(i)}$. So, for any $x^{(i)}$ in the arg max, the vector $\vec{c}s - \sum_i x^{(i)}$ is a subgradient o f f. In particular, $\vec{c}s - \vec{c}d(\vec{c}p)$. □

A useful fact in studying this function is that we have an initial bound on the set of minimizers:

Lemma 110. *If there exists a Walrasian equilibrium, then the set of minimizers $P := \arg\min_{\vec{c}p} f(\vec{c}p)$ is contained in the box $[-2M, 2M]^n$ for $M = \max_i \max_{x \in [\![s]\!]} |v_i(x)|$.*

Proof. Let $\vec{c}p \in P$ be a vector of equilibrium prices and $\vec{c}x$ an optimal allocation. Since the pair $(\vec{c}p, \vec{c}x)$ constitute a Walrasian equilibrium, then for any item j, there is some $x_j^{(i)} \geq 1$ and therefore,

$$v_i(x^{(i)}) - \vec{c}p \cdot x^{(i)} \geq v_i(x^{(i)} - \vec{c}1_j) - \vec{c}p \cdot (x^{(i)} - \vec{c}1_j)$$

where $\vec{c}1_j$ is the unit vector in the j-th coordinate. This gives us: $p_j \leq v_i(x^{(i)}) - v_i(x^{(i)} - \vec{c}1_j) \leq 2M$.

For the lower bound, if there is more than one buyer then p_j must be larger than $-2M$, otherwise all the supply of item j will be demanded by all buyers and therefore $\vec{c}p$ cannot be Walrasian. □

Lemma 111. *The set of Walrasian prices is a polytope P whose vertices have coordinates of the form $p_j = a_j / b_j$ with $a_j, b_j \in \mathbb{Z}$ and $|b_j| \leq (Sn)^n$ for $S = \max_j s_j$.*

Proof. The vertices of P correspond to $\vec{c}p$-coordinates of the basic feasible solutions of the dual linear program. Given that the coefficients of the $\vec{c}p$ variables in the linear program are integers from 0 to S, solving the linear system using Cramer's rule (see Section 5 in [23]) we get that every solution must be a fraction with denominator at most $n! \cdot S^n \leq (Sn)^n$. □

Two More Obstacles The natural approach at this point is to try to apply convex optimization algorithms as a black box to optimize f. There are two issues with this approach. The first, which is easier to address, is that unlike algorithms for linear programming which are exact, algorithms for general convex optimization give only ϵ-approximation guarantees. We will get around this issue by exploiting the connection between f and the linear program from which it is derived.

A second more serious obstacle is that we don't have access to the functional value of f, only to its subgradient. This is a problem for the following reasons. Traditional cutting plane methods work by keeping in each iteration t a set G_t called a *localizer*. The localizer is a subset of the domain which is guaranteed to contain the optimal solution. We start with a large enough localizer set G_0 that is guaranteed to contain the solution. In each iteration t, a point $\vec{cp}_t \in G_{t-1}$ is queried for the subgradient $\partial f(\vec{cp}_t)$. Now, if $\vec{cp}^*$ is an optimal solution we know that:

$$0 \geq f(\vec{cp}^*) - f(\vec{cp}_t) \geq \partial f(\vec{cp}_t) \cdot (\vec{cp}^* - \vec{cp}_t)$$

where the first inequality comes from the fact $\vec{cp}^*$ is an optimal solution and the second inequality comes from the definition of the subgradient. This in particular means that any optimal solution must be contained in $H_t = G_{t-1} \cap \{\vec{cp}; \partial f(\vec{cp}_t) \cdot (\vec{cp} - \vec{cp}_t) \leq 0\}$. The method then updates G_t to either H_t or a superset thereof. The Ellipsoid Methods, for example, updates G_t to the smallest ellipsoid containing H_t. So far, all the steps depend only on the subgradient and not on the actual functional values of $f(\vec{cp}_t)$. The guarantee of the method, however, is that after a sufficient number of steps, one of the iterates is close to the optimal, i.e., $\min_t f(\vec{cp}_t) - f(\vec{cp}^*) \leq \epsilon$. To find the approximate minimizer, however, we need to look at the actual functional values, which we have not access to. See Section 2.2 in Nemirovski [208] for a complete discussion on the guarantees provided by cutting plane methods.

A special case : Walrasian prices with non-empty interior The set of Walrasian prices P forms a convex set, since it corresponds to the set of minimizers of the convex function f. If P has non-zero volume, it is possible to find a Walrasian equilibrium using the Ellipsoid Method without needing to know the functional value. If the set is large, the Ellipsoid method cannot avoid querying a point in the interior of P for too long. And when a point in the interior of P is queried, the only subgradient is zero, or in market terms, each agent has a unique favorite bundle and those bundles clear the market. This means that the aggregate demand oracle returns exactly the supply. Next we make this discussion formal:

Theorem 112. *Assume that the set of Walrasian prices P has non-zero volume, then the Ellipsoid method*[9] *is guaranteed to query a Walrasian price $\vec{cp}^*$ for which the aggregate demand oracle returns $\vec{cd}(\vec{cp}^*) = \vec{cs}$ in $\tilde{O}(n^3)$ iterations.*

Proof. By Lemma 110, $P \subseteq [-2M, 2M]^n$, so we can take the initial set of localizers G_0 in the ellipsoid methods to be the ball of radius $2M\sqrt{n}$ around 0, which has volume $O(M^n)$. The Ellipsoid guarantee (see Section 3 of [208]) is that the volume of G_t is at most $e^{-t/2n}$ of the initial volume. So if the method hasn't queries any point in the interior of P, then we must have G_t containing P.

[9]One may use cutting plane methods to derive a better running time but for simplicity we omit doing it for this special case.

To bound the volume of P we use the fact that if a polytope has vertices $\vec{c}p_0, \vec{c}p_1, \ldots, \vec{c}p_n$ then the volume of the polytope is lower bounded by the volume of the convex hull of those vertices:

$$\text{vol}(P) \geq \frac{1}{n!} \det \begin{bmatrix} 1 & 1 & \ldots & 1 \\ \vec{c}p_0 & \vec{c}p_1 & \ldots & \vec{c}p_n \end{bmatrix} \geq \frac{1}{n!} \left(\frac{1}{O((Sn)^n)} \right)^n = \Omega((Sn)^{-n^2})$$

where the last inequality follows from Lemma 111.

Therefore, if G_t contains P then: $O(M^n)e^{-t/2n} \geq \Omega((Sn)^{-n^2})$ which implies that $t \leq O(n^3 \log(Sn) + n^2 \log(M)) = \tilde{O}(n^3)$. So after so many iterations we are guaranteed to query a point in the interior of P. For a point $\vec{c}p$ in the interior of P, there is a small ball around it for which f is flat, so zero must be the unique subgradient at that point. Therefore, the aggregate demand oracle has no choice but to return $\vec{c}d(\vec{c}p) = \vec{c}s$. $\square$

The main drawback of this method is that it strongly relies on the fact that P has non-empty interior. Some very simple and well-behaved markets have a set of Walrasian prices of zero volume. Consider for example a market with two items with supply one of each and three buyers, each having valuation $v(0) = 0$ and $v(x) = 1$ otherwise. The set of Walrasian prices is $P = \{(1, 1, 1)\}$. Even for $\vec{c}p^* = (1, 1, 1)$, the subgradient is not unique since the aggregate demand oracle can return any $\vec{c}d(\vec{c}p^*) = (d_1, d_2)$ for $0 \leq d_1 + d_2 \leq 3$ and $d_1, d_2 \in [\![3]\!]$. In this case even trying to modify the objective function is hopeless, since there is no point in the domain for which the aggregate demand oracle is guaranteed to return the supply vector. Since there is no point for which the subgradient oracle is guaranteed to return zero, there is no direct way to recognize a vector of Walrasian prices even if we see one!

Approach for the general case Our approach for optimizing f is as follows: first we show how to obtain an ϵ-approximate minimizer of f in time $O(n \log(nM/\epsilon) T_{AD} + n^3 \log^{O(1)}(nM/\epsilon))$. In order to round the ϵ-approximate solution to the optimal solution, we exploit the fact that f came from a linear program and customize the traditional approach of Khachiyan [152] for rounding approximate to exact solutions.

The idea is to use Lemma 111 to argue that if f has a unique minimizer, and we set ϵ small enough, then all ϵ-approximate solutions are in a small neighborhood of the exact optimal. Moreover, for small enough ϵ, there should be only one point in the format indicated by Lemma 111 in a small neighborhood of the approximate minimizer, so we can recognize this as the exact solution by rounding.

For this approach to work, we need f to have a unique minimizer. To achieve that, we perturb the objective function f to $\hat{f}$ in such a way that $\hat{f}$ has a unique minimizer and that this minimizer is still a minimizer of f. There are several ways to implement this approach, the simplest of which uses the isolation lemma (see Lemma 116). We note that the isolation lemma was not available to Khachiyan at the time so he had to resort to something more complicated.

A Cutting Plane Method without using functional values The first part of the algorithm consists in obtaining an ϵ-approximate minimizer of f without using its functional value. In order to do so, we use our previous result on the cutting plane method. Recall that we show an efficient algorithm which either identifies a point inside the desired convex set K or *certifies* that K contains no ball of radius ϵ. We show that our previous main theorem can be used to compute approximate minimizers of convex functions using only the subgradient. Note that our previous application of the main theorem to minimizing convex functions relies on using functional values as we did not assume a separation oracle given by subgradients.

For readers' convenience, our previous main theorem is restated below:

Theorem 113 (Lee, Sidford, Wong (Theorem 31 in [179])). *Let $K \subseteq [-M, M]^n$ be a convex set and consider a separation oracle that can be queried for every point in $\vec{cp} \in [-M, M]^n$ and will either return that $\vec{cp} \in K$ or return a half-space $H = \{\vec{cp} \in \mathbb{R}^n; \vec{ca} \cdot \vec{cp} \leq b\}$ containing K. Then there exists an algorithm with running time $O(nT \log(nM/\delta) + n^3 \log^{O(1)}(nM) \log^2(nM/\delta))$[10] that either produces a point $\vec{cp} \in K$ or finds a polytope $P = \{\vec{cp} \in \mathbb{R}^n; \vec{ca}_i \cdot \vec{cp} \leq b_i, i = 1, \ldots, k\}$, $k = O(n)$ such that:*

- *The constraints $\vec{ca}_i \cdot \vec{cp} \leq \vec{cb}$ are either from the original box, $p_i \leq M$ or $p_i \geq -M$ or are constraints returned by the separation oracle normalized such that $\|\vec{ca}_i\|_2 = 1$.*
- *The width of P is small, i.e., there is a vector $\vec{ca}$ with $\|\vec{ca}\|_2 = 1$ such that*
$$\max_{\vec{cp} \in P} \vec{ca} \cdot \vec{cp} - \min_{\vec{cp} \in P} \vec{ca} \cdot \vec{cp} \leq O(n\delta \log(Mn/\delta))$$
- *The algorithm produces a certificate of the previous fact in the form of a convex combination $t_1, \ldots, t_k$ with $t_i \geq 0$, $\sum_{i=1}^k t_i = 1$ and $k = O(1)$ such that:*
  - $\|\sum_{i=1}^k \vec{ca}_i\|_2 \leq O\left(\frac{\delta}{M}\sqrt{n} \log\left(\frac{M}{\delta}\right)\right)$
  - $\left|\sum_{i=1}^k b_i\right| \leq O\left(n\delta \log\left(\frac{M}{\delta}\right)\right)$

Now, we show that this result can be used to obtain a convex minimization algorithm that uses only subgradients:

Theorem 114. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be an L-Lipschitz convex function equipped with a subgradient oracle with running time T. If f has a minimizer in $[-M, M]^n$ we can find a point $\vec{\varphi}$ such that $f(\vec{\varphi}) - \min_{\vec{cp}} f(\vec{cp}) \leq \epsilon$ in time $O(nT \log(nML/\epsilon) + n^3 \log^{O(1)}(nML) \log^2(nML/\epsilon))$.*

Proof. Let $K = \text{argmin}_{\vec{cp}} f(\vec{cp})$ and use the subgradient as the separation oracle, since if $\partial f(\vec{cp}_t)$ is the subgradient at $\vec{cp}_t$ we know that for all $\vec{cp} \in K$, it holds that $\partial f(\vec{cp}_t) \cdot (\vec{cp} - \vec{cp}_t) \leq 0$. What we will do is to run the algorithm in Theorem 113 starting from a very large box $[-M', M']$ for $M' = n^{O(1)} M$ instead of starting from $[-M, M]$, which appears to be more natural. The reason we do that is to avoid having the constraints defining the bounding box added to P.

Either the algorithm will return a point $\vec{cp} \in K$, in which case we are done or will return a set P like in statement of the theorem. If we are lucky and all constraints added to P are

[10]The running time stated beofre has $\log^{O(1)}(nM/\delta)$ dependence which is, upon a closer examination, actually $\log^{O(1)}(nM) \log^2(nM/\delta))$. (The difference was ignored because both runtimes give the same result for previous applications)

of the type $\partial f(\vec{c}p_t) \cdot \vec{c}p \le \partial f(\vec{c}p_t) \cdot \vec{c}p_t$, then we can use the certificate $t_1, \dots, t_k$ to produce a point $\bar{\vec{p}} = \sum_{i=1}^k t_i \vec{c}p_i$. Now:

$$\begin{aligned}
f(\bar{\vec{p}}) - f(\vec{c}p^*) &\le \sum_i t_i f(\vec{c}p_i) - f(\vec{c}p^*) \le \sum_i \partial f(\vec{c}p_i) \cdot (\vec{c}p_i - \vec{c}p^*) \\
&\le L \cdot \left[\Big\| \sum_i t_i \vec{c}a_i \Big\| \cdot \|\vec{c}p^*\| + \left| \sum_i t_i b_i \right| \right] \\
&\le L \cdot \left[O\left(\frac{\delta}{M'} \sqrt{n} \log\left(\frac{M'}{\delta}\right)\right) \cdot M'\sqrt{n} + O\left(n\delta \log\left(\frac{M'}{\delta}\right)\right)\right] \\
&= O\left(n\delta L \log\left(\frac{M'}{\delta}\right)\right)
\end{aligned}$$

Then setting δ such that $\epsilon = O\left(n\delta L \log\left(\frac{M'}{\delta}\right)\right)$ gives us the desired result.

To be done, we just need to worry about the case where some of the box constraints are present in P. In that case, we argue that the weight put by the coefficients t_i on the box constraints must be tiny, so it is possible to remove those and rescale t. Formally, observe that

$$\sum_i t_i(b_i - \vec{c}a_i \cdot \vec{c}p^*) \le \left| \sum_i t_i b_i \right| + \Big\| \sum_i t_i \vec{c}a_i \Big\|_2 \cdot \|\vec{c}p^*\|_2 = O\left(n\delta \log\left(\frac{M'}{\delta}\right)\right)$$

Now, if i is an index corresponding to a box constraint such as $p_i \le M'$ or $p_i \ge -M'$ then $b_i - \vec{c}a_i \cdot \vec{c}p^* \ge |M' - M| = \Omega(n^{O(1)}M)$. This implies in particular that:

$$O\left(n\delta \log\left(\frac{M'}{\delta}\right)\right) \ge \sum_i t_i(b_i - \vec{c}a_i \cdot \vec{c}p^*) \ge t_i(b_i - \vec{c}a_i \cdot \vec{c}p^*) \ge t_i \Omega(n^{O(1)}M)$$

so $t_i \le O\left(\frac{n^{-O(1)}\delta}{M} \log\left(\frac{M'}{\delta}\right)\right)$. By choosing a very small δ, say $\delta = O\left(\frac{\epsilon}{Ln^{O(1)}M^{O(1)}}\right)$ we guarantee that is B is the set of indices corresponding to box constraints, then $\sum_{i\in B} t_i \le O\left(\frac{\epsilon}{n^{O(1)}M^{O(1)}}\right) \le \frac{1}{2}$ and that $\sum_i t_i(b_i - \vec{c}a_i \cdot \vec{c}p^*) \le O\left(\frac{\epsilon}{Ln^{O(1)}M^{O(1)}}\right)$. This allows us to define for $i \notin B$, $t'_i = t_i / \left(1 - \sum_{i\in B} t_i\right)$, so we have:

$$\begin{aligned}
\frac{\epsilon}{2L} \ge O\left(\frac{\epsilon}{Ln^{O(1)}M^{O(1)}}\right) \ge \sum_i t_i(b_i - \vec{c}a_i \cdot \vec{c}p^*) &\ge \sum_{i\notin B} t_i(b_i - \vec{c}a_i \cdot \vec{c}p^*) \\
&= \left(1 - \sum_{i\in B} t_i\right)\left(\sum_{i\notin B} t'_i(b_i - \vec{c}a_i \cdot \vec{c}p^*)\right) \\
&\ge \frac{1}{2}\left(\sum_{i\notin B} t'_i(b_i - \vec{c}a_i \cdot \vec{c}p^*)\right)
\end{aligned}$$

Now we can repeat the argument in the beginning of the proof with $\bar{\vec{p}} = \sum_{i\notin B} t'_i \vec{c}p_i$. $\square$

Perturbation, Approximation and Rounding The next step is to use the algorithm for finding an approximate solution to find an exact solution. This is impossible for generic convex programs, but since the function we are trying to optimize comes from a linear program, we can do that by exploiting this connection. As done for linear programs, we will do this in three steps: first we perturb the function to be optimized, then we find an approximate solution by Theorem 114 and finally we round it to an exact solution in the format of the optimal solution given by Lemma 111.

We can perturb the objective of the dual linear program by changing it to:

$$\min u + \vec{cp} \cdot (\vec{cs} + \vec{cr}) \text{ s.t. } u \geq \sum_i v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)}, \forall x^{(i)} \in [\![s]\!] \tag{PD}$$

We want to specify the real-valued vector $\vec{cr}$ in such a way that the optimal solution to this linear program is still the optimal solution to the original program, but also in a way that the optimal solution becomes unique. First we observe the following:

Lemma 115. *If $r_i < (nS)^{-(2n+1)}$ then an optimal solution of the perturbed program is also an optimal solution to the original program.*

Proof. Let $\mathcal{C}$ be the set of vertices (basic feasible solutions) of the dual LP. By Lemma 111 we know that the coordinates of those vertices must be of the form a_i/b_i for integers a_i, b_i such that $0 \leq b_i \leq (nS)^n$. For any linear objective function, the optimal solution must be a point in $\mathcal{C}$. Since the original objective has integral coefficients, when evaluated on any vertex, the objective is a fraction with denominator $b_i \leq (nS)^n$. Therefore, the difference between the objective evaluated at an optimal vertex and the objective evaluated at a suboptimal one is at least $\left|\frac{a_i}{b_i} - \frac{a'_i}{b'i}\right| = \left|\frac{a_i b'_i - a'_i b_i}{b_i b'_i}\right| \geq \frac{1}{(nS)^{2n}}$. Therefore, if $r_i < (nS)^{-(2n+1)}$, then its effect in the objective function is at most $nS \cdot (nS)^{-(2n+1)} \leq (nS)^{-2n}$, so it cannot cause a suboptimal solution to the original program to become an optimal solution of the perturbed program. □

Our final ingredient is a lemma by Klivans and Spielman [156] which is in the spirit of the Isolation Lemma of Mulmuley, Vazirani and Vazirani [196].

Lemma 116 (Klivans and Spielman (Lemma 4 in [156]))**.** *Let $\mathcal{C}$ be a set of points in $\mathbb{R}^n$ where all coordinates assume at most K distinct values. Then if $\vec{cr}$ is a vector with coordinates sampled at random from $\{0, 1, \ldots, Kn/\epsilon\}$, then with probability $1-\epsilon$, there is a unique $\vec{cp} \in \mathcal{C}$ minimizing $\vec{cr} \cdot \vec{cp}$.*

We note that although Lemma 4 in [156] is stated for $\mathcal{C}$ having coordinates in $\{0, 1, \ldots, K-1\}$, the proof only uses the fact that the coordinates of $\mathcal{C}$ assume at most K distinct values.

Lemma 117. *If $r_j = z_j / \left[Mn(nS)^{2n+1}\right]$ where z_j is drawn uniformly from $\{0, \ldots, nM(nS)^{2n} - 1\}$, then with probability $\frac{1}{2}$, the dual program has a unique minimizer and this minimizer is a minimizer of the original program.*

Proof. Let $\mathcal{C}$ be the set of vertices (basic feasible solutions) of the dual linear program in $[-M, M]^n$. All the minimizers of the original dual program are guaranteed to be in this set because of Lemma 110. Lemmas 110 and 111 tell us that the coordinates of points in $\mathcal{C}$ are of the form a_i/b_i where a_i and b_i are integers such that $1 \leq b_i \leq (nS)^n$ and $|a_i| \leq Mb_i$. So there are at most $\sum_{b=1}^{(nS)^n} b \cdot 2M \leq 2M(nS)^{2n}$ coordinates.

Now, Lemma 115 guarantees that the magnitude of the perturbation will prevent suboptimal vertices in the original program to become optimal vertices in the perturbed program. Lemma 116 guarantees that with half probability the vertex minimizing the objective is unique. □

The perturbed dual program translates to the following perturbed objective function:

$$\hat{f}(\vec{cp}) = \sum_{i=1}^{m} \left(\max_{x \in [\![\vec{cs}]\!]} v_i(x) - \vec{cp} \cdot x \right) + \vec{cp} \cdot (\vec{cr} + \vec{cs}). \tag{PC}$$

Since the subgradient of $\hat{f}$ can be computed from the aggregate demand oracle $\partial \hat{f}(\vec{cp}) = \vec{cs} + \vec{cr} - \vec{cd}(\vec{cp})$ we can use Theorem 114 to find an ϵ-minimizer.

Lemma 118. *For $\epsilon = (nMS)^{-O(n)}$, if $\vec{cp}$ is an ϵ-approximation to the optimal value of $\hat{f}$, i.e., $\hat{f}(\vec{cp}) - \hat{f}(\vec{cp}^*) \leq \epsilon$ then the optimal solution is the only point $\vec{cp}^*$ such that $\|\vec{cp} - \vec{cp}^*\|_2 \leq \frac{1}{(nMS)^{O(n)}}$ and has coordinates of the form described in Lemma 111.*

Proof. Let $\mathcal{C}$ be the set of vertices of the dual linear program, which are points of the form $(u_i, \vec{cp}_i)$ and let $\vec{cp}$ be an ϵ-approximation to $\hat{f}$. Now, if $u = \sum_i (\max_x v_i(x) - \vec{cp} \cdot x)$ then $(u, \vec{cp})$ is feasible in the dual linear program, and in particular, it can be written as a convex combination of points in $\mathcal{C}$, i.e., $(u, \vec{cp}) = \sum_i t_i (u_i, \vec{cp}_i)$ for $t_i \geq 0$ and $\sum_i t_i = 1$. There is only one vector in $\mathcal{C}$, call it $(u^*, \vec{cp}^*)$ for which the objective evaluates to $\hat{f}^*$. For all other vertices, the objective evaluates to at least $\hat{f}^* + \frac{1}{Mn(nS)^{2n+1}} \cdot \frac{1}{(nS)^n} = \hat{f}^* + \frac{1}{Mn(nS)^{3n+1}}$ due to Lemmas 111 and 117. Therefore, by evaluating $(u, \vec{cp}) = \sum_i t_i (u_i, \vec{cp}_i)$ on the linear objective of the perturbed dual program, we get:

$$\hat{f}^* + \epsilon \geq \hat{f}(\vec{cp}) \geq t_* \hat{f}^* + (1 - t_*) \left(\hat{f}^* + \frac{1}{Mn(nS)^{3n+1}} \right)$$

where t_* is the weight put on $(u^*, \vec{cp}^*)$ by the convex combination, therefore, if $\epsilon = (nMS)^{-O(n)}$ then $t^* \geq 1 - (nMS)^{-O(n)}$. In particular:

$$\|\vec{cp} - \vec{cp}^*\|_2 \leq \Big\| \sum_{i \neq *} t_i (\vec{cp}_i - \vec{cp}^*) \Big\|_2 \leq (1 - t_*) \cdot \max_{i \neq *} \|\vec{cp}_i - \vec{cp}^*\| \leq (nMS)^{-O(n)}$$

Therefore, $\vec{cp}^*$ is in a ball of radius $(nMS)^{-O(n)}$ around $\vec{cp}$. Also, any other point $\vec{cp}' \neq \vec{cp}^*$ with coordinates of the form a_i/b_i with $b_i \leq (nS)^n$ can be in this ball, since two distinct points of this type differ in at least one coordinate by at least $(nS)^{-n}$ so their distance is at least that much. □

Putting it all together:

Theorem 119. *There is an algorithm of running time $O(n^2 T_{AD} \log(SMn) + n^5 \log^{O(1)}(SMn))$ to compute an exact vector of Walrasian prices whenever it exists using only access to an aggregate demand oracle.*

Proof. From the potential function f of the market, use Lemma 117 to construct a perturbed potential function $\hat{f}$. Then use Theorem 114 to optimize $\hat{f}$ with $\epsilon = (nMS)^{-O(n)}$. Finally, use the guarantee to argue that the exact optimal value is the only point in the format of Lemma 111 in the ball of radius $(nMS)^{-O(n)}$ around the ϵ-approximate minimizer. At this point, we can round the solution to the exact point by using the method of continuous fractions in Kozlov et al [163] (see [230] for a complete exposition of this and related methods of rounding). $\square$

20 Walrasian Equilibrium for Gross Substitutes in $\tilde{O}(nT_{AD} + n^3)$

In the previous section we discussed how Walrasian equilibria can be computed without any assumptions on the valuation function using only an aggregate demand oracle. The focus was to address various difficulties in applying optimization tools for this problem.

Here we remain in the aggregate demand oracle model and focus on computing Walrasian equilibria for markets typically studied in economics, which are those where buyers have *gross substitute valuations*. The development of the theory of gross substitute valuations is intertwined with the development of the theory of Walrasian equilibrium for markets with discrete goods. In particular, it is the largest class of valuation functions that is closed under perturbation by an additive function[11] for which Walrasian equilibria always exist.

Gross substitutes play a central role in economics. Hatfield and Milgrom [119] argue that most important examples of valuation functions arising in matching markets belong to the class of gross substitutes. Gross substitutes have been used to guarantee the convergence of salary adjustment processes in the job market [149], to guarantee the existence of stable matchings in a variety of settings [227, 159], to show the stability of trading networks [118, 123], to design combinatorial auctions [14, 204] and even to analyze settings with complementarities [243, 117].

The concept of gross substitutes has been re-discovered in different areas from different perspectives: Dress and Wenzel [63] propose the concept of *valuated matroids* as a generalization to the Grassmann-Plücker relations in p-adic analysis. Dress and Terhalle [64] define the concept of *matroidal maps* which are the exact class of functions that can be optimized by greedy algorithms. Murota [198] generalized the concept of convex functions to discrete lattices, which gave birth to the theory known as Discrete Convex Analysis. One of the central objects in the Discrete Convex Analysis are M-concave and $M^\natural$-concave functions (the latter class was introduced by Murota and Shioura [203]). We refer to a recent survey by Murota [202] for comprehensive discussion on application of Discrete Convex Analysis in economics.

Surprisingly, gross substitutes, valuated matroids, matroidal maps and $M^\natural$-concave functions are all equivalent definitions – despite being originated from very different motivations. We refer to [181] for a detailed historic description and a survey on their equivalence.

[11]A class $\mathcal{C}$ of valuation functions is closed under perturbations by additive functions if for every $v \in \mathcal{C}$ and every vector $\vec{c}w \in \mathbb{R}^n$, the function $w(x) = v(x) + \vec{c}w \cdot x$ is also in $\mathcal{C}$.

Before presenting our algorithms, we first give a quick summary of the standard facts about gross substitutes that are needed. For a more comprehensive introduction, please see [181].

20.1 A crash course on gross substitutes

First we define gross substitute valuations on the hypercube $\{0,1\}^{[n]}$ and then we extend the defn to $[\![\vec{c}s]\!]$ for any supply vector $\vec{c}s$. When talking about functions defined on the hypercube, we will often identify vectors $x \in \{0,1\}^{[n]}$ with the set $S = \{i \in [n] : x_i = 1\}$, so we write $v(S)$ where $S \subseteq [n]$ meaning $v(\vec{c}1_S)$ where $\vec{c}1_S$ is the indicator vector of S.

Next we define three classes of valuation functions. The reader who saw the spoilers in the previous subsection will already suspect that they are equivalent.

Definition 120. (Gross substitutes, Kelso and Crawford [149]) A function $v : 2^{[n]} \to \mathbb{Z}$ is a gross substitute (GS) if for any price $\vec{c}p \in \mathbb{R}^n$ and $S \in D(v, \vec{c}p)$, any price $\vec{c}p' \geq \vec{c}p$ (entry-wise) has some $S' \in D(v, \vec{c}p')$ satisfying $S \cap \{j : p_j = p'_j\} \subseteq S'$.

In other words, price increases for some items can't affect the purchasing decisions for the items whose price stayed the same.

The second definition concerns when the demand oracle problem $\max v(S) - p(S)$ can be solved efficiently:

Definition 121. [Matroidal Maps, Dress and Terhalle [65]]A function $v : 2^{[n]} \to \mathbb{Z}$ is called matroidal if for any price $\vec{c}p \in \mathbb{R}^n$, the set S obtained by the following greedy algorithm solves $\max_{S \subseteq [n]} v(S) - p(S)$:

Greedy algorithm: Start with the empty set $S = \varnothing$. While $|S| < n$ and there exists $i \notin S$ such that $v(S \cup i) - p_i - v(S) > 0$, then update $S \leftarrow S \cup i^*$ where $i^* = \arg\max_{i \in [n] - S} v(S \cup i) - p_i - v(S)$ (break ties arbitrarily).

The third definition generalizes the concept of convexity and concavity to the hypercube. We can define convexity for continuous functions $f : \mathbb{R}^n \to \mathbb{R}$ as being function such that for all vectors $\vec{c}v \in \mathbb{R}^n$, if x^* is a local minimum of $f(x) - \vec{c}v \cdot x$, then x^* is also a global minimum. This definition generalizes naturally to the hypercube as follows:

Definition 122. [Discrete Concavity, Murota [197], Gul and Stachetti [115]] A function $v : 2^{[n]} \to \mathbb{Z}$ is discrete concave function if local optimality implies global optimality for all price vectors $\vec{c}p$. Formally: if $S \subseteq [n]$ is such that for all $i \in S$ and $j \notin S$, $v(S) \geq v(S \cup j) - p_j$, $v(S) \geq v(S - i) + p_i$ and $v(S) \geq v(S \cup j - i) + p_i - p_j$, then $v(S) - \vec{c}p(S) \geq v(T) - \vec{c}p(T), \forall T \subseteq [n]$.

Discrete concave functions have two important properties: (i) the demand oracle has a succinct certificate of optimality and (ii) the function can be optimized via local search for any given prices.

Those definitions arose independently in different communities and, amazingly enough, those definitions turned out to be equivalent. The equivalence of discrete concavity and gross substitutes is due to Fujishige and Yang [92] and the equivalence of discrete concavity and matroidal maps appears explicitly in Paes Leme [181].

Theorem 123. *A valuation function is in gross substitutes iff it is a matroidal map and iff it is a discrete concave valuation.*

The concept of gross substitutes generalizes naturally to multi-unit valuations: given any function $v : [\![\vec{c}s]\!] \to \mathbb{Z}$ we can translate it to a single-unit valuation function $\tilde{v} : 2^{[\sum_i s_i]} \to \mathbb{Z}$ by treating each of the s_i copies of item i as a single item. We say that a multi-unit valuation function v is gross substitutes if its corresponding single item version $\tilde{v}$ is in gross substitutes. We refer to the excellent survey by Shioura and Tamura [237] on gross substitutability for multi-unit valuations.

An important property of gross substitutes is:

Theorem 124 ([149, 53])**.** *If all buyers have gross substitute valuations, then a Walrasian equilibrium always exists.*

The original theorem showing the existence of Walrasian equilibrium for gross substitutes in the single-unit case ($s_j = 1$ for all j) is due to Kelso and Crawford [149]. The extension to multi-units is due to Danilov, Koshevoy and Murota [53].

20.2 Walrasian Prices form an integral polytope

By Theorem 124, the set of Walrasian prices is non-empty. We also know that it forms a convex polyhedral set, since they are the set of minimizers of a convex function that comes from a linear program. Perhaps a more direct way to see it is that given an optimal allocation $\vec{c}x$, the set of Walrasian prices can be characterized by the following set of inequalities:

$$P = \{\vec{c}p \in \mathbb{R}^n \mid v_i(x^{(i)}) - \vec{c}p \cdot x^{(i)} \geq v_i(x) - \vec{c}p \cdot x, \forall i \in [m], x \in [\![s]\!]\}$$

Lemma 111 provided a good characterization of the vertices of this polytope in the general case. For gross substitutes, however, there is an even nicer characterization. The next lemma is a special case of a result by Murota (Theorem 11.16 in [197]) that shows that the set of Walrasian prices form an $L^\natural$-convex polytope and hence is an integral polyhedron with a lattice structure. To keep the Part self-contained we give a proof from first principles.

Lemma 125. *If buyer valuations are gross substitutes, then all the vertices of the feasible region of the dual program D (defined in Section 19) have integral coordinates. In particular, the set of Walrasian prices form an integral polytope.*

Proof. Let $(u, \vec{c}p)$ be a non-integral feasible point of the dual program D. We will write it as a convex combination of integral points. Let $x^{(i)} \in \arg\max v_i(x) - \vec{c}p \cdot x$ and $w = u - \sum_i v_i(x^{(i)}) - \vec{c}p \cdot x^{(i)}$.

Now, we define a distribution over integral points in the following way: sample a random threshold $\theta \in [0, 1]$. If $\theta < p_i - \lfloor p_i \rfloor$, set $\hat{p}_i = \lceil p_i \rceil$, otherwise set $\hat{p}_i = \lfloor p_i \rfloor$. Similarly, if $\theta < w - \lfloor w \rfloor$, set $\hat{w} = \lceil w \rceil$ otherwise set $\hat{w} = \lfloor w \rfloor$. Set $\hat{u} = \hat{w} + \sum_i v_i(x^{(i)}) - \hat{\vec{c}p} \cdot x^{(i)}$. It is easy to see that $(\hat{u}, \hat{\vec{c}p})$ are integral and that $\mathbf{E}[(\hat{u}, \hat{\vec{c}p})] = (u, \vec{c}p)$. We are only left to prove that $(\hat{u}, \hat{\vec{c}p})$ are feasible.

We know that $\hat{u} \geq \sum_i v_i(x^{(i)}) - \hat{\vec{c}p} \cdot x^{(i)}$ since $\hat{w} \geq 0$. If we show that $x^{(i)} \in \arg\max_x v_i(x) - \vec{c}p \cdot x$, then we are done, since it automatically implies that all other constraints in the dual program D are satisfied. To see this notice that since $x^{(i)} \in \arg\max v_i(x) - \vec{c}p \cdot x$, then for all items j and k such that $x_j^{(i)} < s_j$ and $x_k^{(i)} > 0$ we have that:

$$v_i(x^{(i)}) \geq v_i(x^{(i)} + \vec{c}1_j) - p_j, \; v_i(x^{(i)}) \geq v_i(x^{(i)} - \vec{c}1_k) + p_k$$
$$v_i(x^{(i)}) \geq v_i(x^{(i)} - \vec{c}1_k + \vec{c}1_j) + p_k - p_j$$

Since the valuations are integer valued, it is simple to check that rounding using a threshold won't violate any of the above inequalities. Thus:

$$v_i(x^{(i)}) \geq v_i(x^{(i)} + \vec{c}1_j) - \hat{p}_j, \; v_i(x^{(i)}) \geq v_i(x^{(i)} - \vec{c}1_k) + \hat{p}_k$$
$$v_i(x^{(i)}) \geq v_i(x^{(i)} - \vec{c}1_k + \vec{c}1_j) + \hat{p}_k - \hat{p}_j$$

Therefore under price vector $\vec{\hat{p}}$, a buyer can't improve his utility from $x^{(i)}$ by adding, removing or swapping an item. Since gross substitutes are equivalent to discrete concavity (Definition 122), local optimality implies global optimality, i.e., $x^{(i)} \in \arg\max_x v_i(x) - \vec{\hat{p}} \cdot x$. □

Another important property proved by Gul and Stachetti [115] is that the set of Walrasian prices forms a lattice. This property is also a consequence of the $L^\natural$-convex structure in Murota [197].

Theorem 126 (Gul and Stachetti [115], Murota [197]). *If buyer valuations are gross substitutes, then the set of Walrasian prices form a lattice, i.e., if $\vec{c}p$ and $\vec{c}p'$ then $\vec{\bar{p}}$ and $\underline{\vec{c}p}$ are also Walrasian prices for $\bar{p}_i = \max(p_i, p'_i)$ and $\underline{p}_i = \min(p_i, p'_i)$.*

20.3 A simpler and faster algorithm for gross substitutes

Using the fact that the set of Walrasian prices is an integral polytope with a lattice structure we simplify the algorithm described in the previous section and improve its running time. First, since we have a lattice structure we no longer need to randomly perturb the objective function to make the solution unique. A simple and deterministic perturbation suffices. Integrality also allows us to round to an optimal solution from an approximate solution of smaller accuracy (i.e. larger ϵ).

Lemma 127. *If valuations are gross substitutes, then by taking $r_j = \frac{1}{2Sn}$ in the perturbed dual program PD, its optimal solution is unique and also optimal for the original dual program D.*

Proof. Since all the vertices of the polytope are integral and the coefficients are at most S, a perturbation of $r_j = \frac{1}{2Sn}$ can't affect the objective by more than half. So it can't cause a suboptimal vertex to become optimal. Also, since the set of Walrasian prices form a lattice, there is a Walrasian price $\vec{\bar{p}}$ such that $\vec{\bar{p}} \geq \vec{c}p$ for every Walrasian price $\vec{c}p$. Therefore this must be the unique optimal solution to the perturbed program. □

The previous lemma allows us to prove a better version of Lemma 118:

Lemma 128. *For $\epsilon < 1/(4nMS)$, if $\vec{c}p$ is an ϵ-approximation to the optimal value of $\hat{f}$, i.e., $\hat{f}(\vec{c}p) - \hat{f}(\vec{c}p^*) \leq \epsilon$ then the optimal solution is the only integral point $\vec{c}p^*$ such that $\left\|\vec{c}p - \vec{c}p^*\right\|_2 < \frac{1}{2}$.*

Proof. The proof can be obtained following the proof of Lemma 118, replacing the generic guarantees in Lemmas 111 and 117 by the better guarantees provided by Lemma 127 for gross substitutes. □

Putting it all together we have:

Theorem 129. *There is an algorithm of running time $O(nT_{AD}\log(SMn)+n^3\log^{O(1)}(SMn))$ to compute an exact vector of Walrasian prices in a market where buyers have gross substitute valuations.*

Proof. Same proof as in Theorem 119 with $\epsilon = 1/(5nMS)$. Also, since the optimal price vector is integral, instead of using the method of continuous fractions to round a solution, it is enough to round each component to the nearest integer. □

21 Walrasian Equilibrium for Gross Substitutes in $\tilde{O}((mn+n^3)\cdot T_V)$

We now move from the macroscopic view of the market to a microscopic view. We assume access to the market via a *value oracle*, i.e, given a certain buyer i and a bundle $S \subseteq [n]$ of goods, we can query the value of $v_i(S)$. We also assume from this point on that the supply of each good is one, or in other words, that the valuation functions are defined on the hypercube.

The fact that the demand of each buyer for any given price can be computed by the greedy algorithm (Definition 121) let us simulate the aggregate demand oracle by the value oracle model.

Lemma 130. *The outcome of the aggregate demand oracle can be computed in time $O(mn^2T_V)$, where T_V is the running time of the value oracle.*

Proof. The number of queries required for the greedy algorithm described in Definition 121 to compute $S_i \in \arg\max_S v_i(S) - \vec{cp}(S)$ is $n\cdot(|S_i|+1)T_V \le O(n^2T_V)$. Since there are m buyers, the total time to compute the demand of all buyers is $O(mn^2T_V)$. The aggregate demand oracle simply outputs $d(\vec{cp})$ where $d_j(\vec{cp}) = \#\{i : j \in S_i^*\}$. □

Now we can plug Lemma 130 directly into Theorem 129 and obtain a running time of $\tilde{O}(mn^3T_V)$. In the rest of this section, we show how to improve this to $\tilde{O}((mn+n^3)T_V)$.

21.1 Faster Algorithm via regularization

The idea to improve the running time from $\tilde{O}(mn^3T_V)$ to $\tilde{O}((mn+n^3)T_V)$ is to regularize the objective function. As with the use of regularizers in other context in optimization, this is to penalize the algorithm for being too aggressive. The bound of $O(mn^2)$ value oracle calls per iteration of the cutting plane algorithm is so costly precisely because we are trying to take an aggressively large step.

To provide some intuition, imagine that we have a price $\vec{cp}$ that is very close to optimal and that S_i are the set of items demanded by the buyers at price $\vec{cp}$. Intuitively, if $\vec{cp}$ is close to a Walrasian price then the sets $S_1, \ldots, S_m$ should be almost disjoint, which means that the total cost of the greedy algorithm should be $\sum_i n(|S_i|+1) \approx mn + n^2$. So when the prices are good, oracle calls should be cheaper. This tells us that when prices are good, fewer calls to the value oracle suffice to compute the aggregate demand oracle. When prices are far from equilibrium, perhaps a more crude approximation to the aggregate demand oracle is enough.

Based on this idea we define the following regularized objective function:

$$\tilde{f}(\vec{cp}) = \max_{\sum_i |S_i| = n} \left[\sum_{i=1}^{m} v_i(S_i) - \vec{cp}(S_i) \right] + \vec{cp}([n]). \tag{RC}$$

The regularization consists of taking the maximum over all sets $(S_1, \ldots, S_m)$ such that $\sum_i |S_i| = n$. Without this restriction, we have the original market potential function f. The new function $\tilde{f}$ has three important properties: (i) it is still convex, since it is a maximum over linear functions in $\vec{cp}$ and therefore we can minimize it easily; (ii) its set of minimizers is the same as the set of minimizers of f and (iii) subgradients are cheaper to compute. Intuitively, $\tilde{f}$ is very close to f when $\vec{cp}$ is close to equilibrium prices but only a crude estimate when $\vec{cp}$ is far from equilibrium. Next we show those statements formally and present an algorithm for computing the subgradient of $\hat{f}$.

We give an alternate form of $\tilde{f}$ which is nicer to work with algorithmically. A consequence of the next lemma, is that for every $\vec{cp}$ there is a constant γ such that $\tilde{f}(\vec{cp}) = f(\vec{cp} + \gamma \cdot \vec{c}1_{[n]})$ and that such parameter γ can be found algorithmically. In other words:

$$\tilde{f}(\vec{cp}) = \sum_{i=1}^{m} \left(\max_{S \subseteq [n]} v_i(S) - \vec{cp}(S) - \gamma \cdot |S| \right) + \vec{cp}([n]) + \gamma \cdot n$$

for some γ that depends on $\vec{cp}$ (and the tiebreaking rule used by the greedy algorithm). Among other things, this formulation of $\tilde{f}$ resembles common regularizers used in optimization better. One can think of it as if $\vec{cp}$ is changed to $\vec{cp} + \gamma \cdot \mathbf{1}_{[n]}$.

Lemma 131. *Suppose $v_i(j)$ are given and stored as n lists sorted in decreasing order $\mathbf{L}_j = \{v_i(j)\}_{i=1..m}$. With a running time*[12] *of $n^2 \cdot T_V + \tilde{O}(n^2)$, given price $\vec{cp}$, there is an algorithm, which we call* AllGreedy, *that finds*

1. *$S_1^*, \ldots, S_m^*$ maximizing $\max_{\sum_i |S_i| = n} \left(\sum_{i=1}^m v_i(S_i) - \vec{cp}(S_i) \right)$.*
2. *γ such that for all i, $S_i^* \in D(i, \vec{cp} + \gamma \cdot \vec{c}1_{[n]})$. Moreover, for any $\gamma' > \gamma$ and $S_i' \in D(i, \vec{cp} + \gamma' \cdot \vec{c}1_{[n]})$, we have $\sum_i |S_i'| < n$.*
3. *$\tilde{f}(\vec{cp}) = f(\vec{cp} + \gamma \cdot \vec{c}1_{[n]})$.*

Proof. First, we define the algorithm AllGreedy. The algorithm starts with a very large value of γ such that $D(i, \vec{cp} + \gamma \cdot \vec{c}1_{[n]}) = \{\varnothing\}$ for all agents i. Then we gradually decrease γ keeping at each step a set $S_i^*(\gamma) \in D(i, \vec{cp} + \gamma \cdot \vec{c}1_{[n]})$ that monotonically grow as γ decreases,

[12]Assuming the cost of initializing $S_i^* = \varnothing$ is not needed. This is acceptable here because our algorithm would only use S_i^* to compute the subgradient which $S_i^* = \varnothing$ has no effect on.

in the sense that for $\gamma_1 > \gamma_2$, $S_i^*(\gamma_1) \subseteq S_i^*(\gamma_2)$. We stop the algorithm as $\sum_i |S_i^*(\gamma)|$ reaches n.

The algorithm is best visualized as a continuous process, although it admits a very efficient discrete implementation as we will see in a moment. Before, we note that we can use the Greedy algorithm to compute $S_i^*(\gamma) \in D(i, \vec{c}p + \gamma \cdot \vec{c}1_{[n]})$ and if we fix the same tie breaking, the order in which we add the elements is the same for every γ, the only thing that changes is the stopping criteria (the larger γ is, the later we stop).

So this procedure can be implemented by running a greedy algorithm in parallel for each agent i. Initially γ is very large and all $S_i^*(\gamma) = \varnothing$. Then in any given moment, we can compute what is the largest value of γ for which it is possible to add one more item to the demanded set of i. This is simply the largest marginal of an i for the next item:

$$\max_i \max_{j \notin S_i^*} v_i(S_i^* \cup j) - v_i(S_i^*) - p_j$$

We can decrease γ to this value and advance one of the agent's greedy algorithm one step further.

We need to argue that it satisfies the three properties in the lemma and that it can be implemented in $n^2 T_V + O(n^2)$ time. The algorithm achieves this running time by updating lists $\mathtt{L}_j$ such that in each iteration, it is a sorted list of $v_i(S_i^* \cup j) - v_i(S_i^*)$. Since all sets start as the empty set, this is correct in the beginning of the algorithm. Now, in each iteration, we can scan all the lists to find the next largest marginal, taking $O(n)$ to inspect the top of each list. This gives us the next value of γ and the pair i, j to update $S_i^* \leftarrow S_i^* \cup j$. Now, after the update, we go through each list $\mathtt{L}_k$ updating the value of the marginal of agent i for k, since S_i^* was updated. This takes $O(\log(m))$ for each list, so in total, this process takes $nT_V + \tilde{O}(n)$. Since there are at most n iterations, the overall running time is $n^2 T_V + \tilde{O}(n^2)$.

Now, for three properties in the lemma, property 2 is true by construction. For properties 1 and 3, consider the following chain of inequalities:

$$\begin{aligned}
\tilde{f}(\vec{c}p) &= \max_{\sum_i |S_i| = n} \left(\sum_{i=1}^m v_i(S_i) - \vec{c}p(S_i) \right) + \vec{c}p([n]) \\
&= \max_{\sum_i |S_i| = n} \left(\sum_{i=1}^m v_i(S_i) - \vec{c}p(S_i) - \gamma \cdot |S_i| \right) + \vec{c}p([n]) + \gamma \cdot n \\
&\leq \max_{S_i \subseteq [n]} \left(\sum_{i=1}^m (v_i(S_i) - \vec{c}p(S_i) - \gamma \cdot |S_i| \right) + \vec{c}p([n]) + \gamma \cdot n = f(\vec{c}p + \gamma \cdot \vec{c}1_{[n]}) \\
&= \sum_{i=1}^m [v_i(S_i^*) - \vec{c}p(S_i^*) - \gamma \cdot |S_i^*|] + \vec{c}p([n]) + \gamma \cdot n = \sum_{i=1}^m [v_i(S_i^*) - \vec{c}p(S_i^*)] + \vec{c}p([n]) \\
&\leq \max_{\sum_i |S_i| = n} \left(\sum_{i=1}^m v_i(S_i) - \vec{c}p(S_i) \right) + \vec{c}p([n]) = \tilde{f}(\vec{c}p)
\end{aligned}$$

Hence, all inequalities hold with equality, which means in particular that $\tilde{f}(\vec{c}p) = f(\vec{c}p + \gamma \cdot \vec{c}1_{[n]})$ and $S_1^*, \ldots, S_m^*$ maximize $\max_{\sum_i |S_i| = n} \left(\sum_{i=1}^m v_i(S_i) - \vec{c}p(S_i) \right)$. □

Corollary 132. *Suppose $v_i(j)$ are given and stored as n sorted lists $\{v_i(j)\}_j$ each of which has m elements. Then the greedy algorithm computes a subgradient of $\tilde{f}$ in $n^2 \cdot T_V + \tilde{O}(n^2)$ time.*

Proof. This follows directly from Lemma 131 as the gradient of $\sum_{i=1}^{m}(v_i(S_i^*) - \vec{cp}(S_i^*)) - \vec{cp}([n])$ is a subgradient of $\tilde{f}$. □

Corollary 133. *If $\vec{cp}^*$ minimizes $\tilde{f}$, then $\vec{cp}^* + \gamma \cdot \vec{c}1_{[n]}$ is an equilibrium price. Here γ is defined as in Lemma 131 with respect to $\vec{cp}^*$. Conversely, any Walrasian price $\vec{cp}^{eq}$ is a minimizer of $\tilde{f}$.*

The proof of the previous corollary is given in Section 24.

Theorem 134. *For gross substitutes, we can find an equilibrium price in time $mn \cdot T_V + O(mn \log m + n^3 \log(mnM) \cdot T_V + n^3 \log^{O(1)}(mnM))$.*

Proof. Corollary 133 says that it is enough to find a minimizer of $\tilde{f}$. The algorithmic procedure in Theorem 129 can be used to solve $\tilde{f}$ approximately and then round it to an optimal solution.

To bound the overall running time, we note that: computing $v_i(j)$ and storing them as n sorted lists takes $mn \cdot T_V + O(mn \log m)$ time. By Corollary 132, the separation oracle for $\tilde{f}$ can be implemented in $n^2 \cdot T_V + O(n^2 \log(m))$ time. □

22 Robust Walrasian Prices, Market Coordination and Walrasian allocations

So far we focused on computing Walrasian prices. Now we turn to the other component of Walrasian equilibrium, which is to compute the optimal allocation. If we have only access to an aggregate demand oracle, then computing prices is all that we can hope for, since we have no per-buyer information (in fact, we don't even know the number of buyers). If we have access to a value oracle, computing the optimal allocation is possible.

To convince the reader that this is a non-trivial problem, we show that computing an optimal allocation from Walrasian prices is at least as hard as solving the *Matroid Union Problem*. In the Matroid Union problem we are given m matroids defined over the same ground set $M_i = ([n], \mathcal{B}_i)$ and a promise that there exist basis $B_i \in \mathcal{B}_i$ such that $[n] = \cup_i B_i$. The goal is to find those bases. Now, consider the following mapping to the problem of computing an optimal allocation: consider m agents with valuations over a set $[n]$ of items such that $v_i = r_{M_i}$, i.e. the rank of matroid M_i (matroid rank functions are known to be gross substitute valuations [181]). The price vector $\vec{c}1$ is clearly a vector of Walrasian prices. Finding the optimal allocation, however, involves finding $\vec{c}S = (S_1, \ldots, S_m)$ maximizing $\sum_i r_{M_i}(S_i)$.

The previous paragraph hopefully convinced the reader that finding an allocation is not always a simple task even if we know the prices. One approach to solve this problem is based on a modification of standard matroid union algorithms.

The second approach, which we discuss here in details, is based on convex programming and reveals an important structural property of gross substitute valuations that might be of independent interest. Incidentally, this also answers an open question of Hsu et al. [122] who asked what are the conditions for markets to be perfectly coordinated using prices. More precisely, they showed that under some genericity condition the minimal Walrasian price *for a restricted class of gross substitutes* induces an overdemand at most 1 for each item. On the other hand, our argument in this section says that under the same condition *almost every* Walrasian prices *for any gross substitutes* have *no* overdemand, i.e. the market is perfectly coordinated. This follows from the fact that the polytope of Walrasian prices have nonempty interior and that interior Walrasian price induces no overdemand (see section 6.2).

Next, we review two combinatorial lemmas that will be fundamental for the rest of this section and the next one:

22.1 Two combinatorial lemmas

One of the most useful (and largely unknown) facts about gross substitutes is the following analogue to the Unique Matching Theorem for matroids. The version of this theorem for gross substitutes is due to Murota [199, 200] and it was originally proved in the context of valuated matroids, which are known to be equivalent to gross substitutes under a certain transformation. We refer the reader to Lemma 10.1 in [181] for a proof of this lemma in the language of gross substitute valuations. We also refer to chapter 5 of [201] for a comprehensive treatment of such exchange properties.

Lemma 135 (Unique Minimum Matching Lemma)**.** *Let $v : 2^{[n]} \to \mathbb{R}$ be a valuation satisfying gross substitutes, $S \subseteq [n]$, $A = \{a_1, \dots, a_k\} \subseteq S$, $B = \{b_1, \dots, b_k\} \subseteq [n] - S$. Consider weighted bipartite graph G with left set A, right set B and edge weights $w_{a_i,b_j} = v(S) - v(S \cup b_j - a_i)$. If $M = \{(a_1, b_1), (a_2, b_2), \dots, (a_k, b_k)\}$ is the unique minimum matching in G, then:*

$$v(S) - v(S \cup B - A) = \sum_{j=1}^{k} v(S) - v(S \cup b_j - a_j)$$

Lastly, we state a purely combinatorial lemma that is commonly used in conjunction with the previous lemma. We present a sketch of the proof in Section 24 and refer to [201, 181] for a complete proof.

Lemma 136. *Let $G = (V, E, w)$ be a weighted directed graph without negative weight cycles. Let C be the cycle of minimum number of edges among the cycles with minimum weight. Let $M := \{(u_1, v_1), \dots, (u_t, v_t)\}$ be a set of non-consecutive edges in this cycle, $U = \{u_1, \dots, u_t\}$ and $V = \{v_1, \dots, v_t\}$. Construct a bipartite graph G' with left set U, right set V and for each edge from $u \in U$ to $v \in V$ in the original graph, add an edge of the same weight to G'. Under those conditions, M forms a unique minimum matching in G'. The same result holds for a path P of minimum length among all the minimum weight paths between a pair of fixed nodes.*

22.2 Robust Walrasian Prices and Market Coordination

Hsu et al. [122] raise the following important question: when is it possible to find Walrasian prices that coordinate the market? A vector of Walrasian prices $\vec{cp}$ is said to coordinate the market if each agent has a unique demanded bundle under $\vec{cp}$ and those bundles clear the market. If this happens, we say that this vector is *robust*.

Definition 137 (Robust Walrasian Prices)**.** A price vector $\vec{cp}$ is said to be a vector of *robust Walrasian prices* for a certain market if $D(i, \vec{cp}) = \{S_i\}$ and $\vec{cS} = (S_1, \ldots, S_m)$ form a partition of the items.

Notice that by the Second Welfare Theorem (Lemma 104), if the optimal allocation is not unique, then no vector of Walrasian prices is robust, since each vector of Walrasian prices support all the allocations. If the optimal allocation is unique, on the other hand, then we show that a vector of robust Walrasian prices always exists. Moreover, the set of Walrasian prices forms a full-dimensional convex set in which all interior points are robust.

Theorem 138. *If there is a unique partition $\vec{cS} = (S_1, \ldots, S_m)$ maximizing $\sum_i v_i(S_i)$, then there exist a vector $\vec{cp}$ such for all $\vec{cp}' \in \prod_j [p_j - \frac{1}{2n}, p_j + \frac{1}{2n}]$ are Walrasian. In particular, the set of Walrasian prices is a full-dimensional convex set and all price vectors in its interior are robust Walrasian prices.*

The proof involves the concept of the *exchange graph*, which was first introduced by Murota in [200] and characterizes the set of all Walrasian prices as the dual variables of the shortest path polytope for a certain graph. Given an optimal allocation $\vec{cS} = (S_1, \ldots, S_m)$, the Second Welfare Theorem (Lemma 104) combined with the characterization of gross substitute functions from Discrete Convex Analysis (Definition 122) tells us that the set of Walrasian prices can be characterized by:

$$P = \left\{ \vec{cp} \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} v_i(S_i) \geq v(S_i - j) + p_j, & \forall i \in [m], j \in S_i \\ v_i(S_i) \geq v(S_i \cup k) - p_k, & \forall i \in [m], k \notin S_i \\ v_i(S_i) \geq v(S_i \cup k - j) - p_k + p_j, & \forall i \in [m], j \in S_i, k \notin S_i \end{array} \right\}$$

Which is clearly a convex set defined by $O(\sum_i |S_i| \, n) = O(n^2)$ inequalities. A nice combinatorial interpretation of this polytope is that it corresponds to the set of potentials in a graph.

To make the construction nicer, augment the items with m dummy items, one for each buyer. The set of items becomes $[n] \cup [m]$, and the valuations are extended to the subsets of $[n] \cup [m]$ in a way that agents completely ignore the dummy items, i.e., for $T \subset [n] \cup [m]$, $v_i(T) = v_i(T \cap [n])$. Also, augment S_i to contain the dummy item for buyer i. Under this transformation we can simplify the definition of P to:

$$P = \left\{ \vec{cp} \in \mathbb{R}^n \;\middle|\; v_i(S_i) \geq v(S_i \cup k - j) - p_k + p_j, \forall i \in [m], j \in S_i, k \notin S_i \right\}$$

since we can represent adding and removing an item as a swap with a dummy item. Under this transformation, construct a directed graph with one node for each item in $[n]$. For each $i \in [m]$, $j \in S_i$ and $k \notin S_i$, add an edge (j, k) with weight $w_{j,k} = v_i(S_i) - v_i(S_i \cup k - j)$.

Since the allocation $\vec{cS} = (S_1, \ldots, S_m)$ is optimal, there exists at least one vector of Walrasian prices $\vec{cp} \in P$. This guarantees that the exchange graph has no negative cycles, since for any cycle $C = \{(j_1, j_2), \ldots, (j_t, j_1)\}$, we can bound the sum of weights by $\sum_r w_{j_r, j_{r+1}} \geq \sum_r p_{j_r} - p_{j_{r+1}} = 0$, where the inequality follows from the definition of P and the definition of the weights. Now we argue that the exchange graph can't contain any cycles of zero weight:

Lemma 139. *If $\vec{c}S$ is the unique optimal allocation, then the exchange graph has no cycles of zero weight.*

Proof. If there were cycles of zero weight, let C be the cycle of zero weight of minimum length. Now, let $C_i = \{(j_1, t_1), \ldots, (j_a, t_a)\}$ be the edges (j, t) in C with $j \in S_i$ and (consequently) $t \notin S_i$. Now, define $S'_i = S_i \cup \{t_1, \ldots, t_a\} - \{j_1, \ldots, j_a\}$. Notice that we performed the swaps prescribed by the cycle, so each moved item was removed from one set and added to another and as a result, $\vec{c}S' = (S'_1, \ldots, S'_m)$ is still a partition of the items. Using Lemmas 136 and 135 we get that:

$$v_i(S'_i) - v_i(S_i) = \sum_{r=1}^{a} v_i(S_i \cup t_r - j_r) - v_i(S_i) = \sum_{r=1}^{a} w_{j_r t_r}$$

therefore:

$$\sum_i v_i(S'_i) - \sum_i v_i(S_i) = \sum_{e \in C} w_e = 0$$

and so $\vec{c}S' = (S'_1, \ldots, S'_m)$ is an alternative optimal allocation, contradicting the uniqueness of $\vec{c}S$. □

Now we are ready to prove the Theorem 138:

Proof of Theorem 138. Since we know there are no zero weight cycles and all the edge weights are integral, all cycles have weight at least 1. Now perform the following operation: for each vertex $j \in [n]$ in the directed graph, split it into j_1 and j_2 with an edge between j_1 and j_2 with weight $-\frac{1}{n}$. Make all incoming edges to j be incoming to j_1 and all outgoing edges from j to be outgoing from j_2. The resulting graph has again no cycles of negative weight, since the new edges can decrease each cycle by at most 1.

Therefore, it is possible to find a potential in this graph. A potential of a weighted graph with edge weights w_{jt} is a function ϕ from the nodes to $\mathbb{R}$ such that $\phi(t) \leq \phi(j) + w_{jt}$. It can be easily computed by running a shortest path algorithm from any fixed source node and taking the distance from source node to j as $\phi(j)$. For the particular case of the graph constructed, it is useful to take the source as one of the dummy nodes.

After computing a potential from the distance from a dummy node to each node, define the price of j as $p_j = \phi(j_2)$. By the definition of the potential for each edge (j, t) in the graph:

$$p_t = \phi(t_2) \leq \phi(t_1) - \frac{1}{n} \leq \phi(j_2) + w_{jt} - \frac{1}{n} = p_j + w_{jt} - \frac{1}{n}.$$

This means in particular that all inequalities that define P are valid with a slack of $\frac{1}{n}$. Therefore, changing any price by at most $\frac{1}{2n}$ still results in a valid Walrasian equilibrium. This completes the proof of the first part of the theorem.

Since P contains a cube, then it must be a full-dimensional convex body. Finally, let's show that every price vector in the interior of P is a vector of robust Walrasian prices. By the second welfare theorem (Lemma 104), $S_i \in D(i, \vec{c}p)$ for all $\vec{c}p \in P$. Now, assume that for some point in the interior, there is $S'_i \in D(i, \vec{c}p)$, $S'_i \neq S_i$. Then either: (i) There is $j \in S'_i - S_i$. We decrease the price of j by ϵ so that S'_i becomes strictly better than S_i, i.e. $S_i \notin D(i, \vec{c}p - \epsilon \vec{c}1_j)$, which contradicts the second welfare theorem since $\vec{c}p - \epsilon \vec{c}1_j \in P$.

(ii) There is $j \in S_i - S_i'$. We increase the price of j by ϵ so that S_i' becomes strictly better than S_i, i.e. $S_i \notin D_i(i, \vec{c}p + \epsilon\vec{c}1_j)$, which again contradicts the second welfare theorem since $\vec{c}p + \epsilon\vec{c}1_j \in P$. □

22.3 Computing Optimal Allocation

Theorem 138 guarantees that if the optimal allocation is unique, then the set of Walrasian prices has large volume. Since the set of Walrasian prices corresponds to the set of minimizers of the market potential $f(\vec{c}p)$, then there is a large region where zero is the unique subgradient of f. In such situations, convex optimization algorithms are guaranteed to eventually query a point that has zero subgradient. The point $\vec{c}p$ queried corresponds to a set of Walrasian prices and the optimal allocation can be inferred from the subgradient (recall Theorem 112).

Our strategy is to perturb the valuation functions in such a way that the optimal solution is unique and that it is still a solution of the original problem. It is important for the reader to notice that this is a different type of perturbation than the one used in previous sections. While previously we perturbed the objective of the dual program, here we are effectively perturbing the objective of the primal program. One major difference is that if we perturb the objective of the dual, we can still compute the subgradient of $\hat{f}$ in PC using the aggregate demand oracle. If we perturb the objective of the primal, we no longer can compute subgradients using the aggregate demand oracle. With value oracles, however, this is still possible to be done.

To perturb the primal objective function, we use the isolation lemma, a standard technique to guarantee a unique optimum for combinatorial problems.

Lemma 140 (Isolation Lemma [196]). *Let $\vec{c}w \in [N]^n$ be a random vector where each coordinate w_i is chosen independently and uniformly over $[N]$. Then for any arbitrary family $\mathcal{F} \subseteq 2^{[n]}$, the problem $\max_{S\in\mathcal{F}} \sum_{i\in S} w_i$ has a unique optimum with probability at least $1 - n/N$.*

For our application, we would like a unique optimum to the problem of maximizing $\sum_{i=1}^m v_i(S_i)$ over the partition $(S_1, \ldots, S_m)$. To achieve this, we first replace v_i by $\tilde{v}_i(S) = Bv_i(S) + w_i(S)$ where B is some big number to be determined and $w_i(j)$ is set as in the isolation lemma (with N to be determined as well).

Lemma 141. *By setting $B = 2nN$, $N = mn^{O(1)}$ and sampling $w_i(j)$ uniformly from $[N]$ for each $i \in [m]$ and $j \in [n]$, then with probability $1 - 1/n^{O(1)}$, there is a unique partition $(S_1, \ldots, S_m)$ maximizing $\sum_{i=1}^m \tilde{v}_i(S_i)$, for $\tilde{v}_i(S) = B \cdot v_i(S) + w_i(S)$.*

Proof. Let $(S_1', \ldots, S_m')$ be an optimal solution w.r.t the original problem. We first show that any suboptimal partition $(S_1, \ldots, S_m)$ cannot be optimal for the perturbed problem. Since v_i assume integer values, we have $\sum_{i=1}^m v_i(S_i') \geq \sum_{i=1}^m v_i(S_i) + 1$. Now:

$$\sum_{i=1}^m \tilde{v}_i(S_i') \geq B + \sum_{i=1}^m \tilde{v}_i(S_i) + \sum_{i=1}^m w_i(S_i) - \sum_{i=1}^m w_i(S_i') \geq B + \sum_{i=1}^m \tilde{v}_i(S_i) - nN > \sum_{i=1}^m \tilde{v}_i(S_i).$$

which shows that a suboptimal solution to the original problem cannot be optimal for the perturbed one.

Now, consider all partitions $(S'_1, \dots, S'_m)$ that are optimal for the original problem and identify each optimal partition with a subsets of $[mn] = \{(i,j); i \in [m], j \in [n]\}$ in the natural way: add (i,j) to the subset if $j \in S'_i$. This family of subsets corresponds to $\mathcal{F} \subseteq 2^{[mn]}$ in the statement of the Isolation Lemma and $w_i(j)$ corresponds to $\vec{c}w$. The result then follows from applying that lemma. □

The strategy to find an optimal allocation is to perturb the valuation functions, then search for an interior point in the set of minimizers of the market potential function f. When we find such a point we can obtain a vector of Walrasian prices for the original market by rounding and the optimal allocation by inspecting the subgradient. To get the desired running time, we need to apply those ideas to the regularized potential function $\tilde{f}$ (see RC in Section 21) instead of the original one. To apply this to the regularized potential we need an extra lemma:

Lemma 142. *Let $\vec{c}p$ be an interior point of the set of minimizers of the regularized potential function $\tilde{f}$, then the allocation $(S^*_1, \dots, S^*_m)$ produced by the* AllGreedy *algorithm in Lemma 131 is an equilibrium allocation.*

Proof. If $(S^*_1, \dots, S^*_m)$ is a partition over the items, then it is an optimal allocation by part 2 of Lemma 131. To show that it is a partition, observe that since $(S^*_1, \dots, S^*_m)$ is a maximizer of $\sum_i v_i(S_i) - \vec{c}p(S_i)$ subject to $\sum_i |S_i| = n$, then we can use it to build a subgradient $\vec{c}g$ of $\tilde{f}$ such that $g_j = -1 + |\{i; j \in S^*_i\}|$. Since $\vec{c}p$ is an interior point of the set of minimizers, the subgradient must be zero and therefore $|\{i; j \in S^*_i\}| = 1$ for all j. □

Theorem 143. *For gross substitute valuation, we can find a Walrasian equilibrium, i.e. allocation and prices, in time $O((mn+n^3)T_V \log(nmM) + n^3 \log^{O(1)}(mnM))$.*

Proof. Use Lemma 141 to perturb the valuation functions and obtain $\tilde{v}_i$ which are still gross substitutes (since they are the sum of a gross substitute valuation and an additive valuation) and there is a unique optimal allocation. By Lemma 138, the set of minimizer of the market potential function f contains a box of width $1/n$. Since the set of minimizers of the market potential f is contained in the set of minimizers of the regularized potential $\tilde{f}$, then its set of minimizers also contains a box of width $1/n$. We also know that it is contained in the box $[-Mmn^{O(1)}, Mmn^{O(1)}]^n$ since $|\tilde{v}_i(S)| \le O(Mmn^{O(1)})$.

If we apply the algorithm in Theorem 113 with $\delta = O(1/n^{O(1)})$ to optimize the regularized potential $\tilde{f}$ then we are guaranteed to query a point in the interior of minimizers as otherwise the algorithm would certify that there is $\vec{c}a$ with $\|\vec{c}a\|_2 = 1$ such that $\max_{\vec{c}p \in P} \vec{c}a \cdot \vec{c}p - \min_{\vec{c}p \in P} \vec{c}a \cdot \vec{c}p \le 1/n^{O(1)}$, where P is the set of minimizers.

Finally, we can obtain the optimal allocation for the perturbed using Lemma 131, which is an optimal allocation to the original market according to Lemma 141. □

23 Combinatorial approach to Walrasian Equilibrium for Gross Substitutes

In a sequence of two foundational papers [199, 200], Murota shows that the assignment problem for *valuated matroids*, a class of functions introduced by Dress and Wenzel [62] can be solved in strongly polynomial time. We show how this algorithm can be used to obtain an $\tilde{O}(nm + n^3)$ strongly polynomial time algorithm for problem of computing a Walrasian equilibrium for gross substitute valuations. Our contribution is two-fold: first we map the Walrasian equilibrium problem on gross substitute valuations to the assignment problem on valuated matroids and analyze its running time. The straightforward mapping allows us to obtain a strongly polynomial time algorithm with running time $O(mn^3 \log(m+n))$. We note that a different way to reduce the Walrasian equilibrium problem to a standard problem in discrete convex analysis is to map it to the M-convex submodular flow problem as done in Murota and Tamura [205]. We choose to reduce it to the assignment problem on valuated matroids since its running time is simpler to analyze.

Inspired by our $\tilde{O}(mn + n^3)$ algorithm, we revisit Murota's algorithm and propose two optimizations that bring the running time down to $O((mn + n^3)\log(m + n))$. Murota's algorithm works by computing augmenting paths in a structure known as the *exchange graph*. First we show that for the Walrasian equilibrium problem, this graph admits a more succinct representation. Then we propose a data structure to amortize the cost of some operations across all iterations.

In section 23.1 we define valuated matroids and the assignment problem for valuated matroids. Then we describe Murota's algorithm for this problem. We also discuss the relation between the assignment problem for valuated matroids and the welfare problem for gross substitutes. The goal of subsection 23.1 is to provide the reader with the historical context for this result.

The reader interested solely in the welfare problem is welcome to skip to Section 23.2 which can be read independently, without any mention to valuated matroids or the assignment problem. A complete proof is given in that section.

23.1 The assignment problem for valuated matroids

A *valuated matroid* is an assignment of weights to basis of a matroid respecting some valuated analogue of the exchange property.

Definition 144 (Valuated matroid)**.** Let $\mathcal{B}$ be the set of basis of a matroid $\mathcal{M} = (V, \mathcal{B})$. A valuated matroid is a function $\omega : \mathcal{B} \to \mathbb{R} \cup \{\pm\infty\}$ such that for all $B, B' \in \mathcal{B}$ and $u \in B - B'$ there exists $v \in B' - B$ such that:

$$\omega(B) + \omega(B') \leq \omega(B \cup v - u) + \omega(B' \cup u - v)$$

Valuated matroids are related to gross substitutes by the following one-to-one correspondence. We refer the reader to Lemma 7.4 in [181] for a proof.

Proposition 145. *A valuation function $v : 2^{[n]} \to \mathbb{R}$ is a gross substitutes valuation function iff $\omega : \binom{[2n]}{n} \to \mathbb{R}$ defined by $\omega(S) = v(S \cap [n])$ is a valuated matroid defined over the basis of the n-uniform matroid on $2n$ elements.*

Now, we are ready to define the assignment problem for valuated matroids:

Definition 146 (Valuated matroid assignment problem)**.** Given two sets V_1, V_2, matroids $\mathcal{M}_1 = (V_1, \mathcal{B}_1)$ and $\mathcal{M}_2 = (V_2, \mathcal{B}_2)$ of the same rank, valuated matroids $\omega_1 : \mathcal{B}_1 \to \mathbb{R}$ and

$\omega_2 : \mathcal{B}_2 \to \mathbb{R}$ and a weighted bipartite graph $G = (V_1 \cup V_2, E, w)$, find a matching M from V_1 to V_2 maximizing:

$$w(M) + \omega_1(M_1) + \omega_2(M_2)$$

where M is a subset of edges of E forming a matching and M_1 and M_2 are the sets of endpoints of M in V_1 and V_2 respectively.

Murota gives two strongly-polynomial time algorithms based on network flows for the problem above in [200] – the first based on cycle-cancellations and the second based on flow-augmentations. Although the running time is not formally analyzed in his paper, it is possible to see that his algorithm (more specifically the algorithm *Augmenting algorithm with potentials* in Section 3 of [200]) has running time $\tilde{O}(R \cdot (|E| + R \cdot (|V_1| + |V_2|)))$ for $R = \text{rank}(\mathcal{M}_1) + \text{rank}(\mathcal{M}_2)$.

First, we show a simple reduction from the welfare problem for gross substitutes to this problem. Given m gross substitute valuation functions $v_i : 2^{[n]} \to \mathbb{R}$, define the following instance of the valuated matroid assignment problem: define the first matroid as $\binom{[mn]}{n}$ i.e. the n-uniform matroid on mn elements. Interpret the elements of $[mn]$ as "the allocation of item j to agent i" for each pair (i, j). Following this interpretation, each $S \subseteq [mn]$ can be seen as $S = \cup_{i=1}^{m} S_i$ where S_i are the elements assigned to agent i. This allows us to define for each $S \in \binom{[mn]}{n}$, $\omega_1(S) = \sum_i v_i(S_i)$. For the second matroid, let $\mathcal{M}_2 = ([n], 2^{[n]})$ and $\omega_2(S) = 0$ for all S. Finally, define the edges of E such that for each $j \in [n]$, the j-th element of $[n]$ are connected to the element (i, j) in $[mn]$ for each i.

One needs to prove that ω_1 satisfies the properties defining a valuated matroid, but this can be done using the transformations described in [181]. We omit this proof since we are giving a self-contained description of the algorithm in the next section.

In the construction shown below, $|V_1| = |E| = mn$, $|V_2| = n$ and $\text{rank}(\mathcal{M}_1) = \text{rank}(\mathcal{M}_2) = n$. This leads to an $\tilde{O}(m \cdot n^3)$ strongly polynomial time algorithm for the welfare problem.

23.2 Gross substitutes welfare problem in $\tilde{O}(mn + n^3)$ time

In this section we give a self-contained description of a specialized version of Murota's algorithm for the gross substitutes welfare problem and show that the running time of $\tilde{O}(m \cdot n^3)$ can be improved to $\tilde{O}(mn+n^3)$. Murota's algorithm for the case of generic valuated matroids can be quite complicated. Since the underlying matroids of our problem are simple (uniform matroids) and the functions being optimized have additional structure, it is possible to come up with a simpler algorithm. Our presentation also makes the algorithms accessible to the reader not familiar with discrete convex analysis and the theory of valuated matroids.

We consider the setting in which a set $[n]$ of items needs to be allocated to m agents with monotone valuation functions $v_i : 2^{[n]} \to \mathbb{R}$ satisfying the gross substitutes condition. Consider the intermediary problem of computing the optimal allocation for the first k items (in some arbitrary order):

$$\max_i v_i(S_i) \quad \text{s.t.} \quad \cup_i S_i \subseteq [k] := \{1, 2, \dots, k\} \text{ and } S_i \cap S_j = \varnothing \text{ for } i \neq j \qquad (I_k)$$

The central idea of the algorithm is to successively solve $(I_1), (I_2), \dots, (I_n)$ using the solution of (I_{k-1}) to compute (I_k). We will show how a solution to (I_k) can be computed from a solution of (I_{k-1}) via a shortest path computation in a graph with $\tilde{O}(m + n^2)$ edges.

A solution for problem (I_{k-1}) consists of an allocation $S = (S_1, \dots, S_m)$ of items in $[k-1]$ to agents $1, \dots, m$ and a price vector $p_1, \dots, p_{k-1}$ that certifies that the allocation is optimal. Since optimality for gross substitutes can be certified by checking that no agent wants to add an item, remove an item or swap an item (Definition 122) then S, p need to satisfy the following conditions for every agent i and every $j \notin S_i$ and $j' \in S_i$:

$$v_i(S_i \cup j) - p_j \leq v_i(S_i) \tag{Add}$$

$$v_i(S_i - j') + p_{j'} \leq v_i(S_i) \tag{Remove}$$

$$v_i(S_i \cup j - j') - p_j + p_{j'} \leq v_i(S_i) \tag{Swap}$$

23.2.1 Exchange graph

The first step to solve (I_k) is to build a combinatorial object called the *exchange graph* using the solution of (I_{k-1}), expressed as a pair S, p. We define a weighted directed graph on $V = [k] \cup \{\phi_1, \dots, \phi_m\}$. Intuitively, we can think of ϕ_i as an "empty spot" in the allocation of agent i. We add edges as follows:

- (t, j) for all items t and j not acquired by the same agent under S. If i is the agent holding item t, then the edge represents the change in utility (under price p) for agent i to swap his item t by j:

$$w_{tj} = v_i(S_i) - v_i(S_i \cup j - t) + p_j - p_t$$

- (ϕ_i, j) for all items $j \notin S_i$. It represents the change in utility for i to add item j:

$$w_{\phi_i j} = v_i(S_i) - v_i(S_i \cup j) + p_j$$

Notice that problem (I_{k-1}) only defines prices for $1, \dots, k-1$. So for the construction above to be well defined we need to define p_k. We will set p_k in a moment, but before that, note that for all the edges not involving k, $w \geq 0$ which follow by the fact that p is a certificate of optimality for S and therefore conditions Add and Swap hold. Finally, notice that there are only directed edges from k to other nodes, so p_k always appear with positive sign in w. So, we can set p_k large enough such that all edges have non-negative weights, in particular, set:

$$p_k = \max \left\{ \max_{i \in [m]; t \in S_i} [v_i(S_i \cup k - t) - v_i(S_i) + p_t], \ \max_{i \in [m]} [v_i(S_i \cup k) - v_i(S_i)] \right\}$$

23.2.2 Updating prices and allocations via shortest path

After the exchange graph is built, the algorithm is trivial: compute the minimal-length shortest path from some ϕ_i (i.e. among all paths of minimum weight between ϕ_i and k, pick the one with minimum length). Since the edges are non-negative, the shortest path can be computed in the order of the number of edges using Dijkstra's algorithm in $O(|E| + |V| \cdot \log |V|)$ where $|E|$ is the number of edges in the graph and $|V|$ is the number of vertices. Dijkstra's algorithm can be easily modified to compute the minimum weight path with shortest length using the following idea: if weights are integers, then substitute weights w_{ij} by $w_{ij} - \epsilon$. In the end, round the solution up. Or more formally, run Dijkstra in the ordered ring $(\mathbb{Z}^2, +, <)$ with weights $(w_{ij}, 1)$ where $+$ is the componentwise sum and $<$ is the lexicographic order.

Let P be the path output by Dijkstra. Update the allocation by performing the swaps prescribed by P. In other words, if edge $(t, j) \in P$ and $t \in S_i$, then we swap t by j in S_i. Also, if edge $(\phi_i, j) \in P$ we add j in S_i. Formally, let $(t_r, j_r)_{r=1..a}$ be all the edges in P with $t_r \in S_i$ or $t_r = \phi_i$. Then we update S_i to $S_i \cup \{j_1, \ldots, j_a\} - \{t_1, \ldots, t_a\}$.

The execution of Dijkstra also produces a certificate of optimality of the shortest path in the form of the minimum distance from some ϕ_i to any given node. So, there is a distance function d such that

$$d(\phi_i) = 0, \qquad d(j) \leq d(\phi_i) + w_{\phi_i j}, \qquad d(j) \leq w_{tj} + d(t)$$

Moreover, for all edges (t, j) and (ϕ_i, j) in the shortest path P, this holds with equality, i.e.: $d(j) = d(\phi_i) + w_{\phi_i j}$ and $d(j) = w_{jt} + d(t)$. Update the price of each item j from p_j to $p_j - d(j)$.

23.2.3 Running time analysis

Before we show that each iteration produces an optimal pair of allocation S and prices p for problem (I_k) we analyze the running time.

The exchange graph for problem (I_k) as previously described has $O(mk + k^2)$ edges. Running Dijkstra's algorithm on this graph has running time $\tilde{O}(mk + k^2)$ for (I_k), which corresponds to $\tilde{O}(mn^2 + n^3)$ time overall.

In order to get the overall running time down to $\tilde{O}(mn + n^3)$ we need one extra observation. Since we want to compute the shortest path from any of the ϕ_i nodes to k, we can collapse all ϕ_i nodes to a single node ϕ. Now, for any given node j:

$$w_{\phi j} = \min_i w_{\phi_i j} = p_j + \min_i [v_i(S_i) - v_i(S_i \cup j)]$$

Now, the graph is reduced to $O(k^2)$ edges for problem (I_k). So, Dijkstra can be computed in $\tilde{O}(k^2)$. We are only left with the task to compute $w_{\phi j}$. Our task is to compute $\min_i [v_i(S_i) - v_i(S_i \cup j)]$. This can be divided in two parts:

1. *active agents*: the minimum among the agents i for which $S_i \neq \varnothing$. There are at most k of those, so we can iterate over all of them and compute the minimum explicitly;
2. *inactive agents*: the minimum over all agents with $S_i = \varnothing$. In order to do so we maintain the following data structure: in the first iteration, i.e. in (I_1), we compute $v_i(\{j\})$ for every i, j (which takes $O(mn)$ time) and keep for each item j a sorted list $\mathtt{L}_j$ in decreasing order of $v_i(j)$ for all i.

   In the end of each iteration, whenever an inactive agent i becomes active (i.e. we allocate him an item), we remove them from $\mathtt{L}_j$ for all j. This operation takes $O(n)$ time to go over all lists.

Now, once we have this structure, we can compute the minimum among the inactive buyers $\min_i[v_i(S_i) - v_i(S_i \cup j)] = -\max_i v_i(\{j\})$ by looking at the minimum element of the list $\mathtt{L}_j$. Therefore, even though we need to pay $O(mn)$ time in (I_1). In each subsequent iteration we pay only $O(n)$ to update lists $\mathtt{L}_j$ and then we can make query the value of $w_{\phi j}$ in constant time.

This leads to a running time of $O(mn)$ in (I_1) and $\tilde{O}(n + k^2)$ in each subsequent iteration, leading to an overall running time of $\tilde{O}(mn + n^3)$. We also note that for each edge of the graph, we query the value oracle once. So the oracle complexity is $O(nm + n^3)$ value oracle calls.

23.2.4 Correctness

We are left to argue that the solution (S, p) produced by the algorithm is indeed a valid solution for problem (I_k). This can be done by checking that the price vector p obtained certifies the optimality of S. The main ingredients for the proof are Lemmas 135 and 136. We encourage the reader to revisit the statement of those lemmas before reading the proof of the following theorem.

Theorem 147. *Let S_i^k, p_j^k be the solution of problem (I_k), then for all agent i and all $S' \subseteq [k]$,*

$$v_i(S_i^k) - p^k(S_i^k) \geq v_i(S') - p^k(S').$$

Proof. Let S_i^{k-1}, p_j^{k-1} be the solution of problem (I_{k-1}) and let S_i^k, p_j^k be the solution of problem (I_k). If $d(\cdot)$ is the distance function returned by Dijkstra in (I_k), then $p_j^k = p_j^{k-1} - d(j)$ and we also know that $\tilde{w}_{jt} = w_{jt} + d(t) - d(j) \geq 0$ and $\tilde{w}_{\phi_i j} = w_{\phi_i j} - d(j) \geq 0$ by the observation in the end of the Section 23.2.2. This implies that for all i, for all $j \notin S_i^{k-1}$ and $t \in S_i^{k-1}$, it holds that:

$$\tilde{w}_{tj} = v_i(S_i^{k-1}) - v_i(S_i^{k-1} \cup j - t) + p_j^k - p_t^k \geq 0$$

$$\tilde{w}_{\phi_i j} = v_i(S_i^{k-1}) - v_i(S_i^{k-1} \cup j) + p_j^k \geq 0$$

This means that properties Add and Swap are satisfied. To see that Remove is also satisfied for $j < k$ since $v_i(S_i^{k-1}) \geq v_i(S_i^{k-1} - j) + p_j^{k-1}$ for all $j \in S_i^{k-1}$. Since $p_j^k \leq p_j^{k-1}$, this condition must continue to hold.

This means that under the price vector p^k the bundles S_i^{k-1} is still the demanded bundle by agent i (by Definition 122). This means in particular that for all $S' \subseteq [k]$:

$$v_i(S_i^{k-1}) - p^k(S_i^{k-1}) \geq v_i(S') - p^k(S')$$

Now, let $(t_r, j_r)_{r=1..a}$ be the set of edges in the path P output by Dijkstra where $t_r \in S_i^{k-1}$. Then $S_i^k = S_t^{k-1} \cup \{j_1, \ldots, j_a\} - \{t_1, \ldots, t_a\}$.

Using Lemma 136, we note that $(t_1, r_1), \ldots, (t_a, r_a)$ is a unique minimum matching in the sense of Lemma 135. Therefore:

$$v_i(S^i k - 1_i) - v_i(S_i^k) = \sum_{r=1}^{a} v_i(S_i^{k-1}) - v_i(S_i^{k-1} \cup j_r - t_r)$$

Summing $-p^k(S_i^{k-1}) + p^k(S_i^k)$ on both sides, we get:

$$[v_i(S_i^{k-1}) - p^k(S_i^{k-1})] - [v_i(S_i^k) - p^k(S_i^{k-1})] = \sum_{r=1}^{a} \tilde{w}_{t_r j_r} = 0$$

Therefore:

$$v_i(S_i^k) - p^k(S_i^{k-1}) = v_i(S_i^{k-1}) - p^k(S_i^{k-1}) \geq v_i(S') - p^k(S')$$

as desired. □

23.2.5 Descending Auction View

One can reinterpret the procedure above as a descending auction. Initially all items very large price (say like the price set for p_k in the beginning of phase k). Each shortest path computation produces a distance function d that dictates how each price should decrease. Indeed, they monotonically decrease until we reach a Walrasian equilibrium.

We note that it is important in this algorithm that we compute in each step both a primal and a dual solution. Without the dual solution (the price vector), it is still possible to carry out the shortest path computation, but since the edges in the path can have mixed signs, Dijkstra's algorithm is no longer available and one needs to pay an extra factor of n to run Bellman-Ford's algorithm.

24 Missing Proofs

Proof of Lemma 104. Let $\vec{cy} = (y^{(1)}, y^{(2)}, \ldots, y^{(m)})$ be a valid allocation that achieves the optimal social welfare. Then since $x^{(i)} \in D(i, \vec{cp})$, we have

$$v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)} \geq v_i(y^{(i)}) - \vec{cp} \cdot y^{(i)}.$$

Summing up, we get

$$\sum_i \left(v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)}\right) \geq \sum_i \left(v_i(y^{(i)}) - \vec{cp} \cdot y^{(i)}\right).$$

the crucial observation is that $\sum_i \vec{cp} \cdot x^{(i)} = \sum_i \vec{cp} \cdot y^{(i)} = \sum_j p_j s_j$. herefore the inequality above simplifies to

$$\sum_i v_i(x^{(i)}) \geq \sum_i v_i(y^{(i)}),$$

i.e. the social welfare of $\vec{cx}$ is at least that of $\vec{cy}$. But since $\vec{cy}$ gives the optimal social welfare, we must then have equality and $\vec{cx}$ also achieves the optimum.

For the second part, notice that since the last equation holds with equality, then the previous equations should also hold, therefore: $\sum_i v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)} = \sum_i v_i(y^{(i)}) - \vec{cp} \cdot y^{(i)}$, which says that x_i is also a favorite bundle of i under price vector $\vec{cp}$. Therefore, $(\vec{cx}, \vec{cp})$ form a Walrasian equilibrium. □

Proof of Lemma 108. If $(\vec{cp}^{\mathsf{eq}}, \vec{cx})$ is a Walrasian equilibrium it is straightforward to check that setting $\vec{cp} = \vec{cp}^{\mathsf{eq}}$, $u_i = \max_{x_i \in [\![s]\!]} v_i(x) - \vec{cp}^{\mathsf{eq}} \cdot x$ and $z_{i,x} = 1$ when $x = x^{(i)}$ and zero otherwise, we have a primal dual pair of feasible solutions with the same value. Conversely, if the primal program has an integral solution, the definition of Walrasian equilibrium can be obtained from the complementarity conditions.

If the primal program has an optimal integral solution $\vec{cx}$, then for every solution $(\vec{cp}, \vec{cu})$ to the dual program: $\sum_i v_i(x^{(i)}) = \sum_i u_i + \vec{cp} \cdot \vec{cs} \geq \sum_i v_i(x^{(i)}) + \vec{cp} \cdot x_i \geq \sum_i v_i(x^{(i)})$ and therefore all inequalities must hold with equality, so in particular $x^i \in D(v_i, \vec{cp})$, so $\vec{cp}$ is a vector of Walrasian prices. Conversely, if $\vec{cp}$ is a vector of Walrasian prices then $(\vec{cx}, \vec{cp})$ is Walrasian equilibrium by the Second Welfare Theorem (Lemma 104). Therefore by setting $u_i = v_i(x^{(i)}) - \vec{cp} \cdot x^{(i)}$ we obtain a dual feasible solution such that $\sum_i u_i + \vec{cp} \cdot \vec{cs} = \sum_i v_i(x^{(i)})$ and therefore $(\vec{cp}, \vec{cu})$ is an optimal dual solution. □

Proof of Corollary 133. Let $\vec{cp}^{\mathsf{eq}}$ and $\vec{cS} = (S_1, S_2, \ldots, S_m)$ be an equilibrium price and allocation. Consider the following chain of inequalities:

$$\sum_i v_i(S_i) \leq \tilde{f}(\vec{cp}^*) \leq \tilde{f}(\vec{cp}^{\mathsf{eq}}) \leq f(\vec{cp}^{\mathsf{eq}}) = \sum_i v_i(S_i)$$

Where the first inequality follows from the definition of $\tilde{f}$, the second from the fact that $\vec{cp}^*$ is a minimizer of $\tilde{f}$, the third follows from the fact that $\tilde{f} \leq f$ for all prices $\vec{cp}$, since f is a maximization over all $S_i \subseteq [n]$ and $\tilde{f}$ is a maximization over all subsets whose cardinality is exactly $[n]$. The last inequality follows from the fact that $\vec{cp}^{\mathsf{eq}}$ is an equilibrium. This implies that all inequalities should hold with equality, in particular, since $\tilde{f}(\vec{cp}^*) = \sum_i v_i(S_i)$, then it must be that:

$$\max_{S \subseteq [n]} v_i(S) - \vec{cp}^*(S) - \gamma \cdot |S| = v_i(S_i) - \vec{cp}^*(S_i) - \gamma \cdot |S_i|$$

In particular, $S_i \in D(i, \vec{cp}^* + \gamma \cdot \vec{c1}_{[n]})$.

The other direction is similar. We have $\tilde{f}(\vec{cp}^{\mathsf{eq}}) = \sum_i v_i(S_i) = f(\vec{cp}^{\mathsf{eq}})$ and for any price p,

$$\tilde{f}(\vec{cp}) = f(\vec{cp} + \gamma \cdot \vec{c1}_{[n]}) \geq f(\vec{cp}^{\mathsf{eq}}) = \tilde{f}(\vec{cp}^{\mathsf{eq}})$$

which shows that any Walrasian price $\vec{cp}^{\mathsf{eq}}$ minimizes $\tilde{f}$. □

Proof of Lemma 136 (sketch). Assume that the bipartite graph has a different matching with total weight not larger than the one presented. Then it is possible to construct either a cycle of weight less than C or a cycle of the same weight with smaller number of edges.

Let M' be an alternative matching between U and V of weight at most the weight of M. If M' has smaller weight, replace M by M' and $C \cup M' - M$ is a collection of cycles with total weight smaller than the weight of C. Since all cycles have non-negative weight, one of the cycles must have weight less than C, contradicting the fact that C is a minimum weight cycle.

Now, if M and M' have the same weight, consider the following family of cycles: for each edge $e = (u', v') \in M'$, construct a cycle C_e composed of edge e and the path from v' to u' in C (in other words, we use e to shortcut C). There is an integer $k \leq t$ such that the collection of cycles C_e uses in total: one of each edge from M', $k-1$ of each edge from M and k of each edge from $C - M$. So the average weight is at most the weight of C. Since the C_e cycles have strictly less edges than C, there should be a cycle with fewer edges than C and weight at most C, which again contradicts the choice of C.

The argument for paths is analogous. □

Part V
Submodular Function Minimization via First Order Method

This Part is based on joint works with Deeparnab Chakrabarty, Yin Tat Lee and Aaron Sidford.

25 Introduction

Submodular functions are set functions that prescribe a value to every subset of a finite universe U and have the following diminishing returns property: for every pair $S \subseteq T \subseteq U$, and for every element $i \notin T$, $f(S \cup i) - f(S) \geq f(T \cup i) - f(T)$. Such functions arise in many applications. For instance, the utility functions of agents in economics are often assumed to be submodular, the cut functions in directed graphs or hypergraphs are submodular, the entropy of a given subset of random variables is submodular, etc. Submodular functions have been extensively studied for more than five decades [42, 68, 186, 86, 192].

One of the most important problems in this area is submodular function minimization (SFM, henceforth) which asks to find the set S minimizing $f(S)$. Note that submodular functions need not be monotone and therefore SFM is non-trivial. In particular, SFM generalizes the minimum cut problem in directed graphs and hypergraphs, and is a fundamental problem in combinatorial optimization. More recently, SFM has found many applications in areas such as image segmentation [27, 157, 158] and speech analysis [184, 185]. Owing to these large scale problems, fast SFM algorithms are highly desirable.

We assume access to an *evaluation oracle* for the submodular function, and use EO to denote the time taken per evaluation. An amazing property of submodular functions is that SFM can be exactly solved with polynomial many queries and in polynomial time. This was first established via the ellipsoid algorithm [112] in 1981, and the first polynomial combinatorial algorithms were obtained [49, 129, 232, 134] much later.

The current fastest algorithms for SFM are from previous Part which give $O(n^2 \log nM \cdot \mathrm{EO} + n^3 \log^{O(1)} nM)$ time and $O(n^3 \log^2 n \cdot \mathrm{EO} + n^4 \log^{O(1)} n)$ time algorithms for SFM. Here M is the largest absolute value of the integer-valued function. The former running time is a (weakly) polynomial running time, i.e. it depends polylogarithmically on M, while the latter is a strongly polynomial running time, i.e. it does not depend on M at all. Although good in theory, known implementations of the above algorithms are slow in practice [87, 88, 17, 35]. A different algorithm, the so-called Fujishige-Wolfe algorithm [260, 84], seems to have the best empirical performance [17, 144, 22] among general purpose SFM algorithms. Recently the Fujishige-Wolfe algorithm and variants were shown [35, 168] to run in $O((n^2 \cdot \mathrm{EO} + n^3)M^2)$ time, proving them to be *pseudopolynomial* time algorithms, that is having running time $O(\mathsf{poly}(n, \mathrm{EO}, M))$.

In this Part we also consider approximate SFM. More precisely, for submodular functions whose values are in the range $[-1, +1]$ (which is without loss of generality by scaling), we want to obtain *additive* approximations[13], that is, return a set S with $f(S) \le \text{opt} + \epsilon$. Although approximate SFM has not been explicitly studied before, previous works [180, 17, 35] imply $O(n^2\text{EO}\log^{O(1)}(n/\epsilon))$-time and $O((n^2 \cdot \text{EO} + n^3)/\epsilon^2)$-time algorithms. Table 6 summarizes the above discussion.

Regime	Previous Best Running Time	Our Result	Techniques
Strongly Poly	$O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$ [180]		CP + Dimension Collapsing
Weakly Poly	$O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$[180]		Cutting Plane (CP)
Pseudopoly	$O((n^2 \cdot \text{EO} + n^3)M^2)$[35, 168]	$\tilde{O}(nM^3 \cdot \text{EO})$	See Section 25.2
ϵ-Approx	$O(n^2 \cdot \text{EO}/\epsilon^2)$ [35, 168, 17]	$\tilde{O}(n^{5/3} \cdot \text{EO}/\epsilon^2)$	See Section 25.2

Table 6: Running times for minimizing a submodular function defined on a universe of size n that takes integer values between $-M$ and M (except for ϵ-approximate algorithms we assume the submodular function is real-valued with range in $[-1, 1]$). EO denotes the time to evaluate the submodular function on a given set.

In particular, the best known dependence on n is *quadratic* even when the exact algorithms are allowed to be pseudopolynomial, or when the ϵ-approximation algorithms are allowed to have a polynomial dependence on ϵ. This quadratic dependence seems to be a barrier. For exact SFM, the smallest known non-deterministic proof [68, 49] that certifies optimality requires $\Theta(n^2)$ evaluation queries, and even for the approximate case, nothing better is known (see Section 29.4). Furthermore, in this Part we *prove* that a large class of algorithms which includes the Fujishige-Wolfe algorithm [260, 84] and the cutting plane algorithms of Lee et al. [180], as stated need to make $\Omega(n^2)$ queries. More precisely, these algorithms do not exploit the full power of submodularity and work even with the weaker model of having access only to the "subgradients of the Lovasz Extension" (or equivalently, vertices of the base polyhedron) where each subgradient takes $\Theta(n)$ queries. We prove that any algorithm must make $\Omega(n)$ subgradient calls implying the quadratic lower bound for this class of algorithms. Furthermore, our lower bound holds even for functions with range $\{-1, 0, 1\}$, and so trivially the lower bound also holds for approximate SFM as well. In other words, our work shows that faster (approximate) algorithms are only possible when one goes beyond the subgradient model.

25.1 Our Results

In this Part, we describe exact and approximate algorithms for SFM which run in time sub-quadratic in the dimension n. Our first result is a nearly linear time exact SFM algorithm for bounded, integer-valued submodular function. More precisely, for any integer valued submodular function with maximum absolute value M, our algorithm returns the optimum solution in $O(nM^3 \log n \cdot \text{EO})$ time. *This algorithm may especially be of interest in various settings such as market equilibrium computation in economics (e.g. [150]) where M is typically a small constant and querying the oracle is expensive.*

[13]We also show in Section 29.1 how to obtain a multiplicative approximation under a mild condition on f that has been used widely for various submodular *maximization* problems. Such a condition is necessary as multiplicative approximation is equivalent to exact optimization for general SFM instances.

Moreover, this has a few consequences to the complexity theory of SFM. First, this gives a better dependence on n for pseudopolynomial time algorithm. Second, this shows that to get a super-linear lower bound on the query complexity of SFM, one need to consider a function with super constant function values.[14] Third, this completes the following picture on the complexity of SFM: the best known strongly polynomial time algorithms have query complexity $\tilde{O}(n^3)$, the best known (weakly) polynomial time algorithms have query complexity $\tilde{O}(n^2)$, and our result implies the best pseudopolynomial time algorithm has query complexity $\tilde{O}(n)$. These theoretical consequences were the primary driver of our work and we hope it may open the door to further improvements in more structured setting; *indeed, our next two results, which build upon this framework, demonstrate that better running times can be achieved by exploiting different solution concepts (approximation) or problem-specific structures (sparsity).*

Our second result is a *subquadratic approximate SFM* algorithm. More precisely, we give an algorithm which in $\tilde{O}(n^{5/3}\text{EO}/\epsilon^2)$ time, returns an ϵ-additive approximate solution. To break the quadratic barrier, that arise from the need to compute $\Omega(n)$ subgradient each of which individully we do not know how to compute faster than $\Omega(n \cdot \text{EO})$, we wed continuous optimization techniques with properties deduced from submodularity and simple data structures. These allow us to compute and use subgradient updates in a much more economical fashion. We believe that that the ability to obtain subquadratic approximate algorithms for approximate submodular minimization is an interesting structural result that could have further implications.[15]

Finally, we show how to improve upon these results further if we know that the optimal solution is sparse. This may be a regime of interest for certain applications where the solution space is large (e.g. structured predictions have exponentially large candidate sets [223]), and as far as we are aware, no other algorithm gives sparsity-critical results.

25.2 Overview of Techniques

In a nutshell, all are our algorithms are projected, stochastic subgradient descent algorithms on the Lovasz extension $\hat{f}$ of a submodular function with economical subgradient updates. The latter crucially uses submodularity and serves as the point of departure from previous black-box continuous optimization based methods. In this section, we give a brief overview of our techniques.

The Lovasz extension $\hat{f}$ of a submodular function is a convex relaxation. It is a non-smooth convex function whose (approximate) minimizers leads to (approximate) SFM. Subgradient descent algorithms maintain a current iterate $x^{(t)}$ and take a step in the negative direction of a subgradient $g(x^{(t)})$ at $x^{(t)}$ to get the next iterate $x^{(t+1)}$. In general, the subgradient of a Lovasz extension takes $O(n \cdot \text{EO})$ to compute. As stated above, the $\Omega(n)$ lower bound on the number of iterations needed, implies that if we naively recompute the subgradients at every iterations, we cannot beat the quadratic barrier. Our main technical

[14]Conversely, [116, Thm 5.7] shows that we need at least n queries of evaluation oracle to minimize a submodular function with range in $\{0, 1, 2\}$.

[15]Note that simple graph optimization problems, such as directed minimum s-t cut, is not one of these (See Section 29.3).

contribution is to exploit submodularity so that $g(x^{(t+1)})$ can be computed in sublinear time given $x^{(t)}$ and $g(x^{(t)})$.

The first implication of submodularity is the observation (also made by [143, 120]) that ℓ_1-norms of the subgradients are bounded by $O(M)$ if the submodular function is in $[-M, M]$. When the function is integer-valued, this implies that the subgradients are sparse and have only $O(M)$ non-zero entries. Therefore, information theoretically, we need only $O(M)$ bits to get $g(x^{(t+1)})$ from $g(x^{(t)})$. However, we need an algorithm to find the positions at which they differ. To do so, we use submodularity again. We observe that given any point x and non-negative, k-sparse vector e, the difference vector $d := g(x+e) - g(x)$ is non-positive at points corresponding to support of e and non-negative everywhere else. Furthemore, on a "contiguous set" of coordinates, the sum of these entries in d can be computed in $O(\text{EO})$ time. Armed with this, we create a binary search tree (BST) type data structure to find the $O(M)$ non-zero coordinates of d in $O(M \cdot \text{EO} \log n)$ time (as opposed to $O(n \cdot \text{EO})$ time). This, along with standard subgradient descent analysis yields our $O(nM^3 \text{EO} \log n)$-algorithm.

When the submodular function is real-valued between $[-1, 1]$, although the ℓ_1-norm is small the subgradient can have full support. Therefore, we cannot hope to evaluate the gradient in sublinear time. We resort to stochastic subgradient descent where one moves along a direction whose expected value is the negative subgradient and whose variance is bounded. Ideally, we would have liked a fast one-shot random estimation of $g(x^{(t+1)})$; unfortunately we do not know how to do it. What we can do is obtain fast estimates to the difference vector d mentioned above. As discussed above, the vector d has $O(k)$ "islands" of non-negative entries peppered with $O(k)$ non-positive entries. We maintain a data-structure which with $O(k\text{EO} \log n)$ preprocessing time can evaluate the sums of the entries in these islands in $O(\text{EO} \log n)$ time. Given this, we can sample a coordinate $j \in [n]$ with probability proportional to $|d_j|$ in a similar time. Thus we get a random estimate of the vector d whose variance is bounded by a constant.

To get the stochastic subgradient, however, we need to add these difference vectors and this accumulates the variance. To keep the variance in control, we run the final algorithm in batches. In each batch, as we progress we take more samples of the d-vector to keep the variance in check. This unfortunately increases the sparsity (the k parameter), and one needs to balance the effects of the two. At the end of each batch, we spend $O(n\text{EO})$ time computing the deterministic subgradient and start the process over. Balancing the number of iterations and length of batches gives us the $\tilde{O}(n^{5/3}\text{EO}\epsilon^{-2})$-time algorithm for ϵ-approximate SFM.

25.3 Related Work

Submodularity, and indeed SFM, has a rich body of work and we refer the reader to surveys of Fujishige [86] and McCormick[192] for a more detailed pre-2006 version. Here we mention a few subsequent related works which were mostly inspired by applications in machine learning.

Motivated by applications in computer vision [27, 26] which require fast algorithms for SFM, researchers focused on minimization of *decomposable* submodular functions which are expressible as sum of "simple" submodular functions. It is assumed that simple submodular functions can be minimized fast (either in practice or in theory). Such a study was initiated by Stobbe and Krause [242] and Kolmogorov [161] who gave faster (than general SFM) algorithms for such functions. More recently, motivated by work of Jegelka et al. [142], algorithms with *linear* convergence rates [216, 72] have been obtained. That is, they get ϵ-approximate algorithms with dependence on ϵ being $\log(1/\epsilon)$. .

We end our introductory discussion by mentioning the complexity of *constrained* SFM where one wishes to minimize over sets satisfying some constraints. In general constrained SFM is much harder than unconstrained SFM. For instance the minimum cut problem with cardinality constraints becomes the balanced partitioning problem which is APX-hard. More generally, Svitkina and Fleischer [244] show that a large class of constrained SFM problems cannot be approximated to better than $\tilde{O}(\sqrt{n})$ factors without making exponentially many queries. In contrast, Goemans and Soto [104] prove that symmetric submodular functions can be minimized over a large class of constraints. Inspired by machine learning applications, Iyer et al. [136, 135] give algorithms for a large class of constrained SFM problems which have good approximation guarantees if the *curvature* of the functions are small.

26 Preliminaries

Here we introduce notations and general concepts used throughout this Part.

26.1 General Notation

We let $[n] \stackrel{\text{def}}{=} \{1, ..., n\}$ and $[0,1]^n \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n : x_i \in [0,1] \ \forall i \in [n]\}$. Given a permutation $P = (P_1, ..., P_n)$ of $[n]$, let $P[j] \stackrel{\text{def}}{=} \{P_1, P_2, ..., P_j\}$ be the set containing the first j elements of P. Any point $x \in \mathbb{R}^n$ defines the permutation P_x *consistent with* x where $x_{P_1} \geq x_{P_2} \geq ... \geq x_{P_n}$with ties broken lexicographically. We denote by $\mathbf{1}_i \in \mathbb{R}^n$ the indicator vector for coordinate i, i.e. $\mathbf{1}_i$ has a 1 in coordinate i and a 0 in all other coordinates. We call a vector *s-sparse* if it has at most s non-zero entries.

26.2 Submodular Functions

Throughout this Part $f : 2^U \to \mathbb{R}$ denotes a submodular function on a ground set U. For notational convenience we assume without loss of generality that $U = [n]$ for some positive integer n and that $f(\varnothing) = 0$ (as this can be enforced by subtracting $f(\varnothing)$ from for the value of all sets while preserving submodularity). Recall that f is submodular if and only if it obeys the property of diminishing marginal returns: for all $S \subseteq T \subseteq [n]$ and $i \notin T$ we have

$$f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T) \,.$$

We let $\text{opt} \stackrel{\text{def}}{=} \min_{S \subseteq [n]} f(S)$ be the minimum value of f. We denote by EO the time it takes to evaluate f on a set S. More precisely, we assume given a linked list storing a permutation P of $[n]$, and a position k, we can evaluate $f(P[k])$ in EO time.

26.3 The Lovasz Extension

Our results make extensive use of the Lovasz extension, a convex, continuous extension of a submodular function to the interior of the n-dimensional hypercube, i.e. $[0,1]^n$.

Definition 148 (Lovasz Extension)**.** Given a submodular function f, the Lovasz extension of f, denoted as $\hat{f} : [0,1]^n \to \mathbb{R}$, is defined for all $x \in [0,1]^n$ by $\hat{f}(x) = \sum_{j\in[n]}(f([P[j]) - f(P[j-1]))x_{i_j}$ where $P = P_x = (P_1, ..., P_n)$ is the permutation consistent with x.

Note that since $f(\varnothing) = 0$ this definition is equivalent to

$$\hat{f}(x) = f(P[n])x_{P_n} + \sum_{j\in[n-1]} f([P[j])(x_{P_j} - x_{P_{j+1}})\,. \tag{26.1}$$

We make use of the following well known facts regarding submodular functions (see e.g. [186, 86]).

Theorem 149 (Lovasz Extension Properties)**.** *The following are true for all $x \in [0,1]^n$:*

- **Convexity**: The Lovasz extension is convex.
- **Consistency**: For $x \in \{0,1\}^n$ we have $\hat{f}(x) = f(S(x))$ where $S(x) = \{i \in S : x_i = 1\}$.
- **Minimizers**: $\min_{x\in[0,1]^n} \hat{f}(x) = \min_{S\subseteq[n]} f(S)$.
- **Subgradients**: The vector $g(x) \in \mathbb{R}^n$ defined by $g(x)_{P_k} \overset{\text{def}}{=} f(P[k]) - f(P[k-1])$ is a subgradient of $\hat{f}$ at x, where $P = P_x$ is the permutation consistent with x. Let us call this the Lovasz subgradient.

We conclude with a few straightforward computational observations regarding the Lovasz extension and its subgradients. First note that for $x \in [0,1]^n$ we can evaluate $\hat{f}(x)$ or compute $g(x)$ in time $O(n\text{EO} + n \log n)$ simply by sorting the coordinates of f and evaluating f at the n desired sets. Also, note that by (26.1) the Lovasz extension evaluated at $x \in [0,1]^n$ is a non-negative combination of the value of f at n sets. Therefore computing the smallest of these sets gives a set $S \subseteq [n]$ such that $f(S) \le \hat{f}(x)$ and we can clearly compute this, again in $O(n\text{EO} + n \log n)$ time. Therefore for any algorithm which approximately minimizes the Lovasz extension with some (additive) error ϵ, we can always find a set S achieving the same error on f by just paying an additive $O(n\text{EO} + n \log n)$ in the running time.

26.4 Subgradient Descent

Our algorithmic results make extensive use of subgradient descent (or mirror descent) and their stochastic analogs. Recall that for a convex function $h : \chi \to \mathbb{R}$, where $\chi \subseteq \mathbb{R}^n$ is a compact convex set, a vector $g \in \mathbb{R}^n$ is a *subgradient* of h at $x \in \chi$ if for all $y \in \chi$ we have

$$h(y) \ge h(x) + g^\top(y - x)\,.$$

For such an h we let $\partial h(x)$ denote the set of subgradients of h at x. An algorithm that on input x outputs $\tilde{g}(x) \in \partial h(x)$ is a *subgradient oracle* for h. Similarly, an algorithm that on input x outputs a *random* $\tilde{g}(x)$ such that $\mathbf{E}\tilde{g}(x) \in \partial h(x)$ is a *stochastic subgradient oracle* for h.

One of our main algorithmic tools is the well known fact that given a (stochastic) subgradient oracle we can minimize a convex function h. Such algorithms are called *(stochastic) subgradient descent* algorithms and fall into a more general framework of algorithms known as *mirror descent*. These algorithms are very well studied and there is a rich literature on the topic. Below we provide one specific form of these algorithms adapted from [29] that suffices for our purposes.

Theorem 150 (Projected (Stochastic) Subgradient Descent[16])**.** *Let $\chi \subseteq \mathbb{R}^n$ denote a compact convex set, $h : \chi \to \mathbb{R}$ be a convex function, $\tilde{g}$ be a (stochastic) subgradient oracle for which $\mathbf{E}\left\|\tilde{g}(x)\right\|_2^2 \leq B^2$ for all $x \in \chi$, and $R^2 \overset{\text{def}}{=} \sup_{x\in\chi} \frac{1}{2}\left\|x\right\|_2^2$. Now consider the iterative algorithm starting with*

$$x^{(1)} := \text{argmin}_{x\in\chi}\left\|x\right\|_2^2$$

and for all s we compute

$$x^{(s+1)} := \text{argmin}_{x\in\chi}\left\|x - (x^{(s)} - \eta\tilde{g}(x^{(s)}))\right\|_2^2$$

Then for $\eta = \frac{R}{B}\sqrt{\frac{2}{t}}$ we have

$$\mathbf{E}h\left(\frac{1}{t}\sum_{i\in[t]} x^{(s)}\right) - \min_{x\in\chi} h(x) \leq RB\sqrt{\frac{2}{t}}\,.$$

We refer to this algorithm as projected stochastic subgradient descent when $\tilde{g}$ is stochastic and as projected subgradient descent when $\tilde{g}$ is deterministic, though we often omit the term projected *for brevity. Note that when $\tilde{g}$ is deterministic the results are achieved exactly rather than in expectation.*

27 Faster Submodular Function Minimization

In this section we provide faster algorithms for SFM. In particular we provide the first nearly linear time pseudopolynomial algorithm for SFM and the first subquadratic additive approximation algorithm for SFM. Furthermore, we show how to obtain even faster running times when the SFM instance is known to have a sparse solution.

All our algorithms follow the same broad algorithmic framework of using subgradient descent with a specialized subgradient oracle. Where they differ is in how the structure of the submodular functions is exploited in implementing these oracles. The remainder of this section is structured as follows: in Section **27.1** we provide the algorithmic framework we use for SFM, in Section **27.2**, we prove structural properties of submodular functions that we use to compute subgradients, in Section **27.3**, we describe our nearly linear time

[16]This is Theorem 6.1 from [29] restated where we used the "ball setup" with $\Phi(x) = \frac{1}{2}\left\|x\right\|_2^2$ so that $\mathcal{D} = \mathbb{R}^n$ and $D_\Phi(x,y) = \frac{1}{2}\left\|x-y\right\|_2^2$. We also used that $\text{argmin}_{x\in\chi}\eta g^\top x + \frac{1}{2}\left\|x - x_t\right\|_2^2 = \text{argmin}_{x\chi}\left\|x - (x_t - \eta g)\right\|_2^2$.

pseodopolynomial algorithms, in Section **27.4**, we describe our subquadratic additive approximation algorithm, and in Section **27.5**, we show how to improve these results when SFM has a sparse solution.

We make minimal effort to control logarithmic factors in through this section and note that some of the factors come from sorting and therefore maybe can be removed depending on the desired computational model.

27.1 Algorithmic Framework

All our algorithms for SFM follow the same broad algorithmic framework. We consider the Lovasz extension $\hat{f} : [0,1]^n \to \mathbb{R}$, and perform projected (stochastic) subgradient descent on $\hat{f}$ over the convex domain $\chi = [0,1]^n$. While the subgradient oracle construction differs between algorithms (and additional care is needed to improve when the solution is sparse, i.e. Section 27.5) the rest of algorithms for Section 27.3 and Section 27.4 are identical.

In the following, Lemma 151, we encapsulate this framework, bounding the performance of projected (stochastic) subgradient descent for the Lovasz extension, i.e. applying Theorem 150 to $\hat{f}$ over $\chi = [0,1]^n$. Formally, we abstract away the properties of a subgradient oracle data structure needed to achieve a fast algorithm. With this lemma in place the remainder of the work in Section 27.2, Section 27.3, and Section 27.4 is to show how to efficiently implement the subgradient oracle in the particular setting.

Lemma 151. *Suppose that there exists a procedure which maintains $(x^{(i)}, \tilde{g}^{(i)})$ satisfying the invariants: (a) $\tilde{g}^{(i)}$ is k-sparse; (b) $\mathbf{E}[\tilde{g}^{(i)}] = g(x^{(i)})$ is the Lovasz subgradient at $x^{(i)}$; (c) $\mathbf{E}\|\tilde{g}^{(i)}\|_2^2 \le B^2$. Furthermore, suppose given any k-sparse $e^{(i)}$, the procedure can update to $(x^{(i+1)} = x^{(i)} + e^{(i)}, \tilde{g}^{(i+1)})$ in time $\mathrm{T_g}$. Then, for any $\epsilon > 0$, we can compute a set S with $\mathbf{E}[f(S)] \le \mathrm{opt} + \epsilon$ in time $O(nB^2\epsilon^{-2}\mathrm{T_g} + n\mathrm{EO} + n\log n)$. If invariants (b) and (c) hold without expectation, then so does our algorithm.*

Proof. We invoke Theorem 150 on the convex function $\hat{f} : [0,1]^n \to \mathbb{R}$ over the convex domain $\chi = [0,1]^n$ to obtain the iterates where we use the given subgradient oracle. Clearly

$$x^{(1)} = \mathrm{argmin}_{x\in[0,1]^n} \frac{1}{2}\|x\|_2^2 = 0 \in \mathbb{R}^n$$

and

$$R^2 = \sup_{x\in[0,1]^n} \frac{1}{2}\|x\|_2^2 = \frac{1}{2}\|1\|_2^2 = \frac{n}{2}.$$

Consequently, as long as we implement the projection step for $T = O(nB^2\epsilon^{-2})$ steps (each step requiring $\mathrm{T_g}$ time), then Theorem 150 yields

$$\mathbf{E}\hat{f}\left(\frac{1}{T}\sum_{i\in[T]} x^{(i)}\right) - \min_{x\in\chi} \hat{f}(x) \le RB\sqrt{\frac{2}{T}} \le \sqrt{\frac{nB^2}{T}} \le \epsilon\,.$$

Furthermore, as we argued in Section 26.3 we can compute S with $f(S) \le \hat{f}(\frac{1}{T}\sum_{i\in[T]} x^{(i)})$ in the time it takes to compute $\frac{1}{T}\sum_{i\in[T]} x^{(i)}$ plus additional $O(n\mathrm{EO} + n\log n)$ time. To prove the lemma all that remains is to reason about the complexity of computing the projection, i.e. $x^{(t+1)}$, given that all the subgradients we compute are s-sparse. However, since $x^{(t+1)} = \text{argmin}_{x\in[0,1]^n} \left\| x - (x^{(t)} - \eta\tilde{g}(x^{(t)})\right\|_2^2$ decouples coordinate-wise – note that $x^{(t+1)} = \text{median}\{0\,,\, x^{(t)} - \eta\tilde{g}(x^{(t)})\,,\, 1\}$, we subtract $\eta\tilde{g}(x^{(t)})$ from $x^{(t)}$ and if any coordinate is less than 0 we set it to 0 and if any coordinate is larger than 1 we set it to 1. Thus the edit vector $e^{(i)}$ is of sparsity$\leq k$. Combining these facts yields the described running time. □

27.2 Subgradients of the Lovasz Extension

Here we provide structural results of submodular function that we leverage to compute subgradients of submodular functions in $o(n)$ time. First, in Lemma 152 we state a result due to Jegelka and Bilmes [143] (also Hazan and Kale [120]) which puts an upper bound on the ℓ_1 norm of subgradients of the Lovasz extension provided that we have an upper bound on the maximum absolute value of the function. We provide a short proof for completeness.

Lemma 152 (Subgradient Upper Bound)**.** *If* $|f(S)| \leq M$ *for all* $S \subseteq [n]$*, then* $\left\|g(x)\right\|_1 \leq 3M$ *for all* $x \in [0,1]^n$ *and for all subgradients* g *of the Lovasz extension.*

Proof. For notational simplicity suppose without loss of generality (by changing the name of the coordinates) that $P(x) = (1,2,...,n)$, i.e. $x_1 \geq x_2 \geq ... \geq x_n$. Therefore, for any $i \in [n]$,we have $g_i = f([i]) - f([i-1])$. Let $r_1 \leq r_2 \leq ..., \leq r_R$ denote all the coordinates such that $g_{r_i} > 0$ and let $s_1 \leq s_2 \leq ... \leq s_S$ denote all the coordinates such that $g_{s_i} < 0$.

We begin by bounding the contribution of the positive coordinate, the g_{r_i}, to the norm of the gradient, $\left\|g\right\|_1$. For all $k \in [R]$ let $R_k \stackrel{\text{def}}{=} \{r_1, ..., r_k\}$ with $R_0 = \varnothing$. By assumption we know that that $f(R_0) = \varnothing$. Furthermore, by submodularity, i.e. diminishing marginal returns, we know that for all $i \in [R]$

$$f(R_i) - f(R_{i-1}) \geq f([r_i]) - f([r_i - 1]) =: g_{r_i} = |g_{r_i}|$$

Consequently $f(R_R) - f(R_0) = \sum_{i\in[R]} f(R_i) - f(R_{i-1}) \geq \sum_{i\in[R]} |g_{r_i}|$. Since $f(R_0) = 0$ and $f(R_R) \leq M$ by assumption we have that $\sum_{i\in[k]} |g_{r_i}| \leq M$.

Next, we bound the contribution of the negative coordinates, the g_{s_i}, similarly. For all $k \in [S]$ let $S_k \stackrel{\text{def}}{=} \{s_1, ..., s_k\}$ with $S_0 = \varnothing$. By assumption we know that that $f(S_0) = \varnothing$. Define $V := [n] - S$. Note that for all $i \in [S]$, the set $V \cup S_{i-1}$ is a superset of $[s_i - 1]$. Therefore, submodularity gives us for all $i \in [S]$,

$$f(V \cup S_i) - f(V \cup S_{i-1}) \leq f([s_i]) - f([s_i - 1]) = g_{s_i} = -|g_{s_i}|$$

Summing over all i, we get $f([n]) - f(V) \leq \sum_{i\in[S]} -|g_{s_i}|$. Since $f([n]) \geq -M$ and $f(V) \leq M$ we have that $\sum_{i\in[S]} |g_{s_i}| \leq 2M$. Combining these yields that $\left\|g\right\|_1 = \sum_{i\in[n]} |g_i| = \sum_{i\in[R]} |g_{r_i}| + \sum_{i\in[S]} |g_{s_i}| \leq 3M$. □

Next, in Lemma 153 we provide a simple but crucial monotonicity property of the subgradient of $\hat{f}$. In particular we show that if we add (or remove) a positive vector from $x \in [0,1]^n$ to obtain $y \in [0,1]^n$ then the *untouched* coordinates of the subgradients all decrease (or increase).

Lemma 153 (Subgradient Monotonicity)**.** *Let $x \in [0,1]^n$ and let $d \in \mathbb{R}^n_{\geq 0}$ be such that $y = x + d$ (resp. $y = x - d$). Let S denote the non-zero coordinates of d. Then for all $i \notin S$ we have $g(x)_i \geq g(y)_i$ (resp. $g(x)_i \leq g(y)_i$).*

Proof. We only prove the case of $y = x+d$ as the proof of the $y = x-d$ case is analogous. Let $P^{(x)}$and $P^{(y)}$ be the permutations consistent with x and y. Note that $P^{(y)}$ can be obtained from $P^{(x)}$ by moving a subset of elements in S to the left, and the relative ordering of elements *not in* S remains the same. Therefore, for any $i \notin S$, if r is its rank in $P^{(x)}$, that is, $P^{(x)}_r = i$, and $r'$is its rank in $P^{(y)}$, then we must have $P^{(y)}[r'] \supseteq P^{(x)}[r]$. By submodularity, $g(y)_i = f(P^{(y)}[r']) - f(P^{(y)}[r'-1]) \leq f(P^{(x)}[r]) - f(P^{(x)}[r-1])) = g(x)_i$. ☐

Lastly, we provide Lemma 154 giving a simple formula for the sum of multiple coordinates in the subgradient.

Lemma 154 (Subgradient Intervals)**.** *Let $x \in [0,1]^n$ and let P be the permutation consistent with x. For any positive integers $a \leq b$, we have $\sum_{i=a}^{b} g(x)_{P_i} = f(P[b]) - f(P[a-1])$.*

Proof. This follows immediately from the definition of $g(x)$: $\sum_{i=a}^{b} g(x)_{P_i} = \sum_{i=a}^{b}(f(P[i]) - f(P[i-1])) = f(P[b]) + \sum_{i=a}^{b-1} f(P[i]) - \sum_{i=a}^{b-1} f(P[i]) - f(P[a-1])$. ☐

27.3 Nearly Linear in n, Pseudopolynomial Time Algorithm

Here we provide the first nearly linear time pseudopolynomial algorithm for submodular function minimization. Throughout this section we assume that our submodular function f is integer-valued with $|f(S)| \leq M$ for all $S \subseteq [n]$. Our goal is to deterministically produce an exact minimizer of f. The primary result of this section is showing that we can achieve this in $\tilde{O}(nM^3\text{EO})$ time:

Theorem 155. *Given an integer-valued submodular function f with $|f(S)| \leq M$ for all $S \subseteq [n]$ in time $O(nM^3\text{EO}\log n)$ we can compute a minimizer of f.*

We prove the theorem by describing $(x^{(i)}, \tilde{g}(x^{(i)}))$ in Lemma 151. In fact, in this case $\tilde{g}$ will *deterministically* be the subgradient of the Lovasz extension. In Lemma 156, we prove that the Lovasz subgradient is $O(M)$-sparse and so $\|g\|_2^2 \leq O(M^2)$. Thus, Conditions (a), (b), and (c) are satisfied with $B^2 = O(M^2)$. The main contribution of this section is Lemma 157, where we show $\text{T}_\text{g} = O(M \log n \cdot \text{EO})$, that is the subgradient can be updated in this much time. A pseudocode of the full algorithm can be found in Section 29.5.

Lemma 156. *For integer-valued f with $|f(S)| \leq M$ for all S the subgradient $g(x)$ has at most $3M$ non-zero entries for all $x \in [0,1]^n$.*

Proof. By Lemma 152 we know $\left\|g(x)\right\|_1 \leq 3M$. However, since $g(x)_{P_i} = f(P[i]) - f(P[i-1])$ and since f is integer-valued, we know that either $g(x)_{P_i} = 0$ or $|g(x)_{P_i}| \geq 1$. Consequently, there are at most $3M$ values of i for which $g(x)_i \neq 0$. ☐

Lemma 157. *With $O(n \cdot \text{EO})$ preprocessing time the following data structure can be maintained. Initially, one has input $x^{(0)} \in [0,1]^n$ and $g(x_0)$. Henceforth, for all i, given $g(x^{(i)})$ and a k-sparse vector $e^{(i)}$, in $O(k \log n + k\text{EO} + M\text{EO}\log n)$ time one can update $g(x^{(i)})$ to the gradient $g(x^{(i+1)})$ for $x^{(i+1)} = x^{(i)} + e^{(i)}$.*

Proof. The main idea is the following. Suppose $e^{(i)}$ is non-negative (we later show how to easily reduce to the case where all coordinates in $e^{(i)}$ have the same sign and the non-positive case is similar) Thus, by Lemma 153, for all coordinates not in support of $e^{(i)}$, the gradient goes down. Due to Lemma 156, the total number of change is $O(M)$, and since we can evaluate the sum of gradients on intervals by Lemma 154, a binary search procedure allows us to find *all* the gradient changes in $O(M \log n \cdot \text{EO})$ time. We now give full details of this idea.

We store the coordinates of $x^{(i)}$ in a balanced binary search tree (BST) with a node for each $j \in [n]$ keyed by the value of $x_j^{(i)}$; ties are broken consistently, e.g. by using the actual value of j. We take the order of the nodes $j \in [n]$ in the binary search tree to define the permutation $P^{(i)}$ which we also store explicitly in a link-list, so we can evaluate $f(P^{(i)}[k])$ in $O(\text{EO})$ time for any k. Note that each node of the BST corresponds to a subinterval of $P^{(i)}$ given by the children of that node in the tree. At each node of the BST, we store the sum of $g(x^{(i)})_j$ for all children j of that node, and call it the *value* of the node. Note by Lemma 154 each individual such sum can be computed with 2 calls to the evaluation oracle. Finally, in a linked list, we keep all indices j such that $g(x^{(i)})_j$ is non-zero and we keep pointers to them from their corresponding node in the binary search tree. Using the binary search tree and the linked list, one can clearly output the subgradient. Also, given $x^{(0)}$, in $O(n \cdot \text{EO})$ time one can obtain the initialization. What remains is to describe the update procedure.

We may assume that all non-zero entries of $e^{(i)}$ are the same sign; otherwise write $e^{(i)} := e_+^{(i)} + e_-^{(i)}$, and perform two updates. WLOG, lets assume the sign is $+$ (the other case is analogous). Let S be the indices of $e^{(i)}$ which are non-zero.

First, we change the key for each $j \in [n]$ such that $x_j^{(i+1)} \neq x_j^{(i)}$ and update the BST. Since we chose a consistent tie breaking rule for keying, only these elements $j \in [n]$ will change position in the permutation $P^{(i+1)}$. Furthermore, performing this update while maintaining the subtree labels can be done in $O(k \log n)$ time as it is easy to see how to implement binary search trees that maintain the subtree values even under rebalancing. For the time being, we retain the old values as is.

For brevity, let $g^{(i)}$ and $g^{(i+1)}$ denote the gradients $g(x^{(i)})$ and $g(x^{(i+1)})$, respectively. Since we assume all non-zero changes in $e^{(i)}$ are positive, by Lemma 153, we know that $g_j^{(i+1)} \leq g_j^{(i)}$ for all $j \notin S$. First, since $|S| \leq k$, for all $j \in S$, we go ahead and compute $g_j^{(i+1)}$ in $O(k\text{EO})$ time. For each such j we update the value of the nodes from j to the root, by adding the difference $(g_j^{(i+1)} - g_j^{(i)})$ to each of them. Next, we perform the following operation top-down start at the root: at each node we compare the current subtree value stored at this node with what the value actually should be with $g^{(i+1)}$. Note that since we know $P^{(i+1)}$, the latter can be computed with 2 evaluation queries. The simple but crucial observation is that if at any node j these two values match, then we are guaranteed that $g_k^{(i+1)} = g_k^{(i)}$ for all k in the tree rooted at j and we do not need to recurse on the children of this node. The reason for equality is that for all the children, we must have $g_k^{(i+1)} \leq g_k^{(i)}$ by Lemma 153 , and so if the sum is equal then we must have equality everywhere. Since there are at most $O(M)$ coordinates change, this takes $O(M\text{EO} \log n)$ for updating *all* the changes to $g^{(i+1)}$ for the binary search tree. During the whole process, whenever a node changes from non-zero to zero or from zero to non-zero, we can update the linked-list accordingly. □

Proof of Theorem 155. We apply Lemma 151 giving the precise requirements of our subgradient oracle. We know that the subgradients we produce are always $O(M)$ sparse by Lemma 156 and satisfy $B^2 = O(M^2)$. Consequently, we can simply instantiate Lemma 157 with $k = O(M)$ to obtain our algorithm. Furthermore, since f is integral we know that so long as we have a set additive error less than 1, i.e. $\epsilon < 1$, the set is a minimizer. Consequently, we can minimize in the time given by the cost of adding the cost of Lemma 2, with the Lemma 156 initialization cost, plus the Lemma 156 cost for $T = O(nM^2)$ iterations, yielding

$$O\left(n(\mathrm{EO} + \log n + M^3) + n + M\mathrm{EO} + (M \log n + M\mathrm{EO} \log n) \cdot nM^2\right) = O(nM^3\mathrm{EO} \log n)\,.$$

□

27.4 Subquadratic Additive Approximation Algorithm

Here we provide the first subquadratic additive approximation algorithm for submodular function minimization. Throughout this section we assume that f is real-valued with $|f(S)| \leq 1$ for all $S \subseteq [n]$. Our goal is to provide a randomized algorithm that produces a set $S \subseteq [n]$ such $\mathbf{E}f(S) \leq \mathrm{opt} + \epsilon$. Our primary result is showing that we can achieve this in $O(n^{5/3}\epsilon^{-2}\log^4 n)$ time:

Theorem 158. *Given a submodular function $f : 2^{[n]} \to \mathbb{R}$ with $|f(S)| \leq 1$ for all $S \subseteq V$, and any $\epsilon > 0$, we we can compute a random set S such that $\mathbf{E}f(S) \leq \mathrm{opt} + \epsilon$ in time $O(n^{5/3}\epsilon^{-2}\mathrm{EO}\log^4 n)$.*

The proof of this theorem has two parts. Note that the difficulty in the real-valued case is that we can no longer assume the Lovasz gradients are sparse, and so we cannot do naive updates. Instead, we use the fact that the gradient has small ℓ_2 norm to get sparse *estimates* of the gradient. This is the first part where we describe a sampling procedure which given any point x and a k-sparse vector e, returns a good and sparse estimate of the *difference* between the Lovasz gradient at $x + e$ and x. The second issue we need to deal with is that if we naively keep using this estimator, then the error (variance) starts to accumulate. The second part then shows how to use the sampling procedure in a "batched manner" so as to keep the total variance under control, restarting the whole procedure with a certain frequency. A pseudocode of the full algorithm can be found in Section 29.5.

Lemma 159. *Suppose a vector $x \in [0,1]^n$ is stored in a BST sorted by value. Given a k-sparse vector e which is either non-negative or non-positive, and an integer $\ell \geq 1$, there is a randomized sampling procedure which returns a vector z with the following properties: (a) $\mathbf{E}[z] = g(x+e) - g(x)$, (b) $\mathbf{E}[\|z - \mathbf{E}[z]\|_2^2] = O(1/\ell)$, and (c) the number of non-zero coordinates of z is $O(\ell)$. The time taken by the procedure is $O((k+\ell) \cdot \mathrm{EO}\log^2 n)$.*

Proof. We assume that each non-zero value of e is positive as the other case is analogous. Note that x is stored in a BST , and the permutation P_x consistent with x is stored in a doubly linked list. Let S be the set of positive coordinates of e with $|S| = k$ and let y denote the vector $x + e$. We compute P_y in $O(k \log n)$ time.

Let $I_1, \ldots, I_{2k} \subseteq [n]$ denote the subsets of the coordinates that correspond to the intervals which are contiguous in both P_x and P_y. Note that these are $\leq 2k$ such intervals, and some of them can be empty. We store the pointers to the endpoints of each interval in the BST. This can be done in $O(k \log n)$ time as follows. First compute the coarse intervals which are contiguous in P_x in $O(k)$ time. These intervals will be refined when we obtain P_y. In $O(k \log n)$ time, update the BST so that for every node we can figure out which coarse interval it lies in $O(\log n)$ time. This is done by walking up the BST for every end point of all the k intervals and storing which "side" of the interval they lie in. Given a query node, we can figure out which interval it lies in by walking up the BST to the root. Finally, for all nodes in S, when we update the BST in order to obtain P_y, using the updated data structure in $O(\log n)$ time figure out which coarse interval it lies in and refine that interval.

For each $j \in S$, we compute $d_j \stackrel{\text{def}}{=} g(y)_j - g(x)_j$ explicitly. This can be done in $O(k\text{EO})$time using $P_x$and P_y.For $r \in [2k]$, we define $D_r := \sum_{j \in I_r}(g(y)_j - g(x)_j)$. Since each I_r is a contiguous interval in both P_x and P_y, Lemma 154 implies that we can store all D_r in $O(k \cdot \text{EO})$ time in look-up tables. Note that by monotonicity Lemma 153 each summand in D_r is of the same sign, and therefore summing the absolute values of D_r's and d_j's gives $\|g(y) - g(x)\|_1$. We store this value of the ℓ_1 norm.

Now we can state the randomized algorithm which returns the vector z. We start by sampling either a coordinate $j \in S$ with probability proportional to $|d_j|$, or an interval I_r with probability proportional to $|D_r|$. If we sample an interval, then iteratively sample sub-intervals $I' \subset I_r$ proportional to $\sum_{j \in I'}(g(y)_j - g(x)_j)$ till we reach a single coordinate $j \notin S$. Note that any $j \in [n]$ is sampled with probability proportional to $|g(y)_j - g(x)_j|$.

We now show how to do this iterative sampling in $O((\text{EO} + \log n) \log n)$ time. Given I_r, we start from the root of the BST and find a node closest to the root which lies in I_r. More precisely, since for every ancestor of the endpoints of I_r, if it doesn't belong to the interval we store which "side" of the tree I_r lies in, one can start from the root and walk down to get to a node inside I_r. This partitions $I_r$into two subintervals and we randomly select I'proportional to $\sum_{j \in I'}(g(y)_j - g(x)_j)$. Since sub-intervals are contiguous in P_y and P_x, this is done in $O(\text{EO})$ time. We then update the information at every ancestor node of the endpoints of the sampled I' in $O(\log n)$ time. Since each iteration decreases the height of the least common ancestor of the endpoints of I', in $O(\log n)$ iterations (that is the height of the tree), we will sample a singleton $j \notin S$.

In summary, we can sample $j \in [n]$ with probability proportional to $g(y)_j - g(x)_j$ in $O((\text{EO} + \log n) \log n)$ time. If we sample j, we return the (random) vector

$$z := \|g(y) - g(x)\|_1 \cdot \mathsf{sign}(g(y)_j - g(x)_j) \cdot \mathbf{1}_j$$

where recall $\mathbf{1}_j$ is the vector with 1 in the jth coordinate and zero everywhere else. Note that given j, computing z takes $O(\text{EO} + \log n)$ time since we have to evaluate $g(y)_j$and $g(x)_j$. Recall, we already know the $\ell_1$norm. Also note by construction, $\mathbf{E}[z]$ is precisely the vector $g(y) - g(x)$. To upper bound the variance, note that

$$\mathbf{E}[\|z - \mathbf{E}z\|_2^2] \leq \mathbf{E}[\|z\|_2^2] = \|g(y) - g(x)\|_1^2 \leq 9 \cdot \max_{S \subseteq V} |f(S)| \leq 9$$

by Lemma 152 and the fact that $|f(S)| \leq 1$. Also observe that z is 1-sparse.

Given ℓ, we sample independently ℓ such random z's and return their *average*. The expectation remains the same, but the variance scales down by ℓ. The sparsity is at most

ℓ.The total running time is $O(k(\mathrm{EO}+\log n)+\ell(\mathrm{EO}+\log n)\log n)$. This completes the proof of the lemma.

□

We now complete the proof of *Theorem 158*.

Proof. (*Theorem 158*) The algorithm runs in batches (as mentioned before, the pseudocode is in Section 29.5.) At the beginning of each batch, we have our current vector $x^{(0)}$ as usual stored in a BST. We also compute the Lovasz gradient $g^{(0)} = g(x^{(0)})$ spending $O(n \log n \mathrm{EO})$ time. The batch runs for $T = \Theta(n^{1/3})$ steps. At each step $t \in [T]$, we need to specify an estimate $\tilde{g}^{(t)}$ to run the (stochastic) subgradient procedure as discussed in Lemma 151. For $t = 0$, since we know $g^{(0)}$ explicitly, we get $\tilde{g}^{(0)}$ by returning $\left\|g^{(0)}\right\|_1 \mathrm{sign}(g_j^{(0)})\mathbf{1_j}$ with probability proportional to $\left|g_j^{(0)}\right|$. This is a 1-sparse, unbiased estimator of $g^{(0)}$ with $O(1)$ variance. Define $z^{(0)} := \tilde{g}^{(0)}$. Henceforth, for every $t \geq 0$, the subgradient descent step suggests a direction $e^{(t)}$ in which to move whose sparsity is at most the sparsity of $\tilde{g}^{(t)}$. We partition $e^{(t)} = e_+^{(t)} + e_-^{(t)}$ into its positive and negative components. We then apply Lemma 159 twice: once with $x = x^{(t)}$,$e = e_+^{(t)}$, and $\ell = t$, to obtain random vector $z_+^{(t)}$of sparsity t, and then with $x = x^{(t)} + e_+^{(t)}$, $e = e_-^{(t)}$, and $\ell = t$, to obtain the random vector $z_-^{(t)}$ of sparsity t. The estimate of the gradient at time t is the sum of these random vectors. That is, for all $t \geq 1$, define $\tilde{g}^{(t)} := \sum_{s \leq t}(z_+^{(s)} + z_-^{(s)})$. By the property (b) of Lemma 159 , $\tilde{g}^{(t)}$ is a valid stochastic subgradient and can be fed into the framework of Lemma 151. Note that for any $t \in [T]$, the sparsity of $\tilde{g}^{(t)}$is $O(t^2)$ and so is the sparsity of $e^{(t)}$ suggested by the stochastic subgradient routine. Thus, the tth step of estimating $z_+^{(t)}$and $z_-^{(t)}$ requires time $O(t^2 \mathrm{EO} \log^2 n)$, implying we can run T steps of the above procedure in $O(T^3 \mathrm{EO} \log^2 n)$ time.

Finally, to argue about the number of iterations required to get ϵ-close, we need to upper bound $\mathbf{E}[\left\|\tilde{g}^{(t)}\right\|_2^2]$ for every t. Since $\mathbf{E}[\tilde{g}^{(t)}] = g^{(t)}$, the true subgradient at $x^{(t)}$ and since $\left\|g^{(t)}\right\|_2^2 = O(1)$ by Lemma 152 , it suffices to upper bound $\mathbf{E}[\left\|\tilde{g}^{(t)} - \mathbf{E}[\tilde{g}^{(t)}]\right\|_2^2]$. But this follows since $\tilde{g}^{(t)}$ is just a sum of independent z-vectors.

$$\mathbf{E}[\left\|\tilde{g}^{(t)}-\mathbf{E}[\tilde{g}^{(t)}]\right\|_2^2] = \sum_{s \leq t} \mathbf{E}[\left\|z_+^{(s)}-\mathbf{E}[z_+^{(s)}]\right\|_2^2]+\sum_{s \leq t} \mathbf{E}[\left\|z_-^{(s)}-\mathbf{E}[z_-^{(s)}]\right\|_2^2] = O\left(\sum_{s \leq t} 1/s\right) = O(\log n)$$

The second-last inequality follows from (c) of Lemma 159. And so, $\mathbf{E}[\left\|\tilde{g}^{(t)}\right\|_2^2] = \mathbf{E}[\left\|\tilde{g}^{(t)} - \mathbf{E}[\tilde{g}^{(t)}]\right\|_2^2] + \left\|g^{(t)}\right\|_2^2 = O(\log n)$. Therefore, we can apply the framework in Lemma 151 with $B = O(\log n)$ implying the total number of steps to get ϵ-approximate is $N = O(n \log^2 n \epsilon^{-2})$. Furthermore, since each batch takes time $O((n + T^3)\mathrm{EO} \log^2 n)$ and there are N/T batches, we get that the total running time is at most

$$O\left(n \mathrm{EO} \log^4 n \epsilon^{-2} \left(\frac{n + T^3}{T}\right)\right) = \tilde{O}(n^{5/3} \epsilon^{-2} \mathrm{EO})$$

if $T = n^{1/3}$. This ends the proof of Theorem 158.

□

27.5 Improvements when Minimizer is Sparse

Here we discuss how to improve our running times when the submodular function f is known to have a sparse solution, that is, the set minimizing $f(S)$ has at most s elements. Throughout this section we suppose we know s.

Theorem 160. *Let f be a submodular function with a s-sparse minimizer. Then if f is integer valued with $|f(S)| \leq M$ for all $S \subseteq [n]$ we can compute the minimizer deterministically in time $O((n + sM^3)\log n \cdot \mathrm{EO})$. Furthermore if f is real-valued with $|f(S)| \leq 1$ for all $S \subseteq [n]$, then there is a randomized algorithm which in time $\tilde{O}((n + sn^{2/3})\mathrm{EO}\epsilon^{-2})$ returns a set S such that $\mathbf{E}[f(S)] \leq \mathrm{opt} + \epsilon$, for any $\epsilon > 0$.*

Therefore, if we know that the sparsity of the optimum solution is, say $\texttt{polylog}(n)$, then there is a near linear time approximate algorithm to get constant additive error.

To obtain this running time we leverage the same data structures for maintaining subgradients presented in Section 27.3 and Section 27.4. Instead we show how to specialize the framework presented in Section 27.1. In particular we simply leverage that rather than minimizing the Lovasz extension over $[0,1]^n$ we can minimize over $S_s \stackrel{\text{def}}{=} \{x \in [0,1]^n \mid \sum_{i\in[n]} x_i \leq s\}$. This preserves the value of the maximum and minimum, but now improves the convergence of projected (stochastic) subgradient descent (because the quantity R becomes s from n). To show this formally we simply need to show that the projection step does not hurt the performance of our algorithm asymptotically.

We break the proof of this into three parts. First, in Lemma 161 we compute how to project onto S_s. Then in Lemma 162 we show how to update our framework. Using these, we prove Theorem 160.

Lemma 161. *For $k \geq 0$ and $y \in \mathbb{R}^n$ let $S = \{x \in [0,1]^n \mid \sum_i x_i \leq k\}$ and*

$$z = \mathrm{argmin}_{x\in S}\frac{1}{2}\left\|x - y\right\|_2^2.$$

Then, we have that for all $i \in [n]$

$$z_i = \mathit{median}(0, y_i - \lambda, 1)$$

where λ is the smallest non-negative number such that $\sum_i z_i \leq k$.

Proof. By the method of Lagrange multiplier, we know that there is some $\lambda \geq 0$ such that

$$z = \mathrm{argmin}_{x\in[0,1]^n}\frac{1}{2}\left\|x - y\right\|_2^2 + \lambda\sum_{i\in[n]} x_i\,.$$

Since each variable in this problem is decoupled with each other, we can solve this problem coordinate-wise and get that for all $i \in [n]$

$$z_i = \mathrm{med}(0, y_i - \lambda, 1).$$

Since $\sum_{i\in[n]} z_i$ decreases as λ increases, we know that λ is the smallest non-negative number such that $\sum_{i\in[n]} z_i \leq k$. □

In particular we provide Lemma 162, an improvement on Lemma 151 in the sparse regime.

Lemma 162. *Suppose that for $N \geq sB^2\epsilon^{-2}$ and any sequence of $x^{(1)}, .., x^{(N)}$ such that $x^{(i+1)} - x^{(i)}$ is $k^{(i)}$-sparse up to modifications that do not affect the additive distance between non-zero coordinates with $K \overset{\text{def}}{=} \sum_{i\in[T]} k^{(i)} = O(Nk)$, we can implement a subgradient oracle for f and $x^{(i)}$, denoted $\tilde{g}(x^{(i)})$, that is k-sparse and obeys $\mathbf{E}\left\|\tilde{g}\right\|_2^2 \leq B^2$. Then in time $O(n(\mathrm{EO} + \log n) + Nk\log n)$ we can compute a set S such that $\mathbf{E}f(S) \leq \mathrm{opt} + \epsilon$ (and if the subgradient oracle is deterministic then the result holds without the expectation).*

Proof. The proof is the same as before, just that the size of R improves to s and we need to deal with this new projection step. However, in the projection step we set all the coordinates that are less than 0 to 0 and then keep subtracting uniformly (stopping whenever a coordinate reaches 0) until the maximum coordinate is ≤ 1. We can do this efficiently by simply maintaining an additive offset and the coordinate values in sorted order. Then we simply need to know the number of coordinates above some threshold and the maximum and the minimum non-zero coordinate to determine what to subtract up to the point we make the minimum non-zero. We can do this in $O(\log n)$ time easily. Now we are not counting the movements that do not set something to 0 so do not change the additive distances between the non-zero coordinate. Consequently, an iteration may only move many coordinates if it sets many things to 0, however that is paid for by the movement that created it, so we only need $Nk\log n$ time in total to do all the updates. □

We now have everything we need to prove Theorem 160

Proof of Theorem 160. (Sketch) The proof is the same as in Section 27.3 and Section 27.4. We just use Lemma 162 instead of the previous framework lemma. To invoke the first data structure we just do the update in batches. The second data structure was already written for this setting. □

28 Lower Bound

It is well known that $\Omega(n)$ evaluation oracle calls are needed to minimize a submodular function. On the other hand, the best way we know of for certifying minimality takes $\Theta(n)$ subgradient oracle calls (or equivalently, vertices of the base polyhedron). A natural question is whether $\Theta(n)$ subgradient oracle calls are in fact needed to minimize a submodular function. In this section we answer this in the affirmative. Since each gradient oracle needs n evaluation oracle calls, this gives an $\Omega(n^2)$ lower bound on the number of evaluations required for algorithms which only access the function via graident oracles. As mentioned in the introduction, these include the Fujishige-Wolfe heuristic [260, 84], various version of conditional gradient or Franke Wolfe [82, 100] , and the new cutting plane methods [180]. Note that there are known lower bounds for subgradient descent that have a somewhat submodular structure [211] and this suggests that such a lower bound should be possible, however we are unaware of a previous information theoretic lower bound such as we provide.

To prove our lower bound, we describe a distribution over a collection of hard functions and show that any algorithm must make $\Omega(n)$ subgradient calls in expectation[17] and by Yao's

[17]One can also prove a high probability version of the same result but for simplicity we don't do it.

minimax principle this will give an $\Omega(n)$ lower bound on the expected query complexity of any randomized SFM algorithm. The distribution is the following. Choose R to be a random set with each element of the universe selected independently with probability $1/2$. Given R, define the function

$$f_R(S) = \begin{cases} -1 & \text{if } S = R \\ 0 & \text{if } S \not\subseteq R \text{ or } R \not\subseteq S \\ 1 & \text{otherwise.} \end{cases}$$

Clearly the minimizer of f_R is the set R. Any SFM algorithm is equivalent to an algorithm for recognizing the set R via subgradient queries to f_R. A subgradient g of f_R at any point x corresponds to a permutation P of $\{1, 2, \ldots, n\}$ (the sorted order of x). Recall the notation $P[i] := \{P_1, P_2, \ldots, P_i\}$. The following claim describes the structure of subgradients.

Lemma 163. *Let i be the smallest index such that $P[i]$ **is not** a subset of R and j be the smallest index such that $P[j]$ **is** a superset of R. Then $g(i) = 1$, $g(j) = -1$, and $g(k) = 0$ for all $k \in [n] - i, j$.*

Proof. To see $g(i) = 1$, note that $A := P[i-1]$ is a subset of R. Two cases arise: either $A = R$ in which case $P[i]$ is a strict superset of R and so $f_R(A) = -1$ and $f_R(P[i]) = 0$ implying $g(i) = 1$; or A is a strict subset of R in which case $P[i]$ is neither a subset or a superset, implying $f_R(A) = 0$ and $f_R(P[i]) = 1$. Similarly, to see $g[j] = -1$, note that $B := P[j-1]$ is not a superset of R. Two cases arise: either B is a strict subset of R in which case $P[j] = R$ and we have $f_R(B) = 0$ and $f_R(P[j]) = -1$; or B is neither a subset nor a superset in which case $P[j]$ is a strict superset of R and we have $f_R(B) = 1$ and $f_R(P[j]) = 0$.

For any other k, we have either both $P[k]$ and $P[k-1]$ are strict subsets of R (if $k < \min(i,j)$), or both $P[k]$ and $P[k-1]$ are strict supersets of R (if $k > \max(i,j)$) , or both are neither superset nor subset. In all three cases, $g(k) = 0$. □

Intuitively, any gradient call gives the following information regarding R: we know elements in $P[i-1]$ lie in R, P_i doesn't lie in R, P_j lies in R, and all P_k for $k > j$ do not lie in R. Thus we get $i + n - j + 1$ "bits" of information. If R is random, then the expected value of this can be shown to be $O(1)$, and so $\Omega(n)$ queries are required. We make the above intuitive argument formal below.

Suppose at some point of time, the algorithm knows a set $A \subseteq R$ and a set $B \cap R = \varnothing$. The following lemma shows that one may assume wlog that subsequent subgradient calls are at points x whose corresponding permutation P contains the elements of A as a "prefix" and elements of B as a "suffix".

Lemma 164. *Suppose we know $A \subseteq R$ and $B \cap R = \varnothing$. Let g be a subgradient and g' be obtained from g by moving A and B to the beginning and end of the permutation respectively. Then one can compute g from g' without making any more oracle calls.*

Proof. Easy by case analysis and Lemma 163. Let P be the permutation corresponding to g. We show that given g' and P, we can evaluate g. Let us say we are interested in evaluating g_{P_k} and say $P_k = a$. Lemma 163 states that this is 1 iff $P[k-1] \subseteq R$ and $P[k]$ isn't. Now, if $P[k-1] \cap B \neq \varnothing$, then we know g_{P_k} =0. Otherwise, g_{P_k} =1 iff $P[k-1] - B \cup A \subseteq R$ and $P[k] - B \cup A$ is not, since $A \subseteq R$. Therefore, g_{P_k} =1 iff $g'_a = 1$ and $P[k-1] \cap B = \varnothing$. Whether $g_{P_k} = -1$or not can be done analogously. □

For an algorithm, let $h(k)$ be the expected number of subgradient calls required to minimize f_R when the universe if of size k (note R is chosen randomly by picking each element with probability 1/2). For convenience we also define $h(k) = 0$ for $k \leq 0$.

Lemma 165. *For $k \geq 1$, $h(k) \geq 1 + \mathbf{E}_{X,Y}[h(k - X - Y)]$, where X, Y are independent geometric random variables, i.e. $Pr[X = i] = 1/2^i$ for $i \geq 1$.*

Proof. By our observation above, a subgradient of f reveals the identities of $\min\{X + Y, k\}$ elements, where $X - 1 = i - 1$ and $Y - 1 = n - j$ (i, j as defined in Lemma 163) are the lengths of the streaks of 0's at the beginning and end of the subgradient.

Note that X simply follow a geometric distribution because $Pr[P[i-1] \subseteq R, P_i \notin R] = 1/2^i$. Similarly, Y also follow the same geometric distribution. In the case of $X + Y > k$, we have R as a prefix of the permutation.

Finally, as a subgradient call reveals no information about the intermediate elements in the permutation, by Lemma 164 we are then effectively left with the same problem of size $k - X - Y$. More formally, this is because the value of the subgradient queried is independent of the identities of the elements $P_{i+1}, \ldots, P_{j-1}$. □

Theorem 166. *$h(n) \geq n/4$, i.e. any algorithm for SFM requires at least $\Omega(n)$ subgradient calls.*

Proof. We show by induction that $h(k) \geq k/4$. By Lemma 165 and the induction hypothesis,

$$\begin{aligned} h(k) &\geq 1 + \mathbf{E}_{X,Y}[h(k - X - Y)] \\ &\geq 1 + \mathbf{E}_{X,Y}[(k - X - Y)/4] \\ &= 1 + k/4 - \mathbf{E}[X]/4 - \mathbf{E}[Y]/4 \\ &= k/4 \end{aligned}$$

as desired. □

Readers may have noticed that the proofs of the preceding two lemmas essentially imply that $h(k)$ is roughly the expected number of geometric random variables needed to sum up to k. One can use this property together with some concentration inequality for geometric random variables to establish a high probability version of our lower bound.

29 Miscellaneous

29.1 Reduction from Multiplicative to Additive Approximation

Here we show how to obtain a multiplicative approximation for SFM from our $\tilde{O}(n^{5/3} \cdot \mathrm{EO}/\epsilon^2)$ additive-approximate SFM algorithm. Because the minimizer of f is scale- and additive-invariant, it is necessary to make certain regularity assumptions on f to get a nontrivial result. This is akin to submodular function maximization where constant factor approximation is possible only if f is nonnegative everywhere [32, 75]. For SFM, by considering $f - \mathrm{opt}$ we see that finding a multiplicative-approximate solution and an exact solution are equivalent for general f. (Indeed most submodular optimization problems permit multiplicative approximation only in terms of the range of values.)

Similar to submodular maximization, we assume f to be *nonpositive*. Then $f' = f/\mathrm{opt}$ has range $[-1, 0]$ and has minimum value -1 so our additive-approximate algorithm immediately yields multiplicative approximation. This requires knowing opt (or some constant factor approximation of). Alternately we can "binary search" to get factor-2 close to opt by trying different powers of 2. This would lead to a blowup of $O(\log \mathrm{opt})$ in the running time.

29.2 Approximate SFM via Fujishige-Wolfe

Here we show how Frank-Wolfe and Wolfe can give ϵ-additive approximations for SFM. We know that both algorithms in $O(1/\delta)$ iterations can return a point $x \in B_f$, the base polyhedron associated with f, such that $x^\top x \leq p^\top p + \delta$ for all $p \in B_f$. Here we are using the fact implied by Lemma 152 that the diameter of the base-polytope for functions with bounded range is bounded (note that vertices of the base polytope correspond to gradients of the Lovasz extension.) The robust Fujishige Theorem (Theorem 5, [35]) implies that we can get a set S such that $f(S) \leq \mathrm{opt} + 2\sqrt{n\delta}$. Setting $\delta = \epsilon^2/4n$ gives the additive approximation in $O(n\epsilon^{-2})$ gradient calls.

29.3 Faster Algorithm for Directed Minimum Cut

Here we show how to easily obtain faster approximate submodular minimization algorithms in the case where our function when the funciton is an explicitly given s-t cut function. This provides a short illustration of the reasonable fact that when given more structure, our sumbodular minimization algorithms can be improved.

For the rest of this section, let $G = (V, E, w)$ be a graph with vertices V, directed edges $E \subseteq V \times V$, and edge weights $w \in \mathbb{R}^E_{\geq} 0$. Let $s, t \in V$ be two special vertices, $A \overset{\mathrm{def}}{=} V - \{s, t\}$, and for all $S \subseteq A$ let $f(S)$ be defined as the total weight of the edges in leaving the set $S \cup \{s\}$, i.e. where the tail of edge is in $S \cup \{s\}$ and the head of the edges is in $V - (S \cup \{s\})$. The function f is a well known submodular function and minimizing it corresponds to computing the minimum s-t cut, or correspondingly the maximum s-t flow.

Note that clearly, $f(S) \leq W$ where $W = \sum_{e \in E} w_e$. Furthermore, if we pick S by including each vertex in A randomly to be in S with probability independently $\frac{1}{2}$ then we see that $\mathbf{E} f(S) = \frac{1}{2} W$. Consequently, $\frac{1}{2} W \leq \max_{S \subseteq A} f(s) \leq W$ and if we want to scale f to make it have values in $[-1, 1]$ we need to devide by something that is W up to a factor of 2.

Now, note that we can easily extend this problem to a continuous problem over the reals. Let x^+ denote x if $x \geq 0$ and 0 otherwise. Furthermore, for all $x \in \mathbb{R}^A$ let $y(x) \in \mathbb{R}^V$ be given by $y(x)_i = x_i$ if $i \in A$, $y(x)_s = 0$, $y(x)_t = 1$, and let

$$g(x) \overset{\text{def}}{=} \sum_{(a,b) \in E} w_{ab}(y(x_b) - y(x_a))^+ \,.$$

Clearly, minimizing $g(x)$ over $[0,1]^A$ is equivalent to minimizing $f(S)$. Furthermore the subgradient for g decomposes into subgradients for each edge $(a,b) \in E$ each of which is a vector with 2 non-zero entries and norm at most $O(w_{ab})$. If we picking a random edge with probability proportional to w_{ab} and output its subgradient scaled by W/w_{ab} subgradient this yields a stochastic subgradient oracle $\tilde{g}(x)$ with $\mathbf{E}\left\|\tilde{g}(x)\right\|_2^2 = O(\sum_{(a,b)\in E} \frac{w_{ab}}{W}((W/w_{ab}) \cdot w_{ab})^2) = O(W^2)$. Consequently, by Theorem 150 setting $R^2 = O(|V|)$ we see that we can compute z with $g(z) - \min_x g(x) \leq W\epsilon$ in $O(|v|\epsilon^{-2})$. Thus, if we scaled g to make it $[-1,1]$ valued the time to compute an ϵ-approximate solution would be $O(|V|\epsilon^{-2})$.

This shows that an explicit instance of minimum s-t cut does not highlight the efficacy of the approach in this Part. Instantiating our algorithm naively would give an $\tilde{O}(|E|\cdot|V|^{5/3}\cdot\epsilon^{-2})$ to achieve additive error ϵ. Nevertheless, even for such an instance if instead we were simply given access to the an EO time evaluation oracle for f, and the graph was desne, even in this instance, without knowing the structure aprior we do not know how to improve upon the $O(\text{EO} \cdot |V|^{5/3}\epsilon^{-2})$ time bound achieved in this Part (though no serious attempt was made to do this). In short there may be a gap between explicitly given structured instances of submodular functions and algorithms that work with general evaluation oracles as focused on in this Part.

29.4 Certificates for Approximate SFM

The only certificate we know to prove that the optimum value of SFM is $\geq F$ is to show a certain vector x lies in the base polyhedron. For example, one proof via Edmond's Theorem [68] is by demonstrating $x \in B_f$ whose negative entries sum to $\geq F$. The only way to do this is via Carathedeory's Theorem which requires n vertices of B_f, each of which requires n function evaluations. For approximate SFM, one thought might to be to use approximate Caratheodory's Theorems [19, 194] to describe a nearby point x'. Unfortunately, for ϵ-additive SFM approximation, one needs $x'$and x to be close in ℓ_1-norm and approximate Caratheodory works only for ℓ_2-norm and higher. If one uses the ℓ_2-norm approximation, then unfortunately one doesn't get anything better than quadratic. More precisely, approximate Caratheodory states that one can obtain $||x' - x||_2 \leq \delta$ with support of x' being only $O(1/\delta^2)$-sparse. But to get ℓ_1 approximations, we need to set $\delta = \epsilon\sqrt{n}$ leading to linear sized support for x'. The approximate Caratheodory Theorems are tight [194] for general polytopes. Whether one can get better theorems for the base polyhedron is an open question.

29.5 Pseudocodes for Our Algorithms

We provide guiding pseudocodes for our two algorithms.

Algorithm 1 Near Linear Time Exact SFM Algorithm.

```
Initialization.
    • $x^{(1)} \stackrel{\text{def}}{=} 0^n$
    • Evaluate $g^{(1)}$ is the Lovasz subgradient at $x^{(1)}$. *(Takes $O(n \cdot \text{EO})$ time. Store as (coordinate, value) pair in set $S^{(1)}$. $|S^{(1)}| \le 3M$. )*
    • Store $x^{(1)}$ in a balanced Binary search tree. At each node store the **value** that is the sum of the gradient coordinates corr. to children in the tree. *(Takes $O(n)$ time to build.)*
    • Set $T \stackrel{\text{def}}{=} 20nM^2$. Set $\eta \stackrel{\text{def}}{=} \frac{\sqrt{n}}{18M}$.
For $t = 1, 2, \ldots, T$ :
    • Define $e^{(t)}$ which is non-zero in coordinates corresponding to $S^{(t)}$: *(Takes time $|S^{(t)}| \le 3M$ .)*
        – if $g_i^{(t)} > 0$, then $e_i^{(t)} = \min(x_i^{(t)}, \eta g_i^{(t)})$
        – if $g_i^{(t)} < 0$, then $e_i^{(t)} = \max(x_i^{(t)} - 1, \eta g_i^{(t)})$
    • **Update**$(x^{(t)}, e^{(t)}, S^{(t)})$ to get $(x^{(t+1)}, g^{(t+1)}, S^{(t+1)})$ where $g^{(t+1)}$ is stored as coordinate,value pairs in $S^{(t+1)}$. as described in Lemma 157. *(Update takes time $O(M \log n + M \cdot \text{EO} + M \cdot \text{EO} \log n)$)*
Obtain the $O(n)$ sets given the order of $x_T$, that is, if $P$ is the permutation corresponding to $x_T$, then the sets are $\{P[1], \ldots, P[n]\}$. Return the minimum valued set among them.
```

Algorithm 2 Subquadratic Approximate SFM Algorithm.

```
Initialization
    • Set $N \overset{\text{def}}{=} 10n\log^2 n\epsilon^{-2}$, $T = \lceil n^{1/3} \rceil$
    • Initialize $x$ as the all zeros vector and store it in a BST.
For $i = 1, 2, \ldots N/T$ :
    • $x^{(1)} \overset{\text{def}}{=}$the current $x$.
    • Compute $g^{(1)}$, the gradient to the Lovasz extension given $x^{(1)}$. //This takes $O(n\text{EO})$ time).
    • Sample $z^{(1)}$ by picking $j \in [n]$ with probability proportional to $\left|g_j^{(1)}\right|$ and returning
      $z^{(1)} \overset{\text{def}}{=} \left\|g^{(1)}\right\|_1 sign(g_j^{(1)}) \cdot \mathbf{1}_j$. //This takes $O(n\text{EO})$ time.
    • Set $\tilde{g}^{(1)} \overset{\text{def}}{=} z^{(1)}$.
    • For $t = 1, 2, \ldots, T$ :
        – Define $e^{(t)}$ as in Algorithm 1 using $\tilde{g}^{(t)}$ instead of $g^{(t)}$. //This takes time
          $O(\text{supp}(\tilde{g}^{(t)}))$ which will be $O(t^2)$
        – Obtain $z^{(t)}$ using Sample$(x^{(t)}, e^{(t)}, \ell = t)$ where Sample is the randomized pro-
          cedure describe in Lemma 159. //This takes $O(t^2\text{EO}\log n)$ time.
        – Update $\tilde{g}^{(t+1)} \overset{\text{def}}{=} \sum_{s \le t} z^{(s)}$. //This takes $O(t^2 \log n)$ time to update the relevant
          BSTs.
    • Set current $x$ to $x_T$.
Obtain the $O(n)$ sets given the order of the final $x$, that is, if $P$ is the permutation corre-
sponding to $x$, then the sets are $\{P[1], \ldots, P[n]\}$. Return the minimum valued set among
them.
```

Part VI
Tight Algorithms for Vertex Cover with Hard Capacities on Multigraphs and Hypergraphs

No coauthor for this Part.

30 Introduction

The minimum vertex cover problem is one of the earliest NP-hard problems studied in combinatorial optimization. In its most basic form, given a graph $G = (V, E)$ we are asked to find a subset $U \subseteq V$, called vertex cover, so that every edge $e \in E$ intersects U in at least one of its two endpoints. The objective is to minimize the size of U. Curiously, despite decades of efforts the best known algorithms for this problem are the 2-approximation which can either be done by LP relaxation or a simple greedy procedure. The minimum vertex cover problem lends itself to a natural generalization to f-hyergraphs where an edge $e \in E$ can have as many as f endpoints. It is not a difficult matter to generalize the 2-approximation to f-approximation for this version. The seminal result of Khot showed that these algorithms are in fact optimal assuming the Unique Game Conjecture (UGC) [154].

Chuzhoy and Naor [46] initiated the study of vertex cover with *hard capacity* constraints (VCHC) where we have a capacity of $k_v \geq 0$ for each $v \in V$ and (a copy of) v can cover at most k_v of its incident edges. The objective is still to minimize the size of the vertex cover found. They gave a natural LP relaxation for VCHC from which a 3-approximation is derived via randomized rounding for graphs with *no multiple edges.* Their analysis is based on Chebyshev inequality. Subsequently Gandhi et al. [98] improved this to a tight 2-approximation by using Chernoff in place of Chebyshev with a much more involved analysis. Both of these algorithms fail to work for multigraphs (graphs possibly with multiple edges) or hypergraphs essentially because in such cases the random variables in their analyses become unbounded and standard concentration inequalities do not apply.

Progress had been stagnated until Saha and Khuller gave a $\min\{6f, 65\}$ approximation for VCHC on hypergraphs [228]. Their idea is to apply randomized rounding for random variables at different scales to salvage Chernoff. Partly inspired by their result, Cheung, Goemans and Wong (CGW) surprisingly gave simple deterministic rounding algorithms which achieve significantly better approximation ratios of 2.155 (for graphs) and $2f$ [40]. Their method is to formulate the coverage requirement of randomized rounding, used in all previous works, in terms of another LP and study the property of its extreme point solutions. In other words, their approach is a 2-stage LP rounding procedure which solves the same LP relaxation followed by the "coverage requirement LP".

30.1 Our contribution

We propose a new simple approach to the problem based on iterative rounding without using new LPs. Our algorithm achieves the best possible approximation ratio f and essentially settles its approximability. Our approach is inspired by ideas used in previous works, most notably CGW which considers an extreme point solution to certain covering LPs. In hindsight their method suggested the possibility of a better approximation obtained by iterative rounding, which often exploits the structure of extreme point solutions. We also show that when combined with iterative rounding, CGW approach can be extended to give another f-approximation. Although more contrived, this alternate algorithm may be preferred as it involves iteratively rounding the solution to so-called *covering LPs*, which can be solved faster than general LPs using dedicated algorithms [222].

Authors	approx. ratio (graphs, hypergraphs)	multigraphs and hypergraphs okay?
Chuzhoy, Naor [46]	3, *	no
Gandhi et al. [98]	2, *	no
Saha, Khuller [228]	12, $\min\{6f, 65\}$	yes
Cheung, Goemans, Wong [40]	2.155, $2f$	yes
This work, [146]	2, f	yes

30.2 Other related works

Prior to the work of Chuzhoy and Naor [46] which initiated the study of vertex cover with *hard* capacity constraints, Guha et al. [114] resolved the problem with *soft* capacity constraints (where a vertex can be used an arbitrary number of times) using a clever primal-dual algorithm. Notably, their result holds even for the weighted setting whereas the hard capacity version is as hard as set cover in the weighted case and an approximation ratio of $O(\log n)$ is optimal [46]. Using dependent randomized rounding, another 2-approximation for the soft capacity version was given by Gandhi et al. [99].

31 Preliminaries

Let $G = (V, E)$ be a multigraph. We write $u \in e$ to indicate that u is an endpoint of edge $e \in E$.

The minimum Vertex Cover problem with Hard Capacity constraints (VCHC) is specified by (V, E, k, m), where

- $G = (V, E)$ is the input multigraph,
- For each $v \in V$, m_v denotes the maximum number of copies of v one can select,
- For each $v \in V$, k_v is the number of incident edges (a copy of) v can cover.

A solution to VCHC consists of $(x, y) = (\{x_v\}_{v \in V}, \{y(e, v)\}_{e \in E, v \in e})$. Here x_v is the number of copies of vertex v selected, and the assignment variable $y(e, v) \in \{0, 1\}$ represents whether edge e is covered by v, for each $e \in E$ and $v \in e$. A solution (x, y) is *feasible* for VCHC if

1. For all $v \in V$: $x_v \in \{0, 1, \cdots, m_v\}$,
2. For all $e \in E$: $\sum_{v \in e} y(e, v) = 1$ (i.e. any edge must be covered by one of its endpoints),
3. For all $v \in V$: $|\{e : y(e, v) = 1\}| \leq k_v x_v$ (i.e. the total number of edges assigned to v does not exceed its total capacity).

The objective of VCHC is to find a feasible solution (x, y) for VCHC that minimizes $\sum_{v \in V} x_v$, the size of the vertex cover. As VCHC generalizes the classical minimum vertex cover problem which is already NP-hard, we provide efficient algorithms for finding good approximate solutions. Our approach is based on rounding a fractional solution to the following **LP1** relaxation, which has been used extensively in the literature [46, 98, 228, 40].

$$\begin{aligned}
\min \quad & \sum_{v \in V} x_v & & \\
\text{s.t.} \quad & \sum_{v \in e} y(e, v) = 1 & \forall e \in E & \quad (31.1a)\\
& y(e, v) \leq x_v & \forall e \in E, v \in e & \quad (31.1b)\\
& \sum_{e \in \delta(v)} y(e, v) \leq k_v x_v & \forall v \in V & \quad (31.1c)\\
& x_v \leq m_v & \forall v \in V & \quad (31.1d)\\
& x, y \geq 0 & & \quad (31.1e)
\end{aligned}$$

Here x_u denotes the number of copies of u selected and $y(e, v)$ indicates whether e is covered by v. The first constraint says that each $e \in E$ should be covered by one of its endpoints, the second ensures that e can be covered by a vertex selected, and the third is the capacity constraint.

The following lemma shows that, when constructing a feasible solution to VCHC, we only need the integrality of x, and not of y. This follows easily by the integrality of flows in networks with integer capacities. We refer readers to [46, 228] for a proof.

Lemma 167 (Chuzhoy and Naor [46], generalized to hypergraphs by Saha and Khuller [228]). *If (x, y) is feasible for LP1, and x is integral, there exists an integral y' such that (x, y') is feasible for LP 31.1, and y' can be found efficiently by a maximum flow computation.*

In light of this lemma, it suffices to identify a feasible integral solution x with a good approximation guarantee.

32 f-approximation for VCHC on f-hypergraphs

Let (x^*, y^*) be an optimal extreme point solution to LP1, and $U = \{u \in V : x^*_u \geq 1/f\}$. A natural idea used in all previous works is round up $u \in U$ which involves only a factor f blowup, and select judiciously a subset of $W = \{w \in V : 0 < x^*_w < 1/f\}$.

Covering tight edges Our iterative rounding scheme[18] is based on the observation that a *tight* edge $e \in \delta(u)$ with $y(e,u) = x_u \geq 1/f$ can be rounded up while respecting the capacity constraint. This follows from the capacity constraint used in the LP1 where R.H.S. is $k_u x_u$. Therefore we may effectively remove e from the LP by covering e with u and decreasing k_u by 1, and solve the new smaller LP relaxation. A similar argument was used in the (non-iterative) rounding algorithms in [46, 98].

Nevertheless, one complication arises as any $x^*_u \geq 1/f$ can in principle drop below $1/f$ in later iterations of the algorithm and end up not being selected, i.e. $x_u = 0$ in the final solution. In this case covering e by u is not justified. Here we introduce the constraint $1/f \leq x_u$ to the rescue. It ensures that any u with $x^*_u \geq 1/f$ will stay above $1/f$ ever after.

Fixing $x^*_u = 1/f$ To further simplify the LP, we observe that any $x^*_u = 1/f$ can be readily rounded up and removed from the LP. In terms of cost this is a good idea as the approximation ratio incurred is exactly f, meaning that we are not being lossy. Moreover, it ensures that any edge would always have an endpoint in U (see Lemma 171) which is important when we bound the approximation ratio by exploiting the structure of the extreme point solution in the proof of Lemma 172. The idea of examining an extreme point solution was inspired by [40].

Modified LP relaxation We incorporate these insights into **LP2** below. LP2 resembles the form of LP1 with a few important modifications. The first constraint involves $\bar{y}(e,v)$ which is the coverage of v towards e in the final solution. This is a result of fixing $x^*_u = 1/f$ as discussed above. The third constraint has $k_u - |T_u|$ in place of k_u; here $|T_u|$ is the number of tight edges covered by u so we simply subtract $|T_u|$ from the capacity k_u. The fourth constraint now sums over only non-tight edges as any tight edges have already benn covered by its endpoint in U. Finally a new constraint $x_u \geq 1/f$ is introduced to ensure that $x^*_u \geq 1/f$ cannot drop below $1/f$. The last new constraint $x_w \leq 1/f$ is, strictly speaking, not needed but is included to simplify our exposition.

[18]See e.g. [171] for the background on iterative rounding which was introduced by Jain [138].

$$
\begin{aligned}
\min \quad & \sum_{v \in V\setminus D} x_v \\
\text{s.t.} \quad & \sum_{v \in e\setminus D} y(e,v) = 1 - \sum_{v \in e \cap D} \bar{y}(e,v) && \forall e \in E\setminus T && (32.1a)\\
& y(e,v) \le x_v && \forall e \in E\setminus T, v \in e\setminus D && (32.1b)\\
& \sum_{e \in \delta(u)\setminus T} y(e,u) \le (k_u - |T_u|)x_u && \forall u \in U_> && (32.1c)\\
& \sum_{e \in \delta(w)\setminus T} y(e,w) \le k_w x_w && \forall w \in W && (32.1d)\\
& 1/f \le x_u \le m_u && \forall u \in U_> && (32.1e)\\
& 0 \le x_w \le 1/f && \forall u \in W && (32.1f)\\
& y \ge 0 && && (32.1g)
\end{aligned}
$$

Algorithm We adopt the following notations:

- $U = \{u \in V : x^*_u \ge 1/f\}$, $W = \{w \in V : 0 < x^*_w < 1/f\}$, $Z = \{z \in V : x^*_z = 0\}$.
- Further divide U into $U_> = \{u \in U : x^*_u > 1/f\}$ and $U_= = \{u \in U : x^*_u = 1/f\}$.
- $T = \bigcup_{u \in V} T_u$ is a disjoint union of edges $e \in T_u$ covered by $u \in V$. We call T the set of tight edges.
- $D \subseteq V$ is the set of vertex v whose $\bar{x}_v$ has been determined.

Our algorithm is based on performing iterative rounding on LP2. It incrementally builds up a feasible solution $(\bar{x}, \bar{y})$ to VCHC for which $\bar{x}$ is integral.

Algorithm 3 Iterative Rounding Algorithm for VCHC

```
Solve LP1 for an extreme point solution $(x^*, y^*)$. Initially $T_u = \varnothing$ and $D = \varnothing$; initialize
$U, U_>, U_=, W, Z$ based on $(x^*, y^*)$. **repeat**
    **for** $v \in Z$ **do**
        set $\bar{x}_v = x^*_v = 0$, $\bar{y}(e,v) = y^*(e,v) = 0$ for $e \in \delta(v)$  $D \longleftarrow D \cup \{v\}$
    **end**
    **if** $y^*(e,u) = x^*_u$ *for some* $u \in U$ *and* $e \in \delta(u)\setminus T$ **then**
        set $\bar{y}(e,u) = 1$, $\bar{y}(e,v) = 0$ for $v \in e\setminus\{u\}$  $T_u \longleftarrow T_u \cup \{e\}$
    **end**
    **for** $u \in U_=$ **do**
        set $\bar{x}_u = 1$, $\bar{y}(e,u) = y^*(e,u)$ for $e \in \delta(u)\setminus T$  $D \longleftarrow D \cup \{u\}$
    **end**
    Solve updated LP2 for a new extreme point solution $(x^*, y^*)$. Update $U, U_>, U_=, W, Z$
    based on $(x^*, y^*)$.
**until** $U_= = Z = \varnothing$ *and* $y^*(e,u) < x^*_u \forall u \in U, e \in \delta(u)\setminus T$;
For $v \notin D$, set $\bar{x}_v = \lceil x^*_v \rceil$ and $\bar{y}(e,v) = y^*(e,v)$ for $e \in \delta(v)\setminus T$
```

Analysis The algorithm works mostly by design. First we demonstrate feasibility.

Lemma 168. *Suppose a vertex u satisfies $x^*_u \geq 1/f$ at some time during the execution of the algorithm. We must then have $x^*_u \geq 1/f$ after so long as $u \notin D$. Moreover, in the final solution $\bar{x}_u \geq 1$.*

Proof. As soon as $x^*_u \geq 1/f$, we either have $u \in U_=$ or $u \in U_>$. In the former case, u is immediately inserted into D (line 13) and $\bar{x}_u = 1$. In the latter case the constraint $x_u \geq 1/f$ ensures $x^*_u \geq 1/f$ ever after. Eventually we either have $u \in U_=$ (so $\bar{x}_u = 1$) or $\bar{x}_u = \lceil x^*_u \rceil \geq \lceil 1/f \rceil \geq 1$. $\square$

Lemma 169. *The final solution $(\bar{x}, \bar{y})$ is feasible with $\bar{x}$ integral.*

Proof. The fact that $\bar{x}$ is integral simply follows from the description of the algorithm.

We first argue that all edges are covered. For tight edges $e \in T$, we must have set $\bar{y}(e,u) = 1$ for some $u \in U$ at some point (line 8) so e is covered by u. For non-tight edges $e \notin T$, when the algorithm terminates we have

$$\sum_{v \in e \setminus D} \bar{y}(e,v) = \sum_{v \in e \setminus D} y^*(e,v) = 1 - \sum_{v \in e \cap D} \bar{y}(e,v)$$

so e is indeed covered.

It remains to argue that the capacity constraint is satisfied. Lines 3-6 and 11-14 are clearly okay. For Lines 7-10, by Lemma 168 any $u \in U$ satisfies $\bar{x}_u \geq 1$ so we may simply set $\bar{y}(e,u) = 1$ and subtract 1 from the capacity k_u in the LP. This is exactly why we have $(k_u - |T_u|)x_u$ in the third type of constraints. $\square$

Now we bound the approximation ratio. Our argument consists of two ingredients. The first is to observe that an edge $e \notin T$ always intersects U as only vertices with $x_v \leq 1/f$ is put into D. The second, which is much more crucial, exploits the structure of an extreme point solution to show that there cannot be too many fractional x^*_w left at the end of the while loop.

Lemma 170. *In line 15 of the algorithm, the old (x^*, y^*) (from before this line but restricted to only the variables appearing in updated LP2) is still feasible for updated LP2.*

Proof. Clear by inspection. $\square$

Lemma 171. *When the algorithm terminates, for any $e \notin T$ we have*

$$\sum_{v \in e \setminus D} x^*_v \geq \sum_{v \in e \setminus D} y^*(e,v) \geq |e \setminus D|/f.$$

Proof. Note that vertices are assigned to D in Lines 3-6 and 11-14, where we have $x^*_v \in \{0, 1/f\}$ (note that this x^*_v is the one from that particular iteration of the algorithm and not necessarily the final one). Therefore $\bar{y}(e,v) = y^*(e,v) \leq x^*_v \leq 1/f$, which implies

$$\sum_{v \in e \setminus D} x^*_v \geq \sum_{v \in e \setminus D} y^*(e,v) = 1 - \sum_{v \in e \cap D} \bar{y}(e,v) \geq 1 - |e \cap D|/f \geq |e \setminus D|/f,$$

where the last inequality follows from $|e| \leq f$. $\square$

Now we show that there cannot be too many elements in W by a simple counting argument based on examining the extreme point.

Lemma 172. *When the algorithm terminates,* $|W| \le |U^=|$ *where* $U^= := \{u \in U : x^*_u = m_u\}$.

Proof. As an extreme point solution, (x^*, y^*) is obtained by setting some of the constraints as equalities. We call these constraints, which form an **invertible** matrix, **tight**. The proof is based on examining the structure and number of these tight constraints.

First note that $0 \le x_w \le 1/f$ and $1/f \le x_u$ cannot be tight since $U_= = Z = \varnothing$. Similarly, $y(e,u) \le x_u$ for $u \in U$ cannot be tight since no more edges can be added to T. Furthermore, we disregard any edge $e \subseteq D$ since all of its $y(e,v)$ has been determined.

Observe that by the last lemma each edge $e \notin T$ (with $e \backslash D \neq \varnothing$) must have an $x^*_v \ge y^*(e,v) \ge 1/f$. In other words, $1 + |e \cap W| \le |e \backslash D|$.

Below we count the number of tight constraints of different types. The first, second and third in the table are self-explanatory.

For the fourth one in the table there is a total of $\sum_{e \notin T} |e \cap W| + |W|$ such constraints but we claim that only $\sum_{e \notin T} |e \cap W|$ can be tight. This follows from the fact that for each $w \in W$, setting all of the $1 + |\delta(w) \backslash T|$ corresponding constraints tight would give a singular system. More precisely, this would give $x_w = y(e,w) = 0$ which renders setting the constraint $\sum_{e \in \delta(w) \backslash T} y(e,w) \le k_w x_w$ tight unnecessary. In other words, including all of these constraints would give rise to a singular matrix. Thus the total number is at most $\sum_w |\delta(w) \backslash T| = \sum_{e \notin T} |e \cap W|$.

For the fifth one, by Lemma 171 each edge $e \notin T$ (with $e \backslash D \neq \varnothing$) must satisfy $x^*_v \ge y^*(e,v) \ge 1/f$ for some $v \in e$. Therefore at least one of the $|e \cap U_>|$ constraints $y(e,u) \ge 0$ is not tight.

constraints	#tight ones
$\sum_{v \in e \backslash D} y(e,v) = 1 - \sum_{v \in e \cap D} \bar{y}(e,v)$	$= \|E \backslash T\|$
$\sum_{e \in \delta(u) \backslash T} y(e,u) \le (k_u - \|T_u\|) x_u$	$\le \|U_>\|$
$x_u \le m_u$	$= \|U^=\|$
$y(e,w) \le x_w,\ y(e,w) \ge 0\ \&\ \sum_{e \in \delta(w) \backslash T} y(e,w) \le k_w x_w\ (w \in W)$	$\le \sum_{e \notin T} \|e \cap W\|$
$y(e,u) \ge 0\ (u \in U_>)$	$\le \sum_{e \notin T} \|e \cap U_>\| - 1$

Now the number of tight constraints is at most

$$\begin{aligned} & |E \backslash T| + |U_>| + |U^=| + \textstyle\sum_{e \notin T} (|e \cap W| + |e \cap U_>| - 1) \\ = \quad & |E \backslash T| + |U_>| + |U^=| + \textstyle\sum_{e \notin T} (|e \backslash D| - 1) \\ = \quad & \textstyle\sum_{e \notin T} |e \backslash D| + |U_>| + |U^=|. \end{aligned}$$

On the other hand, the number of variables is $|U_>| + |W| + \sum_{e \notin T} |e \backslash D|$. Since there is an equal number of tight constraints and variables, we have $|W| \le |U^=|$. □

We are ready to derive our main theorem.

Theorem 173. *Our algorithm is a f-approximation.*

Proof. Feasibility follows from Lemmas 169 and 167. We bound the approximation ratio. By Lemma 170 the old (x^*_{old}, y^*_{old}) is still feasible for the updated LP so the objective value of the new (x^*_{new}, y^*_{new}) is no worse:

$$\text{cost}(x^*_{new}, y^*_{new}) = \sum_{v\in V\setminus D_{new}} x^*_{new,v} \leq \sum_{v\in V\setminus D_{new}} x^*_{old,v} = \text{cost}(x^*_{old}, y^*_{old}) - \sum_{v\in U_{old,=}\cup Z} x^*_{old,v}$$

which says that the cost $|U_{old,=}|$ incurred to round up vertex in $U_{old,=}$ can be charged to $\sum_{v\in U_{old,=}\cup Z} x^*_{old,v} = |U_{old,=}|/f$ with a factor f blowup.

For the last (x^*_{last}, y^*_{last}), the cost of rounding up the remaining $\bar{x}_v = \lceil x^*_{last,v}\rceil$ is

$$\begin{aligned} |W| + \sum_{u\in U^=} m_u + \sum_{u\in U\setminus U^=} \lceil x^*_{last,u}\rceil &\leq |U^=| + \sum_{u\in U^=} m_u + \sum_{u\in U\setminus U^=} \lceil x^*_{last,u}\rceil \\ &\leq f\left(\sum_{u\in U^=} m_u + \sum_{u\in U\setminus U^=} x^*_{last,u}\right) \\ &= f\cdot \text{cost}(x^*_{last}, y^*_{last}), \end{aligned}$$

where we used Lemma 172 and $m_u \geq 1, f\geq 2, x^*_{last,u}\geq 1/f$. This proves the theorem. □

32.1 Implementation using only the original LP

It is possible to implement a similar algorithm without appealing to LP2, which essentially introduces the new constraint $x_u \geq 1/f$. Without them previous $u\in U$ may fall into W and u may not be selected in the final solution, in which case setting $\bar{y}(e,u) = 1$ for $y^*(e,u) = x^*_u$ is not justified as u does not have the capacity to cover e.

The key idea is to continuously move from the old optimum to the new one. More concretely, consider moving along the line from (x^*_{old}, y^*_{old}) to (x^*_{new}, y^*_{new}). We stop whenever $x^*_v = 1/f$ at an intermediate point (x^*, y^*), where we perform lines 11-14 by fixing $x_v = 1$ and $y(e,u) = y^*(e,u)$, and solve the updated LP again. If we arrive at (x^*_{new}, y^*_{new}) without stopping, then we perform lines 7-10 by covering any tight edge $y^*(e,u) = x^*_u \geq 1/f$ using u, i.e. removing e from the LP and decreasing k_u by 1; and solve the updated LP again (one can also eliminate Z which, as with the previous approach, is only introduced to simplify the exposition). If none of these operations are possible, then we are at an extreme point where the same proof would show that rounding up the remaining vertices would give a f-approximation.

Readers can easily make the previous proof work for this new implementation. We refrain from giving a proof because in the next section, we would give a faster implementation using this idea on another LP with a full proof.

33 Faster Implementation via Solving Covering LPs

The approach in the previous section essentially redistributes the coverage relation $y(e,v)$ and cost x_v from one iteration to the next. Upon a closer examination, readers may have noticed that the constraint $y(e,w)\leq x_w$ for $w\in W$ plays a relatively smaller role. One may wonder if there could a faster implementation without solving LP2 again. We answer this in the affirmative in this section. This alternate approach performs iterative rounding on LP3, which is a packing LP and can be solved faster using various specialized algorithms. The formulation of LP3 has been heavily inspired by a similar construction in [40].

33.1 LP relaxation and algorithm

In this section we use the same notations $U, U_>, U_=, W, Z$ as before. Our covering **LP3** is as follows:

$$\begin{aligned}
\min \quad & \sum_{w\in W} x_w + \sum_{u\in U_>} x_u \\
\text{s.t.} \quad & \sum_{w\in W} M(u,w)x_w + (k_u - |T_u|)x_u \geq \sum_{w\in W} M(u,w)x^*_w + \sum_{e\in\delta(u)\setminus T} y^*(e,u) \quad && \forall u \in U_> && (33.1a)\\
& 0 \leq x_w \leq 1/f && \forall w \in W && (33.1b)\\
& 1/f \leq x_u \leq m_u && \forall u \in U_> && (33.1c)
\end{aligned}$$

where

$$M(u,w) = \sum_{e\in E_u\cap\delta(w)} \frac{y^*(e,w)}{x^*_w}.$$

Redistributing coverage At the high level, LP3 is based on doing book-keeping of how edges $\delta(u)$ incident to $u \in U$ are covered. It attempts to redistribute the coverage by maintaining the proportion of capacity used for different $y(e,w)$'s (for a given w). While an edge e can have more than one endpoint in U, we simply arbitrarily assign e to one such u so that E_u is the collection of edges assigned to u.

For $w \in W$ we distribute its capacity in the same proportion as before (hence step 4(a) and the definition of $M(u,w)$; see also Lemma 176). The coverage $y(e,u)$ for $e \in E_u$ would then be the remaining amount not yet covered. However this amount can become negative if the other endpoints of e in W contribute more than before towards covering e. Similarly, if they contribute much less now $y(e,u)$ can become larger than x_u.

The key idea is to slowly move from an old solution (x^*, y^*) to the new (x^*_{new}, y^*_{new}), and stop whenever any of these desired conditions is about to fail (step 5). At this point we may simplify the solution by appropriately modifying x, y.

The algorithm is as follows. Readers should recognize the resemblance to the previous one, except with the notable difference that we need to ensure $y(e,u) \geq 0$ in addition to $x_u \geq 1/f$ and $y(e,u) \leq x_u$ (pardon us for not using the algorithm environment here as it would clutter the longer description).

Iterative Rounding Algorithm using Covering LPs

1. Solve **LP1** for an extreme point solution (x^*, y^*). Initially $T_u = \varnothing$ and $D = \varnothing$; initialize $U, U_>, U_=, W, Z$ based on (x^*, y^*).

2. For $v \in Z$, set $\bar{x}_v = x^*_v = 0$, $\bar{y}(e,v) = y^*(e,v) = 0$ for $e \in \delta(v)$ and $D \longleftarrow D \cup \{v\}$.

3. Partition edges into a disjoint union $E \backslash T = \bigcup_{u \in U} E_u$ by assigning $e \notin T$ to an arbitrary E_u where $u \in e \cap U$.

4. Solve updated **LP3** for an extreme point solution x^*_{new} and set

   (a) $y^*_{new}(e,w) = \frac{y^*(e,w)}{x^*_w} x^*_{new,w}$ for $w \in W, e \in \delta(w) \backslash T$

   (b) $y^*_{new}(e,u) = y^*(e,u) + \sum_{w \in e \cap W} (y^*(e,w) - y^*_{new}(e,w))$ for $u \in U, e \in E_u$

   (c) $y^*_{new}(e,u) = y^*(e,u)$ for $u \in U, e \notin E_u \cup T$

5. Let $x^t = (1-t)x^* + t x^*_{new}$ and $y^t = (1-t)y^* + t y^*_{new}$, where t **continuously** increases from 0 to 1. **Stop** whenever:

   (a) $y^t(e,u) = x^t_u$ for $u \in U$ and $e \in \delta(u) \backslash T$. Set $\bar{y}(e,u) = 1$, $\bar{y}(e,v) = 0$ for $v \in e \backslash \{u\}$ and $T_u \longleftarrow T_u \cup \{e\}$.

   (b) $x^t_v = 1/f$ for some v. Set $\bar{x}_u = 1$, $\bar{y}(e,u) = y^*(e,u)$ for $e \in \delta(u) \backslash T$ and $D \longleftarrow D \cup \{u\}$.

   (c) $y^t(e,u) = 0$ for $u \in U$ and $e \in \delta(u) \backslash T$. Set $\bar{y}(e,u) = 0$ and $e \longleftarrow e \backslash \{u\}$.

6. Update $U, U_>, U_=, W, Z$ based on $(x^*, y^*) \longleftarrow (x^t, y^t)$.

7. **Repeat** steps 2-7 until no more updates are possible (reaching x^*_{new} without stopping).

8. For $v \notin D$, set $\bar{x}_v = \lceil x^*_v \rceil$ and $\bar{y}(e,v) = y^*(e,v)$ for $e \in \delta(v) \backslash T$.

First we show that our algorithm is well-defined. Interestingly, this is analogous to Lemma 171 which is used to establish approximation guarantee instead.

Lemma 174. *We have* $\sum_{v \in e \backslash D} x^*_v \geq \sum_{v \in e \backslash D} y^*(e,v) \geq |e \backslash D|/f$. *In particular, there is some* $u \in e \cap U$ *so step 3 of the algorithm is well-defined.*

Proof. Note that vertices are assigned to D in steps 2 and 5(b), where we have $x^*_v \leq 1/f$. Therefore $\bar{y}(e,v) = y^*(e,v) \leq x^*_v \leq 1/f$, which implies

$$\sum_{v \in e \backslash D} x^*_v \geq \sum_{v \in e \backslash D} y^*(e,v) = 1 - \sum_{v \in e \cap D} \bar{y}(e,v) \geq 1 - |e \cap D|/f \geq |e \backslash D|/f,$$

where the last inequality follows from $|e| \leq f$. □

Now we argue for feasibility. As before, any $x^*_u \geq 1/f$ would stay above $1/f$ which is necessary to cover tight edges in step 5(a).

Lemma 175. *Suppose u satisfies $x^*_u \geq 1/f$ at some time during the execution of the algorithm. We must then have $x^*_u \geq 1/f$ after so long as $u \notin D$. Moreover, in the final solution $\bar{x}_u \geq 1$.*

Proof. Any such x^*_u cannot drop below $1/f$ thanks to step 5(b) of the algorithm, and would be rounded up eventually. □

We justify the capacity constraint in the next lemma, which goes hand-in-hand with the way **LP3** is formulated.

Lemma 176. *$(x^*, y^*), (x^*_{new}, y^*_{new}), (x^t, y^t)$ satisfy the capacity constraint $\sum_{e\in\delta(w)\setminus T} y(e,w) \leq k_w x_w$ for $w \in W$ and $\sum_{e\in\delta(u)\setminus T} y(e,u) \leq (k_u - |T_u|)x_u$ for $u \in U$.*

Proof. We proceed by induction.

For $w \in W$, we have $\sum_{e\in\delta(w)\setminus T} y^*_{new}(e,w) = \sum_{e\in\delta(u)\setminus T} \frac{y^*(e,w)}{x^*_w} x^*_{new,w} \leq k_w x^*_{new,w}$ since by the induction hypothesis we have $\sum_{e\in\delta(w)\setminus T} y^*_{new}(e,w) \leq k_w x^*_w$. Now (x^t, y^t) (and therefore the new (x^*, y^*)) also satisfy the capacity constraint since it is a convex combination of $(x^*, y^*), (x^*_{new}, y^*_{new})$.

For $u \in U$, we have

$$\begin{aligned}\sum_{e\in\delta(u)\setminus T} y^*_{new}(e,u) &= \sum_{e\in E_u}\left[y^*(e,u) + \sum_{w\in e\cap W}(y^*(e,w) - y^*_{new}(e,w))\right] + \sum_{e\notin E_u\cup T} y^*(e,u)\\ &= \sum_{e\in\delta(u)\setminus T} y^*(e,u) + \sum_{w\in e\cap W}\left(y^*(e,w) - \frac{y^*(e,w)}{x^*_w}x^*_{new,w}\right)\\ &\leq (k_u - |T_u|)x^*_{new,u}\end{aligned}$$

where the last inequality follows from the first constraint of **LP3**. Hence (x^*_{new}, y^*_{new}) satisfies the capacity constraint for $u \in U$, and so is (x^t, y^t) since it is a convex combination of $(x^*, y^*), (x^*_{new}, y^*_{new})$.

Finally, for the new (x^*, y^*) note that T_u may have changed in size but this is okay by design since $(k_u - |T_u|)x^t_u$ changes exactly by (#new elements in T_u) $\cdot\, x^t_u$. □

Lemma 177. *The final solution $(\bar{x}, \bar{y})$ is feasible with $\bar{x}$ integral.*

Proof. The fact that $\bar{x}$ is integral simply follows from the description of the algorithm. The capacity constraints are satisfied by Lemma 176 and the fact that any $u \in U$ would be rounded up to $\bar{x}_u \geq 1$ eventually (Lemma 175) and therefore can pay for covering the tight edges T_u.

The other constraints $0 \leq x_v \leq m_v, y \geq 0$ are guaranteed by step 5 of the algorithm where we stop at (x^t, y^t) before they can be violated. □

Finally we prove the approximation guarantee, which again mostly follows from examining the structure of an extreme point solution.

Lemma 178. *When the algorithm terminates, $|W| \leq |U^=|$ where $U^= := \{u \in U : x^*_u = m_u\}$.*

Proof. This is very similar to the previous proof. Note that the number of variables is $|W| + |U|$. On the other hand, $0 \leq x_w \leq 1/f$ and $x_u \geq 1/f$ cannot be tight. The number of tight constrains $x_u \leq m_u$ is $|U^=|$ while there can be at most $|U|$ tight first covering constraints. So $|W| \leq |U^=|$. □

Theorem 179. *Our algorithm is a f-approximation.*

Proof. This is very similar to Theorem 173. The cost of rounding up intermediate $x_v^t = 1/f$ incurs a factor of f. Since (x^*, y^*) is feasible by design and Lemma 176, it remains to show that the last step incurs a factor of at most f. The same inequality in the proof of Theorem 173 works. □

34 Even Faster Implementation for Graphs ($f = 2$)

For the case of graphs $f = 2$ we may in fact drop the variables x_u from **LP3** altogether. This is exactly the same covering LP used in CGW's two-stage rounding algorithm. In this section we show that by performing iterative rounding while moving from old to new optima, their algorithm actually achieves a 2-approximation for graphs.

In our opinion, this approach nicely explains why the randomized rounding scheme in [98] would work. The algorithm of [98] gives a 2-approximation for VCHC on simple graphs by essentially rounding **LP2** via a simple randomized rounding and performing some patching work after. However, a drawback of their work is that the analysis involves many pages of calculations and gives little insights into why the algorithm would give a 2-approximation.

In a way, their argument is a probabilistic proof that a 2-approximate solution exists for **LP4**, and our one-page analysis can be viewed as a simple deterministic proof of that claim.

LP relaxation Instead of **LP3**, we use the following simpler **LP4**:

$$\begin{aligned}
\min \quad & \sum_{w \in W} x_w & & \\
\text{s.t.} \quad & \sum_{w \in W} M(u,w) x_w \geq \sum_{w \in W} M(u,w) x_w^* & \forall u \in U \qquad & (34.1a) \\
& 0 \leq x_w \leq 1 & \forall w \in W \qquad & (34.1b)
\end{aligned}$$

where

$$M(u,w) = \sum_{e = uw \in E \setminus T} \frac{y^*(e,w)}{x_w^*}.$$

We first give an overview of the algorithm. The algorithm is mostly a specialization of the previous one to $f = 2$ but the constraint $x_u \geq 1/2$ is now redundant as an edge cannot have both endpoints in W for graphs. One crucial difference in the algorithm is that we are rounding up $y^*(e,u)$ whenever $y^*(e,u) \cdot (\lceil x_u^* \rceil / x_u^*) \geq 1$ (rather than just when $x_u^* = y^*(e,u)$). This is still okay as the final capacity is $k_u \lceil x_u^* \rceil$ so we can blow up the coverage by a factor of $\lceil x_u^* \rceil / x_u^*$.

As for the analysis, to bound the cost of the extreme point solution it is not enough to use only a cardinality inequality like $|W| \le |U^=|$. Instead we need a finer *matching* structure between U and W that was also used by CGW.

Our algorithm is as follows. Unlike before, *we never redefine or update U and W.*

Iterative Rounding Algorithm using Simpler Covering LPs for Graphs

1. Solve **LP1** for x^*, y^*. Let $T = \varnothing$.
2. For any $u \in U$ and $e \in \delta(u)\backslash T$ satisfying $y^*(e,u) \cdot (\lceil x_u^* \rceil / x_u^*) \ge 1$, set $T \longleftarrow T \cup \{e\}$.
3. Solve updated **LP4** for an extreme point optimum x_{new}^*.
4. Consider $x^t = (1-t)x^* + tx_{new}^*$, where t **continuously** increases from 0 to 1. Let
$$y^t(e,w) = \frac{y^*(e,w)}{x_w^*} x_w^t,\ y^t(e,u) = 1 - y^t(e,w) \quad \forall e = uw \in (U \times W)\backslash T$$
5. Stop whenever $y^t(e,u) \cdot (\lceil x_u^* \rceil / x_u^*) = 1$ for some $u \in U$ and $e \in (U \times W)\backslash T$, set $T \longleftarrow T \cup \{e\}$.
6. Update $M(u,w)$ with $(x^*, y^*) \longleftarrow (x^t, y^t)$. Repeat steps 3-6.
7. If $t = 1$ (i.e. no pair (e,u) satisfies the condition in step 5 for any intermediate t), output $\bar{x}_v = \lceil x_v^* \rceil$ with a corresponding y defined by
$$\bar{y}(e,u) = y^*(e,u),\ \bar{y}(e,v) = y^*(e,v) \quad \forall e \notin T$$
$$\bar{y}(e,u) = 1, \bar{y}(e,v) = 0 \quad \forall e = uw \in T, u \in U, w \in W$$

One difference is that we are rounding up $y^*(e,u)$ whenever $y^*(e,u) \cdot (\lceil x_u^* \rceil / x_u^*) \ge 1$ (rather than just when $x_u^* = y^*(e,u)$). This is still okay as the final capacity is $k_u \lceil x_u^* \rceil$ so we can blow up the coverage by a factor of $\lceil x_u^* \rceil / x_u^*$. Curiously this modification is needed to handle the case $1 < x_u^* \le 2$ when bounding the approximation ratio.

The feasibility of the algorithm is largely the same as before so we skip the proof (in fact, it is more like an easier special case). To bound the cost of the extreme point solution, it is not enough to use only a cardinality inequality like $|W| \le |U^=|$ (one fact, now one does not have this but $|W| \le |U|$). Instead we need a finer *matching* structure between U and W as given in Lemma 181.

Lemma 180. *$(\bar{x}, \bar{y})$ is a feasible solution to VCHC.*

Proof. Similar to Lemmas 176 and 177. □

Lemma 181 (Similar to Theorem 3.2 of [40])**.** *For any extreme point solution x^* to LP4, let $W_f = \{w \in W : 0 < x_w^* < 1\}$. Then there exists a matching between W_f and U that fully matches W_f.*

Proof. Fractional x_w^* can only arise from setting $|W_f|$ covering constraints $\sum_{w \in W} M(u,w)x_w \geq \sum_{w \in W} M(u,w)x_w^*$ tight. Consider the system of equations $Ax = b$ obtained from setting these $|W_f|$ constraints tight. Note that A is $|W_f| \times |W_f|$ with the rows and columns indexed by $|U|$ and $|W_f|$ respectively. A is invertible so

$$\det A = \sum_{\text{permutation } \pi} \left(\pm \prod_i a_{i,\pi(i)} \right) \neq 0$$

which implies that there is a permutation with all $a_{i,\pi(i)} \neq 0$. Such a permutation gives our desired matching since nonzero entries of A corresponds to $M(u,w) \neq 0$ in which case uw is an edge. □

Theorem 182. *$(\bar{x}, \bar{y})$ is a 2-approximation.*

Proof. The cost of U and $x_v = 0$ clearly incurs a factor 2 blowup only. We need to account for the cost of selecting W_f. Now that we don't have $|W_f| \leq |U^=|$, we cannot naively charge $|W_f|$ to $U^=$. However, we have a matching between W_f and U and it suffices to show that for any edge $e = uw$ in the matching,

$$\lceil x_u^* \rceil + 1 = \lceil x_u^* \rceil + \lceil x_w^* \rceil \leq 2(x_u^* + x_w^*).$$

Note that we have $1 = y^*(e,u) + y^*(e,w) \leq x_u^* + x_w^*$ and $y^*(e,u) \cdot (\lceil x_u^* \rceil / x_u^*) \leq 1$ (otherwise step 5 would have applied). We have three cases:

- $\lceil x_u^* \rceil = 1$. Then we are done.
- $\lceil x_u^* \rceil \geq 3$. Then $\lceil x_u^* \rceil + 1 \leq 2x_u^*$.
- $x_u^* + x_w^* \geq 1.5$ and $x_u^* \leq 2$. Then $\lceil x_u^* \rceil + 1 \leq 3 \leq 2(x_u^* + x_w^*)$.
- $x_u^* + x_w^* \leq 1.5$ and $1 < x_u^* \leq 2$, which cannot happen since $1 = y^*(e,u) + y^*(e,w) \leq x_u^*/2 + x_w^* = x_u^* + x_w^* - x_u^*/2 < 1.5 - 1/2 = 1$. Contradiction.

□

In hindsight, CGW came close but failed to obtain a 2-approximation because one does not have $x_{new,u}^* + x_{new,w}^* \geq 1$.

Open Problem

In this Part we have settled the approximability of VCHC. While it is UGC-hard to do better [154], the current best NP-hardness inapproximability stands at $f - 1$ [59] and 1.36 for graphs [60]. Is there any hope of proving that it is NP-hard to beat f? In principle it should be easier than doing it for vertex cover since VCHC is more general.

Part VII
Network Design for s-t Effective Resistance

This Part is based on joint works with Pak Hay Chan, Lap Chi Lau, Aaron Schild and Hong Zhou. My contribution includes formulating the problem and being the main architect behind an earlier $O(log^2 n)$ approximation cycle-pushing algorithm which was later improved to a constant factor approximation by my coauthors Lap Chi Lau and Hong Zhou.

35 Introduction

Network design problems are generally about finding a minimum cost subgraph that satisfies certain "connectivity" requirements. The most well studied problem is the survivable network design problem [103, 2, 105, 139, 96], where the requirement is to have a specified number $r_{u,v}$ of edge-disjoint paths between every pair of vertices u, v. Other combinatorial requirements are also well studied in the literature, including vertex connectivity [162, 74, 36, 45, 169, 38] and shortest path distances [61, 58]. Some spectral requirements are also studied, including spectral expansion [160, 5], total effective resistances [102, 214], and mixing time [25], but in general much less is known about these problems. See Section 35.1 for more discussions of previous work.

In this work, we study a basic problem in designing networks with a spectral requirement – the effective resistance between two vertices.

Definition 183 (The s-t effective resistance network design problem)**.** The input is an undirected graph $G = (V, E)$ where each edge e has a non-negative cost c_e and a non-negative resistance r_e, two specified vertices $s, t \in V$, and a cost budget k. The goal is to find a subgraph H of G that minimizes $\mathrm{Reff}_H(s, t)$ subject to the constraint that the total edge cost of H is at most k, where $\mathrm{Reff}_H(s, t)$ denotes the effective resistance between s and t in the subgraph H with resistances r_e on the edges. See Section 36.2 for the definition of effective resistance and Section 37.1 for a mathematical formulation of the problem.

The s-t effective resistance is an interpolation between s-t shortest path distance and s-t edge connectivity. Let $f \in \mathbb{R}^{|E|}$ be a unit s-t flow in an unweighted graph G and define the ℓ_p-energy of f as $\mathcal{E}_p(f) := (\sum_e |f_e|^p)^{1/p}$. Let $\mathcal{E}_p(s, t) := \min_f \{\mathcal{E}_p(f) \mid f \text{ is a unit } s\text{-}t \text{ flow}\}$ be the minimum ℓ_p-energy of a unit s-t flow that the graph G can support. Thomson's principle (see Section 36.2) states that $\mathrm{Reff}_G(s, t) = \mathcal{E}_2^2(s, t)$, so that a graph of small s-t effective resistance can support a unit s-t flow with small ℓ_2-energy. Note that the shortest path distance between s and t is $\mathcal{E}_1(s, t)$ (as the ℓ_1-energy of a flow is just the average path length and is minimized by a shortest s-t path), and so a graph with small $\mathcal{E}_1(s, t)$ has a short path between s and t. Note also that the edge-connectivity between s and t is equal to the reciprocal of $\mathcal{E}_\infty(s, t)$ (because if there are k edge-disjoint s-t paths, we can set the flow value on each path to be $1/k$), and so a graph with small $\mathcal{E}_\infty(s, t)$ has many edge-disjoint

s-t paths. As ℓ_2 is between ℓ_1 and ℓ_∞, the objective function $\text{Reff}(s,t) = \mathcal{E}_2^2(s,t)$ takes both the s-t shortest path distance and the s-t edge-connectivity into consideration.

A simple property suggests that ℓ_2-energy may be even more desirable than ℓ_1 and ℓ_∞ as a connectivity measure. Conceptually, adding an edge e to G would make s and t more connected. For ℓ_1 and ℓ_∞, however, adding e would not yield a better energy if e does not improve the shortest path and the edge connectivity respectively. In contrast, the ℓ_2-energy would typically improve after adding an edge, and so ℓ_2-energy provides a smoother quantitative measure that better captures our intuition how well s and t are connected in a network.

Also, the effective resistance has a probabilistic interpretation as the expected commute time between vertices in a random walk [37] (see Section 36.2), and is used as a distance function in the study of social networks. Therefore, the effective resistance is a nice and natural alternative connectivity measure in network design.

Thomson's principle also states that the electrical flow between s and t is the unique flow that minimizes the ℓ_2-energy. So, designing a network with small s-t effective resistance has natural applications in designing electrical networks [71, 102, 137]. One natural formulation is to keep at most k wires in the input electrical network to minimize $\text{Reff}(s,t)$, so that the electrical flow between s and t can still be sent with small energy while we switch off many wires in the electrical network.

35.1 Our Results

Unlike the classical problems of shortest path and min-cost flow (corresponding to the ℓ_1 and ℓ_∞ versions of the problem), we prove that the s-t effective resistance network design problem is NP-hard, and is APX-hard assuming the small-set expansion conjecture [224, 225].

Theorem 184. *The s-t effective resistance network design problem is NP-hard, even when every edge has the same cost and the same resistance ($c_e = r_e = 1$ for every edge e).*

Theorem 185. *Assuming the small-set expansion conjecture, it is NP-hard to approximate the s-t effective resistance network design problem within a factor of $2 - \epsilon$ for any $\epsilon > 0$, even when every edge has the same cost.*

We also obtain some approximation algorithms for the problem. We analyze a natural convex programming relaxation for the problem (Section 37.1), and use it to design a constant factor approximation algorithm when every edge has the same cost and the same resistance.

Theorem 186. *There is a convex programming based 8-approximation randomized algorithm for the s-t effective resistance network design problem when $c_e = 1$ and $r_e = 1$ for every edge e.*

We note that the convex program has unbounded integrality gap when the costs could be arbitrary or the resistances could be arbitrary (Section 37.1). When every edge has the same cost and the same resistance, we derive a nice characterization of the optimal solutions to the convex program (Lemma 193), and use it to design a randomized path-rounding algorithm (Section 37.2) based on a flow decomposition of the fractional solution to prove Theorem 186.

There is a simple example showing that the integrality gap is at least two when $c_e = r_e = 1$ for all $e \in E$. When the budget k is much larger than the length of a shortest s-t path, we show how to achieve an approximation ratio close to two with a slightly modified randomized "short" path rounding algorithm (Section 37.5).

Theorem 187. *There is a $(2+O(\epsilon))$-approximation algorithm for the s-t effective resistance network design problem, when $c_e = 1$ and $r_e = 1$ for every edge e and $k \geq 2d_{st}/\epsilon^{10}$ where d_{st} is the length of a shortest s-t path.*

As our problem is related to electrical network design, it is natural to consider the special case when the input graph is a series-parallel graph. In this setting, we can use dynamic programming to design a fully polynomial time approximation scheme for the problem when the ratio between the maximum and minimum resistance is bounded, and an exact algorithm when every edge has the same cost.

Theorem 188. *There is a dynamic programming based $(1+\epsilon)$-approximation algorithm for the s-t effective resistance network design problem when the input graph is a series-parallel graph. The running time of the algorithm is $O(|E|^7U^2/\epsilon^2)$ where $U = \max_e r_e / \min_e r_e$ is the ratio between the maximum and minimum resistance. If we assume further that $c_e = 1$ for all edges e, there is an exact algorithm for the problem with running time $O(|E| \cdot k^2)$.*

We note that the integrality gap examples in Section 37.1 are actually series-parallel graphs, and so the dynamic programming algorithms go beyond the limitation of the natural convex program. We leave it as an open problem whether the general case admits a good approximation algorithm (possibly by combining these techniques).

We also consider a "dual" problem where the effective resistance is a hard constraint and the objective is to minimize the cost. We present similar results in Section 37.6.

35.2 Related Work

In the survivable network design problem, we are given an undirected graph and a connectivity requirement $r_{u,v}$ for every pair of vertices, and the goal is to find a minimum cost subgraph such that there are at least $r_{u,v}$ edge-disjoint paths for all u, v. This problem is very well-studied and captures many interesting special cases [103, 2, 105, 96]. The best approximation algorithm for this problem is due to Jain [139], who introduced the technique of iterative rounding to design a 2-approximation algorithm. His result has been extended in different directions, including element-connectivity [79, 39], directed graphs [95, 96], and with degree constraints [170, 73, 94, 172].

Other combinatorial connectivity requirements were also considered. A natural variation is to require that there are $r_{u,v}$ internally vertex disjoint paths for every pair of vertices u, v. This problem is much harder to approximate [162, 169], but there are good approximation algorithms for global connectivity [74, 38] and when the maximum connectivity requirement is small [36, 45]. Another natural problem is to require that there is a path of length $l_{u,v}$ between every pair of vertices u, v. This problem is also hard to approximate in general but there are better approximation algorithms when every edge has the same cost and the same length [61].

Spectral connectivity requirements were also studied, including spectral gap [101, 160] (closely related to graph expansion), total effective resistances [102], and mixing time [25]. Some of the earlier works only proposed convex programming relaxations and heuristic algorithms. Approximation guarantees are only obtained in two recent papers for the more general experimental design problem. When every edge has the same cost, there is a $(1+\epsilon)$-approximation algorithm for minimizing the total effective resistance when the budget is at least $\Omega(|V|/\epsilon)$ [214], and there is a $(1+\epsilon)$-approximation algorithm for maximizing the spectral gap when the budget is at least $\Omega(|V|/\epsilon^2)$ [5]. For our problem, the interesting regime is when k is much smaller than $|V|$, where the techniques in [5, 214] do not apply. We have developed a set of new techniques for analyzing and rounding the solutions to the convex program that will hopefully find applications for solving related problems.

Traditionally, the effective resistance has many useful probabilistic interpretations, such as the commute time [37], the cover time [190], and the probability of an edge in a random spanning tree [155]. See Section 36 for more details. Recently, effective resistance has found surprising applications in solving problems about graph connectivity, including constructing spectral sparsifiers [241] (by using the effective resistance of an edge as the sampling probability), computing maximum flow [43], finding thin trees [6], and generating random spanning trees [188, 229].

35.3 Techniques

The main technical contribution is in designing rounding techniques for a convex programming relaxation. There is a natural convex programming relaxation for our problem, by using the conductance of the edges as variables, and writing the s-t effective resistance as the objective function and noting that it is convex with respect to the variables (Section 37.1).

When every edge has the same cost and the same resistance, we show that the optimal solution of this convex program has some nice properties[19]. Given an optimal fractional solution x^* and the unit s-t electrical flow f^* supported in x^*, we derive from the KKT optimality conditions that there is a flow-conductance ratio $\alpha > 0$ such that $f^*_e = \alpha x^*_e$ for every fractional edge e with $0 < x^*_e < 1$ and $f^*_e \geq \alpha$ for every integral edge e with $x^*_e = 1$. The flow-conductance ratio α is crucial in the rounding algorithm and the analysis.

The rounding techniques in the two recent papers on experimental design [5, 214] considered each edge/vector as a unit. In [5], a potential function as in spectral sparsification is used to guide a local search algorithm to swap two edges/vectors at a time to improve the current solution. In [214], a probability distribution on the edges/vectors is carefully designed for an independent randomized rounding. These techniques are only known to work in the case when the solutions form a spanning set so that the "contribution" of each individual edge/vector is well-defined. This is basically the reason why the results in [5, 214] only apply when the budget k is at least $\Omega(n)$.

Our approach is based on a randomized rounding procedure on s-t paths. Given x^*, we compute the unit s-t electrical f^* supported in x^*, and then decompose f^* as a convex combination of s-t paths. The rounding algorithm has $T = 1/\alpha$ iterations, where we pick a

[19]We can also show that the fractional edges in the optimal solution form a forest, but this is not included in the Part as we have not used this property in the rounding algorithm.

random path P_i from the convex combination in each iteration, and return $H := \cup_{i=1}^{T} P_i$ as our solution. One difference from the previous techniques is that each unit in the rounding algorithm is an s-t path, so in particular s and t are always connected in our solution. Another difference is that our problem has some extra structure, so that we can compute the electrical flow f^* to guide our rounding procedure, where the variables f_e^* are not in the convex program. These allow us to obtain a constant factor approximation algorithm for all budget $k \geq d_{st}$ (note that when $k < d_{st}$ there is no feasible integral solution).

In the analysis, we prove in Lemma 197 that the expected number of edges in H is at most k, and in Lemma 198 that the expected effective resistance is $\mathrm{Reff}_H(s,t) \leq 2\mathrm{Reff}_{x^*}(s,t)$. To bound the expected effective resistance, we use Thomson's principle and construct a unit s-t flow F to show that $\mathrm{Reff}_H(s,t) \leq \mathcal{E}_H(F) \leq 2\mathrm{Reff}_{x^*}(s,t)$. To construct the unit s-t flow F, the idea is to keep the flow-conductance ratio and send α units of flow on each sampled path P_i (i.e. $f_e = \alpha$ and $x_e = 1$). The flow-conductance ratio plays a crucial role in the proofs of both lemmas. This is because the rounding algorithm is based on the flow variables f_e^*, and thus the performance guarantees are in terms of f_e^*, but the ratio α allows us to relate them back to the variables x_e^* in the convex program. Combining the two lemmas give us a constant factor bicriteria approximation algorithm for the problem. This can be turned into a true approximation algorithm by scaling down the budget to $k/2$ and run the bicriteria approximation algorithm with some additional claims (Section 37.4).

The improvement on the approximation ratio when budget k is large comes from two observations. The first observation is that if k is much larger than the length of the shortest s-t path, then the number of independent iterations in the rounding scheme is large (Lemma 194). The second observation is that we can ignore some s-t paths in the flow decomposition with many fractional edges without affecting the performance much. Combining these, we can apply a Chernoff-Hoeffding bound to show that the number of edges is at most $(1+\epsilon)k$ with high probability. Then, we do not need to scale down the budget by a factor of 2 and we can prove a stronger bound that the effective resistance is at most $2 + O(\epsilon)$ times the optimal value.

35.4 Organization

In Section 36, we define the notations used in this Part and also present some background knowledge about effective resistances. We present the convex programming relaxation and our two rounding procedures in Section 37, and the dynamic programming algorithm in Section 38. The NP-hardness and small set expansion-hardness results are provided in Section 39.

36 Preliminaries

We introduce the notations and definitions for graphs and matrices in Section 36.1, and then we define electrical flow and effective resistance and state some basic results in Section 36.2.

36.1 Graphs and Matrices

Let $G = (V, E)$ be an undirected graph with a non-negative edge weight w_e on each edge $e \in E$. The number of vertices and the number of edges are denoted by $n := |V|$ and $m := |E|$. For a subset of edges $F \subseteq E$, the total weight of edges in F is denoted by $w(F) := \sum_{e \in F} w_e$. For a subset of vertices $S \subseteq V$, the set of edges with one endpoint in S and one endpoint in $V - S$ is denoted by $\delta(S)$. For a vertex v, the set of edges incident on a vertex v is denoted by $\delta(v) := \delta(\{v\})$, and the weighted degree of v is denoted by $\deg(v) := w(\delta(v))$. The volume of a set $\mathrm{vol}(S) := \sum_{v \in S} \deg(v)$ is defined as the sum of the weighted degrees of vertices in S. The conductance of a set $\phi(S) := w(\delta(S))/\mathrm{vol}(S)$ is defined as the ratio of the total weight on the boundary of S to the total weighted degrees in S. For two subsets $S_1, S_2 \subseteq V$, the set of edges with one endpoint in S_1 and one endpoint in S_2 is denoted by $E(S_1, S_2)$.

In this Part, an undirected graph $G = (V, E)$ with non-negative edge weights $w \in \mathbb{R}^E$ is interpreted as an electrical network, where each edge e is a resistor with conductance w_e (not to be confused with the conductance $\phi(S)$ of a set S as defined above), or equivalently with resistance $r_e := 1/w_e$. The adjacency matrix $A \in \mathbb{R}^{V \times V}$ of the graph is defined as $A_{u,v} = w_{u,v}$ for all $u, v \in V$. The Laplacian matrix $L \in \mathbb{R}^{V \times V}$ of the graph is defined as $L = D - A$ where $D \in \mathbb{R}^{V \times V}$ is the diagonal degree matrix with $D_{u,u} = \deg(u)$ for all $u \in V$. For each edge $e = uv \in E$, let $b_e := \chi_u - \chi_v$ where $\chi_u \in \mathbb{R}^n$ is the vector with one in the u-th entry and zero otherwise. The Laplacian matrix can also be written as

$$L = \sum_{e \in E} w_e b_e b_e^T.$$

Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of L with corresponding orthonormal eigenvectors $v_1, v_2, \ldots, v_n$ so that $L = \sum_{i=1}^n \lambda_i v_i v_i^T$. It is well-known that the Laplacian matrix is positive semidefinite and $\lambda_1 = 0$ with $v_1 = \vec{1}/\sqrt{n}$ as the corresponding eigenvector, and $\lambda_2 > 0$ if and only if G is connected. The pseudo-inverse of the Laplacian matrix L of a connected graph is defined as

$$L^\dagger = \sum_{i=2}^n \frac{1}{\lambda_i} v_i v_i^T,$$

which maps every vector x orthogonal to v_1 to a vector y such that $Ly = x$.

36.2 Electrical Flow and Effective Resistance

Before defining s-t electrical flow, we first define the standard unit s-t flow. For each edge $e = uv$, we have two variables $f(uv)$ and $f(vu)$ with $f(uv) = -f(vu)$, where $f(uv)$ is positive if the flow is going from u to v and negative otherwise. A unit s-t flow f satisfies the following flow conservation constraints:

$$\sum_{u \in \delta(v)} f(vu) = \begin{cases} 1 & v = s \\ -1 & v = t \\ 0 & \text{otherwise.} \end{cases}$$

Given a unit s-t flow f, we overload the notation and define its undirected flow vector $f : E \to \mathbb{R}_{\geq 0}$ with $f_e := |f(uv)|$ for each edge $e = uv$. A unit s-t electrical flow is a unit s-t

flow f that also satisfies the Ohm's law: There exists a potential vector $\varphi \in \mathbb{R}^V$ such that for all $u, v \in V$,

$$f(uv) = w_{uv} \cdot (\varphi(u) - \varphi(v)).$$

The effective resistance between s and t is defined as

$$\text{Reff}(s,t) := \varphi(s) - \varphi(t),$$

which is the potential difference between s and t when one unit of electrical flow is sent from s to t. The s-t effective resistance can be interpreted as the resistance of the whole graph G as a big resistor when an electrical flow is sent from s to t.

One can write the effective resistance in terms of the Laplacian matrix. For $u, v \in V$, let $b_{uv} = \chi_u - \chi_v$, where $\chi_v \in \mathbb{R}^n$ is the unit vector with 1 in the v-th entry and 0 in other entries. Combining the flow conservation constraint and the Ohm's law, it can be checked that the potential vector $\varphi \in \mathbb{R}^V$ of a unit s-t electrical flow is a solution to the linear system

$$L \cdot \varphi = b_{st}.$$

Note that $\varphi = L^\dagger b_{st}$ is a solution, and if G is connected then any solution is given by $p + c \cdot \vec{1}$ for $c \in \mathbb{R}$. Therefore, we can write

$$\text{Reff}(s,t) = \varphi(s) - \varphi(t) = b_{st}^T L^\dagger b_{st}.$$

The effective resistance can also be characterized by the energy of a flow. The energy of an s-t flow f is defined as

$$\mathcal{E}(f) := \sum_{e \in E} \frac{f_e^2}{w_e} = \sum_{e \in E} r_e f_e^2.$$

Thomson's principle [151] states that the unit s-t electrical flow is the unique unit s-t flow that minimizes the energy. This can be verified by writing down the optimality condition of the minimization problem. Moreover, this energy is exactly the s-t effective resistance. To see this, note that the flow value on edge uv in the unit s-t electrical flow satisfies $f(uv) = w_{uv} \cdot (\varphi(u) - \varphi(v)) = w_{uv} \cdot b_{uv}^T L^\dagger b_{st}$ and thus

$$\mathcal{E}(f) = \sum_{uv \in E} w_{uv} (b_{uv}^T L^\dagger b_{st})^2 = b_{st}^T L^\dagger \left(\sum_{uv \in E} w_{uv} b_{uv} b_{uv}^T \right) L^\dagger b_{st} = b_{st}^T L^\dagger L L^\dagger b_{st} = \text{Reff}(s,t).$$

To summarize, we will use the following result from Thomson's principle.

Fact 189 (Thomson's principle [151])**.** *Let f^* be the unit electrical s-t flow in G. Then*

$$\text{Reff}_G(s,t) = \min_f \{\mathcal{E}(f) \mid f \text{ is a unit } s\text{-}t \text{ flow in } G\} = \mathcal{E}(f^*).$$

A corollary of Thomson's principle is the following intuitive result known as the Rayleigh's monotonicity principle.

Fact 190 (Rayleigh's monotonicity principle)**.** *The s-t effective resistance cannot increase if the resistance of an edge is decreased.*

We will also use the following result to write a convex programming relaxation of our problem.

Fact 191 ([102])**.** *The s-t effective resistance is a convex function with respect to the conductance of the edges.*

37 Convex Programming Algorithm for Unit-Cost Unit-Resistance

In this section, we analyze a convex programming relaxation for our problem. We first describe the convex program and prove a characterization of the optimal solutions in Section 37.1. We then present a randomized rounding algorithm using flow decomposition in Section 37.2, and show that it is a constant factor bicriteria approximation algorithm in Section 37.3. Then, we show how to turn the bicriteria approximation algorithm into a true approximation algorithm in Section 37.4, and how to modify the algorithm slightly to achieve a better approximation guarantee when the budget k is large in Section 37.5. Finally, we discuss the dual problem of minimizing the cost while satisfying the effective resistance constraint in Section 37.6.

37.1 Convex Programming Relaxation

In the convex programming formulation, we introduce a variable x_e for each edge e to indicate whether e is chosen in our subgraph. Let

$$L_x := \sum_{e \in E} x_e w_e b_e b_e^T$$

be the Laplacian matrix of the fractional solution x, and $\mathrm{Reff}_x(s,t)$ be the s-t effective resistance of the graph with conductance $x_e w_e$ on edge $e \in E$. The following is a natural convex programming relaxation for the problem.

$$\begin{aligned} \min_{x \in \mathbb{R}^m} \quad & \mathrm{Reff}_x(s,t) = b_{st}^T L_x^\dagger b_{st} \\ \text{subject to} \quad & \sum_{e \in E} c_e x_e \leq k, \\ & 0 \leq x_e \leq 1, \qquad \forall e \in E. \end{aligned} \tag{CP}$$

This is an exact formulation if $x_e \in \{0,1\}$ for all $e \in E$. The objective function is convex in x by Fact 191. The convex program can be solved in polynomial time by the ellipsoid method to inverse exponential accuracy or by the techniques described in [5] to inverse polynomial accuracy, which are both sufficient for the rounding algorithm.

37.1.1 Integrality Gap Examples

We show some limitations of the convex program for general w_e and c_e, which partly justifies our assumption $w_e = c_e = 1$ for all $e \in E$. The following figure shows a simple example where the integrality gap is unbounded if the cost could be arbitrary.

In this graph, the top path has length $n-2$ where each edge has cost $1/(n-2)$. The bottom path has two edges with cost 1. The resistance of each edge is 1, and the budget is $k = 1$. The integrality gap of this example is $\Omega(n)$. To see this, the integral solution can only afford the top path, and the effective resistance is $n-2$. However, the fractional solution can set $x_e = 1/2$ for each of the two bottom edges, and the effective resistance of this fractional solution is 4.

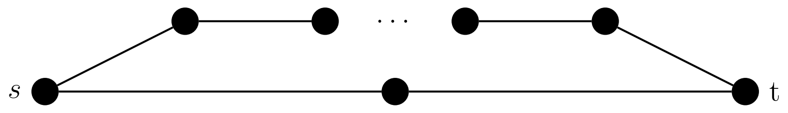

Figure 37.1: Integrality gap example with arbitrary cost and unit resistance.

The following figure shows another simple example where the integrality gap is unbounded if the edge costs are the same but the resistances could be arbitrary.

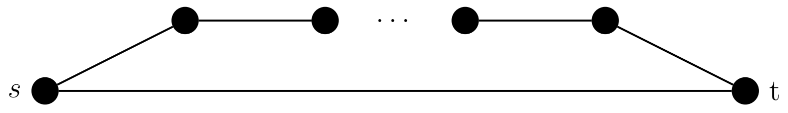

Figure 37.2: Integrality gap example with arbitrary resistance and unit cost.

In this example, the top path has length $n-1$ with each edge of resistance 1. The bottom path has only one edge with resistance R. All edges have cost 1 and the budget $k = n-2$. The integral solution can only afford the bottom path, with effective resistance R. The fractional solution can set $x_e = (n-2)/(n-1)$ for each edge in the top path, with effective resistance $O(n)$. When $R \gg n$, the integrality gap could be arbitrarily large.

Even in the unit-cost unit-resistance case, the integrality gap is unbounded if k is smaller than the s-t shortest path distance. Henceforth, we assume the following in the rest of this section.

Assumption 192. *We assume that $c_e = w_e = r_e = 1$ for every edge $e \in E$, and the budget k is at least the shortest path distance d_{st} between s and t in the input graph.*

The integrality gap of the convex program is still at least two with Assumption 192. For a simple example, consider a graph with two vertex-disjoint s-t paths, each of length $k/2+1$, and the budget is k. Then the optimal integral value is $k/2+1$ while the optimal fractional value is close to $k/4$, and so the integrality gap gets arbitrarily close to two.

We will show that the integrality gap of the convex program is at most 8 with these assumptions. Note that just to connect s and t, then k must be at least the s-t shortest path distance. It is interesting that this small additional assumption could reduce the integrality gap from unbounded to a constant.

37.1.2 Characterization of Optimal Solutions

In the case when $c_e = w_e = r_e = 1$ for all edges $e \in E$, we will prove that the electrical flow f^* supported in the optimal solution x^* to (CP) satisfies a useful property about the flow-conductance ratio f^*_e/x^*_e.

Lemma 193 (Characterization of Optimal Solution)**.** *Let $G = (V, E)$ be the input graph with $c_e = w_e = 1$ for all edges $e \in E$. Let $x^* : E \to \mathbb{R}_{\geq 0}$ be an optimal solution to the convex program (CP). Let $E_F \subseteq E$ be the set of fractional edges with $0 < x^*_e < 1$, and $E_I \subseteq E$ be the set of integral edges with $x^*_e = 1$. Let $f^* : E \to \mathbb{R}_{\geq 0}$ be the undirected flow vector of the unit s-t electrical flow supported in x^*. There exists $\alpha > 0$ such that*

$$f^*_e = \alpha x^*_e \quad \forall e \in E_F \quad \text{and} \quad f^*_e \geq \alpha \quad \forall e \in E_I.$$

Proof. By removing edges with $x_e^* = 0$, we can assume $x_e^* > 0$ for every $e \in E$. By removing isolated vertices, we can further assume that the nonzero edges form a connected graph. So, we can write $\text{Reff}_{x^*}(s,t) = b_{st}^T L_{x^*}^\dagger b_{st}$, where L_{x^*} has rank $n-1$ and the null space of L_{x^*} is $\text{span}(\vec{1})$. Since $b_{st} \perp \vec{1}$, we have $L_{x^*} L_{x^*}^\dagger b_{st} = b_{st}$ and $L_{x^*}^\dagger b_{st} \perp \vec{1}$, which implies that $L_{x^*}^\dagger b_{st} = (L_{x^*} + \frac{1}{n}J)^{-1} b_{st}$ where J is the all-ones matrix. Using the fact that $\partial A^{-1} = -A^{-1}(\partial A)A^{-1}$ (see e.g. [221]), we derive

$$\begin{aligned}
\nabla_{x_e^*} \text{Reff}_{x^*}(s,t) &= \nabla_{x_e^*} b_{st}^T \left(L_{x^*} + \frac{1}{n}J \right)^{-1} b_{st} \\
&= -b_{st}^T \left(L_{x^*} + \frac{1}{n}J \right)^{-1} (\nabla_{x_e^*} L_{x^*}) \left(L_{x^*} + \frac{1}{n}J \right)^{-1} b_{st} \\
&= -b_{st}^T L_{x^*}^\dagger \left(\nabla_{x_e^*} \sum_{e \in E} x_e^* w_e b_e b_e^T \right) L_{x^*}^\dagger b_{st} \\
&= -b_{st}^T L_{x^*}^\dagger b_e b_e^T L_{x^*}^\dagger b_{st} \\
&= -(b_{st}^T L_{x^*}^\dagger b_e)^2,
\end{aligned}$$

where we used the assumption that $w_e = 1$ for all $e \in E$. With this, we write down the KKT conditions for the convex program. Let μ be the dual variable for the budget constraint $\sum_{e \in E} c_e x_e^* \leq k$, and λ_e^+ and λ_e^- be the dual variables for the upper bound $x_e^* \leq 1$ and the nonnegative constraint $x_e^* \geq 0$ respectively. The KKT conditions states if x^* is an optimal solution to (CP), then there exist λ^+, λ^- and μ such that

$$\sum_{e \in E} x_e^* \leq k, \quad 0 \leq x_e^* \leq 1 \ \ \forall e \in E, \qquad \text{(Primal feasibility)}$$

$$\mu \geq 0, \lambda_e^+ \geq 0 \text{ and } \lambda_e^- \geq 0 \ \ \forall e \in E, \qquad \text{(Dual feasibility)}$$

$$\mu \cdot \left(k - \sum_{e \in E} x_e^* \right) = 0, \lambda_e^+ \cdot (x_e^* - 1) = 0, \lambda_e^- \cdot x_e^* = 0 \ \ \forall e \in E, \qquad \text{(Complementary slackness)}$$

$$(b_e^T L_{x^*}^\dagger b_{st})^2 = \lambda_e^+ - \lambda_e^- + c_e \mu = \lambda_e^+ - \lambda_e^- + \mu, \qquad \text{(Lagrangian optimality)}$$

where we used the assumption that $c_e = 1$ for all $e \in E$. For an integral edge with $x_e^* = 1$, we have $\lambda_e^- = 0$ by the complementary slackness condition. Since $\lambda_e^+ \geq 0$, it follows from the Lagrangian optimality condition that $(b_e^T L_{x^*}^\dagger b_{st})^2 \geq \mu$. For a fractional edge with $0 < x_e^* < 1$, we have $\lambda_e^+ = \lambda_e^- = 0$ by the complementary slackness condition, and therefore $(b_e^T L_{x^*}^\dagger b_{st})^2 = \mu$ by the Lagrangian optimality condition. We can assume that $\mu > 0$. Otherwise, $\mu = 0$ implies that the flow on all fractional edges are zero, and so we can delete them from the graph without affecting the s-t effective resistance, and we have an integral solution.

Let φ be a potential vector of the electrical flow f^* supported in x^*. For an edge $e = uv \in E$,

$$\left(\frac{f_e^*}{x_e^*} \right)^2 = (\varphi(u) - \varphi(v))^2 = \left(b_e^T L_{x^*}^\dagger b_{st} \right)^2,$$

where the first equality is by Ohm's law and the assumption that $w_{uv} = 1$ for all $uv \in E$, and the second equality uses that $L_{x^*}\varphi = b_{st}$ as explained in Section 36.2. The lemma then follows from the above paragraph and writing μ as α^2. □

The flow-conductance ratio α will be crucial in the rounding algorithm and its analysis. The following lemma shows an upper bound on α using the budget k and the shortest path distance d_{st} between s and t.

Lemma 194. *Under the conditions in Assumption 192, it holds that $\alpha^2 \le d_{st}/k \le 1$.*

Proof. Let x^* be an optimal solution to (CP), and f^* be the unit s-t electrical flow supported in x^*. As $k \ge d_{st}$, a shortest path is a feasible solution to (CP), and thus $\mathrm{Reff}_{x^*}(s,t) \le d_{st}$. On the other hand, by Thomson's principle and Lemma 193,

$$\mathrm{Reff}_{x^*}(s,t) = \sum_{e \in E} \frac{(f_e^*)^2}{x_e^*} = \sum_{e \in E_I} (f_e^*)^2 + \sum_{e \in E_F} \frac{(f_e^*)^2}{x_e^*} \ge \sum_{e \in E_I} \alpha^2 + \sum_{e \in E_F} \alpha^2 x_e^* = \alpha^2 \sum_{e \in E} x_e^* = \alpha^2 k,$$

where the last equality holds since we can assume $\sum_{e \in E} x_e^* = k$ for the optimal solution x^* without loss of generality by Rayleigh's principle (or otherwise we have an integral optimal solution). The lemma follows by combining the upper bound and the lower bound. □

37.2 Randomized Path-Rounding Algorithm

Our rounding algorithm uses the unit electrical flow f^* supported in the optimal solution x^* to construct an integral solution. The algorithm will first decompose the flow f^* as a convex combination of flow paths, and then randomly choose the flow paths and return the union of the chosen flow paths as our solution.

The following lemma about flow decomposition is by the standard argument to remove one (fractional) flow path at a time, which holds for any unit directed acyclic s-t flow.

Lemma 195 (Flow Decomposition)**.** *Given a unit s-t electrical flow f, there is a polynomial time algorithm to find a set $\mathcal{P}$ of s-t paths with $|\mathcal{P}| \le |E|$ such that the undirected flow vector $f : E \to \mathbb{R}_{\ge 0}$ can be written as a convex combination of the characteristic vectors of the paths in $\mathcal{P}$, i.e.*

$$f = \sum_{p \in \mathcal{P}} v_p \cdot \chi_p \quad \text{and} \quad \sum_{p \in \mathcal{P}} v_p = 1 \quad \text{and} \quad v_p > 0 \text{ for each } p \in \mathcal{P},$$

where $\chi_p \in R^{|E|}$ is the characteristic vector of the path p with one on each edge $e \in p$ and zero otherwise.

With the flow decomposition, we are ready to present the rounding algorithm.

Randomized Path Rounding Algorithm

1. Let x^* be an optimal solution to the convex program (CP). Let f^* be the unit s-t electrical flow supported in x^*. Let α be the flow-conductance ratio defined in Lemma 193.
2. Compute a flow decomposition $\mathcal{P}$ of f^* as defined in Lemma 195.

3. For i from 1 to $T := \lfloor 1/\alpha \rfloor$ do
    - Let P_i be a random path sampled from $\mathcal{P}$ where each path $p \in \mathcal{P}$ is sampled with probability v_p.
4. Return the subgraph H formed by the edge set $\cup_{i=1}^{T} P_i$.

The following lemma shows that the rounding algorithm will always return a non-empty subgraph.

Lemma 196. *Suppose the input instance satisfies the conditions in Assumption 192. Let x^* be an optimal solution to (CP) and $\alpha > 0$ be the flow-conductance ratio as defined in Lemma 193. Then*

$$\frac{1}{\alpha} \geq T \geq \frac{1}{2\alpha} > 0.$$

Proof. Since we assumed that the budget k is at least the length d_{st} of a shortest s-t path, it follows from Lemma 194 that $\alpha \leq 1$. This implies that

$$\frac{1}{\alpha} \geq T = \left\lfloor \frac{1}{\alpha} \right\rfloor \geq \max\left\{1, \frac{1}{\alpha} - 1\right\} \quad \Longrightarrow \quad 1 \geq T\alpha \geq \max\{\alpha, 1-\alpha\} \geq \frac{1}{2}.$$

□

37.3 Bicriteria Approximation

The analysis of the approximation guarantee goes as follows. First, we show that the expected number of edge in the returned subgraph is at most the budget k. Then, we prove that the expected effective resistance of the returned subgraph is at most two times that of the optimal fractional solution. Both of these steps use the flow-conductance ratio α crucially. These combine to show that the randomized path rounding algorithm is a constant factor bicriteria approximation algorithm.

Let x^* be an optimal solution to (CP). Let E_F and E_I be the set of fractional edges and integral edges in x^*. We assume that each edge $e \in E_I$ will be included in the subgraph H returned by the rounding algorithm. We focus on bounding the number of edges in E_F that will be included in H.

Lemma 197 (Expected Budget)**.** *Let x^* be an optimal solution to (CP) when $c_e = w_e = 1$ for all edges $e \in E$. Let X_e be an indicator variable of whether e is included in the returned subgraph H by the rounding algorithm, Then,*

$$\mathbf{E}\left[\sum_{e \in E_F} X_e\right] \leq T\alpha \sum_{e \in E_F} x_e^* \leq \sum_{e \in E_F} x_e^*.$$

Proof. Note that an edge e is contained in P_i with probability $\sum_{p \in \mathcal{P}: p \ni e} v_p$. By the union bound, an edge e is included in the returned subgraph H by the rounding algorithm with probability

$$\mathbf{Pr}[X_e = 1] \leq \sum_{i=1}^{T} \sum_{p \in \mathcal{P}: p \ni e} v_p = T \sum_{p \in \mathcal{P}: p \ni e} v_p = T f_e^*,$$

where the last equality holds by the property of the flow decomposition $\mathcal{P}$ of the electrical flow f^* in Lemma 195.

By Lemma 193, $f_e^* = \alpha x_e^*$ for each fractional edge $e \in E_F$, and this implies that

$$\mathbf{Pr}[X_e = 1] \leq T f_e^* = T\alpha x_e^* \quad \forall e \in E_F.$$

Therefore,

$$\mathbf{E}\left[\sum_{e \in E_F} X_e\right] = \sum_{e \in E_F} \mathbf{Pr}[X_e = 1] \leq T\alpha \sum_{e \in E_F} x_e^* = \left\lfloor \frac{1}{\alpha} \right\rfloor \alpha \sum_{e \in E_F} x_e^* \leq \sum_{e \in E_F} x_e^*.$$

□

The key step is to show that $\mathbf{E}[\mathrm{Reff}_H(s,t)] \leq 2\mathrm{Reff}_{x^*}(s,t)$. To prove this, we construct a unit s-t flow F and show that $\mathbf{E}[\mathcal{E}_H(F)] \leq 2\mathrm{Reff}_{x^*}(s,t)$, and hence by Thomson's principle $\mathbf{E}[\mathrm{Reff}_H(s,t)] \leq \mathbf{E}[\mathcal{E}_H(F)] \leq 2\mathrm{Reff}_{x^*}(s,t)$. To construct the flow F, the idea is to follow the ratio α in the fractional solution x^* and send α units of flow on each path P_i selected.

Lemma 198 (Expected Effective Resistance)**.** *Suppose the input instance satisfies the conditions in Assumption 192. Let x^* be an optimal solution to (CP) and f^* be the unit s-t electrical flow supported in x^*. The expected s-t effective resistance of the subgraph H returned by the rounding algorithm is*

$$\mathbf{E}\left[\mathrm{Reff}_H(s,t)\right] \leq \left(1 - \frac{1}{T} + \frac{1}{T\alpha}\right) \cdot \mathcal{E}_{x^*}(f^*) = \left(1 - \frac{1}{T} + \frac{1}{T\alpha}\right) \cdot \mathrm{Reff}_{x^*}(s,t) \leq 2\mathrm{Reff}_{x^*}(s,t).$$

Proof. Consider the undirected flow vector $F : E \to \mathbb{R}_{\geq 0}$ defined by sending α units of flow on each path P_i chosen by the rounding algorithm, i.e. the random variable $F = \sum_{i=1}^{T} \alpha \cdot \chi_{P_i}$ with $F_e = \alpha \cdot |\{P_i \mid 1 \leq i \leq T, P_i \ni e\}|$ for each edge $e \in E$. We would like to upper bound the expected energy $\mathcal{E}_H(F)$ in order to upper bound $\mathrm{Reff}_H(s,t)$.

Each P_i is a random s-t path sampled from the flow decomposition $\mathcal{P}$ of the undirected flow vector $f^* : E \to \mathbb{R}_{\geq 0}$ of the unit s-t electrical flow supported in x^*, and $\chi_{P_i} \in \mathbb{R}^m$ is its characteristic vector with expected value

$$\mathbf{E}[\chi_{P_i}] = \sum_{p \in \mathcal{P}} v_p \cdot \chi_p = f^*.$$

Since each edge in H is of conductance one, the expected energy of F in H is

$$\mathbf{E}\left[\mathcal{E}_H(F)\right] = \mathbf{E}[\langle F, F \rangle] = \mathbf{E}\left[\left\langle \sum_{i=1}^{T} \alpha \cdot \chi_{P_i}, \sum_{j=1}^{T} \alpha \cdot \chi_{P_j} \right\rangle\right] = \sum_{i=1}^{T} \sum_{j=1}^{T} \alpha^2 \cdot \mathbf{E}[\langle \chi_{P_i}, \chi_{P_j} \rangle].$$

As each path P_i is sampled independently, for $i \neq j$,

$$\mathbf{E}[\langle \chi_{P_i}, \chi_{P_j} \rangle] = \langle \mathbf{E}[\chi_{P_i}], \mathbf{E}[\chi_{P_j}] \rangle = \langle f^*, f^* \rangle = \sum_{e \in E} (f_e^*)^2.$$

For $i = j$,

$$\mathbf{E}[\langle \chi_{P_i}, \chi_{P_i} \rangle] = \sum_{p \in \mathcal{P}} v_p \langle \chi_p, \chi_p \rangle = \sum_{p \in \mathcal{P}} v_p \sum_{e \in p} 1 = \sum_{e \in E} \sum_{p \in \mathcal{P} : p \ni e} v_p = \sum_{e \in E} f_e^*,$$

where the last equality follows from the property of the flow decomposition in Lemma 195. Combining these two terms, it follows that

$$\mathbf{E}\left[\mathcal{E}_H(F)\right] = \alpha^2 T \sum_{e \in E} f_e^* + \alpha^2 T(T-1) \sum_{e \in E} (f_e^*)^2.$$

Thomson's principle states that the $\mathrm{Reff}_H(s,t)$ is upper bounded by the energy of any one unit s-t flow. Note that F is an s-t flow of $T\alpha$ units, and $T\alpha > 0$ by Lemma 196. Scaling F to a one unit s-t flow by dividing the flow on each edge by $T\alpha$ gives an upper bound on

$$\begin{aligned}
\mathbf{E}[\mathrm{Reff}_H(s,t)] \leq \frac{\mathbf{E}\left[\mathcal{E}_H(F)\right]}{T^2\alpha^2} &= \frac{1}{T} \sum_{e \in E} f_e^* + \left(1 - \frac{1}{T}\right) \sum_{e \in E} (f_e^*)^2 \\
&\leq \frac{1}{T\alpha} \sum_{e \in E} \frac{(f_e^*)^2}{x_e^*} + \left(1 - \frac{1}{T}\right) \sum_{e \in E} \frac{(f_e^*)^2}{x_e^*} \\
&= \left(1 - \frac{1}{T} + \frac{1}{T\alpha}\right) \cdot \mathcal{E}_{x^*}(f^*) \\
&= \left(1 - \frac{1}{T} + \frac{1}{T\alpha}\right) \cdot \mathrm{Reff}_{x^*}(s,t),
\end{aligned}$$

where the second inequality follows from Lemma 193 that $f_e^*/x_e^* \geq \alpha$ for every edge $e \in E$ and also $x_e^* \leq 1$ for every edge $e \in E$, and the last equality is from Thomson's principle that $\mathrm{Reff}_{x^*}(s,t) = \mathcal{E}_{x^*}(f^*)$. Finally, notice that $1 - 1/T + 1/(T\alpha) \leq 2$ as $1/\alpha - 1 \leq \lfloor 1/\alpha \rfloor = T$. □

Combining Lemma 197 and Lemma 198, it follows from a simple application of Markov's inequality that there is an outcome of the randomized path-rounding algorithm which uses at most $2k$ edges with s-t effective resistance at most $4\mathrm{Reff}_{x^*}(s,t)$. In the following, we apply Markov's inequality more carefully to show that the success probability is at least $\Omega(\alpha)$. In the next subsection, we will argue that α can be assumed to be $\Omega(1/m)$ and so the path-rounding algorithm is a randomized polynomial time algorithm.

Theorem 199 (Bicriteria Approximation)**.** *Suppose the input instance satisfies the conditions in Assumption 192. Let x^* be an optimal solution to (CP). Given x^*, the randomized path rounding algorithm will return a subgraph H with at most $2k$ edges and $\mathrm{Reff}_H(s,t) \leq 4\mathrm{Reff}_{x^*}(s,t)$ with probability at least $\Omega(\alpha)$.*

Proof. First, we bound the probability that the subgraph H has more than $2k$ edges. Let X_e be an indicator variable of whether the edge e is included in the returned subgraph H. Recall that E_F and E_I denote the set of fractional edges and integral edges in x^* respectively. We assume pessimistically that all edges in E_I will be included in the subgraph H returned by the rounding algorithm. Then, by Markov's inequality and Lemma 197,

$$\Pr\left(\sum_{e\in E} X_e > 2k\right) \le \Pr\left(\sum_{e\in E_F} X_e > 2k - |E_I|\right) \le \frac{\mathbf{E}\left[\sum_{e\in E_F} X_e\right]}{2k-|E_I|} \le \frac{T\alpha \sum_{e\in E_F} x^*_e}{2k-|E_I|} \le \frac{T\alpha}{2},$$

where the last inequality is by $\sum_{e\in E_F} x^*_e \le k - |E_I|$.

Next, we bound the probability that $\text{Reff}_H(s,t) > 4\text{Reff}_{x^*}(s,t)$. By Markov's inequality and Lemma 198,

$$\Pr\left(\text{Reff}_H(s,t) > 4\text{Reff}_{x^*}(s,t)\right) \le \frac{1}{4}\left(1 - \frac{1}{T} + \frac{1}{T\alpha}\right) = \frac{T\alpha+1}{4T\alpha} - \frac{1}{4T} \le \frac{T\alpha+1}{4T\alpha} - \Omega(\alpha),$$

where the last inequality is because $T = \lfloor 1/\alpha \rfloor \le 1/\alpha$.

To prove the lemma, it remains to show that

$$\frac{T\alpha}{2} + \frac{T\alpha+1}{4T\alpha} \le 1 \quad \iff \quad 2(T\alpha)^2 - 3(T\alpha) + 1 = (2T\alpha - 1)(T\alpha - 1) \le 0,$$

which follows from Lemma 196. □

37.4 Constant Factor Approximation

We showed that the randomized path rounding algorithm is a bicriteria approximation algorithm. To achieve a true approximation algorithm, a natural idea is to scale down the budget from k to $k/2$ and apply the randomized path rounding algorithm. The following lemma takes care of the case of $k/2 < d_{st}$, when the shortest path assumption does not hold after scaling, by showing that simply returning a shortest s-t path is already a good enough approximation.

Lemma 200. *When the budget k is at least the length d_{st} of a shortest s-t path, any s-t shortest path is a (k/d_{st})-approximate solution for the s-t effective resistance network design problem.*

Proof. When $k \ge d_{st}$, an s-t shortest path is a feasible solution to the problem with s-t effective resistance at most d_{st}. To prove the lemma, we will show that $\text{Reff}_x(s,t) \ge d_{st}^2/k$ for any feasible solution x to (CP), and so an s-t shortest path is already a (k/d_{st})-approximation.

Let G_x be the graph G with fractional conductance x_e on each edge $e \in E$. To show a lower bound on $\text{Reff}_x(s,t)$, we identify the vertices in G_x to a form a path graph P_x as follows: For each $i \ge 0$, let U_i be the set of vertices in G with shortest path distance i to s, where the shortest path distance is defined where each edge in G is of length one. First, for each $0 \le i \le d_{st} - 1$, we identify the vertices in U_i to a single vertex u_i. Then, we identify all the vertices in $\cup_{i \ge d_{st}} U_i$ to a single vertex $u_{d_{st}}$. The path graph P_x has vertex set $\{u_0, \ldots, u_{d_{st}}\}$ and edge set $\{ab \in E \mid a \in U_i \text{ and } b \in U_{i+1} \text{ for } 0 \le i \le d_{st} - 1\}$. For each edge e in P_x, its conductance x_e in P_x is the same as that in G_x. As an electrical network, identifying two vertices uv is equivalent to adding an edge of resistance zero between u and v. So, it follows from Rayleigh's monotonicity principle (Fact 190) that $\mathrm{Reff}_{G_x}(s,t) \geq \mathrm{Reff}_{P_x}(u_0, u_{d_{st}})$ as $s \in U_0$ and $t \in U_{d_{st}}$.

As P_x is a series-parallel graph, we can compute $\mathrm{Reff}_{P_x}(s,t)$ directly. For each $1 \leq i \leq d_{st}$, let E_i be the set of parallel edges connecting u_{i-1} and u_i in P_x, and $c_i = \sum_{e \in E_i} x_e$ be the effective conductance between u_{i-1} and u_i in P_x. Then, by Fact 214,

$$\mathrm{Reff}_{P_x}(u_{i-1}, u_i) = \frac{1}{c_i} \quad \text{and} \quad \mathrm{Reff}_{P_x}(u_0, u_{d_{st}}) = \sum_{i=1}^{d_{st}} \mathrm{Reff}_{P_x}(u_{i-1}, u_i) = \sum_{i=1}^{d_{st}} \frac{1}{c_i}.$$

Note that $\sum_{i=1}^{d_{st}} c_i = \sum_{i=1}^{d_{st}} \sum_{e \in E_i} x_e \leq \sum_{e \in E} x_e \leq k$ for any feasible solution x. Using Cauchy-Schwarz inequality,

$$d_{st} = \sum_{i=1}^{d_{st}} \sqrt{c_i} \cdot \frac{1}{\sqrt{c_i}} \leq \sqrt{\sum_{i=1}^{d_{st}} c_i} \cdot \sqrt{\sum_{i=1}^{d_{st}} \frac{1}{c_i}} \leq \sqrt{k} \cdot \sqrt{\mathrm{Reff}_{P_x}(u_0, u_{d_{st}})}.$$

Therefore, we conclude that $\mathrm{Reff}_{G_x}(s,t) \geq \mathrm{Reff}_{P_x}(u_0, u_{d_{st}}) \geq d_{st}^2/k$. □

We are ready to prove our main approximation result.

Theorem 201. *Suppose the input instance satisfies the conditions in Assumption 192. There is a polynomial time* 8*-approximation algorithm for the s-t effective resistance network design problem.*

Proof. If the budget $k \leq 2d_{st}$, then Lemma 200 shows that simply returning an s-t shortest path would give a 2-approximation. Henceforth, we assume that $k \geq 2d_{st}$.

Let $\mathrm{opt}(k)$ be the objective value of an optimal solution to the convex program (CP) with budget k. Let x^* be an optimal solution to (CP) with budget k and $\mathrm{Reff}_{x^*}(s,t) = \mathrm{opt}(k)$. As $\frac{1}{2}x^*$ is a feasible solution to (CP) with budget $\frac{1}{2}k$, by Thomson's principle,

$$\mathrm{opt}\left(\frac{k}{2}\right) \leq \mathrm{Reff}_{\frac{1}{2}x^*}(s,t) = b_{st}^T \Big(\sum_{e \in E} \frac{x_e^*}{2} b_e b_e^T \Big)^{\dagger} b_{st} = 2 b_{st}^T \Big(\sum_{e \in E} x_e^* b_e b_e^T \Big)^{\dagger} b_{st} = 2\mathrm{Reff}_{x^*}(s,t) = 2\mathrm{opt}(k).$$

Given the original budget $k \geq 2d_{st}$, our algorithm is to find an optimal solution z^* to (CP) with budget $k/2 \geq d_{st}$, and then use the path-rounding algorithm with input z^* to return a subgraph H. By Theorem 199, with probability $\Omega(\alpha)$, the subgraph H satisfies

$$|E(H)| \leq 2 \sum_{e \in E} z_e^* \leq 2 \left(\frac{k}{2}\right) = k \quad \text{and} \quad \mathrm{Reff}_H(s,t) \leq 4\mathrm{opt}\left(\frac{k}{2}\right) \leq 8\mathrm{opt}(k),$$

and so H is an 8-approximate solution to the s-t effective resistance network design problem.

Finally, we consider the time complexity of the algorithm. The number of iterations in the path rounding algorithm is $O(1/\alpha)$, and we need to run the path rounding algorithm $O(1/\alpha)$ times to boost the success probability to a constant. This is a randomized polynomial time algorithm when $\alpha = \Omega(1/m)$.

In the following, we show that when $\alpha \le 1/(4m)$, it is easy to obtain a 2-approximate solution without running the path-rounding algorithm. Let x^* be an optimal solution to (CP) with budget k, and f^* be the unit s-t electrical flow supported in x^*. Let $\mathcal{P}$ be the flow decomposition of f^* as in Lemma 195. We call a path $p \in \mathcal{P}$ an integral path if every edge $e \in p$ has $x^*_e = 1$; otherwise we call p a fractional path. When $\alpha \le 1/(4m)$, we simply return the union of all integral paths as our solution H. Clearly, H has at most k edges as it only contains integral edges. Next, we bound $\mathrm{Reff}_H(s,t)$ by the energy of the flow supported in the integral paths. By Lemma 193, an edge e with $x^*_e < 1$ has $f^*_e = \alpha x^*_e < \alpha \le 1/(4m)$. This implies that each fractional path p has $v_p \le 1/(4m)$. Since $\mathcal{P}$ has at most m paths (Lemma 195), the total flow in the fractional paths is at most $1/4$, and thus the total flow in the integral paths is at least $3/4$. By scaling the flow supported in the integral paths to a one unit s-t flow, we see that

$$\mathrm{Reff}_H(s,t) \le \frac{\mathcal{E}_{x^*}(f^*)}{(3/4)^2} \le 2\mathcal{E}_{x^*}(f^*) = 2\mathrm{Reff}_{x^*}(s,t).$$

To summarize, in all cases including $k < 2d_{st}$ and $\alpha \le 1/(4m)$, there is a polynomial time algorithm to return an 8-approximate solution to the s-t effective resistance network design problem. □

We have two remarks about improvements of Theorem 201.

Remark 202 (Approximation Ratio). The analysis of the 8-approximation algorithm is not tight. By a more careful analysis of the expected energy in Lemma 198 and the short path idea used in the next subsection, we can show that the approximation guarantee of the same algorithm in Theorem 201 is less than 5. However, the analysis is quite involved and not very insightful, so we have decided to omit those details and only keep the current analysis.

Remark 203 (Deterministic Algorithm). Using the standard pessimistic estimator technique, we can derandomize the path-rounding algorithm to obtain a deterministic 8-approximation algorithm. The analysis is standard and we omit the details that would take a few pages.

37.5 The Large Budget Case

In this subsection, we show how to modify the algorithm in Theorem 201 to achieve a better approximation ratio when the budget is much larger than the s-t shortest path distance.

The observation is that when $k \gg d_{st}$, then α is small by Lemma 194, and so there are many iterations in the path-rounding algorithm. Since each iteration is independent, we can use Chernoff-Hoeffding's bound to prove a stronger bound on the probability that the number of edges in the returned solution is significantly more than k (outperforms the bound proved in Lemma 197 using Markov's inequality). Then, we can show that the expected s-t effective resistance is close to two times the optimal value by argument similar to the proof of Lemma 198.

Modified Rounding Algorithm

For our analysis, we slightly modify the path-rounding algorithm to ignore "long" paths in the flow decomposition, so that we have a worst case bound to apply Chernoff-Hoeffding's bound.

Unlike the flow decomposition in Lemma 195, the short path flow decomposition definition is specific to the electrical flow of an optimal solution to (CP). In the following definition, c is a parameter which will be set to be $1/\epsilon > 1$ to achieve a $(2 + O(\epsilon))$-approximation.

Definition 204 (Short Path Decomposition of Electrical Flow of Optimal Solution)**.** Let x^* be an optimal solution to the convex program (CP). Let f^* be the unit s-t electrical flow supported in x^*. Let α be the flow-conductance ratio defined in Lemma 193.

Let $\mathcal{P}^*$ be a flow decomposition of f^* as defined in Lemma 195. Let $x^*_F := \sum_{e \in E_F} x^*_e$ be the total fractional value on the fractional edges E_F in the optimal solution x^*.

We call a path $p \in \mathcal{P}^*$ a long path if p has at least $c\alpha x^*_F$ edges in E_F, i.e. $|p \cap E_F| \geq c\alpha x^*_F$. Otherwise we call a path $p \in \mathcal{P}^*$ a short path.

Let $\mathcal{P} := \{p \in \mathcal{P}^* \mid p \text{ is a short path}\}$ be the collection of short paths in $\mathcal{P}^*$. Let $f_{\mathcal{P}} := \sum_{p \in \mathcal{P}} v_p \chi_p$ be the s-t flow defined by the short paths, and $v_{\mathcal{P}} := \sum_{p \in \mathcal{P}} v_p$ be the total flow value of $f_{\mathcal{P}}$.

The modified algorithm is very similar to the randomized path-rounding algorithm in Section 37.3. The only difference is that we only use the paths in the short path flow decomposition in Definition 204, and we adjust the sampling probability of a path p to $v_p/v_{\mathcal{P}}$ so that the sum is one.

Randomized Short Path Rounding Algorithm

1. Let x^* be an optimal solution to the convex program (CP). Let f^* be the unit s-t electrical flow supported in x^*. Let α be the flow-conductance ratio defined in Lemma 193.

2. Compute a short path flow decomposition $\mathcal{P}$ of f^* as described in Definition 204.

3. For i from 1 to $T = \lfloor 1/\alpha \rfloor$ do

    - Let P_i be a random path sampled from $\mathcal{P}$ where each path $p \in \mathcal{P}$ is sampled with probability $v_p/v_{\mathcal{P}}$.

4. Return the subgraph H formed by the edge set $\cup_{i=1}^T P_i$.

The following simple lemma shows that the total flow on the long paths is negligible when c is large, which will be useful in the analysis.

Lemma 205. *In the short path flow decomposition in Definition 204, $v_{\mathcal{P}} \geq 1 - \frac{1}{c}$.*

Proof. Using $\alpha x^*_e = f^*_e$ for $e \in E_F$ from Lemma 193 and the properties of the flow decomposition $\mathcal{P}^*$ of f^* in Lemma 195,

$$\alpha x^*_F = \sum_{e \in E_F} f^*_e = \sum_{p \in \mathcal{P}^*} v_p \cdot |p \cap E_F| \geq \sum_{p \in \mathcal{P}^* - \mathcal{P}} v_p \cdot |p \cap E_F| \geq c\alpha x^*_F \sum_{p \in \mathcal{P}^* - \mathcal{P}} v_p = c\alpha x^*_F (1 - v_{\mathcal{P}}),$$

where the last inequality is by the definition of long paths and the last equality is because f^* is a unit s-t flow. □

Analysis of Approximation Guarantee

First, we consider the expected s-t effective resistance of the returned subgraph H. For intuition, we can think of the modified rounding algorithm as applying the rounding algorithm in the scaled flow $f_{\mathcal{P}}/v_{\mathcal{P}}$, and so it should follow from Lemma 198 that

$$\mathbf{E}[\mathrm{Reff}_H(s,t)] \leq 2\mathcal{E}_{x^*}\left(\frac{f_{\mathcal{P}}}{v_{\mathcal{P}}}\right) = \frac{2}{v_{\mathcal{P}}^2}\mathcal{E}_{x^*}(f_{\mathcal{P}}) \leq \frac{2}{v_{\mathcal{P}}^2}\mathcal{E}_{x^*}(f^*) = \frac{2}{v_{\mathcal{P}}^2}\mathrm{Reff}_{x^*}(s,t),$$

which will be at most $(2+O(\epsilon))\mathrm{Reff}_{x^*}(s,t)$ when $c = 1/\epsilon$ from Lemma 205.

We cannot directly apply Lemma 198 as stated, as the flow $f_{\mathcal{P}}$ does not satisfy the flow-conductance ratio α in Lemma 193, but essentially the same proof will work to get the same conclusion (but not exactly the same intermediate step).

Lemma 206. *Suppose the input instance satisfies the conditions in Assumption 192. Let x^* be an optimal solution to (CP) and f^* be the unit s-t electrical flow supported in x^*. The expected s-t effective resistance of the subgraph H returned by the randomized short path rounding algorithm is*

$$\mathbf{E}\left[\mathrm{Reff}_H(s,t)\right] \leq \frac{2}{v_{\mathcal{P}}^2}\mathcal{E}_{x^*}(f^*) = \frac{2}{v_{\mathcal{P}}^2}\mathit{Reff}_{x^*}(s,t),$$

where $\mathcal{P}$ is the short path flow decomposition of f^* as described in Definition 204.

The main difference of the analysis is to apply the following Hoeffding's inequality (instead of Markov's inequality) to bound the probability that the returned subgraph has significantly more than k edges.

Fact 207 (Hoeffding's Inequality)**.** *Let $X_1, \ldots, X_n$ be independent random variables such that $X_i \in [0, M]$ with probability one. Let $X = \sum_{i=1}^n X_i$, and $\mu = \mathbf{E}[X]$, then for any $\delta > 0$,*

$$\Pr(X \geq (1+\delta)\mu) \leq \exp\left(-\frac{2\delta^2\mu^2}{nM^2}\right).$$

Lemma 208. *Suppose the input instance satisfies the conditions in Assumption 192. Let x^* be an optimal solution to (CP) and f^* be the unit s-t electrical flow supported in x^*. Let H be the subgraph returned by the randomized short path rounding algorithm given x^* as input, and $|E(H)|$ be the number of edges in H. Then, for any $\delta > 0$,*

$$\Pr\left(|E(H)| \geq (1+\delta)k\right) \leq \exp\left(-\frac{2\delta^2}{c^2\alpha}\right),$$

where c is the parameter in the short path flow decomposition in Definition 204 and α is the flow-conductance ratio of f^* and x^* as defined in Lemma 193.

Proof. As in Lemma 197, we assume pessimistically that all integral edges E_I will be included in H, and so we focus on the fractional edges E_F. Let $X_{i,e}$ be the indicator variable of whether the edge e is sampled in the i-th iteration of the short path rounding algorithm, and $X_{i,F} := \sum_{e \in E_F} X_{i,e}$ be the total number of fractional edges sampled in the i-th iteration. Let

X_F be the total number of fractional edges in H. Note that $X_F \leq \sum_{i=1}^T X_{i,F}$, since if some fractional edge was sampled multiple times in different iterations, we only count it once in X_F. By linearity of expectation, $\mathbf{E}[X_F] \leq \sum_{i=1}^T \mathbf{E}[X_{i,F}]$.

Let $\mathcal{P}^*$ be the flow path decomposition of f^* in Lemma 195, and $\mathcal{P}$ be the short path flow decomposition of f^* as described in Definition 204. For an edge e, recall that $(f_{\mathcal{P}})_e := \sum_{p \in \mathcal{P}: p \ni e} v_p$ is the total flow value on e from the short paths in $\mathcal{P}$. As we scaled the probability of each path by $1/v_{\mathcal{P}}$ in the rounding algorithm, the probability that edge e is sampled in the i-th iteration is $(f_{\mathcal{P}})_e / v_{\mathcal{P}}$. Let $\overline{f}_e := f_e^* - (f_{\mathcal{P}})_e$ be the total flow value on e from the long paths in $\mathcal{P}^* - \mathcal{P}$. The expected value of $X_{i,F}$ is

$$\mathbf{E}[X_{i,F}] = \sum_{e \in E_F} \mathbf{E}[X_{i,e}] = \sum_{e \in E_F} \frac{(f_{\mathcal{P}})_e}{v_{\mathcal{P}}} = \sum_{e \in E_F} \frac{f_e^* - \overline{f}_e}{v_{\mathcal{P}}} = \sum_{e \in E_F} \frac{\alpha x_e^* - \overline{f}_e}{v_{\mathcal{P}}}$$

By the definition of the long paths,

$$\sum_{e \in E_F} \overline{f}_e = \sum_{e \in E_F} \sum_{p \in \mathcal{P}^* - \mathcal{P}} v_p = \sum_{p \in \mathcal{P}^* - \mathcal{P}} v_p \cdot |p \cap E_F| \geq c\alpha x_F^* \sum_{p \in \mathcal{P}^* - \mathcal{P}} v_p = c\alpha x_F^* (1 - v_{\mathcal{P}}),$$

where we recall that $x_F^* = \sum_{e \in E_F} x_e^*$. Therefore,

$$\mathbf{E}[X_{i,F}] = \sum_{e \in E_F} \frac{\alpha x_e^* - \overline{f}_e}{v_{\mathcal{P}}} \leq \alpha x_F^* \cdot \frac{1 - c + c v_{\mathcal{P}}}{v_{\mathcal{P}}} = \alpha x_F^* \cdot (c - \frac{c-1}{v_{\mathcal{P}}}) \leq \alpha x_F^*,$$

where the last inequality uses that $v_{\mathcal{P}} \leq 1$ and $c > 1$. It follows that $\mathbf{E}[X_F] \leq T\alpha x_F^* \leq x_F^*$.

As each iteration is independent, the random variables $X_{i,F}$ for $1 \leq i \leq T$ are independent. Since we only use short paths, the maximum value of each $X_{i,F}$ is at most $c\alpha x_F^*$. So, we can apply Hoeffding's inequality to show that

$$\Pr\left(X_F \geq (1+\delta)x_F^*\right) \leq \exp\left(-\frac{2\delta^2 (x_F^*)^2}{Tc^2\alpha^2 (x_F^*)^2}\right) \leq \exp\left(-\frac{2\delta^2}{c^2\alpha}\right).$$

Let X_I be the total number of integral edges in H. As $X_I \leq |E_I|$, we conclude that

$$\Pr\Big(|E(H)| \geq (1+\delta)k\Big) = \Pr\Big(X_I + X_F \geq (1+\delta)(|E_I| + x_F^*)\Big)$$

$$\leq \Pr\Big(X_F \geq (1+\delta)x_F^*\Big) \leq \exp\left(-\frac{2\delta^2}{c^2\alpha}\right).$$

□

As in Section 37.3, we can combine Lemma 208 and Lemma 206 to show that the randomized short path rounding algorithm is a bicriteria approximation algorithm.

Theorem 209. *Suppose the input instance satisfies the conditions in Assumption 192. Suppose further that $k \geq d_{st}/\epsilon^{10}$, where $\epsilon > 0$ is an error parameter satisfying $\epsilon \leq \eta$ for a small constant η. Let x^* be an optimal solution to (CP). Given x^*, the randomized short path rounding algorithm with $c = 1/\epsilon$ will return a subgraph H with at most $(1+\epsilon)k$ edges and $Reff_H(s,t) \leq (2 + 10\epsilon) \cdot Reff_{x^*}(s,t)$ with probability at least ϵ.*

Proof. The additional assumption $k \geq d_{st}/\epsilon^{10}$ implies that $\alpha \leq \epsilon^5$ by Lemma 194.

Setting $c = 1/\epsilon$ and $\delta = \epsilon$, it follows from Lemma 208 that

$$\Pr(|E(H)| \geq (1+\epsilon)k) \leq \exp\left(-\frac{2\delta^2}{c^2\alpha}\right) \leq \exp\left(-\frac{2}{\epsilon}\right) < \epsilon,$$

where the last inequality holds for $\epsilon > 0$.

Since $c = 1/\epsilon$, Lemma 205 implies that $v_{\mathcal{P}} \geq 1 - \epsilon$ for the short path flow decomposition in Definition 204. Using Markov's inequality and Lemma 206, for sufficiently small ϵ we have

$$\begin{aligned}\Pr\Big(\mathrm{Reff}_H(s,t) \geq (2+10\epsilon)\cdot \mathrm{Reff}_{x^*}(s,t)\Big) &\leq \frac{\mathbf{E}\left[\mathrm{Reff}_H(s,t)\right]}{(2+10\epsilon)\cdot \mathrm{Reff}_{x^*}(s,t)} \\ &\leq \frac{2}{v_{\mathcal{P}}^2(2+10\epsilon)} \leq \frac{2}{(1-\epsilon)^2(2+10\epsilon)} < 1-2\epsilon\end{aligned}$$

Therefore, with probability at least ϵ, the subgraph H returned by the randomized short path rounding algorithm satisfies both properties. ☐

Using the same arguments as in Section 37.4, we can turn the above bicriteria approximation algorithm into a true approximation algorithm.

Theorem 210. *Suppose the input instance satisfies the conditions in Assumption 192. Suppose further that $k \geq 2d_{st}/\epsilon^{10}$, where $\epsilon > 0$ is an error parameter satisfying $\epsilon \leq \eta$ for a small constant η. There is a polynomial time $(2+O(\epsilon))$-approximation algorithm for the s-t effective resistance network design problem.*

Proof. As in the proof of Theorem 201, we apply the bicriteria approximation algorithm in Theorem 209 with input x^*, an optimal solution to (CP) with the scaled-down budget $k/(1+\epsilon)$, to return a subgraph H. As the new budget $k/(1+\epsilon)$ is still greater than d_{st}/ϵ^{10}, by Theorem 209, with probability at least ϵ the subgraph H satisfies $|E(H)| \leq (1+\epsilon)k/(1+\epsilon) = k$ and

$$\mathrm{Reff}_H(s,t) \leq \big(2+O(\epsilon)\big)\cdot \mathrm{opt}\left(\frac{k}{1+\epsilon}\right) \leq \big(2+O(\epsilon)\big)(1+\epsilon)\cdot \mathrm{opt}(k) \leq \big(2+O(\epsilon)\big)\cdot \mathrm{opt}(k),$$

where we used the notations and arguments in Theorem 201.

For the time complexity, note that $\alpha \leq \epsilon^5$ by Lemma 194 and the large budget assumption, and so we can assume that $\epsilon^5 \geq \alpha \geq 1/(4m)$, as otherwise there is a simple 2-approximation algorithm in the case $\alpha \leq 1/(4m)$ described in Theorem 201. Therefore, the success probability can be boosted to a constant in polynomial number of executions of the bicriteria algorithm in Theorem 209. ☐

37.6 Cost Minimization with s-t Effective Resistance Constraint

In this subsection, we consider a "dual" problem of the s-t effective resistance minimization problem. In the dual problem, we are given a graph $G = (V,E)$ with a cost c_e and a resistance r_e on each edge and a target effective resistance R, and the objective is to find a subgraph H of minimum cost such that $\mathrm{Reff}_H(s,t) \leq R$. The same NP-hardness proof in

Section 39.1 can be used to show that the dual problem is NP-complete even when the edges of G have unit cost and unit resistance.

Using the same techniques for the s-t effective resistance minimization problem, we can obtain a constant factor bicriteria approximation algorithm for this problem. As the proofs are very similar, we will just state the results and highlight the differences. The main difference is that the convex program has unbounded integrality gap even in the case when $c_e = r_e = 1$ for all $e \in E$, and as a consequence we cannot turn the bicriteria approximation algorithm into a true approximation algorithm as in the s-t effective resistance network design problem. Using the same technique as in Theorem 201, however, we can return an 8-approximation to the cost without violating the effective resistance constraint, if we are allowed to buy up to four copies of the same edge (see Theorem 211).

Convex Programming Relaxation

We consider the following natural convex programming relaxation for the dual problem.

$$\begin{aligned} \min_{x \in \mathbb{R}^m} \quad & \sum_{e \in E} c_e \cdot x_e \\ \text{subject to} \quad & \mathrm{Reff}_x(s,t) = b_{st}^T L_x^\dagger b_{st} \le R, \\ & 0 \le x_e \le 1 \qquad \forall e \in E. \end{aligned} \tag{DCP}$$

Integrality Gap Examples

As in the s-t effective resistance network design problem, there are simple examples showing that the integrality gap of (DCP) is unbounded, when the costs are arbitrary or when the resistances are arbitrary. So we will make the assumptions that $c_e = r_e = 1$ for all edges $e \in E$.

Unlike the s-t effective resistance network design problem, even in the special case when $c_e = r_e = 1$ for all edges $e \in E$, the convex program (DCP) still has unbounded integrality gap. Consider the following example in Figure 37.3, where the top path has length $n-1$ with each edge of resistance 1, and the bottom path has only one edge with resistance 1. All edges have cost 1. The target effective resistance is $R = (n-1)^2/((n-1)^2 + \epsilon)$ for some constant $\epsilon > 0$. Since $R < 1$, to satisfy the effective resistance constraint, any integral solution must contain both paths and thus has cost n. However, the fractional solution can set $x_e = \epsilon/(n-1)$ for each edge in the top path and set $x_e = 1$ for the bottom edge. It can be checked that this fractional solution satisfies the constraint, and the total cost is $1+\epsilon$. Therefore, the integrality gap of this example is $\Omega(n)$.

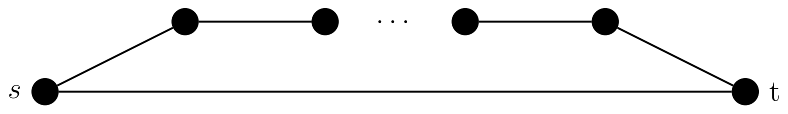

Figure 37.3: Integrality gap example with unit cost and unit resistance.

Optimal Solutions

Although the convex program (DCP) has a large integrality gap, the same rounding technique can be used to obtain a constant factor bicriteria approximation algorithm under the assumption that $c_e = r_e = 1$ for all edges $e \in E$. Exactly the same characterization of the optimality conditions as in the s-t effective resistance network design problem holds, such that any optimal solution satisfies the flow-conductance ratio $\alpha > 0$ as described in Lemma 193.

Analogous to Lemma 194, we can prove an upper bound on α that

$$\alpha^2 \leq \frac{R}{d_{st}}.$$

Analogous to Lemma 200, we can prove a lower bound on any optimal solution x that

$$\text{opt} := \sum_{e \in E} x_e \geq \frac{d_{st}^2}{R}.$$

We can assume that $R < d_{st}$, as otherwise a shortest s-t path is an optimal solution, and so we can assume that $0 < \alpha < 1$.

Rounding Algorithm

The rounding algorithm is exactly the same as in Section 37.3. The same proofs as in Lemma 197 and Lemma 198 will imply that, with probability $\Omega(\alpha)$, the subgraph H returned by the randomized path rounding algorithm satisfies

$$|E(H)| \leq 2 \sum_{e \in E} x_e^* \quad \text{and} \quad \text{Reff}_H(s,t) \leq 4\text{Reff}_{x^*}(s,t),$$

where x^* is an optimal solution to (DCP) and so $|E(H)| \leq 2\text{opt}$. The same lower bound on $\alpha = \Omega(1/m)$ as described in Theorem 201 applies, and so this is a randomized polynomial time algorithm.

An Alternative Bicriteria Approximation Algorithm

In the s-t effective resistance network design problem, we turn a bicriteria approximation algorithm into a true approximation algorithm, by scaling down the budget k by a factor of two and run the bicriteria approximation algorithm. For the proof, we argue that $\text{opt}(k/2) \leq 2 \cdot \text{opt}(k)$ by scaling down an optimal solution x^* with budget k to a solution $x^*/2$ with budget $k/2$.

In the dual problem, we can also try a similar approach, by scaling down the target effective resistance R by a factor 4 and run the bicriteria approximation algorithm. However, we cannot argue that $\text{opt}(R/4) \leq 4 \cdot \text{opt}(R)$, as an optimal solution x^* with effective resistance R may not be able to scale up to $4x^*$ with effective resistance $R/4$ because of the capacity constraints $0 \leq x_e \leq 1$ for $e \in E$. This approach would work if we are allowed to violate the capacity constraint by a factor of 4.

Theorem 211. *Suppose the input graph $G = (V, E)$ satisfies $c_e = r_e = 1$ for every edge $e \in E$. There is a polynomial time algorithm for the dual problem which returns a multi-subgraph H with $|E(H)| \leq 8\mathrm{opt}$ and $\mathrm{Reff}_H(s,t) \leq R$ where there are at most 4 parallel copies of each edge.*

38 Dynamic Programming Algorithms for Series-Parallel Graphs

In this section, we will present the dynamic programming algorithms for solving the s-t effective resistance network design problem on series-parallel graphs. We first review the definitions of series-parallel graphs in Section 38.1. Then, we present the exact algorithm in Theorem 188 when every edge has the same cost in Section 38.2, and the fully polynomial time approximation scheme in Theorem 188 in Section 38.3.

38.1 Series-Parallel Graphs

Definition 212 (two-terminal series-parallel graph)**.** A two-terminal series-parallel graph (SP graph) is a graph with two distinguished vertices (the source vertex s and the target vertex t) that can be constructed recursively as follows:

- Base case: A single edge (s, t)
- Compose step: If G_1 and G_2 are two series parallel graphs with source s_i and target t_i ($i = 1, 2$), then we can combine them in two ways:
  - Series-composition: We identify t_1 with s_2 as the same vertex, the source of the new graph is s_1 and the target is t_2.
  - Parallel-composition: We identify s_1 with s_2 as the same vertex and t_1 with t_2 as the same vertex, the new source is $s_1 = s_2$ and the new target is $t_1 = t_2$.

Given the sequence of steps of constructing a series-parallel graph G, we can define a tree T (SP-tree) as follows.

Definition 213 (SP-tree)**.**

- Leaf node: If G is a single edge, then T is a single node containing the edge.
- Recurse step: G is either a series-composition (S) or a parallel-composition (P) of G_1 and G_2, then T is a S-node (P-node) containing G, and its children are roots of the SP-trees of G_1 and G_2.

For a tree node v in a SP-tree T, let G_v be the subgraph that v represents, s_v, t_v be the two terminals of G_v, and v_l, v_r to be its left and right child if v is an internal node. Note that the SP-tree is a fully binary tree with $2m-1$ nodes.

Given a two-terminal SP graph, the corresponding SP-tree can be computed in $O(n+m)$ time. The linear time SP-graph recognition algorithm in [251] will give us the construction sequence of G, and we can build the SP-tree bottom-up.

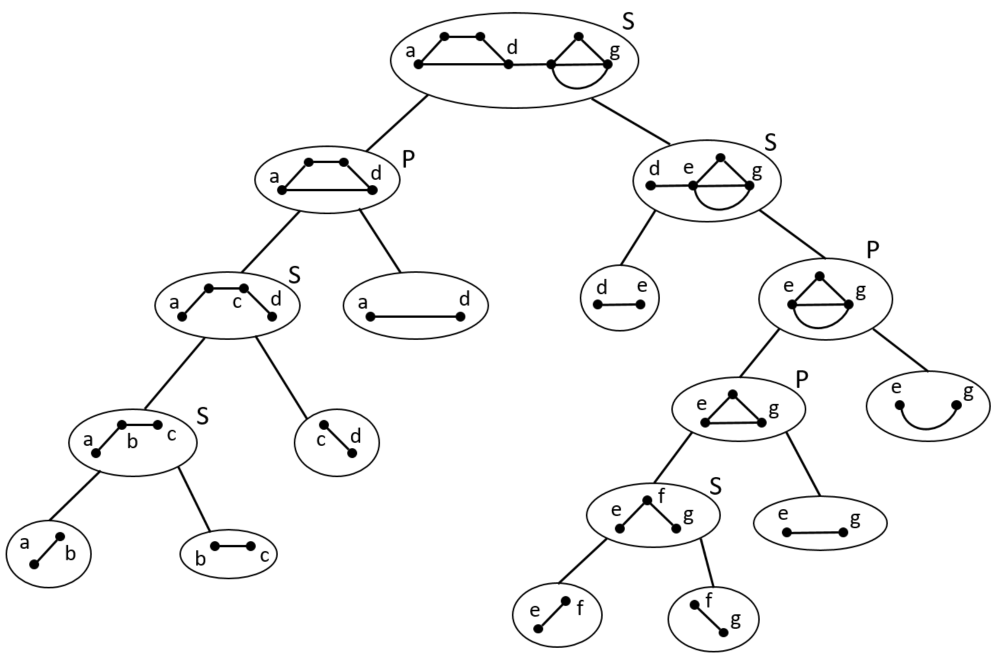

Figure 38.1: An example of a SP-tree.

38.2 Exact Algorithm for Unit-Cost

The following fact shows that the s-t effective resistance can be computed easily from the SP-tree.

Fact 214 (Resistance of series-parallel network)**.** *Let G be a two-terminal SP graph and each edge e has a non-negative resistance r_e. Let T be the corresponding SP-tree. For every tree node v, we can compute the source-target effective resistance as follows.*

Leaf node: $\mathit{Reff}_{G_v}(s_v, t_v) = r_e$ *if v is a leaf node with a single edge e.*

S-node: $\mathit{Reff}_{G_v}(s_v, t_v) = \mathit{Reff}_{G_{v_l}}(s_{v_l}, t_{v_l}) + \mathit{Reff}_{G_{v_r}}(s_{v_r}, t_{v_r})$.

P-node: $\mathit{Reff}_{G_v}(s_v, t_v) = \dfrac{\mathit{Reff}_{G_{v_l}}(s_{v_l}, t_{v_l}) \cdot \mathit{Reff}_{G_{v_r}}(s_{v_r}, t_{v_r})}{\mathit{Reff}_{G_{v_l}}(s_{v_l}, t_{v_l}) + \mathit{Reff}_{G_{v_r}}(s_{v_r}, t_{v_r})}$.

We can design the dynamic programming algorithm by defining the subproblems using the SP-tree T. For every tree node v and $b = 0, 1 \dots k$, we define the subproblem

$$R(v, b) := \min_{H \subseteq G_v} \{\mathrm{Reff}_H(s_v, t_v) \mid \sum_{e \in H} c_e \le b\}.$$

Since we assume that every edge e has cost $c_e = 1$, there are at most $2mk$ subproblems, as the SP-tree has at most $2m$ nodes and there are at most k possibilities for the cost of a subgraph.

It follows from the definition that $R(v_{\text{root}}, k)$ would be the optimal s-t effective resistance for our problem. To compute $R(v, b)$, with Fact 214, we can use the following recurrence which exhausts all possible distributions of the budget among the two children:

$$R(v,b) = \begin{cases} \infty & \text{if } v \text{ is a leaf node and } b < c_e \\ r_e & \text{if } v \text{ is a leaf node and } b \geq c_e \\ \min\limits_{b'=0\ldots b} R(v_l, b') + R(v_r, b-b') & \text{if } v \text{ is a S-node} \\ \min\limits_{b'=0\ldots b} \dfrac{R(v_l,b') \cdot R(v_r, b-b')}{R(v_l,b') + R(v_r,b-b')} & \text{if } v \text{ is a P-node.} \end{cases}$$

As there are $O(mk)$ subproblems and each subproblem can be computed in $O(k)$ time, the time complexity of this dynamic programming algorithm is $O(mk^2)$.

38.3 Fully Polynomial Time Approximation Scheme

In this subsection, we use dynamic programming to design a fully polynomial time approximation scheme to prove Theorem 188. In the previous subsection, we assume that every edge has the same cost to obtain an exact algorithm, by having a bounded number of subproblems in dynamic programming. When the cost could be arbitrary, the number of subproblems can no longer be bounded by a polynomial. Since the cost constraint must be satisfied, we do not change the cost of the edges, but instead we discretize the resistance of the edges and optimize over the cost, and show that it gives an arbitrarily good approximation when the discretization is fine enough.

Rescaling: First, by rescaling, we assume that $\min_e r_e = 1$ and $\max_e r_e = U$ in G. Let $m = |E|$ and $L = \epsilon/m^2$ where $\epsilon > 0$ is the parameter in the approximation guarantee. We further rescale the resistance by setting $r_e \leftarrow r_e/L$. This rescaling ensures that for any subgraph of G in which s-t is connected, the s-t effective resistance is upper bounded by Um/L (when all the edges are in series) and is lower bounded by $1/(mL)$ (when all the edges are in parallel).

Subproblems and Recurrence: Let T be the SP-tree of G and let v_{root} be the root of T. We define two similar sets of subproblems. For every tree node v and a value $R \in [1/(mL), Um/L]$, we define the subproblem

$$C(v,R) := \min_{H \subseteq G_v} \{\sum_{e \in H} c_e \mid \text{Reff}_H(s_v,t_v) \leq R\}.$$

Similar to the reasoning in the previous subsection, the subproblems satisfy the following recurrence relation:

$$C(v,R) = \begin{cases} c_e \quad \text{if } v \text{ is a leaf node with a single edge } e \text{ and } R \geq r_e \\ \infty \quad \text{if } v \text{ is a leaf node with a single edge } e \text{ and } R < r_e \\ \min\limits_{R_1,R_2 \in [1/(mL), Um/L]} \{C(v_l,R_1) + C(v_r,R_2) \mid R_1 + R_2 \leq R\} & \text{if } v \text{ is a S-node} \\ \min\limits_{R_1,R_2 \in [1/(mL), Um/L]} \{C(v_l,R_1) + C(v_r,R_2) \mid \dfrac{R_1 R_2}{R_1+R_2} \leq R\} & \text{if } v \text{ is a P-node.} \end{cases}$$

Discretized subproblems: We cannot use dynamic programming to solve the above recurrence relation efficiently as there are unbounded number of subproblems. Instead, we use dynamic programming to compute the solution of all the "discretized" subproblems using the same recurrence relation. For every integer R from $\lceil 1/(mL) \rceil$ to $\lceil Um/L \rceil$, we define

$$\overline{C}(v,R) := \begin{cases} c_e \quad \text{if } v \text{ is a leaf node with a single edge } e \text{ and } R \geq \lceil r_e \rceil \\ \infty \quad \text{if } v \text{ is a leaf node with a single edge } e \text{ and } R < \lceil r_e \rceil \\ \min\limits_{R_1,R_2} \{\overline{C}(v_l,R_1) + \overline{C}(v_r,R_2) \mid R_1 + R_2 \leq R\} & \text{if } v \text{ is a S-node} \\ \min\limits_{R_1,R_2} \{\overline{C}(v_l,R_1) + \overline{C}(v_r,R_2) \mid \left\lceil \dfrac{R_1R_2}{R_1+R_2} \right\rceil \leq R\} & \text{if } v \text{ is a P-node.} \end{cases}$$

We can think of $\overline{C}(v,R)$ as the minimum cost required to select a subset of edges such that the effective resistance between s_v and t_v is at most R, when the effective resistance is rounded up to an integer during each step of the computation in the recurrence relation.

Algorithm and Complexity: After computing all $\overline{C}(v,R)$, the algorithm will return

$$\min\{R \mid \overline{C}(v_{\text{root}},R) \leq k\}$$

as the approximate minimum s-t effective resistance. Given a tree node v, by trying all possible integral values of R_1 and R_2, we can compute the values of $\overline{C}(v,R)$ for each possible R in $O((Um/L)^2)$ time. Therefore, the total running time of computing all $\overline{C}(v,R)$ is $O(m) \cdot O((Um/L)^2) = O(m^7U^2/\epsilon^2)$. To output the optimal edge set, we can store the optimal values of R_1, R_2 for each pair of (v,R) to reconstruct the edge set.

Correctness and Approximation Guarantee: Since we have not changed the edge cost, the solution returned by the algorithm will have total cost at most k. It remains to show that the s-t effective resistance is at most $(1+\epsilon)$ times the optimal s-t effective resistance. For every tree node v and every $b \in [0,k]$, we define

$$\begin{aligned} R(v,b) &:= \min\{R \mid C(v,R) \leq b, R \in [1/(mL), Um/L]\} \\ \overline{R}(v,b) &:= \min\{R \mid \overline{C}(v,R) \leq b, R \in \{\lceil 1/(mL) \rceil, \dots \lceil Um/L \rceil\}\}. \end{aligned}$$

It follows from the definitions that the optimal s-t effective resistance is $R(v_{\text{root}},k)$, and the output of our algorithm will be $\overline{R}(v_{\text{root}},k)$. The following lemma establishes the approximation guarantee.

Lemma 215. *For every tree node v and for every $b \in [0,k]$, it holds that*

$$\overline{R}(v,b) \leq \left(1 + \frac{\epsilon|E(G_v)|}{m}\right) R(v,b).$$

Proof. We prove the lemma by induction on the tree node of the SP-tree.

Base Case: Suppose v is a leaf node of T and G_v is a graph of a single edge e.

- For $b < c_e$, we have $\overline{R}(v,b) = R(v,b) = \infty$.
- For $b \geq c_e$, we have $R(v,b) = r_e$ and

  $$\overline{R}(v,b) = \lceil r_e \rceil \leq r_e + 1 = r_e + (\frac{\epsilon}{m})(\frac{1}{mL}) \leq r_e + \frac{\epsilon}{m} r_e = \left(1 + \frac{\epsilon|E(G_v)|}{m}\right) R(v,b),$$

  where the second inequality uses that every resistance is at least $1/(mL)$, and the last equality uses that $|E(G_v)| = 1$ and $r_e = R(v,b)$.

S-node: Suppose v is a S-node. For every $b \in [0, k]$, we have

$$\begin{aligned}
\overline{R}(v,b) &= \min_{b_1,b_2|b_1+b_2=b}\{\overline{R}(v_l,b_1)+\overline{R}(v_r,b_2)\} \\
&\le \min_{b_1,b_2|b_1+b_2=b}\left\{\left(1+\frac{\epsilon|E(G_{v_l})|}{m}\right)R(v_l,b_1)+\left(1+\frac{\epsilon|E(G_{v_r})|}{m}\right)R(v_r,b_2)\right\} \\
&\le \min_{b_1,b_2|b_1+b_2=b}\left\{\left(1+\frac{\epsilon|E(G_v)|}{m}\right)(R(v_l,b_1)+R(v_r,b_2))\right\} \\
&= \left(1+\frac{\epsilon|E(G_v)|}{m}\right)\min_{b_1,b_2|b_1+b_2=b}\{(R(v_l,b_1)+R(v_r,b_2))\} \\
&= \left(1+\frac{\epsilon|E(G_v)|}{m}\right)R(v,b),
\end{aligned}$$

where the first inequality follows from the induction hypothesis, and the second inequality follows from the fact that $\max(|E(G_{v_l})|, |E(G_{v_r})|) \le |E(G_v)| - 1$.

P-node: Suppose v is a P-node. For every $b \in [0, B]$, we have

$$\begin{aligned}
\overline{R}(v,b) &= \min_{b_1,b_2|b_1+b_2=b}\left\{\left\lceil\frac{1}{1/\overline{R}(v_l,b_1)+1/\overline{R}(v_r,b_2)}\right\rceil\right\} \\
&\le \min_{b_1,b_2|b_1+b_2=b}\left\{\left\lceil\left(1+\frac{\epsilon(|E(G_v)|-1)}{m}\right)\frac{1}{1/R(v_l,b_1)+1/R(v_r,b_2)}\right\rceil\right\} \\
&= \left\lceil\left(1+\frac{\epsilon(|E(G_v)|-1)}{m}\right)R(v,b)\right\rceil \\
&\le \left(1+\frac{\epsilon(|E(G_v)|-1)}{m}\right)R(v,b)+1 \\
&= \left(1+\frac{\epsilon(|E(G_v)|-1)}{m}\right)R(v,b)+\frac{\epsilon}{m}\frac{1}{mL} \\
&\le \left(1+\frac{\epsilon(|E(G_v)|-1)}{m}\right)R(v,b)+\frac{\epsilon}{m}R(v,b) \\
&= \left(1+\frac{\epsilon|E(G_v)|}{m}\right)R(v,b),
\end{aligned}$$

where the first inequality follows from the induction hypothesis and the fact that $\max(|E(G_{v_l})|, |E(G_{v_r})|) \le |E(G_v)| - 1$, and the last inequality holds as the minimum resistance of any subgraph is at least $1/(mL)$.

Therefore, the lemma follows by an induction on the SP-tree. □

By substituting $v = v_{\text{root}}$ and $b = k$, we have

$$\overline{R}(v_{\text{root}},k) \le \left(1+\frac{m\epsilon}{m}\right)R(v_{\text{root}},k) = (1+\epsilon)R(v_{\text{root}},k),$$

and this completes the proof of Theorem 188.

39 Hardness Results

In this subsection, we present two hardness results for the s-t effective resistance network design problem. First, we prove that the problem is NP-hard even if every edge has the same cost and the same resistance in Section 39.1. Then, we prove that the problem is APX-hard assuming the small-set expansion conjecture in Section 39.2.

39.1 NP-Hardness for Unit-Cost Unit-Resistance

We will prove Theorem 184 in this subsection. The following is the decision version of the problem.

Problem 216 (s-t effective resistance network design with unit-cost unit-resistance)**.**

> **Input:** *An undirected graph $G = (V, E)$ where each edge $e \in E$ has resistance one, two vertices $s, t \in V$, and two parameters k and R.*
>
> **Question:** *Does there exist a subgraph H of G with at most k edges and $Reff_H(s,t) \leq R$?*

We will show that this problem is NP-complete by a reduction from the 3-Dimensional Matching (3DM) problem.

Problem 217 (3-Dimensional Matching)**.**

> **Input:** *Three disjoint sets of elements $X = \{x_1, \ldots, x_q\}, Y = \{y_1, \ldots, y_q\}$ and $Z = \{z_1, \ldots, z_q\}$, a set of triples $\mathcal{T} \subseteq X \times Y \times Z$ where each triple contains exactly one element in X, Y and Z.*
>
> **Question:** *Does there exist a subset of q pairwise disjoint triples in $\mathcal{T}$?*

Reduction: Given an instance of 3DM with $\{(X, Y, Z), \mathcal{T}\}$, we let $\tau = |\mathcal{T}|$ and denote the triples by $\mathcal{T} = \{T_1, \ldots, T_\tau\}$.

We construct a graph $G = (V, E)$ as follows.

> **Vertex Set:** The vertex set V of the graph G is the disjoint union of five sets $\{s\}, \{t\}, V_A$, V_B, and D. Each vertex in V_A corresponds to a triple in $\mathcal{T}$, that is $V_A = \{T_1, \ldots, T_\tau\}$. Each vertex in V_B corresponds to an element in $X \cup Y \cup Z$, that is $V_B = \{x_1, \ldots, x_q, y_1, \ldots, y_q, z_1, \ldots, z_q\}$. Let $l = 3\tau + 3q$. The set D consists of $\tau \cdot l$ "dummy" vertices $\{d_{i,j} \mid 1 \leq i \leq \tau, 1 \leq j \leq l\}$. So, there are totally $\tau + 3q + 2 + \tau(3\tau + 3q)$ vertices in G, which is a polynomial in the input size of the 3DM instance.
>
> **Edge Set:** The edge set E of the graph G is the disjoint union of three edge sets F_1, F_2 and P. There are 3τ edges in F_1, where there are three edges (T, x_a), (T, y_b) and (T, z_c) for each triple $T = (x_a, y_b, z_c) \in \mathcal{T}$. There are $3q$ edges in F_2, where there is an edge from each vertex in V_B to t. There are $\tau(l+1)$ edges in P, where there is a path $P_i := (s, d_{i,1}, d_{i,2}, \ldots, d_{i,l}, T_i)$ for each triple $T_i \in \mathcal{T}$ for $1 \leq i \leq \tau$. So, there are totally $3\tau + 3q + \tau(3\tau + 3q + 1)$ edges in E, which is a polynomial in the input size of the 3DM instance.

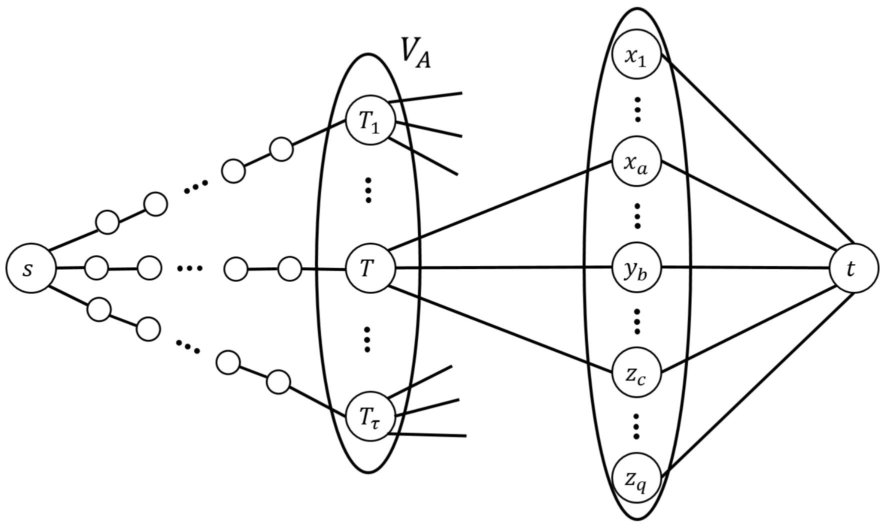

Figure 39.1: An illustration of the construction of the graph G from a 3DM instance.

The proof of the following claim completes the proof of Theorem 184.

Lemma 218. *Let $k = q(l+1) + 3\tau + 3q$ and $R = (3(l+1)+2)/3q$. The 3DM instance has q disjoint triples if and only if the graph G has a subgraph H with at most k edges and $\mathrm{Reff}_H(s,t) \le R$.*

Proof. One direction is easy. If there are q disjoint triples in the 3DM instance, say $\{T_1, \dots, T_q\}$, then H will consist of the q paths $P_1, \dots, P_q$, the $3q$ edges in F_1 incident on $T_1, \dots, T_q$, and all the $3q$ edges in F_2. There are $(l+1)q + 3q + 3q \le k$ edges in H, and $\mathrm{Reff}_H(s,t) = (l+1)/q + 1/3q + 1/3q = (3(l+1)+2)/3q = R$, as in the graph in Figure 39.2.

The other direction is more interesting. If there do not exist q disjoint triples in the 3DM instance, then we need to argue that $\mathrm{Reff}_H(s,t) > R$ for any H with at most k edges. First, note that $k < (q+1)(l+1)$, and so the budget is not enough for us to buy more than q paths. As it is useless to buy only a proper subset of a path, we can thus assume that H consists of q paths and all the edges in F_1 and all the edges in F_2, a total of exactly $q(l+1) + 3\tau + 3q = k$ edges. For any such H, we will argue that $\mathrm{Reff}_H(s,t) > R$. Without loss of generality, we assume that H consists of $P_1, \dots, P_q$ and all edges in F_1 and F_2. As $T_1, \dots, T_q$ are not disjoint, there are some vertices in V_B that are not neighbors of $T_1 \cup \ldots \cup T_q$, call those vertices U.

We consider the following modifications of H to obtain H', and use $\mathrm{Reff}_{H'}(s,t)$ to lower bound $\mathrm{Reff}_H(s,t)$. For every pair of vertices in V_B, we add an edge of zero resistance. For each edge incident on $T_{q+1}, \dots, T_\tau$, we decrease its resistance to zero. By the monotonicity principle, the modifications will not increase the s-t effective resistance, as we either add edges with zero resistance or decreasing the resistance of existing edges. The modifications are equivalent to contracting the vertices with zero resistance edges in between, and so H' is equivalent to the graph in Figure 39.2. Therefore, we have $\mathrm{Reff}_H(s,t) \ge \mathrm{Reff}_{H'}(s,t) \ge R$.

We will prove that one of the inequalities in $\mathrm{Reff}_H(s,t) \ge \mathrm{Reff}_{H'}(s,t) \ge R$ must be strict when $U \ne \varnothing$ (Figure 39.3). To argue the strict inequality, we look at the unit s-t electrical flow f in H and consider two cases.

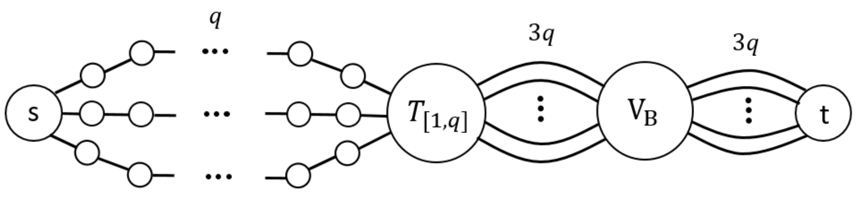

Figure 39.2: The subgraph H when the 3DM instance has q disjoint triples.

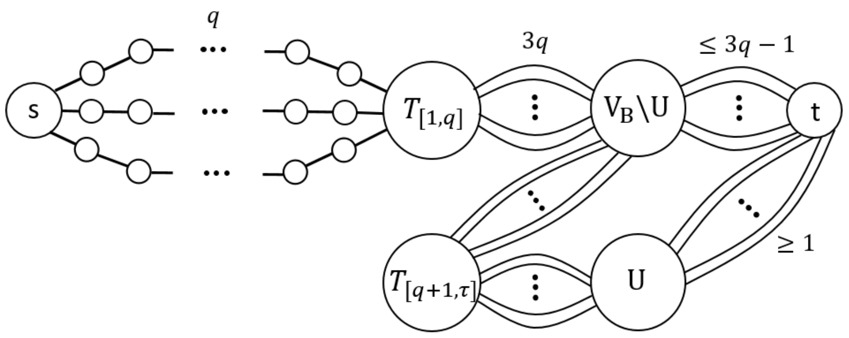

Figure 39.3: The subgraph H when U is non-empty.

- If there exists some vertex $u \in U$ with no incoming electrical flow, then we can delete such a vertex without changing $\mathrm{Reff}_H(s,t)$. But then in the modified graph H', the number of parallel edges to t is strictly smaller than $3q$, and therefore $\mathrm{Reff}_{H'}(s,t) > R$.

- If there exists some vertex $u \in U$ with some incoming electrical flow, then $f(T_j u) > 0$ for some $j \geq q+1$. Since we have decrease the resistance of such an edge $T_j u$ to zero, the energy of f in H' is strictly smaller than the energy of f in H. By Thomson's principle, we have $\mathrm{Reff}_{H'}(s,t) \leq \mathcal{E}_{H'}(f) < \mathcal{E}_H(f) = \mathrm{Reff}_H(s,t)$.

Since the 3DM instance has no q disjoint triples, it follows that $U \neq \varnothing$ and thus one of the above two cases must apply. In either case, we have $\mathrm{Reff}_H(s,t) > R$ and this completes the proof of the other direction. □

39.2 Improved Hardness Assuming Small-Set Expansion Conjecture

In this subsection, we will prove Theorem 185 that it is NP-hard to approximate the s-t effective resistance network design problem within a factor smaller than 2. First, we will state the small-set expansion conjecture and its variant on bipartite graphs, and present an overview of the proof in Section 39.2.1. Then, we will reduce the bipartite small-set expansion problem to the s-t effective resistance network design problem in Section 39.2.2, and then reduce the small-set expansion problem to the bipartite small-set expansion problem in Section 39.2.3 to complete the proof.

39.2.1 The Small-Set Expansion Conjecture and Proof Overview

The gap small-set expansion problem is formulated by Raghavendra and Steurer [224]. We use the version that is stated in [225].

Definition 219 (Gap Small-Set Expansion Problem [224, 225])**.** Given an undirected graph $G = (V, E)$, two parameters $0 < \beta < \alpha < 1$ and $\delta > 0$, the (α, β)-gap δ-small-set expansion problem, denoted by $\mathrm{SSE}_\delta(\alpha, \beta)$, is to distinguish between the following two cases.

- YES: There exists a subset $S \subseteq V$ with $\mathrm{vol}(S) = \delta|E|$ and $\phi(S) \leq \beta$.
- NO: Every subset $S \subseteq V$ with $\mathrm{vol}(S) = \delta|E|$ has $\phi(S) \geq \alpha$.

It is conjectured in [224] that the gap small-set expansion problem becomes harder when δ becomes smaller.

Conjecture 220 (Small-Set Expansion Conjecture [224, 225])**.** *For any $\epsilon \in (0, \frac{1}{2})$, there exists sufficiently small $\delta > 0$ such that $SSE_\delta(1-\epsilon, \epsilon)$ is NP-hard even for regular graphs.*

It is known that the small-set expansion conjecture implies the Unique Game conjecture [224] and is equivalent to some variant of the Unique Game Conjecture [225].

We will show the SSE-hardness of the s-t effective resistance network design problem in two steps, and use the small-set expansion problem on regular *bipartite* graphs as an intermediate problem.

Proposition 221. *For any $\epsilon > 0$, there is a polynomial time reduction from $SSE_\delta(1-\epsilon, \epsilon)$ on d-regular graphs to $SSE_\delta(1-8\epsilon, \epsilon)$ on d-regular bipartite graphs.*

Proposition 222. *Given an instance of $SSE_\delta(\alpha, \beta)$ on a d-regular bipartite graph B, there is a polynomial time algorithm to construct an instance of the s-t effective resistance network design problem with graph G and cost budget k satisfying the following properties.*

- *If B is a* YES*-instance, then there is a subgraph H of G with cost at most k and*
$$\mathit{Reff}_H(s,t) \leq \frac{2}{(1-\beta)dk}.$$
- *if B is a* NO*-instance, then every subgraph H of G with cost at most k has*
$$\mathit{Reff}_H(s,t) \geq \frac{2}{(1-\frac{\alpha}{2})dk}.$$

Theorem 185 will follow immediately from the two propositions.

Theorem 223. *For any $\epsilon' > 0$, it is NP-hard to approximate the s-t effective resistance network design problem to within a factor of $2-\epsilon'$, assuming that $SSE_\delta(1-\epsilon, \epsilon)$ is NP-hard on regular graphs for sufficiently small $\epsilon > 0$.*

Proof. First, given a d-regular instance of $\mathrm{SSE}_\delta(1-\epsilon, \epsilon)$, we apply Proposition 221 to obtain a d-regular bipartite instance of $\mathrm{SSE}_\delta(1-8\epsilon, \epsilon)$. Then, we apply Proposition 222 with $\alpha = 1-8\epsilon$ and $\beta = \epsilon$ and see that the ratio between the s-t effective resistance of the NO-case and the YES-case is at least

$$\frac{(1-\beta)dk}{(1-\frac{\alpha}{2})dk} = \frac{1-\epsilon}{\frac{1}{2}+4\epsilon} = \frac{2(1-\epsilon)}{1+8\epsilon} > 2-\epsilon',$$

when ϵ is sufficiently small. □

We will first prove Proposition 222 in Section 39.2.2, and then prove Proposition 221 in Section 39.2.3.

39.2.2 From Bipartite Small-Set Expansion to s-t Effective Resistance Network Design

We prove Proposition 222 in this subsection. In the YES-case of bipartite SSE, we use the small dense subgraph (from the small low conductance set) to construct a small subgraph with small s-t effective resistance. In the NO-case of bipartite SSE, we argue that every small subgraph has considerably larger s-t effective resistance.

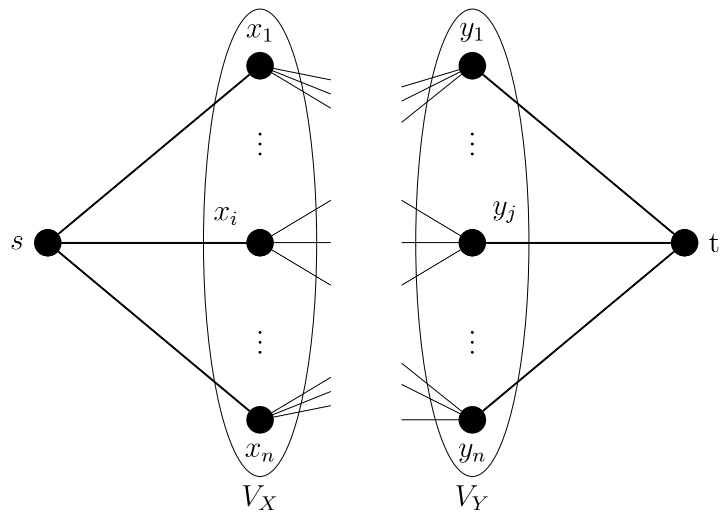

Figure 39.4: Reduction from bipartite small set expansion to s-t effective resistance network design problem.

Construction: Given an $\mathrm{SSE}_\delta(\alpha, \beta)$ instance with a d-regular bipartite graph $B = (V_X, V_Y; E_B)$, we construct an instance of the s-t effective resistance network design problem with graph $G = (V, E)$ as follows. See Figure 39.4 for an illustration.

Vertex Set: The vertex set V of G is simply the disjoint union of $\{s\}, V_X, V_Y, \{t\}$.

Edge Set: The edge set E of G is the disjoint union of three edge sets E_s, E_B, E_t. The edge set E_s has $|V_X|$ edges, where there is an edge from s to each vertex $v \in V_X$. The edge set E_t has $|V_Y|$ edges, where there is an edge from each vertex $v \in V_Y$ to t.

Costs and Resistances: Every edge e in E_B has $c_e = 1$ and $r_e = 0$. Every edge $e \in E_s \cup E_t$ has $c_e = 0$ and $r_e = 1$.

Budget: The cost budget k is equal to $\delta|V_X \cup V_Y|$.

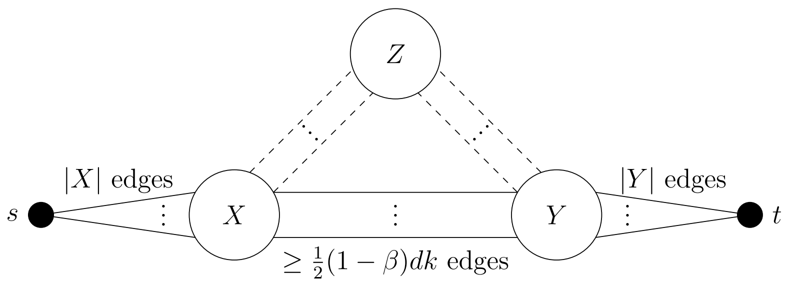

Figure 39.5: In the YES-case, the solid edges are included in H and the dashed edges are deleted.

YES-**case:** Suppose B is a YES-instance of $\mathrm{SSE}_\delta(\alpha, \beta)$. Since B is regular, there exist subsets $X \subseteq V_X$ and $Y \subseteq V_Y$ such that $|X \cup Y| = \delta|V_X \cup V_Y| = k$ and $\phi_B(X \cup Y) \leq \beta$. We construct the subgraph H of G as follows.

Subgraph H: The subgraph H includes all the edges from s to X, all the edges from X to Y, and all the edges from Y to t. Since edges from X to Y are of cost zero, the total cost in H is equal to $|X| + |Y| = k$.

The following claim will complete the proof of the first item of Proposition 222.

Lemma 224. *$Reff_H(s, t) \leq 2/((1-\beta)dk)$.*

Proof. Since B is a d-regular bipartite graph, we have

$$d(|X| + |Y|) = \mathrm{vol}_B(X \cup Y) = |\delta_B(X \cup Y)| + 2|E_B(X, Y)|,$$

where $E_B(X, Y)$ denotes the set of edges with one endpoint in X and one endpoint in Y. Since $\phi_B(X \cup Y) \leq \beta$, we have $|\delta_B(X \cup Y)| \leq \beta \cdot \mathrm{vol}_G(X \cup Y) = d\beta(|X| + |Y|)$. Hence, the number of edges between X and Y is

$$|E_B(X, Y)| = \frac{d(|X| + |Y|) - |\delta_B(X \cup Y)|}{2} \geq \frac{1}{2}(1-\beta)d(|X| + |Y|) = \frac{1}{2}(1-\beta)dk.$$

In terms of s-t effective resistance, H is equivalent to the graph in Figure 39.5, where $Z = (V_X \backslash X) \cup (V_Y \backslash Y)$ is the set of vertices not in X and Y. Since the edges from s to X and from Y to t have zero resistance and edges between X and Y have resistance one, we have $\mathrm{Reff}_H(s, t) \leq 2/((1-\beta)dk)$. □

NO-**case:** We will prove the second item of Proposition 222 by arguing that every subgraph of B with total cost at most k has considerably larger s-t effective resistance. Since all the edges between V_X and V_Y have zero cost and adding edges never increases s-t effective resistance (by Rayleigh's monotonicity principle), we can assume without loss of generality that any solution H to the s-t effective resistance network design problem takes all edges between V_X and V_Y and also takes exactly k edges from $E_s \cup E_t$. Consider an arbitrary subgraph H with the above properties. Let $X \subseteq V_X$ be the set of neighbors of s in H and $Y \subseteq V_Y$ be the set of neighbors of t in H, with $|X| + |Y| = k$. Let $\phi := \phi_B(X \cup Y)$. Note that $\phi \geq \alpha$ as we are in the NO-case where $\phi_B(X \cup Y) \geq \alpha$ for every $|X \cup Y| = k$. Using the same calculation as above, we have

$$|E_B(X,Y)| = \frac{1}{2}(1 - \phi_B(X \cup Y))dk = \frac{1}{2}(1 - \phi)dk.$$

The subgraph H is shown in Figure 39.6, where $Z = (V_X \backslash X) \cup (V_Y \backslash Y)$ is the set of vertices not in X and Y, and the edges within Z are not shown. To lower bound $\mathrm{Reff}_H(s,t)$, we modify H to obtain H' and argue that $\mathrm{Reff}_H(s,t) \geq \mathrm{Reff}_{H'}(s,t)$ and then show a lower bound on $\mathrm{Reff}_{H'}(s,t)$.

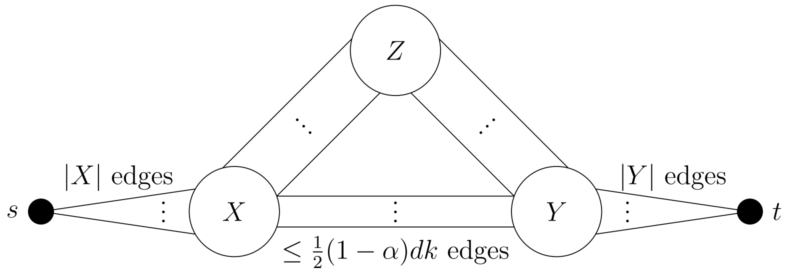

Figure 39.6: The subgraph H' is obtained by identifying the subsets X, Y, Z into single vertices.

To obtain H' from H, we simply identify the three subsets of vertices X, Y, Z to three vertices, which is equivalent to adding a clique of zero resistance edges to each of these three subsets. By Rayleigh's monotonicity principle, this could only decrease the s-t effective resistance and so we have $\mathrm{Reff}_H(s,t) \geq \mathrm{Reff}_{H'}(s,t)$.

In terms of s-t effective resistance, the subgraph H' is equivalent to the graph with two paths between X and Y (with parallel edges): one path P_1 of length one with $|E_B(X,Y)|$ parallel edges between X and Y, another path P_2 of length two with $|E_B(X,Z)|$ parallel edges between X and Z and $|E_B(Z,Y)|$ parallel edges between Z and Y. To lower bound $\mathrm{Reff}_{H'}(s,t)$, we lower bound the resistance of P_1 and P_2, denoted by $r(P_1)$ and $r(P_2)$. Note that

$$r(P_1) = \frac{1}{E_B(X,Y)} = \frac{2}{(1-\phi)dk}.$$

For $r(P_2)$, let $x = |\delta_B(X,Z)|$ and $y = |\delta_B(Y,Z)|$, then

$$r(P_2) = \frac{1}{x} + \frac{1}{y} = \frac{1}{x+y} \cdot \frac{(x+y)^2}{xy} = \frac{1}{x+y} \cdot \left(\frac{x}{y} + \frac{y}{x} + 2\right) \geq \frac{4}{x+y} = \frac{4}{\phi dk},$$

where the inequality holds since $a + 1/a \geq 2$ for any $a > 0$, and the last equality holds because $x + y = |\delta_B(X \cup Y)| = \phi dk$. Finally, by Fact 214,

$$\mathrm{Reff}_H(s,t) \geq \mathrm{Reff}_{H'}(s,t) = \frac{1}{1/r(P_1) + 1/r(P_2)} \geq \frac{1}{\frac{1}{2}(1-\phi)dk + \frac{1}{4}\phi dk}$$
$$= \frac{2}{(1-\phi/2)dk} \geq \frac{2}{(1-\alpha/2)dk},$$

where the last inequality is because we are in the No-case. This completes the proof of the second item of Proposition 222.

Remark 225. In this subsection, we show the hardness of the s-t effective resistance network design problem, when the edge cost and the edge resistance could be arbitrary. Using a similar argument as in the proof of Theorem 184, the reduction can be modified to the unit-cost case if we replace the edges from s to V_X and V_Y to t by sufficiently long paths (so that the cost of connecting s to a vertex in V_X is much larger than the cost of connecting a vertex in V_X to a vertex in V_Y). Therefore, the same $(2-\epsilon)$-SSE-hardness also holds in the case when every edge has the same cost.

39.2.3 From Small Set Expansion to Bipartite Small Set Expansion

We prove Proposition 221 in this subsection.

Construction: Given an instance $\mathrm{SSE}_\delta(1-\epsilon,\epsilon)$ on a d-regular graph $G = (V,E)$, we construct a d-regular bipartite graph $B = (V_X, V_Y; E_B)$ as follows. For each vertex v in V, we create a vertex $v_X \in V_X$ and a vertex $v_Y \in V_Y$, so that $|V_X| = |V_Y| = |V|$. For each edge $uv \in E$, we add two edges $u_X v_Y$ and $u_Y v_X$ to E_B. It is clear from the construction that B is d-regular.

Correctness: To prove Proposition 221, we will establish the following two claims.

1. **Yes-case:** If there is a set $S \subseteq V$ with $|S| = \delta|V|$ and $\phi_G(S) \leq \epsilon$ in G, then there exist $X \subseteq V_X$ and $Y \subseteq V_Y$ with $|X| + |Y| = \delta(|V_X| + |V_Y|)$ and $\phi_B(X \cup Y) \leq \epsilon$ in B.

2. **No-case:** If every set $S \subseteq V$ with $|S| = \delta|V|$ has $\phi_G(S) \geq 1-\epsilon$ in G, then every sets $X \subseteq V_X$ and $Y \subseteq V_Y$ with $|X| + |Y| = \delta(|V_X| + |V_Y|)$ has $\phi_B(X \cup Y) \geq 1 - 8\epsilon$ in B.

Yes-case: Let $S \subseteq V$ be a subset with $|S| = \delta|V|$ and $\phi_G(S) \leq \epsilon$ in G. Let $S_X := \{v_X \mid v \in S\}$ and $S_Y := \{v_Y \mid v \in S\}$, with $|S| = |S_X| = |S_Y|$. By construction, an edge $uv \in \delta_G(S)$ if and only if both $u_X v_Y$ and $v_X u_Y$ are in $\delta_B(S_X \cup S_Y)$, ans thus $|\delta_B(S_X \cup S_Y)| = 2|\delta_G(S)|$. Since $|S_X \cup S_Y| = |S_X| + |S_Y| = 2|S|$ and B is d-regular, we have

$$\phi_B(S_X \cup S_Y) = \frac{|\delta_B(S_X \cup S_Y)|}{\mathrm{vol}_B(S_X \cup S_Y)} = \frac{|\delta_B(S_X \cup S_Y)|}{d(|S_X| + |S_Y|)} = \frac{2|\delta_G(S)|}{2d|S|} = \phi_G(S) \leq \epsilon.$$

No-case: Consider arbitrary subsets $X \subseteq V_X$ and $Y \subseteq V_Y$ with $|X| + |Y| = \delta(|V_X| + |V_Y|) = 2\delta|V|$. To lower bound $\phi_B(X \cup Y)$, we will upper bound $|E_B(X,Y)|$. We partition X into groups $X_1, \ldots, X_a$ where every group except the last group is of size $\delta|V|/2$ and the last group is of size at most $\delta|V|/2$. We partition Y into groups $Y_1, \ldots, Y_b$ in a similar way. The following claim uses the small-set expansion property in G to show that there is no small dense subset in B.

Lemma 226. *Suppose G is a* NO*-instance of* $\text{SSE}_\delta(1-\epsilon,\epsilon)$*. Then, for any $1 \le i \le a$ and $1 \le j \le b$,*

$$|E_B(X_i, Y_j)| \le \epsilon\delta d|V|.$$

Proof. We first argue that there is no small dense subset in G, and then we will use it to bound $|E_B(X_i, Y_j)|$. Suppose $S \subseteq V$ with $|S| = \delta|V|$. As G is a NO-instance, we know that $\phi_G(S) \ge 1-\epsilon$ and thus $|\delta_G(S)| \ge (1-\epsilon)\text{vol}_G(S) = (1-\epsilon)d|S|$. Since $d|S| = \text{vol}_G(S) = |\delta_G(S)| + 2|E_G(S,S)|$, it follows that $|E_G(S,S)| \le \epsilon d|S|/2 = \epsilon\delta d|V|/2$. Note that this also implies trivially that $|E_G(Z,Z)| \le \epsilon\delta d|V|/2$ for any Z with $|Z| \le \delta|V|$.

Given X_i and Y_j, let $Z := \{v \in G \mid v_X \in X_i \text{ or } v_Y \in Y_j\}$. In words, Z is the set of vertices in G which have at least one copy in $X_i \cup Y_j$ in B. Since each X_i and Y_j is of size at most $\delta|V|/2$, it follows that $|Z| \le \delta|V|$. Also, note that $|E_B(X_i, Y_j)| \le 2|E_G(Z,Z)|$, as each edge in $E_B(X_i, Y_j)$ corresponds to one edge in $E_G(Z,Z)$ while each edge in $E_G(Z,Z)$ is corresponded to at most two edges in $E_B(X_i, Y_j)$. Therefore, we can apply the bound in the previous paragraph to conclude that $|E(X_i, Y_j)| \le 2|E_G(Z,Z)| \le \epsilon\delta d|V|$. □

We now use the lemma to bound $|E_B(X,Y)|$. Since $|X| + |Y| = 2\delta|V|$, it follows that $a \le 4$ and $b \le 4$, and therefore

$$|E_B(X,Y)| \le \sum_{i=1}^{a}\sum_{j=1}^{b} |E_B(X_i, Y_j)| \le ab\epsilon\delta d|V| \le 16\epsilon\delta d|V|.$$

As B is bipartite,

$$|\delta_B(X \cup Y)| = \text{vol}_B(X \cup Y) - 2|E_B(X,Y)| \ge 2\delta d|V| - 16\epsilon\delta d|V| = 2(1-8\epsilon)\delta d|V|.$$

Therefore, we have

$$\phi_B(X \cup Y) = \frac{|\delta_B(X \cup Y)|}{\text{vol}_B(X \cup Y)} \ge \frac{2(1-8\epsilon)\delta d|V|}{2\delta d|V|} = 1 - 8\epsilon.$$

This completes the proof of Proposition 221. We remark that a more careful argument gives $|E_B(X,Y)| \le 6\epsilon\delta d|V|$ and thus $\phi_B(X \cup Y) \ge 1 - 3\epsilon$, but this constant does not matter for the proof of Theorem 223.

Concluding Remarks

We have formulated a natural problem and presented some hardness and algorithmic results. It opens up a number of interesting problems.

1. For the s-t effective resistance network design problem when $c_e = r_e = 1$ for all $e \in E$, we conjecture that the integrality gap of the convex program is exactly two. As mentioned in Remark 202, the analysis of the 8-approximation is not tight, and we can show that the same algorithm achieves an approximation ratio strictly smaller than 5. It would be very good to close the gap completely.

2. The general case of arbitrary costs and arbitrary resistances is wide open. It will be very interesting if there are stronger convex programming relaxations for the problem (perhaps adding some knapsack constraint as suggested by the dynamic programming algorithms for series-parallel graphs).

3. As in survivable network design, one could study the general problem when there are multiple source-sink pairs and each pair has a different effective resistance requirement. It will be very interesting if it is still possible to achieve a constant factor approximation in this very general setting.

4. An interesting intermediate problem is to find a minimum cost network so that the maximum effective resistance over pairs (the resistance diameter) is minimized. This is an analog of the global connectivity problem in traditional network design.

A more open-ended question is to unify and extend the techniques for network design problems with spectral requirements.

40 References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[2] Ajit Agrawal, Philip Klein, and R Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[3] Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.

[4] Zeyuan Allen-Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive sdp solver. *arXiv preprint arXiv:1507.02259*, 2015.

[5] Zeyuan Allen-Zhu, Yuanzhi Li, Aarti Singh, and Yining Wang. Near-optimal discrete optimization for experimental design: A regret minimization approach. *arXiv preprint arXiv:1711.05174*, 2017.

[6] Nima Anari and Shayan Oveis Gharan. Effective-resistance-reducing flows, spectrally thin trees, and asymmetric tsp. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 20–39. IEEE, 2015.

[7] Kurt M Anstreicher. Large step volumetric potential reduction algorithms for linear programming. *Annals of Operations Research*, 62(1):521–538, 1996.

[8] Kurt M Anstreicher. On vaidya's volumetric cutting plane method for convex programming. *Mathematics of Operations Research*, 22(1):63–89, 1997.

[9] Kurt M Anstreicher. Towards a practical volumetric cutting plane method for convex programming. *SIAM Journal on Optimization*, 9(1):190–206, 1998.

[10] Kurt M Anstreicher. The volumetric barrier for semidefinite programming. *Mathematics of Operations Research*, 25(3):365–380, 2000.

[11] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 339–348. IEEE, 2005.

[12] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 227–236. ACM, 2007.

[13] David S Atkinson and Pravin M Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69(1-3):1–43, 1995.

[14] Larry Ausubel and Paul Milgrom. Ascending auctions with package bidding. *Frontiers of Theoretical Economics*, 1(1), 2002.

[15] Lawrence M Ausubel. An efficient dynamic auction for heterogeneous commodities. *The American economic review*, pages 602–629, 2006.

[16] Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 2013.

[17] Francis Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2–3):145–373, 2013.

[18] Francisco Barahona and William H Cunningham. A submodular network simplex method. In *Mathematical Programming at Oberwolfach II*, pages 9–31. Springer, 1984.

[19] Siddharth Barman. Approximating Nash equilibria and dense bipartite subgraphs via an approximate version of Caratheodory's theorem. *ACM Symp. on Theory of Computing (STOC)*, 2015.

[20] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *Journal of the ACM (JACM)*, 51(4):540–556, 2004.

[21] Sushil Bikhchandani and John W. Mamer. Competitive equilibrium in an exchange economy with indivisibilities. *Journal of Economic Theory*, 74(2):385–413, June 1997.

[22] Jeff Bilmes. Submodularity in machine learning applications. Twenty-Ninth Conference on Artificial Intelligence, AAAI-15 Tutorial Forum, January 2015.

[23] Robert G Bland, Donald Goldfarb, and Michael J Todd. The ellipsoid method: A survey. *Operations research*, 29(6):1039–1091, 1981.

[24] Liad Blumrosen and Noam Nisan. On the computational power of demand queries. *SIAM Journal on Computing*, 39(4):1372–1391, 2009.

[25] Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing markov chain on a graph. *SIAM review*, 46(4):667–689, 2004.

[26] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9):1124 – 1137, 2004.

[27] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization viagraph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(11):1222 – 1239, 2001.

[28] Carl Brezovec, Gerard Cornuéjols, and Fred Glover. Two algorithms for weighted matroid intersection. *Mathematical Programming*, 36(1):39–53, 1986.

[29] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

[30] Sébastien Bubeck and Yin-Tat Lee. Black-box optimization with a politician. *arXiv preprint arXiv:1602.04847*, 2016.

[31] Sébastien Bubeck, Yin Tat Lee, and Mohit Singh. A geometric alternative to nesterov's accelerated gradient descent. *arXiv preprint arXiv:1506.08187*, 2015.

[32] Niv Buchbinder, Moran Feldman, Seffi Naor, and Roy Schwartz. A tight linear time 1/2-approximation for unconstrained submodular maximization. *SIAM J. Comput.*, 44(5):1384 – 1402, 2015.

[33] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Optimal multi-dimensional mechanism design: Reducing revenue to welfare maximization. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 130–139. IEEE, 2012.

[34] Yang Cai, Constantinos Daskalakis, and S Matthew Weinberg. Reducing revenue to welfare maximization: Approximation algorithms and other generalizations. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 578–595. SIAM, 2013.

[35] Deeparnab Chakrabarty, Prateek Jain, and Pravesh Kothari. Provable submodular minimization via Fujishige-Wolfe algorithm. *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2014.

[36] Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 167–176. ACM, 2008.

[37] Ashok K Chandra, Prabhakar Raghavan, Walter L Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4):312–340, 1996.

[38] Joseph Cheriyan and László A Végh. Approximating minimum-cost k-node connected subgraphs via independence-free graphs. *SIAM Journal on Computing*, 43(4):1342–1362, 2014.

[39] Joseph Cheriyan, Santosh Vempala, and Adrian Vetta. Network design via iterative rounding of setpair relaxations. *Combinatorica*, 26(3):255–275, 2006.

[40] Wang Chi Cheung, Michel X Goemans, and Sam Chiu-wai Wong. Improved algorithms for vertex cover with hard capacities on multigraphs and hypergraphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1714–1726. SIAM, 2014.

[41] Yun Kuen Cheung, Richard Cole, and Nikhil R. Devanur. Tatonnement beyond gross substitutes?: gradient descent to the rescue. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 191–200, 2013.

[42] Gustave Choquet. Theory of capacities. *Annales de l'institut Fourier*, 5:131–295, 1955.

[43] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282. ACM, 2011.

[44] Nam-Kee Chung and Dong-Wan Tcha. A dual algorithm for submodular flow problems. *Operations research letters*, 10(8):489–495, 1991.

[45] Julia Chuzhoy and Sanjeev Khanna. An o (kˆ 3 log n)-approximation algorithm for vertex-connectivity survivable network design. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 437–441. IEEE, 2009.

[46] Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM J. Comput.*, 36(2):498–515, 2006.

[47] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. *CoRR*, abs/1408.5099, 2014.

[48] Vincent Cohen-Addad, Alon Eden, Michal Feldman, and Amos Fiat. The invisible hand of dynamic market pricing. *CoRR*, abs/1511.05646, 2015.

[49] William Cunningham. On submodular function minimization. *Combinatorica*, 5:185 – 192, 1985.

[50] William H Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985.

[51] William H Cunningham. Improved bounds for matroid partition and intersection algorithms. *SIAM Journal on Computing*, 15(4):948–957, 1986.

[52] William H Cunningham and András Frank. A primal-dual algorithm for submodular flows. *Mathematics of Operations Research*, 10(2):251–262, 1985.

[53] Vladimir Danilov, Gleb Koshevoy, and Kazuo Murota. Discrete convexity and equilibria in economies with indivisible goods and money. *Mathematical Social Sciences*, 41(3):251–273, 2001.

[54] Constantinos Daskalakis and S Matthew Weinberg. Bayesian truthful mechanisms for job scheduling from bi-criterion approximation algorithms. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1934–1952. SIAM, 2015.

[55] Sven de Vries, James Schummer, and Rakesh V Vohra. On ascending Vickrey auctions for heterogeneous objects. *Journal of Economic Theory*, 132(1):95–118, 2007.

[56] James Demmel, Ioana Dumitriu, Olga Holtz, and Robert Kleinberg. Fast matrix multiplication is stable. *Numerische Mathematik*, 106(2):199–224, 2007.

[57] EA Dinic. An algorithm for the solution of the max-flow problem with the polynomial estimation. *Doklady Akademii Nauk SSSR*, 194(4):1277–1280, 1970.

[58] Michael Dinitz and Zeyu Zhang. Approximating low-stretch spanners. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 821–840. Society for Industrial and Applied Mathematics, 2016.

[59] Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered pcp and the hardness of hypergraph vertex cover. *SIAM J. Comput.*, 34(5):1129–1146, 2005.

[60] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

[61] Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 750–759. ACM, 1999.

[62] Andreas W.M. Dress and Walter Wenzel. Valuated matroids: a new look at the greedy algorithm. *Applied Mathematics Letters*, 3(2):33 – 35, 1990.

[63] Andreas WM Dress and Walter Wenzel. Valuated matroids. *Advances in Mathematics*, 93(2):214–250, 1992.

[64] A.W.M. Dress and W. Terhalle. Rewarding maps: On greedy optimization of set functions. *Advances in Applied Mathematics*, 16(4):464 – 483, 1995.

[65] A.W.M. Dress and W. Terhalle. Well-layered maps: A class of greedily optimizable set functions. *Applied Mathematics Letters*, 8(5):77 – 80, 1995.

[66] Jack Edmonds. Matroid partition. *Mathematics of the Decision Sciences*, 11:335–345, 1968.

[67] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Edited by G. Goos, J. Hartmanis, and J. van Leeuwen*, page 11, 1970.

[68] Jack Edmonds. Submodular functions, matroids and certain polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87, 1970.

[69] Jack Edmonds. Matroid intersection. *Annals of discrete Mathematics*, 4:39–49, 1979.

[70] Jack Edmonds and Rick Giles. A min-max relation for submodular functions on graphs. *Studies in Integer Programming (PL Hammer, EL Johnson and BH Korte, eds.), Ann. Discrete Math*, 1:185–204, 1977.

[71] Jeremy Elson, Richard M Karp, Christos H Papadimitriou, and Scott Shenker. Global synchronization in sensornets. In *Latin American Symposium on Theoretical Informatics*, pages 609–624. Springer, 2004.

[72] Alina Ene and Huy L. Nguyen. Random coordinate descent methods forminimizing decomposable submodular functions. *Proc, Int. Conf. on Machine Leanring (ICML)*, 2015.

[73] Alina Ene and Ali Vakilian. Improved approximation algorithms for degree-bounded network design problems with node connectivity requirements. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 754–763. ACM, 2014.

[74] Jittat Fakcharoenphol and Bundit Laekhanukit. An $o(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 153–158. ACM, 2008.

[75] Uriel Feige, Vahab Mirrokni, and Jan Vondrak. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133 – 1153, 2011.

[76] Lisa Fleischer and Satoru Iwata. Improved algorithms for submodular function minimization and submodular flow. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 107–116. ACM, 2000.

[77] Lisa Fleischer and Satoru Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics*, 131(2):311–322, 2003.

[78] Lisa Fleischer, Satoru Iwata, and S Thomas McCormick. A faster capacity scaling algorithm for minimum cost submodular flow. *Mathematical Programming*, 92(1):119–139, 2002.

[79] Lisa Fleischer, Kamal Jain, and David P Williamson. An iterative rounding 2-approximation algorithm for the element connectivity problem. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 339–347. IEEE, 2001.

[80] András Frank. A weighted matroid intersection algorithm. *Journal of Algorithms*, 2(4):328–336, 1981.

[81] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[82] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[83] S. Fujishige. Algorithms for solving the independent-flow problems. *Journal of the Operations Research Society of Japan*, 1978.

[84] Satoru Fujishige. Lexicographieally optimal base of a polymatroid with respect to a weight vector. *Math. Oper. Res.*, 5:186–196, 1980.

[85] Satoru Fujishige. An out-of-kilter method for submodular flows. *Discrete applied mathematics*, 17(1):3–16, 1987.

[86] Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.

[87] Satoru Fujishige, Takumi Hayashi, and Shigueo Isotani. The minimum-norm-point algorithm applied to submodular function minimization and linear programming. Publications of the Research Institute for Mathematical Sciences (RIMS), Kyoto, 2006.

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

[89] Satoru Fujishige and Satoru Iwata. Algorithms for submodular flows. *IEICE TRANSACTIONS on Information and Systems*, 83(3):322–329, 2000.

[90] Satoru Fujishige, Hans Röck, and Uwe Zimmermann. A strongly polynomial algorithm for minimum cost submodular flow problems. *Mathematics of Operations Research*, 14(1):60–69, 1989.

[91] Satoru Fujishige and Zhang Xiaodong. An efficient cost scaling algorithm for the independent assignment problem. *Journal of the Operations Research Society of Japan*, 38(1):124–136, 1995.

[92] Satoru Fujishige and Zaifu Yang. A note on Kelso and Crawford's gross substitutes condition. *Math. Oper. Res.*, 28(3):463–469, July 2003.

[93] Mituhiro Fukuda, Masakazu Kojima, Kazuo Murota, and Kazuhide Nakata. Exploiting sparsity in semidefinite programming via matrix completion i: General framework. *SIAM Journal on Optimization*, 11(3):647–674, 2001.

[94] Takuro Fukunaga, Zeev Nutov, and R Ravi. Iterative rounding approximation algorithms for degree-bounded node-connectivity network design. *SIAM Journal on Computing*, 44(5):1202–1229, 2015.

[95] Harold N Gabow. On the l-inf-norm of extreme points for crossing supermodular directed network lps. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 392–406. Springer, 2005.

[96] Harold N Gabow, Michel X Goemans, Éva Tardos, and David P Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. *Networks*, 53(4):345–357, 2009.

[97] François Le Gall. Powers of tensors and fast matrix multiplication. *arXiv preprint arXiv:1401.7714*, 2014.

[98] Rajiv Gandhi, Eran Halperin, Samir Khuller, Guy Kortsarz, and Aravind Srinivasan. An improved approximation algorithm for vertex cover with hard capacities. *J. Comput. Syst. Sci.*, 72(1):16–33, 2006.

[99] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding in bipartite graphs. In *FOCS*, pages 323–332, 2002.

[100] Dan Garber and Elad Hazan. A linearly convergent variant of the conditional gradient algorithm under strong convexity, with applications to online and stochastic optimization. *SIAM Journal on Optimization*, 26(3):1493–1528, 2016.

[101] Arpita Ghosh and Stephen Boyd. Growing well-connected graphs. In *Decision and Control, 2006 45th IEEE Conference on*, pages 6605–6611. IEEE, 2006.

[102] Arpita Ghosh, Stephen Boyd, and Amin Saberi. Minimizing effective resistance of a graph. *SIAM review*, 50(1):37–66, 2008.

[103] Michel X Goemans, Andrew V Goldberg, Serge A Plotkin, David B Shmoys, Eva Tardos, and David P Williamson. Improved approximation algorithms for network design problems. In *SODA*, volume 94, pages 223–232, 1994.

[104] Michel X. Goemans and Jose A. Soto. Symmetric submodular function minimization under hereditary family constraints. 27(2):1123 – 1145, 2013.

[105] Michel X Goemans and David P Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[106] Jean-Louis Goffin, Zhi-Quan Luo, and Yinyu Ye. Complexity analysis of an interior cutting plane method for convex feasibility problems. *SIAM Journal on Optimization*, 6(3):638–652, 1996.

[107] Jean-Louis Goffin and Jean-Philippe Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. *Mathematical Programming*, 84(1):89–103, 1999.

[108] Jean-Louis Goffin and Jean-Philippe Vial. Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17(5):805–867, 2002.

[109] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[110] Andrew V Goldberg and Robert E Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[111] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[112] Martin Grotschel, Laszlo Lovasz, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169 – 197, 1981.

[113] Martin Grötschel, László Lovász, and Alexander Schrijver. Geometric algorithms and combinatorial optimization. Springer, 1988.

[114] Sudipto Guha, Refael Hassin, Samir Khuller, and Einat Or. Capacitated vertex covering with applications. In *SODA*, pages 858–865, 2002.

[115] Faruk Gul and Ennio Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87(1):95–124, July 1999.

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

[117] John William Hatfield and Scott Duke Kominers. Hidden substitutes. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC '15, Portland, OR, USA, June 15-19, 2015*, page 37, 2015.

[118] John William Hatfield, Scott Duke Kominers, Alexandru Nichifor, Michael Ostrovsky, and Alexander Westkamp. Full substitutability in trading networks. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation, EC '15, Portland, OR, USA, June 15-19, 2015*, pages 39–40, 2015.

[119] John William Hatfield and Paul R. Milgrom. Matching with Contracts. *American Economic Review*, 95(4):913–935, September 2005.

[120] Elad Hazan and Satyen Kale. Online submodular minimization. *J. Mach. Learn. Res.*, 13:2903 – 2922, 2012.

[121] Christoph Helmberg and Franz Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[122] Justin Hsu, Jamie Morgenstern, Ryan M. Rogers, Aaron Roth, and Rakesh Vohra. Do prices coordinate markets? 2016.

[123] Yoshiko T Ikebe, Yosuke Sekiguchi, Akiyoshi Shioura, and Akihisa Tamura. Stability and competitive equilibria in multi-unit trading networks with discrete concave utility functions. *Japan Journal of Industrial and Applied Mathematics*, 32(2):373–410, 2015.

[124] Satoru Iwata. A capacity scaling algorithm for convex cost submodular flows. *Mathematical programming*, 76(2):299–308, 1997.

[125] Satoru Iwata. A fully combinatorial algorithm for submodular function minimization. *Journal of Combinatorial Theory, Series B*, 84(2):203–212, 2002.

[126] Satoru Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing*, 32(4):833–840, 2003.

[127] Satoru Iwata. Submodular function minimization. *Mathematical Programming*, 112(1):45–64, 2008.

[128] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.

[129] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

[130] Satoru Iwata, S Thomas McCormick, and Maiko Shigeno. A faster algorithm for minimum cost submodular flows. In *SODA*, pages 167–174, 1998.

[131] Satoru Iwata, S Thomas McCormick, and Maiko Shigeno. A strongly polynomial cut canceling algorithm for the submodular flow problem. In *Integer Programming and Combinatorial Optimization*, pages 259–272. Springer, 1999.

[132] Satoru Iwata, S Thomas McCormick, and Maiko Shigeno. A fast cost scaling algorithm for submodular flow. *Information Processing Letters*, 74(3):123–128, 2000.

[133] Satoru Iwata and James B Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237. Society for Industrial and Applied Mathematics, 2009.

[134] Satoru Iwata and James B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *SODA*, pages 1230–1237, 2009.

[135] Rishabh Iyer and Jeff A. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2013.

[136] Rishabh Iyer, Stefanie Jegelka, and Jeff A. Bilmes. Curvature and optimal algorithms for learning and minimizing submodular functions. *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2013.

[137] Rabih A Jabr, Ravindra Singh, and Bikash C Pal. Minimum loss network reconfiguration using mixed-integer convex programming. *IEEE Transactions on Power systems*, 27(2):1106–1115, 2012.

[138] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[139] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[140] Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 463–471. IEEE, 2011.

[141] Klaus Jansen. Approximate strong separation with application in fractional graph coloring and preemptive scheduling. *Theoretical Computer Science*, 302(1):239–256, 2003.

[142] Stefanie Jegelka, Francis Bach, and Suvrit Sra. Reflection methods for user-friendly submodular optimization. *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2013.

[143] Stefanie Jegelka and Jeff Bilmes. Online submodular minimization for combinatorial structures. In *ICML*, pages 345–352, 2011.

[144] Stefanie Jegelka, Hui Lin, and Jeff A. Bilmes. On fast approximate submodular minimization. In *NIPS*, pages 460–468, 2011.

[145] Ravindran Kannan and Hariharan Narayanan. Random walks on polytopes and an affine interior point method for linear programming. *Mathematics of Operations Research*, 37(1):1–20, 2012.

[146] Mong-Jen Kao. Iterative partial rounding for vertex cover with hard capacities. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2638–2653. SIAM, 2017.

[147] Richard M Karp and Christos H Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM Journal on Computing*, 11(4):620–632, 1982.

[148] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *SODA*, pages 217–226. SIAM, 2014.

[149] Jr Kelso, Alexander S and Vincent P Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50(6):1483–1504, November 1982.

[150] Alexander S Kelso Jr and Vincent P Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica: Journal of the Econometric Society*, pages 1483–1504, 1982.

[151] William Thomson Baron Kelvin and Peter Guthrie Tait. *Treatise on natural philosophy*, volume 1. Clarendon Press, 1867.

[152] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[153] LG Khachiyan, SP Tarasov, and II Erlikh. The method of inscribed ellipsoids. In *Soviet Math. Dokl*, volume 37, pages 226–230, 1988.

[154] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[155] Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

[156] Adam R Klivans and Daniel Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 216–223. ACM, 2001.

[157] Pushmeet Kohli, M. Pawan Kumar, and Philip H. S. Torr. P3 and beyond: Move making algorithms for solving higher order functions. *IEEE Trans. Pattern Anal. and Machine Learning*, 31, 2008.

[158] Pushmeet Kohli and Philip H. S. Torr. Dynamic graph cuts and their applications in computer vision. *Computer Vision: Detection, Recognition and Reconstruction.*, 2010.

[159] Fuhito Kojima, Akihisa Tamura, and Makoto Yokoo. Designing matching mechanisms under constraints: An approach from discrete convex analysis. Mpra paper, University Library of Munich, Germany, 2014.

[160] Alexandra Kolla, Yury Makarychev, Amin Saberi, and Shang-Hua Teng. Subgraph sparsification and nearly optimal ultrasparsifiers. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 57–66. ACM, 2010.

[161] Vladimir Kolmogorov. Minimizing a sum of submodular functions. *Discrete Appl. Math.*, 160(15):2246 – 2258, 2012.

[162] Guy Kortsarz, Robert Krauthgamer, and James R Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM Journal on Computing*, 33(3):704–720, 2004.

[163] Mikhail K Kozlov, Sergei P Tarasov, and Leonid G Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Computational Mathematics and Mathematical Physics*, 20(5):223–228, 1980.

[164] Andreas Krause. http://submodularity.org/.

[165] Kartik Krishnan and John E Mitchell. A unifying framework for several cutting plane methods for semidefinite programming. *Optimization methods and software*, 21(1):57–74, 2006.

[166] Kartik Krishnan and John E Mitchell. Properties of a cutting plane method for semidefinite programming. *Pacific Journal of Optimization*, 8(4):779–802, 2012.

[167] Kartik Krishnan and Tamás Terlaky. Interior point and semidefinite approaches in combinatorial optimization. In *Graph theory and combinatorial optimization*, pages 101–157. Springer, 2005.

[168] S. Lacoste-Julien and M. Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2015.

[169] Bundit Laekhanukit. Parameters of two-prover-one-round game and the hardness of connectivity problems. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1626–1643. SIAM, 2014.

[170] Lap Chi Lau, Joseph Naor, Mohammad R Salavatipour, and Mohit Singh. Survivable network design with degree or order constraints. *SIAM Journal on Computing*, 39(3):1062–1087, 2009.

[171] Lap Chi Lau, Ramamoorthi Ravi, and Mohit Singh. *Iterative methods in combinatorial optimization*, volume 46. Cambridge University Press, 2011.

[172] Lap Chi Lau and Hong Zhou. A unified algorithm for degree bounded survivable network design. *Mathematical Programming*, 154(1-2):515–532, 2015.

[173] Eugene L Lawler. Matroid intersection algorithms. *Mathematical programming*, 9(1):31–56, 1975.

[174] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *The 45th ACM Symposium on Theory of Computing (STOC)*, pages 755–764, 2013.

[175] Yin Tat Lee and Aaron Sidford. Path finding ii: An\~ o (m sqrt (n)) algorithm for the minimum cost flow problem. *arXiv preprint arXiv:1312.6713*, 2013.

[176] Yin Tat Lee and Aaron Sidford. Path-finding methods for linear programming : Solving linear programs in õ(sqrt(rank)) iterations and faster algorithms for maximum flow. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, 18-21 October, 2014, Philadelphia, PA, USA*, pages 424–433, 2014.

[177] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. *arXiv preprint arXiv:1503.01752*, 2015.

[178] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2015)*, 2015.

[179] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. *CoRR*, abs/1508.04874, 2015.

[180] Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. *Proceedings, IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

[181] Renato Paes Leme. Gross substitutability: An algorithmic survey. *Games and Economic Behavior*, 106:294–316, 2017.

[182] A. Yu Levin. On an algorithm for the minimization of convex functions. *Soviet Math. Doklady*, 1965.

[183] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. 2012.

[184] Hui Lin and Jeff Bilmes. An application of the submodular principal partition to training data subset selection. *NIPS workshop on Discrete Optimization in Machine Learning*, 2010.

[185] Hui Lin and Jeff Bilmes. Optimal selection of limited vocabulary speech corpora. *Proc. Ann. Conf. Int. Speech Comm. Ass. (INTERSPEECH)*, 2011.

[186] Laszlo Lovasz. Submodular functions and convexity. *Mathematical Programming – The State of the Art.* A. Bachem, M. Grotschel, B. Korte eds.,*Springer*, pages 235 – 257, 1983.

[187] László Lovász and Santosh Vempala. Simulated annealing in convex bodies and an $o^*(n^4)$ volume algorithm. *J. Comput. Syst. Sci.*, 72(2):392–417, 2006.

[188] Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 2019–2036. Society for Industrial and Applied Mathematics, 2015.

[189] Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011.

[190] Peter Matthews. Covering problems for brownian motion on spheres. *The Annals of Probability*, pages 189–199, 1988.

[191] S McCormick. *Submodular Function Minimization.* 2013.

[192] S. T. McCormick. Submodular function minimization. *Chapter 7 in the Handbook of Discrete Optimization*, 2006.

[193] S Thomas and McCormick. Canceling most helpful total submodular cuts for submodular flow. In *IPCO*, pages 343–353, 1993.

[194] Vahab Mirrokni, Renato Paes Leme, Adrian Vladu, and Sam Chiu-Wai Wong. Tight bounds for approximate Caratheodory and beyond. *arXiv,* `http://arxiv.org/abs/1512.08602`, December, 2015.

[195] Renato DC Monteiro. First-and second-order methods for semidefinite programming. *Mathematical Programming*, 97(1-2):209–244, 2003.

[196] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 345–354. ACM, 1987.

[197] K. Murota. *Discrete Convex Analysis.* Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2003.

[198] Kazuo Murota. Convexity and Steinitz's exchange property. *Advances in Mathematics*, 124(2):272 – 311, 1996.

[199] Kazuo Murota. Valuated matroid intersection i: Optimality criteria. *SIAM J. Discrete Math.*, 9(4):545–561, 1996.

[200] Kazuo Murota. Valuated matroid intersection ii: Algorithms. *SIAM J. Discrete Math.*, 9(4):562–576, 1996.

[201] Kazuo Murota. *Matrices and matroids for systems analysis*, volume 20. Springer, 2000.

[202] Kazuo Murota. Discrete convex analysis: A tool for economics and game theory. *Journal of Mechanism and Institution Design*, 1(1):151–273, 2016.

[203] Kazuo Murota and Akiyoshi Shioura. M-convex function on generalized polymatroid. *Mathematics of Operations Research*, 24(1):pp. 95–105, 1999.

[204] Kazuo Murota, Akiyoshi Shioura, and Zaifu Yang. Computing a walrasian equilibrium in iterative auctions with multiple differentiated items. In *International Symposium on Algorithms and Computation*, pages 468–478. Springer, 2013.

[205] Kazuo Murota and Akihisa Tamura. Application of M-convex submodular flow problem to mathematical economics. *Japan Journal of Industrial and Applied Mathematics*, 20(3):257–277, 2003.

[206] Kazuhide Nakata, Katsuki Fujisawa, Mituhiro Fukuda, Masakazu Kojima, and Kazuo Murota. Exploiting sparsity in semidefinite programming via matrix completion ii: Implementation and numerical results. *Mathematical Programming*, 95(2):303–327, 2003.

[207] Arkadi Nemirovski. Efficient methods in convex programming. 1994.

[208] Arkadi Nemirovski. Efficient methods in convex programming. 2005.

[209] D. B. Nemirovsky, A. S., & Yudin. Problem complexity and method efficiency in optimization. 1983.

[210] Yu Nesterov. Complexity estimates of some cutting plane methods based on the analytic barrier. *Mathematical Programming*, 69(1-3):149–176, 1995.

[211] Yu Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume I. 2003.

[212] Yu Nesterov and Arkadi Nemirovskiy. *Self-concordant functions and polynomial-time methods in convex programming.* USSR Academy of Sciences, Central Economic & Mathematic Institute, 1989.

[213] Yu Nesterov and A Nemirovsky. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[214] Aleksandar Nikolov, Mohit Singh, and Uthaipon Tao Tantipongpipat. Proportional volume sampling and approximation algorithms for a-optimal design. *arXiv preprint arXiv:1802.08318*, 2018.

[215] Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting prices. *J. Economic Theory*, 129(1):192–224, 2006.

[216] Robert Nishihara, Stefanie Jegelka, and Michael I. Jordan. On the convergence rate of decomposable submodular function minimization. *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2014.

[217] James B Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.

[218] James B Orlin, John VandeVate, et al. On a" primal" matroid intersection algorithm. 1983.

[219] David C Parkes. i bundle: an efficient ascending price bundle auction. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 148–157. ACM, 1999.

[220] David C Parkes and Lyle H Ungar. An ascending-price generalized Vickrey auction. *manuscript, Harvard University*, 2002.

[221] Kaare Brandt Petersen and Michael Syskind Pedersen. The matrix cookbook (version: November 15, 2012), 2012.

[222] Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[223] Adarsh Prasad, Stefanie Jegelka, and Dhruv Batra. Submodular meets structured: Finding diverse subsets in exponentially-large structured item sets. In *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2014.

[224] Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 755–764. ACM, 2010.

[225] Prasad Raghavendra, David Steurer, and Madhur Tulsiani. Reductions between expansion problems. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 64–73. IEEE, 2012.

[226] Srinivasan Ramaswamy and John E Mitchell. A long step cutting plane algorithm that uses the volumetric barrier. *Department of Mathematical Science, RPI, Troy, NY*, 1995.

[227] Alvin E. Roth. Stability and polarization of interests in job matching. *Econometrica*, 52(1):47–57, 1984.

[228] Barna Saha and Samir Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *ICALP (1)*, pages 762–773, 2012.

[229] Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. *arXiv preprint arXiv:1711.06455*, 2017.

[230] Alexander Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.

[231] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

[232] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Comb. Theory, Ser. B*, 80(2):346–355, 2000.

[233] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[234] Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, pages 124–127, 1950.

[235] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science*, 2013.

[236] Maiko Shigeno and Satoru Iwata. A dual approximation approach to weighted matroid intersection. *Operations research letters*, 18(3):153–156, 1995.

[237] Akiyoshi Shioura and Akihisa Tamura. Gross substitutes condition and discrete concavity for multi-unit valuations: A survey. *Journal of Operations Research Society of Japan*, 58(1):61–103, 2015.

[238] Naum Z Shor. Cut-off method with space extension in convex programming problems. *Cybernetics and systems analysis*, 13(1):94–96, 1977.

[239] Maurice Sion. On general minimax theorems. *Pacific J. Math*, 8(1):171–176, 1958.

[240] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[241] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[242] Peter Stobbe and Andreas Krause. Efficient minimization of decomposable submodular functions. *Adv. in Neu. Inf. Proc. Sys. (NIPS)*, 2010.

[243] Ning Sun and Zaifu Yang. Equilibria and indivisibilities: gross substitutes and complements. *Econometrica*, 74(5):1385–1402, 2006.

[244] Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM J. Comput.*, 40, 2011.

[245] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[246] Michael J Todd. Semidefinite optimization. *Acta Numerica 2001*, 10:515–560, 2001.

[247] Nobuaki Tomizawa and Masao Iri. Algorithm for determining rank of a triple matrix product axb with application to problem of discerning existence of unique solution in a network. *ELECTRONICS & COMMUNICATIONS IN JAPAN*, 57(11):50–57, 1974.

[248] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets (extended abstract). In *FOCS*, pages 338–343, 1989.

[249] Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 332–337. IEEE, 1989.

[250] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical Programming*, 73(3):291–341, 1996.

[251] Jacobo Valdes, Robert E Tarjan, and Eugene L Lawler. The recognition of series parallel digraphs. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 1–12. ACM, 1979.

[252] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996.

[253] Vijay V Vazirani. *Approximation algorithms.* Springer Science & Business Media, 2013.

[254] Jens Vygen. A note on schrijver's submodular function minimization algorithm. *Journal of Combinatorial Theory, Series B*, 88(2):399–402, 2003.

[255] S. Fujishige W. Cui. A primal algorithm for the submodular flow problem with minimum mean cycle selection. *Journal of the Operations Research Society of Japan*, 1988.

[256] C Wallacher and Uwe T Zimmermann. A polynomial cycle canceling algorithm for submodular flows. *Mathematical programming*, 86(1):1–15, 1999.

[257] L. Walras. *Éléments d'économie politique pure; ou, Théorie de la richesse sociale.* Corbaz, 1874.

[258] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012.

[259] David P Williamson and David B Shmoys. *The design of approximation algorithms.* Cambridge university press, 2011.

[260] Philip Wolfe. Finding the nearest point in a polytope. *Math. Programming*, 11:128 – 149, 1976.

[261] Yinyu Ye. Complexity analysis of the analytic center cutting plane method that uses multiple cuts. *Mathematical Programming*, 78(1):85–104, 1996.

[262] David B Yudin and Arkadii S Nemirovski. Evaluation of the information complexity of mathematical programming problems. *Ekonomika i Matematicheskie Metody*, 12:128–142, 1976.

[263] U Zimmermann. Minimization on submodular flows. *Discrete Applied Mathematics*, 4(4):303–323, 1982.

[264] Uwe Zimmermann. Negative circuits for flows and submodular flows. *Discrete applied mathematics*, 36(2):179–189, 1992.