Efficient algorithms for online convex optimization and their applications

Elad Hazan

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

September 2006

Abstract

In this thesis we study algorithms for online convex optimization and their relation to approximate optimization.

In the first part, we propose a new algorithm for a general online optimization framework called *online convex optimization*. Whereas previous efficient algorithms are mostly gradient-descent based, the new algorithm is inspired by the Newton-Raphson method for convex optimization, and hence called Online Newton Step. We prove that in certain scenarios Online Newton Step guarantees logarithmic regret, as opposed to polynomial bounds achieved by previous algorithms. The analysis is based on new insights concerning the natural "follow-the-leader" (FTL) method for online optimization, and answers some open problems regarding FTL.

One application is for the portfolio management problem, for which we describe experimental results over real market data.

In the second part of the thesis, we describe a general scheme of utilizing online game playing algorithms to obtain efficient algorithms for offline optimization. Using new and old online convex optimization algorithms we show how to derive the following:

1. Approximation algorithms for convex programming with linear dependence on the approximation guarantee.

2. Efficient algorithms for haplotype frequency estimation.

3. Fast algorithms for approximate semidefinite programming

Acknowledgments

First and foremost, I would like to thank my advisor Sanjeev Arora. Most of what I've learned in the past four years is a direct result of his patience, intuition, crisp and honest criticism (which was far too lenient). Thanks for the guidance, support, dinners, lunches and everything.

Many thanks to my primary collaborator Satyen Kale. It was a great pleasure discussing research and random topics with someone as brilliant as Satyen. I look forward to further collaboration, as well as his upcoming wedding...

I was fortunate enough to be in the active research environment of Princeton, and as a consequence had the opportunity to work and interact with many wonderful people. In particular I'd like to thank Eran Halperin for teaching me perspective on research and life in general. I'm looking forward to working more with Eran soon. It was an invaluable opportunity to have Rob Schapire at Princeton. Rob manages to create such a great atmosphere for research, and learning from him was particularly enjoyable. Many thanks to the other great people I worked with: Amit Agarwal, Eli Berger, Adam Kalai, Guy Kindler and Muli Safra.

I would like to thank colleagues that influenced my research. Thanks to my thesis committee members Boaz Barak, Moses Charikar, Bernard Chazelle, Rob Schapire and Robert Tarjan for their helpful comments and suggestions. I will not attempt to list all my friends and colleagues at Princeton, yet I want to thank them for creating a fun and productive environment. I'm greatly indebted to Amos Fiat for introducing me to academic research and his continued support.

I want to thank Princeton University for giving me this amazing opportunity to study and do research. I gratefully acknowledge funding through Sanjeev Arora's David and Lucile Packard Fellowship and NSF grants MSPA-MCS 0528414, CCF 0514993, ITR 0205594, as well as Princeton's Wu fellowship.

Most importantly, I want to thank my family. My parents Tzafi and Giora for all the love and support they have provided during my graduate studies and always. My dear wife Dana followed me half way across the world for my graduate studies. If that's not enough, she has already agreed to cross another continent next year... I cannot imagine this work, or anything at all, without her, and our infinite source of happiness, Hadar.

To my parents, Tzafi and Giora,
and my darlings Dana and Hadar

Contents

Abstract iii
List of Figures ix

1 Introduction **1**
1.1 Online Convex Optimization 1
1.2 Approximate optimization and Lagrangian relaxation 2
1.3 Our results 3
1.4 Structure of the thesis 4

2 Online Convex Optimization: A survey **5**
2.1 The online convex optimization model 5
2.1.1 A formal definition of the model 6
2.2 Applications 7
2.2.1 Learning from expert advice 7
2.2.2 Online zero-sum game playing 8
2.2.3 Portfolio Management 9
2.3 Previous algorithms for online convex optimization 10
2.3.1 Notation and Definitions 10
2.3.2 Online Gradient Descent 12
2.3.3 The Multiplicative Weights Algorithm 13
2.3.4 Follow the Leader 15
2.3.5 Other Algorithms for Linear cost functions 16
2.3.6 Cover's Algorithm 17
2.4 Linear versus curved cost functions 18
2.5 Lower bounds 19

3 The Online Newton Method **20**
3.1 Comparison of new and old results 20
3.2 Online Newton Step 21
3.2.1 Implementation and running time 26
3.3 The connection to the Follow The Leader algorithm 27
3.3.1 Implementation and running time 31
3.4 Other logarithmic regret algorithms 31
3.4.1 Online Gradient Descent 32

3.4.2 Exponentially Weighted Online Optimization 33
3.5 Projections onto convex sets 35
3.6 Experiments with Portfolio Management 39
3.6.1 Performance vs. Portfolio Size 40
3.6.2 Random stocks from S&P 500 41
3.6.3 Cover's Experiments 42
3.6.4 Stock volatility 43
3.6.5 Margins loans 44
3.6.6 Sharpe Ratio and Mean-Variance Optimal CRPs 44

4 A Game Playing Framework for Offline Optimization 46
4.1 Introduction 46
4.1.1 Lagrangian relaxation and solving zero sum games 50
4.2 The general scheme 50
4.3 Applications 55
4.3.1 Strictly convex programs 55
4.3.2 Linear and Convex Programs 57
4.3.3 Derivation of previous results 59
4.4 Lower bounds 60
4.5 A general min-max theorem 61

5 Estimating Haplotype Frequencies Efficiently 63
5.1 Introduction 63
5.2 Estimating Haplotype Frequencies 65
5.2.1 A maximum likelihood approach 65
5.2.2 Working with phased data 66
5.3 Estimating Haplotype Frequencies from a Phased Sample 67
5.3.1 Efficient Approximation Scheme 68
5.3.2 The HaploFreq algorithm 71
5.4 Estimating Haplotype Frequencies from Unphased Genotypes 74
5.4.1 Asymptotic Behavior of the Likelihood Function. 75
5.4.2 The HaploFreq2 algorithm 76
5.5 Lower Bounds 79

6 Fast Approximation Algorithms for Semidefinite Programming 82
6.1 Introduction 82
6.1.1 Overview of our results 85
6.2 An illustration of the method 86
6.3 Proofs of the Main Theorems 90
6.4 Implementing Oracle using the approximate eigenvector computations . 92
6.5 Applications 94
6.5.1 SDP relaxations of Quadratic Programs 94
6.5.2 SDP relaxations of biological probability estimation problems . . . 95
6.5.3 SDP relaxations in Side Chain Positioning Problems 96

6.5.4 Embedding of finite metric spaces into ℓ_2 98
6.5.5 SDP relaxation of Sparsest Cut 99
6.5.6 SDP relaxations for MIN UNCUT and related problems 100
6.6 Matrix sparsification 101
6.7 Discussion 104

A A brief history of Newton's method 105

List of Figures

2.1 The ONLINE GRADIENT DESCENT algorithm definition. 12
2.2 The Multiplicative Weights algorithm for online convex optimization over the simplex 14
2.3 The FOLLOW THE LEADER algorithm definition. 16
2.4 The UNIVERSAL algorithm for portfolio management. 17

3.1 Previous and New algorithms for online convex optimization. 21
3.2 The ONLINE NEWTON STEP algorithm. 22
3.3 The FOLLOW THE APPROXIMATE LEADER algorithm 27
3.4 The EXPONENTIALLY WEIGHTED ONLINE OPTIMIZATION Algorithm. . . 33
3.5 A Procedure for projecting onto the Simplex 37
3.6 Algorithms used in the experiments. 39
3.7 The ONLINE NEWTON STEP variant for portfolio management. 40
3.8 Performance vs. Portfolio Size 41
3.9 How ONS tracks CMC. 41
3.10 Performance vs. Trading Period. 42
3.11 Four pairs of stocks tested by [Cov91] and [HSSW96]. 42
3.12 Wealth achieved by various algorithms on a portfolio consisting of IBM and Coke. 43
3.13 Performance of algorithms on high and low volatility datasets. 43
3.14 Sharpe ratios for various algorithms on different datasets. 44
3.15 Minimum variance CRPs for various algorithms on different datasets. The number to the left of the slash is the volatility of the minimum variance CRP and the number to the right is the volatility of the algorithm on the dataset. 45

4.1 meta algorithms for approximate optimization by online game playing . . 53

5.1 The Multiplicative Weights algorithm applied to FREQUENCY ESTIMATION OF PHASED GENOTYPES 69
5.2 Algorithm HAPLOFREQ 71
5.3 Algorithm HAPLOFREQ2 77

6.1 Running time obtained for several applications. 86
6.2 Procedure SPARSIFY 102

Chapter 1

Introduction

The focus of this thesis is efficient algorithms for optimization, both in the online and the offline settings.

An example of an online optimization problem we consider is the problem of online portfolio management. An online investor wants to distribute her wealth on a set of available financial instruments without knowing the market outcome in advance. The goal of the online investor is to maximize her change in wealth over a sequence of many trading periods.

An example of offline optimization is the problem of Linear Programming, in which one wants to maximize a linear objective function subject to a set of linear constraints. Another example is Haplotype Frequency Estimation. In this computational problem, motivated from a biological application, we are given a set of noisy binary strings, in which several strings contain missing/erroneous bits. The goal is to compute the most likely distribution over binary strings which generated the noisy data.

In this thesis we propose a new algorithm for the online setting, which is based on the Newton-Raphson method (see Appendix A). We then describe how the new online algorithm and other related algorithms can be used to derive efficient approximation algorithms for several offline optimization problems.

1.1 Online Convex Optimization

In *online convex optimization*, an online player chooses a point in a convex set. After the point is chosen, a concave payoff function is revealed, and the online player receives payoff which is the concave function applied to the point she chose. This scenario is repeated for many iterations.

The online convex optimization framework generalizes many previous online optimization problems. For example, in the problem of online portfolio management an online investor wants to distribute her wealth on a set of n available financial instruments without knowing the market outcome in advance. The wealth distribution of the online investor can be thought of as a point in the set of all distributions over n items (the financial instruments), which is a convex set. The payoff to the online player is the change in wealth, which is a concave function of her distribution. Other examples which fit into this online framework include the problems of prediction from expert advice and online zero-sum game playing.

To measure the performance of the online player we consider two metrics. The first is information theoretic: how well did the player perform considering that she did not know in advance the concave payoff functions which she will encounter (e.g. the market behavior) ? In other words, how is the performance of the player limited by lack of foresight.

The obvious measure would be to compare the performance of the online player to a player that would have complete knowledge of the future, i.e. knows the concave payoff functions to be encountered. Indeed such performance metrics have been used to analyze various online algorithms for interesting problems such as computer cache management. However, for many real world problems no player which cannot anticipate the future can even slightly compare with an all-knowing prophet.

There are many approaches to cope with the aforementioned difficulty, but by far the most common is to measure the performance of the online player versus a "restricted prophet", i.e. versus a player that has complete knowledge of the future but is restricted to choose a single point for all iterations. This performance measure is called *regret*. Regret measures the difference in payoff between the online player and the "restricted prophet" (the best fixed point in hindsight).

The second metric by which we measure performance is computational complexity, i.e. the amount of computer resources required to compute the online player's point for the upcoming iteration given the history of payoff functions encountered thus far.

Previous approaches for online convex optimization are based on first-order optimization, i.e. optimization using the first derivatives of the payoff functions. The regret achieved by these algorithms is proportional to a polynomial (square root) in the number of iterations. Besides the general framework, there are specialized algorithms, e.g. for portfolio management, which attain regret proportional to the logarithm of the number of iterations. However, these algorithms do not apply to the general online convex optimization framework and are less efficient in terms of computational complexity.

1.2 Approximate optimization and Lagrangian relaxation

In the second part of the thesis we consider the computational problem of minimizing a convex function over a convex domain. Special cases of this general problem include Linear Programming, Semidefinite Programming and various network flow problems.

Polynomial time algorithms for this task include interior point methods and the ellipsoid method. Although of great theoretical and practical importance, for certain convex and linear programs which occur in practice these methods are too slow.

Lagrangian relaxation is one approach to deal with this difficulty, in which an approximate solution is computed efficiently. A characteristic of lagrangian relaxation methods is the polynomial dependence of the running time on the approximation guaranty and on the *width* - a measure of the size of the instance numbers. This is in contrast to interior point methods and the ellipsoid method, whose running time depend poly-logarithmically on the approximation guaranty and on the width, and are thus much superior to Lagrangian relaxation in this respect. The advantage of Lagrangian relaxation algorithms is their simplicity, low running time in terms of input size and the use of elementary operations. On the other hand, interior point methods require “expensive” computations such as matrix inversions or computing the Cholesky decomposition.

Many researchers have previously observed the close relation between Lagrangian relaxation and solving zero sum games. Perhaps the earliest reference goes back to von Neumann, who noted the connection between the min-max theorem of zero sum games and linear programming duality. The connection to the online optimization setting stems from the fact that online convex optimization algorithms can be used to solve zero sum games.

1.3 Our results

We introduce a new algorithm, Online Newton Step, which uses second-order information of the payoff functions and is based on the well known Newton-Raphson method for offline optimization. The Online Newton Step algorithm attains regret which is proportional to the logarithm of the number of iterations when the payoff functions are concave, and is computationally efficient.

The intuition leading to the Online Newton Step algorithm stems from new observations regarding a very natural approach for online optimization called *follow-the-leader*. Informally, this natural approach suggests to use the best point so far for the upcoming iteration. The connection to the Newton method allows us to prove a strong performance guarantee for the ”follow-the-leader” approach in a general setting, resolving some open problems.

After discussing online convex optimization, we proceed to describe how online convex optimization algorithms can be used to derive efficient offline approximate optimization algorithms. Admittedly, this sounds counterintuitive: online optimization is a more difficult problem than offline optimization. Yet a general algorithmic scheme called Lagrangian relaxation is related to online convex optimization.

We generalize previous approaches and describe a framework for deriving approximate optimization algorithms using online convex optimization algorithms. The main observation is that the guarantee on the regret for a given online algorithm can be converted to a convergence guarantee when this algorithm is used to derive an offline approximation algorithm.

Using the new online convex optimization algorithms, and in particular the Online Newton Step algorithm, we derive approximation algorithms for convex programming with linear dependence on the approximation guarantee. Previous approximation algorithms had quadratic dependence on the approximation guarantee. Another advantage of the new algorithms is that similar to the ellipsoid method, they allow the convex set to be described implicitly by a separation oracle.

Finally we give concrete examples of Lagrangian relaxation algorithms. First, we describe a problem arising from computational biology called Haplotype Frequency Estimation. We show how to easily derive an approximation algorithm using the general framework for deriving approximate optimization algorithms from online convex optimization algorithms. Then we describe an algorithm called HaploFreq which is custom designed (although related to Lagrangian relaxation) and more efficient.

The next example concerns efficient algorithms for semi-definite programming. The overall framework is also Lagrangian relaxation, but we also introduce some new ideas, such as composing lagrangian relaxation with the ellipsoid method, efficient eigenvalue computation and matrix sparsification.

1.4 Structure of the thesis

The first two chapters deal with the online setting. In chapter 2 we give a survey of the online convex optimization model, applications and previous algorithms. Chapter 3 contains our new algorithms for the online setting, as well as some experimental results.

In chapter 4 we describe the general framework for deriving approximate optimization algorithms from online convex optimization algorithms. This chapter also describes applications to Linear and Convex Programming.

In the final two chapters we discuss other lagrangian relaxation algorithms. In chapter 5 we describe the Haplotype Frequency Estimation problem and efficient approximation algorithms designed for it. Chapter 6 concerns efficient algorithms for semi-definite programming.

Most chapters can be read independently. An exception is chapter 3, which relies on definitions and preliminaries given in chapter 2. Chapter 4 connects the chapters which precede and succeed it, and reading it before chapters 5 and 6 is recommended (although not necessary).

Chapter 2

Online Convex Optimization: A survey

In this chapter we describe the online model which we study in this thesis and its applications. After defining the model formally, we illustrate how several problem can be cast as special cases of online convex optimization.

We then proceed to survey some of the previous algorithms for this model. In some cases we include performance analysis which is different from the original and may be of independent interest.

We conclude with lower bounds and a discussion on the room left for improvement over previous algorithms.

2.1 The online convex optimization model

Consider the following simple example: an investor tries to invest in a certain stock. For simplicity, think of its price movements as up/down. If the investor succeeds in predicting the market she gains one dollar, and otherwise loses one dollar.

In making her predictions, the online investor is allowed to watch the predictions of n "experts" (who could be arbitrarily correlated, and who may or may not know what they are talking about). The investor is also allowed to use random coin tosses for her decision. For example, she may decide to follow the advice of the first expert with certain probability, and the second expert otherwise.

The algorithms we are interested in will enable the investor to perform roughly the same as the *best* of these experts (in expectation and with high probability if randomization is allowed). Further, the computation that the investor need perform before each trading day can be carried out efficiently. At first sight this may seem an impossible goal, since it is not known until the end of the sequence who the best expert was, whereas the algorithm is required to make predictions all along.

Below we define the *online convex optimization* framework formally. After providing all formal definitions and notation, we proceed to show how the above simple prediction problem, as well as several other online problems, can be derived as instantiations of this model.

2.1.1 A formal definition of the model

In online convex optimization, an online player iteratively chooses a point from a set in Euclidean space denoted $\mathcal{P} \subseteq \mathbb{R}^n$. Following Zinkevich [Zin03], we assume that the set $\mathcal{P}$ is non-empty, bounded and closed. For reasons that will be apparent in section 3.5, we also assume the set $\mathcal{P}$ to be convex.

We denote the number of iterations by T (which is unknown to the online player). At iteration t, the online player chooses $\mathbf{x}_t \in \mathcal{P}$. After committing to this choice, a convex cost function $f_t : \mathcal{P} \mapsto \mathbb{R}$ is revealed. The cost incurred to the online player is the value of the cost function at the point she committed to $f_t(\mathbf{x}_t)$.

Consider an online player using a (possibly randomized) algorithm for online game playing $\mathcal{A}$. At iteration t, the algorithm $\mathcal{A}$ takes as input the history of cost functions $f_1, ..., f_{t-1}$ and produces a feasible point $\mathcal{A}(\{f_1, ..., f_{t-1}\}) \in \mathcal{P}$. When there is no ambiguity concerning the algorithm used, we simply denote $\mathbf{x}_t = \mathcal{A}(\{f_1, ..., f_{t-1}\})$. The regret of the online player using algorithm $\mathcal{A}$ at time T, is defined to be the total cost minus the cost of the best single decision, where the best is chosen with the benefit of hindsight. Formally

$$\text{Regret}(\mathcal{A}, \{f_1, ..., f_T\}) = \mathbf{E}[\textstyle\sum_{t=1}^T f_t(\mathbf{x}_t)] - \min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T f_t(\mathbf{x}).$$

Regret measures the difference in performance between the online player and a “static” player with the benefit of hindsight - i.e a player that is constrained to choose a fixed point over all iterations. It is tempting to compare the online player to an adversary which has the benefit of hindsight but is otherwise unconstrained (i.e. can dynamically change her point every iteration). However, this allows the adversary to choose the optimum point $\mathbf{x}_t^* \triangleq \min_{x\in\mathcal{P}} f_t(x)$ each iteration, and the comparison becomes trivial in many interesting applications.

We are usually interested in an upper bound on the worst case guaranteed regret, denoted

$$\text{Regret}_T(\mathcal{A}) = \sup_{\{f_1,...,f_t\}} \{\text{Regret}(\mathcal{A}, \{f_1, ..., f_t\})\}$$

Intuitively, an algorithm attains non-trivial performance if its regret is sublinear as a function of T, i.e. $\text{Regret}_T(\mathcal{A}) = o(T)$, since this implies that “on the average” the algorithm performs as good as the best fixed strategy in hindsight.

Remark: For some problems it is more natural to talk of “payoff” given to the online player rather than cost she incurs. In such cases, the online player receives payoff $f_t(\mathbf{x}_t)$, where f_t is a concave utility function. Regret is then defined to be

$$\text{Regret}(\mathcal{A}, \{f_1, ..., f_T\}) = \max_{\mathbf{x}\in\mathcal{P}} \textstyle\sum_{t=1}^T f_t(\mathbf{x}) - \mathbf{E}[\sum_{t=1}^T f_t(\mathbf{x}_t)].$$

The running time of an algorithm for online game playing is defined to be the worst-case expected time to produce $\mathbf{x}_t$, for an iteration $t \in [T]$ [1] in a T iteration repeated game.

[1]here and henceforth we denote by $[n]$ the set of integers $\{1, ..., n\}$

Typically, the running time will depend on n, T and parameters of the cost functions and underlying convex set.

2.2 Applications

2.2.1 Learning from expert advice

Learning from expert advice is the generic problem of machine learning, which we informally described in the beginning of this chapter with the stock prediction example. This problem fits into the online game playing framework as follows. The convex set $\mathcal{P}$ is taken to be the set of all distributions, namely the n dimensional simplex denoted $\mathbb{S}_n$.

$$\mathbb{S}_n = \left\{ \mathbf{x} \in \mathbb{R}^n, \forall i \in [n] \ \mathbf{x}_i \geq 0 \ , \ \sum_{i=1}^{n} \mathbf{x}_i = 1 \right\}$$

For day t, let the payoff (i.e. the profit we would make if we would take this expert's advice) associated with expert $i \in [n]$ be $\mathbf{a}_t(i)$. Denote the entire payoff vector by $\mathbf{a}_t$. Define the t'th payoff function as

$$\forall x \in \mathcal{P} \quad f_t(\mathbf{x}) \triangleq \sum_{i=1}^{n} \mathbf{a}_t(i) \cdot \mathbf{x}(i) = \mathbf{x}^\top \mathbf{a}_t$$

Thus, choosing expert i in day t, parallels the choice of the point $\mathbf{e}_i \in S_n$ [2], and results in payoff of $\mathbf{a}_t(i)$. Even more generally, this models the use of a randomized algorithm that picks expect i with probability $\mathbf{x}_t(i)$. The expected payoff is exactly $f_t(\mathbf{x}_t)$.

Let j be the best expert in hindsight, i.e $j \triangleq \arg\max_{i\in[n]} \sum_{t=1}^{T} \mathbf{e}_i^\top \mathbf{a}_t$. Then regret for an algorithm $\mathcal{A}$ in this setting is

$$\text{Regret}_T(\mathcal{A}) = \max_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^{T} \mathbf{x}^\top \mathbf{a}_t - \mathbf{E}[\sum_{t=1}^{T} \mathbf{x}_t^\top \mathbf{a}_t] \leq \sum_{t=1}^{T} \mathbf{e}_j^\top \mathbf{a}_t - \mathbf{E}[\sum_{t=1}^{T} \mathbf{x}_t^\top \mathbf{a}_t]$$

Which is simply the difference between the profit of the best distribution over experts in hindsight (which is just the best expert in hindsight) and the payoff attained by $\mathcal{A}$. Thus, sublinear regret amounts to having the average difference in payoff converge to zero.

Prediction from expert advice is closely related to Boosting —combining several moderately accurate rules-of-thumb into a singly highly accurate prediction rule— which is a central idea of AI today. As such, many algorithms for this learning problem were proposed and analyzed. We refer the reader to the survey by Schapire [Sch03].

Efficient algorithms, measured by notions of regret, algorithmic complexity and other metrics, were proposed in the seminal work of Freund and Schapire on Boosting [FS97]. Their algorithms achieve regret which is bounded by $O(\sqrt{T \log n})$ for T prediction iterations, and can be implemented in time $O(n)$ per iteration, where n is the number of experts. The technique underlying their algorithms is called the Multiplicative Weights Update Method, which we shall revisit in later sections and chapters.

[2] $\mathbf{e}_i$ denotes the i'th standard basis vector, i.e. the n dimensional vector with coordinates all zero except for one in the i'th position.

2.2.2 Online zero-sum game playing

Consider the following scenario: Let $\mathcal{M}$ be a matrix. On each of a series of rounds, one player chooses a row $i \in [n]$ and the other chooses a column $j \in [m]$. The selected entry $\mathcal{M}(i,j)$ is the loss suffered by the row player and the profit gained by the column player. We denote by $\mathcal{M}(\mathbf{x},\mathbf{y})$ the expected loss to the row player if she plays according to the distribution $\mathbf{x} \in \mathbb{S}_n$ over the rows and the column player plays distribution $\mathbf{y} \in \mathbb{S}_m$ over the columns, i.e.

$$\mathcal{M}(\mathbf{x},\mathbf{y}) = \sum_{i\in[n]} \sum_{j\in[m]} \mathbf{x}(i)\mathbf{y}(j)\mathcal{M}(i,j)$$

The game value is defined to be

$$\lambda^* = \min_{\mathbf{x}\in\mathbb{S}_n} \max_{\mathbf{y}\in\mathbb{S}_m} \mathcal{M}(\mathbf{x},\mathbf{y})$$

According to the von Neumann min-max theorem $\lambda^* = \min_{\mathbf{x}\in\mathbb{S}_n} \max_{\mathbf{y}\in\mathbb{S}_m} \mathcal{M}(\mathbf{x},\mathbf{y}) = \max_{\mathbf{y}\in\mathbb{S}_m} \min_{\mathbf{x}\in\mathbb{S}_n} \mathcal{M}(\mathbf{x},\mathbf{y})$. Denote the optimal row/column distributions (also called *mixed strategies*) by

$$\mathbf{x}^* \triangleq \arg \min_{\mathbf{x}\in\mathbb{S}_n} \max_{\mathbf{y}\in\mathbb{S}_m} \mathcal{M}(\mathbf{x},\mathbf{y})$$

$$\mathbf{y}^* \triangleq \arg \max_{\mathbf{y}\in\mathbb{S}_m} \min_{\mathbf{x}\in\mathbb{S}_n} \mathcal{M}(\mathbf{x},\mathbf{y})$$

The optimal strategies can be computed in polynomial time using linear programming. Hence by computing $\mathbf{x}^*$ the row player can guarantee an expected loss of at most λ^*.

There are several drawbacks to this approach. First, the matrix $\mathcal{M}$ may be very large and computing $\mathbf{x}^*$ becomes infeasible. Second, the matrix $\mathcal{M}$ may be unknown to the row player. Third, the column player may not be truly adversarial and allow loss significantly smaller than the game value.

To phrase this problem in the online convex optimization setting, the convex set $\mathcal{P}$ is chosen to be the set of all distributions over the n rows, i.e. $\mathbb{S}_n$. The cost functions are defined to be the expected cost according to a given distribution. If the column player plays according to distribution $\mathbf{y}_t$ at iteration t, define

$$\forall \mathbf{x} \in \mathcal{P} \ . \ f_t(\mathbf{x}) = \mathcal{M}(\mathbf{x},\mathbf{y}_t) = \sum_{i\in[n]} \mathbf{x}_t(i)\mathcal{M}(i,\mathbf{y}_t)$$

Notice that the cost functions are linear. An algorithm attaining regret at most $R_T(\mathcal{A})$ over T iterations satisfies

$$\textstyle\sum_{t=1}^T \mathcal{M}(\mathbf{x}_t,\mathbf{y}_t) - \min_{\mathbf{x}\in\mathbb{S}_n} \sum_{t=1}^T \mathcal{M}(\mathbf{x},\mathbf{y}_t) \leq R_t(\mathcal{A})$$

Note that according to the definition of the game value $\min_{\mathbf{x}\in\mathbb{S}_n} \mathcal{M}(\mathbf{x},\mathbf{y}_t) \leq \lambda^*$. Shifting sides in the above inequality and normalizing we obtain

$$\frac{1}{T}\textstyle\sum_{t=1}^T \mathcal{M}(\mathbf{x}_t,\mathbf{y}_t) \leq \lambda^* + \frac{R_t(\mathcal{A})}{T}$$

Thus, if $R_T(\mathcal{A}) = o(T)$, as the number of game iterations grows $T \mapsto \infty$, the average cost becomes closer to the game value (or possibly better, to the best fixed strategy in hindsight).

Algorithms of this type were first proposed by Blackwell [Bla56] and Hannan [Han57], and later algorithms were proposed by Foster and Vohra [FV99, FV93, FV98] and Freund and Schapire [FS99].

2.2.3 Portfolio Management

In the universal portfolio management problem, we seek an online wealth investment strategy which enables an investor to maximize his wealth by distributing it on a set of available financial instruments without knowing the market outcome in advance or even any statistical assumptions on its behavior. In fact, our model permits the market to be adversarial. The study of such a model was started in the 1950s by Kelly [Kel56] followed by Bell and Cover [BC80, BC88] and Algoet and Cover [AC88].

Absolute wealth maximization in an adversarial market is of course a hopeless task; we therefore aim to maximize our wealth relative to that achieved by a reasonably sophisticated investment strategy.

One option that comes to mind is versus the best single stock in hindsight. We consider an even more powerful investment strategy, the constant-rebalanced portfolio [Cov91], abbreviated CRP. A CRP strategy rebalances the wealth each trading period to have a fixed proportion in every stock in the portfolio. For example, in a two commodity market the $(\frac{1}{2}, \frac{1}{2})$ CRP rebalances the wealth evenly over the two commodities after each trading period. A special case is, of course, a single stock in the market which corresponds to the CRP that reinvests all wealth in that stock.

However, a CRP is potentially much more powerful strategy, as shown by the following example. Consider a market with only two stocks: the first increases it's value ten fold on even days, and decreases its value to 1/100 on odd days. The second stock changes its value with inverse correlation - on odd days increases its value ten fold, and even days decreases its value a hundred fold. Obviously, investing all wealth in any of the two options results in net decrease of wealth by $10 \cdot \frac{1}{100} = \frac{1}{10}$ every two days. However, reinvesting 50% in each stock every day guarantees $\frac{1}{2} \cdot 10 + \frac{1}{2} \cdot \frac{1}{100} = 500.5\%$ increase in wealth daily !

Let the number of stocks in the portfolio be n. On every trading period t, for $t = 1, \ldots, T$, the investor observes a *price relative vector* $\mathbf{r}_t \in \mathbb{R}^n$, such that $\mathbf{r}_t(j)$ is the ratio of the closing price of stock j on day t to the closing price on day $t-1$. A portfolio $\mathbf{p}$ is a distribution on the n stocks, so it is a point in the n-dimensional simplex S_n. If the investor uses a portfolio $\mathbf{p}_t$ on day t, his wealth changes by a factor of $\mathbf{p}_t^\top \mathbf{r}_t$. Thus, after T periods, the wealth achieved per dollar invested is $\prod_{t=1}^T (\mathbf{p}_t^\top \mathbf{r}_t)$. Taking the logarithm of the wealth change gives the *logarithmic growth ratio* $\sum_{t=1}^T \log(\mathbf{p}_t^\top \mathbf{r}_t)$. Thus, to fit the online game playing framework, we define the cost function at time t to be $f_t(\mathbf{p}) = \log(\mathbf{p}^\top \mathbf{r}_t)$. Notice that this function is concave.

An investor using a CRP $\mathbf{p} \in \mathbb{S}_n$ achieves the logarithmic growth ratio $\sum_{t=1}^T \log(\mathbf{p}^\top \mathbf{r}_t)$.

The best CRP in hindsight $\mathbf{p}^*$ is the one which maximizes this quantity, i.e.

$$\mathbf{p}^* \in \arg\max_{p \in \mathbb{S}_n} \sum_{t=1}^{T} \log(\mathbf{p}^\top \mathbf{r}_t)$$

The regret of an online algorithm, $\mathcal{A}$, which produces portfolios $\mathbf{p}_t$ for $t = 1, \ldots, T$, is then

$$\text{Regret}_T(\texttt{Alg}) \ = \sum_{t=1}^{T} \log((\mathbf{p}^*)^\top \mathbf{r}_t) - \sum_{t=1}^{T} \log(\mathbf{p}_t^\top \mathbf{r}_t).$$

Thus, an investor using an algorithm with low regret achieves asymptotically the same daily wealth increase as the best CRP in hindsight.

Since scaling $\mathbf{r}_t$ by a constant affects the logarithmic growth ratios of both the best CRP and $\mathcal{A}$ by the same additive factor, the regret does not change. So we assume without loss of generality that for all t, $\mathbf{r}_t$ is scaled so that $\max_j \mathbf{r}_t(j) = 1$. We also make the assumption that after this scaling, all the $\mathbf{r}_t(j)$ are bounded below by the *market variability parameter* $\xi > 0$. This has been called the *no-junk-bond assumption* by Agarwal and Hazan, and can be interpreted to mean that no stock crashes to zero value over the trading period.

With this setup, Cover [Cov91] gave the first portfolio selection algorithm which had the optimal regret $O(n \log T)$, without dependence on the market variability parameter ξ. The running time of his algorithm is exponential: the algorithm computes the portfolio $\mathbf{p}_t$ in $\Omega(t^n)$ time. Kalai et al. [KV03] gave a polynomial implementation of the algorithm using sampling of logconcave functions from convex domains [LV03b, LV03a], and this results in a randomized $poly(T, n)$ time algorithm, though the polynomial is still quite large. Helmbold et al. [HSSW96] gave an algorithm which needs just linear $O(n)$ time and space per period but has suboptimal regret of $O(\sqrt{T \log n})$ (assuming $\xi > 0$).

In Chapter 3 we describe an algorithm that is both deterministically computable in time $\tilde{O}(n^2)$ and attains logarithmic regret $O(n \log T)$ for markets with constant market variability $\xi > 0$.

2.3 Previous algorithms for online convex optimization

We begin by describing the parameters used to evaluate algorithms for online convex optimization. We then describe some previously known algorithms. At times, we present a different analysis than the original for some of these algorithms. This may be of independent interest to the readers.

2.3.1 Notation and Definitions

Recall that in online convex optimization, the online player iteratively chooses points from a closed, bounded and non-empty convex set $\mathcal{P} \subseteq \mathbb{R}^n$ and encounters convex cost functions $\{f_t : \mathcal{P} \mapsto \mathbb{R}\}$.

Denote by D the diameter of the underlying convex set $\mathcal{P}$, i.e.

$$D = \max_{\mathbf{x},\mathbf{y}\in\mathcal{P}} \|\mathbf{x}-\mathbf{y}\|_2$$

Unless stated otherwise, we assume that the cost functions $\{f_t\}$ are twice differentiable and convex. These assumptions are satisfied by all applications described previously. For some of the algorithms we shall describe, these smoothness conditions can be somewhat relaxed. For the sake of brevity we leave to the reader to note where such generalization are applicable.

Recall that the gradient for a $f : \mathbb{R}^n \mapsto \mathbb{R}$ at point $\mathbf{x} \in \mathbb{R}^n$ is the vector $\nabla f(\mathbf{x})$ whose components are the partial derivatives of the function at $\mathbf{x}$. Its direction is the one in which the function has the largest rate of increase, and its magnitude is the actual rate of increase. We denote the upper bound (not necessarily finite) on the gradients of the cost functions by

$$G = \sup_{\mathbf{x}\in\mathcal{P},t\in[T]} \nabla f_t(\mathbf{x})$$

In some cases we are concerned with the ℓ_∞ norm of the gradient rather than the Euclidean norm, in which case we denote the upper bound by G_∞.

We also consider the analogue of second derivatives for multivariate functions. The Hessian of a function f at point $\mathbf{x}$ is a matrix $\nabla^2 f(\mathbf{x})$, such that $\nabla^2 f(\mathbf{x})[i,j] = \frac{\partial^2}{\partial \mathbf{x}_i,\mathbf{x}_j} f(\mathbf{x})$. Analogous to the one-dimensional case, a function f is convex at point $\mathbf{x}$ if and only if its Hessian is positive-semi-definite $\nabla^2 f(\mathbf{x}) \succeq 0$. We denote a lower bound on the Hessian of all cost functions by the real number $\mathbb{R} \ni H \geq 0$:

$$\inf_{\mathbf{x}\in\mathcal{P},t\in[T]} \nabla^2 f_t(\mathbf{x}) \succeq H \cdot \mathbf{I}_n$$

Here, $\mathbf{I}_n$ is the n-dimensional identity matrix and we denote $\mathbf{A} \succeq \mathbf{B}$ if the matrix $\mathbf{A}-\mathbf{B} \succeq 0$ is positive semi-definite (PSD), i.e. all its eigenvalues are nonnegative. Thus, H is a lower bound on the eigenvalues of all the Hessians of the constraints.

In the following chapters we will consider functions which have bounded gradient and are strictly convex (i.e. $H > 0$). An alternative to a bound on G and H is a constant $\alpha > 0$ such that $\exp(-\alpha f_t(\mathbf{x}))$ is a concave function of $\mathbf{x} \in \mathcal{P}$, for all t, i.e.

$$\inf_{\mathbf{x}\in\mathcal{P},t\in[T]} \nabla^2 \exp(-\alpha f_t(\mathbf{x})) \preceq \mathbf{0}$$

This condition is weaker then a bounded gradient and strict convexity, since if G is bounded and $H > 0$, one can show that $\alpha \leq H/G^2$. One can easily verify this for one-dimensional functions $f_t : \mathbb{R} \to \mathbb{R}$ by taking two derivatives,

$$h_t''(x) = ((\alpha f_t'(x))^2 - \alpha f_t''(x)) \exp(-\alpha f_t(x)) \leq 0 \iff \alpha \leq \frac{f_t''(x)}{(f_t'(x))^2}.$$

When it is more natural to talk of maximization of payoff rather than minimization of cost (i.e. for portfolio management), we require the payoff functions to be concave instead of convex. The parameter H is then defined to be

$$\sup_{\mathbf{x}\in\mathcal{P},t\in[T]} \nabla^2 f_t(\mathbf{x}) \preceq -H \cdot \mathbf{I}_n$$

2.3.2 Online Gradient Descent

Perhaps the simplest algorithm that applies to the most general setting of online convex optimization is online gradient descent. This algorithm, which is based on the standard gradient descent algorithm from offline optimization, was introduced to the online setting by Zinkevich [Zin03].

Pseudo-code for the algorithm is given in figure 2.1. In each iteration the point chosen by the algorithm is the previous point plus a multiple of the gradient of the previous cost function. Adding a multiple of the previous gradient may take the current point out of the underlying convex set. In such cases, the algorithm projects the point back to the convex set, i.e. finds the point in the convex set which is closest to the current. Despite the fact that the upcoming cost function may be completely different than all those which occurred thus far, the regret attained by the algorithm is sublinear, as shown in the following theorem by Zinkevich.

ONLINE GRADIENT DESCENT.
Inputs: convex set $\mathcal{P} \subset \mathbb{R}^n$, step sizes $\eta_1, \eta_2, \ldots \geq 0$, initial $\mathbf{x}_1 \in \mathcal{P}$.

- In iteration 1, use point $\mathbf{x}_1 \in \mathcal{P}$.
- In iteration $t > 1$: use point

  $$\mathbf{x}_t = \Pi_{\mathcal{P}}(\mathbf{x}_{t-1} - \eta_t \nabla f_{t-1}(\mathbf{x}_{t-1}))$$

  Here, $\Pi_{\mathcal{P}}$ denotes the *projection* onto nearest point in $\mathcal{P}$, $\Pi_{\mathcal{P}}(\mathbf{y}) = \arg\min_{\mathbf{x}\in\mathcal{P}} \|\mathbf{x} - \mathbf{y}\|_2$.

Figure 2.1: The ONLINE GRADIENT DESCENT algorithm definition.

Theorem 2.1. *[Zinkevich]* ONLINE GRADIENT DESCENT *with step sizes* $\eta_t = \frac{G}{D\sqrt{t}}$ *achieves the following guarantee, for all* $T \geq 1$.

$$\text{Regret}_T(\text{OGD}) = \sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^{T} f_t(\mathbf{x}) \ \leq 3GD\sqrt{T}$$

Proof. Let $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^{T} f_t(\mathbf{x})$. Define $\nabla_t \triangleq \nabla f_t(\mathbf{x}_t)$. By convexity

$$f_t(\mathbf{x}_t) - f_t(\mathbf{x}^*) \quad \leq \quad \nabla_t^\top(\mathbf{x}_t - \mathbf{x}^*) \tag{2.1}$$

Let's upper-bound $\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^*)$. Using the update rule for $\mathbf{x}_{t+1}$ and standard properties of projections onto convex sets (see section 3.5 lemma 3.9)

$$\|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 \ = \ \|\Pi(\mathbf{x}_t - \eta_t\nabla_t) - \mathbf{x}^*\|^2 \leq \|\mathbf{x}_t - \eta_t\nabla_t - \mathbf{x}^*\|^2. \tag{2.2}$$

Hence,

$$\begin{aligned}\|\mathbf{x}_{t+1}-\mathbf{x}^*\|^2 &\leq \|\mathbf{x}_t-\mathbf{x}^*\|^2+\eta_t^2\|\nabla_t\|^2-2\eta_t\nabla_t^\top(\mathbf{x}_t-\mathbf{x}^*)\\ 2\nabla_t^\top(\mathbf{x}_t-\mathbf{x}^*) &\leq \frac{\|\mathbf{x}_t-\mathbf{x}^*\|^2-\|\mathbf{x}_{t+1}-\mathbf{x}^*\|^2}{\eta_t}+\eta_t G^2 \qquad (2.3)\end{aligned}$$

Sum up (1) and (2) from $t=1$ to T, and having $\eta_t=\frac{D}{G\sqrt{t}}$ (with $\frac{1}{\eta_0}\triangleq 0$):

$$\begin{aligned}2\sum_{t=1}^T f_t(\mathbf{x}_t)-f_t(\mathbf{x}^*) &\leq D^2\sum_{t=1}^T\left(\frac{1}{\eta_t}-\frac{1}{\eta_{t-1}}\right)+G^2\sum_{t=1}^T\eta_t\\ &\leq D^2\frac{1}{\eta_T}+G^2\sum_{t=1}^T\eta_t\leq 3DG\sqrt{T}\end{aligned}$$

The last inequality follows since $\sum_{t=1}^T t^{-\frac{1}{2}}\leq 2\sqrt{T}$. $\square$

The Online Gradient Descent algorithm is straightforward to implement, and updates take time $O(n)$ given the gradient. However, there is a projection step which may take longer. For details on computing projections onto convex sets see section 3.5.

2.3.3 The Multiplicative Weights Algorithm

In this subsection we present an algorithm based on the ubiquitous Multiplicative Weights Update method (for more applications of the method see survey [AHK05a]). For very detailed analysis of the method in similar settings see also [KW97]. Freund and Schapire [FS99] analyze exactly the algorithm below, although for linear payoff functions rather than for general convex functions. Helmbold et al [HSSW96] analyze another instantiation where the payoff functions are logarithmic. Below we apply the method to online convex optimization over the simplex.

The Multiplicative Weights online algorithm is depicted in figure 2.2. This algorithm has been called "exponentiated gradient" in the machine learning literature [KW97]. As the algorithm is phrased, it needs to know T and G_∞ in advance. [3] Standard techniques can be used so that the algorithm need not accept any input: the dependence on T can be removed by doubling the value of T as it is being exceeded. The dependence on G_∞ can be removed by similar means: maintaining an estimate of G_∞ and doubling this estimate whenever a gradient with larger infinity norm is encountered. Implementation is straightforward, and updates take time $O(n)$ given the gradient.

The Multiplicative Weights algorithm attains similar performance guarantees to the "online gradient descent" algorithm of Zinkevich [Zin03]. Despite being less general than Zinkevich's algorithm (we only give an application to the n-dimensional simplex, whereas online gradient descent can be applied over any convex set in Euclidean space), it attains better regret with respect to the dimension as given in the following theorem.

[3]Recall from the previous sections that G_∞ is an upper bound on the ℓ_∞ norm of the cost functions.

Multiplicative Weights.
Inputs: parameters T, G_∞.

- In iteration 1, use the uniform distribution $x_1 = \frac{1}{n}\mathbf{1} \in S_n$. Let $\forall i \in [n]$. $w_i^1 = 1$
- In iteration t, update
$$w_i^t = w_i^{t-1} \cdot (1 + \frac{\eta}{G_\infty} \nabla_{t-1}(i))$$
where $\nabla_t \triangleq \nabla f_t(\mathbf{x}_t)$, $\eta = \sqrt{\frac{\log n}{T}}$ and use point $\mathbf{x}_t$ defined as
$$\mathbf{x}_t \ \triangleq \ \frac{w^t}{\|w^t\|_1}$$

Figure 2.2: The Multiplicative Weights algorithm for online convex optimization over the simplex

Theorem 2.2. *The Multiplicative Weights algorithm achieves the following guarantee, for all $T \geq 1$.*

$$Regret_{MW}(T) = \sum_{t=1}^T f_t(\mathbf{x}_t) - \min_{\mathbf{x} \in S_n} \sum_{t=1}^T f_t(\mathbf{x}) \ \leq O(G_\infty \sqrt{\log n}\sqrt{T})$$

Proof. Define $\Phi^t = \sum_i w_i^t$. Since $\frac{1}{G_\infty}\nabla_t(i) \in [0,1]$,

$$\begin{aligned}
\Phi^{t+1} &= \sum_i w_i^{t+1} = \sum_i w_i^t(1 - \frac{\eta}{G_\infty}\nabla_t(i)) \\
&= \Phi^t - \frac{\eta \Phi^t}{G_\infty} \sum_i \nabla_t(i)\mathbf{x}_t(i) && \text{since } \mathbf{x}_t(i) = w_i^t/\Phi^t \\
&= \Phi^t(1 - \eta \nabla_t^\top \mathbf{x}_t / G_\infty) \\
&\leq \Phi^t e^{-\eta \nabla_t^\top \mathbf{x}_t / G_\infty} && \text{since } 1 - x \leq e^{-x} \text{ for } |x| \leq 1
\end{aligned}$$

After T rounds, we have

$$\Phi^T \leq \Phi^1 e^{-\eta \sum_t \nabla_t^\top \mathbf{x}_t / G_\infty} = n e^{-\eta \sum_t \nabla_t^\top \mathbf{x}_t / G_\infty} \tag{2.4}$$

Also, for every $i \in [n]$, using the following facts which follow immediately from the convexity of the exponential function

$$\begin{aligned}
(1-\eta)^x &\leq (1 - \eta x) \quad \text{if } x \in [0,1] \\
(1+\eta)^{-x} &\leq (1 - \eta x) \quad \text{if } x \in [-1,0]
\end{aligned}$$

We have

$$\begin{aligned}\Phi^T &= \sum_t w_i^T \geq w_i^T \\ &= \prod_t (1-\eta \nabla_t(i)/G_\infty) \\ &\geq (1-\eta)^{\sum_{t>0} \nabla_t(i)/G_\infty} (1+\eta)^{\sum_{t<0} -\nabla_t(i)/G_\infty}\end{aligned}$$

where the subscripts ≥ 0 and < 0 refer to the rounds t where $\nabla_t(i)$ is ≥ 0 and < 0 respectively. So together with (2.4)

$$ne^{-\eta \sum_t \nabla_t^\top \mathbf{x}_t/G_\infty} \geq (1-\eta)^{\sum_{t>0} \nabla_t(i)/G_\infty} (1+\eta)^{\sum_{t<0} -\nabla_t(i)/G_\infty}$$

Taking logarithms and using $\ln(\frac{1}{1-\eta}) \leq \eta + \eta^2$ and $\ln(1+\eta) \geq \eta - \eta^2$ for $\eta \leq \frac{1}{2}$ we get for all $i \in [n]$ and $\mathbf{x}^* \in S_n$

$$\begin{aligned}\sum_t \nabla_t^\top \mathbf{x}_t &\leq (1+\eta)\sum_{\geq 0} \nabla_t(i) + (1-\eta)\sum_{<0} \nabla_t(i) + \frac{G_\infty \log n}{\eta} \\ &\leq \sum_t \nabla_t(i) + \eta \sum_t |\nabla_t(i)| + \frac{G_\infty \log n}{\eta} \\ &\leq \sum_t \nabla_t^\top \mathbf{x}^* + \eta T G_\infty + \frac{G_\infty \log n}{\eta}\end{aligned}$$

Therefore

$$\begin{aligned}\sum_t f_t(\mathbf{x}_t) - f_t(\mathbf{x}^*) &\leq \sum_t \nabla_t^\top (\mathbf{x}_t - \mathbf{x}^*) \\ &\leq \eta T G_\infty + \frac{G_\infty \log n}{\eta}\end{aligned}$$

And the theorem follows since $\eta = \sqrt{\frac{\log n}{T}}$

$\square$

2.3.4 Follow the Leader

Perhaps the most intuitive algorithm for online game playing can be described as follows: at iteration t, choose $\mathbf{x}_t$ which is the best strategy so far, i.e. the point in $\mathcal{P}$ that minimizes the sum of all cost functions encountered thus far. A precise mathematical definition is given in figure 2.3. [4]

Given the natural appeal of this algorithm, it was considered in the game theory literature for over 50 years. It is not difficult to show that for linear cost functions, the Follow The Leader algorithm does not attain any non-trivial regret guarantee (in the

[4]In case of degenerate cost functions, $\mathbf{x}_t$ may not be uniquely defined. In such cases we let $\mathbf{x}_t \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{\tau=1}^{t-1} f_\tau(\mathbf{x})$.

Follow The Leader.

Inputs: convex set $\mathcal{P} \subset \mathbb{R}^n$, initial $\mathbf{x}_1 \in \mathcal{P}$.

- In iteration 1, use $\mathbf{x}_1 \in \mathcal{P}$.
- In iteration $t > 1$: use point

$$\mathbf{x}_t = \arg\max_{\mathbf{x}\in\mathcal{P}} \sum_{\tau=1}^{t-1} f_\tau(\mathbf{x})$$

Figure 2.3: The Follow The Leader algorithm definition.

worst case it can be $\Omega(T)$ if the cost functions are chosen adversarially). However, already in 1957 Hannan [Han57] proposed a randomized variant of FTL, called perturbed-follow-the-leader, which attained $O(\sqrt{T})$ regret in the online game playing setting for linear functions over the simplex. [5]

As we show later, this regret bound is optimal. Merhav and Feder [MF92] extend the FTL approach to strictly convex cost functions over the simplex, and prove that for such functions FTL attains regret which is logarithmic in the number of iterations. Similar results were obtained by Cesa-Bianchi and Lugosi [CBL06], and Gaivoronski and Stella [GS00].

A natural question, asked explicitly by Cover and Ordentlich, Kalai and Vempala, and others, is whether follow-the-leader provides any non-trivial guarantee for curved (but not necessarily strictly convex) cost functions ? One application which is not covered by previous results is the problem of portfolio management.

In Chapter 3 We answer this question in the affirmative, and prove that in fact Follow The Leader attains optimal regret for curved functions.

2.3.5 Other Algorithms for Linear cost functions

Given the generality and ubiquitousness of applications for online convex optimization with linear payoff functions, much work has been devoted to various special cases and many fields of study. It is beyond the scope of this thesis to describe all historical developments of these interesting results. The preceding algorithms were selected as the culmination of efforts spanning the last 50 years and more, attain the best theoretical guarantees to date, and are most relevant to our subsequent discussion.

For more historical context and algorithms the reader is referred to the surveys of Fudenberg and Levin [FL99], Foster and Vohra [FV99], Cover [Cov96] and Blum [Blu98].

[5]this algorithm was rediscovered in [KV05], who provide a much simpler analysis and many applications

2.3.6 Cover's Algorithm

In [Cov91] Cover introduced the Universal algorithm for portfolio management, which attains $O(n \log T)$ regret. As detailed in section 2.2.3, portfolio management is a special case of online convex optimization where the underlying set is the simplex and the payoff functions are defined by $f_t(x) = \log(\mathbf{p}^\top \mathbf{r}_t)$, where $\mathbf{r}_t$ is a vector of returns on the market commodities.

The original algorithm was defined in terms of integrals over the simplex. We choose to provide a discretized version which permits an easier analysis. [6] The algorithm is defined in figure 2.4.

Universal

Let S_n^k be the k-discretized n-dimensional simplex, for $k = T^{3n}$:

$$S_n^k \triangleq \left\{ \mathbf{p} | \mathbf{p} \in S_n \ ; \ \forall i \in [n] \quad \sqrt[n]{k} \cdot \mathbf{p}(i) \in \mathbb{Z} \right\}$$

- On period 1, use the uniform portfolio $\mathbf{p}_1 = \frac{1}{n}\mathbf{1}$.
- On period $t > 1$: use portfolio

$$\mathbf{p}_t \triangleq \frac{\sum_{\mathbf{p} \in S_n^k} \mathbf{p} \cdot \prod_{i=1}^{t-1} \mathbf{p}^\top \mathbf{r}_i}{\sum_{\mathbf{p} \in S_n^k} \prod_{i=1}^{t-1} \mathbf{p}^\top \mathbf{r}_i}$$

Figure 2.4: The Universal algorithm for portfolio management.

As can be seen, Cover's algorithm takes the portfolio at day t to be a convex combinations of all portfolios in the set S_n^k, weighted according to their performance so far. The performance of this algorithm, as observed by Cover, is the average performance over all portfolios in the set S_n^k. Formally,

$$\prod_{t=1}^{T} \mathbf{p}_t^\top \mathbf{r}_t = \prod_{t=1}^{T} \frac{\sum_{\mathbf{p} \in S_n^k} \mathbf{p} \cdot \prod_{i=1}^{t-1} \mathbf{p}^\top \mathbf{r}_i}{\sum_{\mathbf{p} \in S_n^k} \prod_{i=1}^{t-1} \mathbf{p}^\top \mathbf{r}_i} \cdot \mathbf{r}_t = \prod_{t=1}^{T} \frac{\sum_{\mathbf{p} \in S_n^k} \prod_{i=1}^{t} \mathbf{p}^\top \mathbf{r}_i}{\sum_{\mathbf{p} \in S_n^k} \prod_{i=1}^{t-1} \mathbf{p}^\top \mathbf{r}_i} = \frac{1}{|S_n^k|} \sum_{\mathbf{p} \in S_n^k} \prod_{i=1}^{T} \mathbf{p}^\top \mathbf{r}_i \tag{2.5}$$

This observation suffices to prove Universal's performance:

Theorem 2.1 (Cover, discretized version)**.** *For any sequence of returns* $\{\mathbf{r}_1, ..., \mathbf{r}_t\}$*, denote by* $\mathbf{p}^*$ *the optimal CRP in hindsight. Then*

$$\textit{Regret}_T(\textit{Universal}) = \sum_{t=1}^{T} \log\left(\frac{(\mathbf{p}^*)^\top \mathbf{r}_t}{\mathbf{p}_t^\top \mathbf{r}_t}\right) \leq 3n \log T + O(1)$$

[6] Blum and Kalai [BK97] provide for a much simplified analysis of the original algorithm along the same lines

Proof. By observation (2.5):

$$\prod_{t=1}^{T} \frac{(\mathbf{p}^*)^\top \mathbf{r}_t}{\mathbf{p}_t^\top \mathbf{r}_t} = \frac{\prod_{t=1}^{T} (\mathbf{p}^*)^\top \mathbf{r}_t}{\frac{1}{|S_n^k|}\sum_{\mathbf{p}\in S_n^k} \prod_{t=1}^{T} \mathbf{p}^\top \mathbf{r}_t} \leq |S_n^k| \cdot \min_{\mathbf{p}\in S_n^k} \left\{ \frac{\prod_{t=1}^{T} (\mathbf{p}^*)^\top \mathbf{r}_t}{\prod_{t=1}^{T} \mathbf{p}^\top \mathbf{r}_t} \right\}$$

Taking logs:

$$\begin{aligned}\sum_{t=1}^{T} \log(\mathbf{p}^*)^\top \mathbf{r}_t - \log \mathbf{p}_t^\top \mathbf{r}_t &\leq \log |S_n^k| + \min_{\mathbf{p}\in S_n^k} \left\{ \sum_{t=1}^{T} \log \frac{(\mathbf{p}^*)^\top \mathbf{r}_t}{\mathbf{p}\mathbf{r}_t} \right\} \\ &\leq 3n \log T + T \cdot \min_{\mathbf{p}\in S_n^k} \left(\frac{(\mathbf{p}^*)^\top \mathbf{r}_t}{\mathbf{p}\mathbf{r}_t} - 1 \right)\end{aligned}$$

It remains to bound the second term above. By definition of S_n^k, we can choose $\mathbf{p}$ to be one of the closest points to $\mathbf{p}^*$ in ℓ_∞ as follows (in order for $\mathbf{p}$ to be in S_n^k a few coordinates may need to be increased. This will only improve the bound below):

$$\forall i \in [n] \;.\; \mathbf{p}(i) \triangleq \begin{cases} T^{-3} \cdot \lceil T^3 \mathbf{p}^*(i) \rceil & \mathbf{p}^*(i) \leq T^{-1} \\ T^{-2} \cdot \lfloor T^2 \mathbf{p}^*(i) \rfloor & o/w \end{cases}$$

For this $\mathbf{p}$ it holds that

$$\forall \mathbf{r} \;.\; \exists i \in [n] \;.\; \frac{\mathbf{r}^\top \mathbf{p}^*}{\mathbf{r}^\top \mathbf{p}} \leq \frac{\mathbf{p}^*(i)}{\mathbf{p}(i)} \leq \frac{T^{-1}}{T^{-1} - T^{-2}} \leq 1 + \frac{2}{T}$$

And the theorem follows. □

Remark: No attempt was made to optimize the constant 3 above. Indeed, Cover's original paper had a better (and in fact optimal) constant.

2.4 Linear versus curved cost functions

Much of the work in online game playing focused on the case where the payoff functions are linear and the underlying convex set is the set of all distributions over n items, namely the n-dimensional simplex. This special case is particularly interesting in machine learning since it generalizes the problems of prediction from expert advice and online zero sum game playing.

Optimal algorithms, both in terms of regret and computational complexity, are known for this case. The algorithms of Freund and Schapire [FS97] attain $O(\sqrt{T \log n})$ regret and run in time $O(n)$.

For the more general case, efficient algorithms which attain $O(GD\sqrt{T})$ regret were described by Zinkevich [Zin03]. However, an observation made by Zinkevich himself is that linear cost functions are in some sense the "hardest" to optimize over. In fact, his algorithm reduces the general case by approximating convex functions by linear functions, and then applies an algorithm for the linear case.

A complementary fact we prove in the following section is that any algorithm for online game playing incurs $\Omega(GD\sqrt{T})$ in the general case.

What remains then to be shown ? A hint is given by Cover's algorithm: for logarithmic payoff function his algorithm, albeit being inefficient, attains $O(n \log T)$ regret — potentially exponentially lower ! Is it possible that curved, rather than linear, functions allow for more efficient algorithms in general ?

In the next chapter we answer this question on the affirmative. Our main insight is to exploit the curvature of the cost functions, using the second derivative, in order to obtain logarithmic (in T) regret. Surprisingly, unlike Cover, we do not need to sacrifice computational efficiency: the algorithm runs in time which is polynomial in n and independent of the number of iterations. For many applications (i.e. portfolio management) the running time is $O(n^3)$.

2.5 Lower bounds

The following theorem shows that Zinkevich's online gradient descent algorithm is essentially tight in terms of regret guarantees.

Theorem 2.3. *Any algorithm for online convex optimization incurs $\Omega(DG\sqrt{T})$ regret for linear cost functions. This is true even if the adversary is non-malicious, i.e. the cost functions come from a distribution.*

Proof. Consider an instance of online game playing where the convex set $\mathcal{P}$ is the n-dimensional hypercube, i.e.

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n \ , \ \|\mathbf{x}\|_\infty \leq 1\}$$

There are 2^n cost functions, one for each vertex $\mathbf{v} \in \{\pm 1\}^n$, defined as

$$\forall \mathbf{v} \in \{\pm 1\}^n \ . \ f_{\mathbf{v}}(\mathbf{x}) = \mathbf{v}^\top \mathbf{x}$$

That is, the cost functions are linear - inner products with the vectors which are the vertices of the hypercube. Notice that both the diameter of $\mathcal{P}$ and the gradients of the cost functions are

$$G = \sqrt{\sum_{i=1}^{n} (\pm 1)^2} = \sqrt{n} \ , \ D = \sqrt{\sum_{i=1}^{n} 2^2} = 2\sqrt{n}$$

The cost functions are chosen each iteration uniformly at random from the set $\{f_{\mathbf{v}}\}$. Denote by v_t the vertex chosen at iteration t. By uniformity and independence, for any t and $\mathbf{x}_t$ chosen online $\mathbf{E}_{\mathbf{v}}[f_t(\mathbf{x}_t)] = \mathbf{E}_{\mathbf{v}}[\mathbf{x}_t^\top \mathbf{v}_t] = 0$. However,

$$\mathbf{E}_{\mathbf{v}_1,\ldots,\mathbf{v}_T}[\min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^{T} f_t(\mathbf{x})] = \mathbf{E}[\min_{\mathbf{x}\in\mathcal{P}} \sum_{i\in[n]} \sum_{t=1}^{T} \mathbf{v}_t(i) \cdot \mathbf{x}(i)] = n\mathbf{E}[-\left|\sum_{t=1}^{T} \mathbf{v}_t(1)\right|] = -\Omega(n\sqrt{T})$$

□

Chapter 3

The Online Newton Method

In this chapter we describe the Online Newton Step algorithm, proposed in [HKKA06].

The intuition behind the Online Newton Step algorithm stem from new observations regarding the well studied *follow-the-leader* method, which is detailed in the second section of this Chapter. However, once the connection to the Newton-Raphson method is realized, the original analysis can be simplified, and the analysis we present does not bear any resemblance to the typical analysis of "follow-the-leader" type algorithms.

After analyzing the Online Newton Step algorithm and its connection to follow-the-leader, we describe two other new algorithms, which achieve logarithmic regret for curved payoff functions. We proceed to discuss efficient implementation of projections onto convex sets, which are important building blocks of both old and new online convex optimization algorithms. We conclude this chapter with experimental results for portfolio management, for which we applied many of the online convex optimization algorithms described both in this and the previous chapters.

3.1 Comparison of new and old results

Recall the online convex optimization setting: there is a fixed closed, bounded and non-empty convex set $\mathcal{P} \subset \mathbb{R}^n$ and an *arbitrary, unknown* sequence of convex cost functions $f_1, f_2, \ldots : \mathcal{P} \to \mathbb{R}$. The online player must make a sequence of decisions, where the t^{th} decision is a selection of a point $\mathbf{x}_t \in \mathcal{P}$ and there is a cost of $f_t(\mathbf{x}_t)$ on period t. However, $\mathbf{x}_t$ is chosen with only the knowledge of the set $\mathcal{P}$, previous points $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$, and the previous functions $f_1, \ldots, f_{t-1}$. Our performance measure is *regret*:

$$\text{Regret}_T \triangleq \sup_{f_1,\ldots,f_T} \left\{ \sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}^* \in \mathcal{P}} \sum_{t=1}^{T} f_t(\mathbf{x}^*) \right\} \tag{3.1}$$

Figure 3.1 summarizes previous and new algorithms for online convex optimization. The performance parameters considered include: n is the dimension of the convex set $\mathcal{P}$, T is the number of iterations, G is a bound on the norm of the cost functions (either G for ℓ_2 norm or G_∞ for the ℓ_∞ norm), H is a lower bound on the smallest eigenvalue of the Hessians of the cost functions and α is a constant such that the cost functions are

α-exp-concave. See section 2.3.1 for a detailed definition of the performance parameters. The O notation for the regret bounds hides constant factors. For the running time bounds the $\tilde{O}$ notation hides constant factors as well as polylogarithmic factors in n, T, G, H, D, α.

In the running time column, T_{proj} stands for the time it takes to compute a projection onto the underlying convex set (see section 3.5). Similarly, T^g_{proj} stands for the time to compute a generalized projection. All algorithms assume an oracle that given a point in the convex set returns the value of the cost function on that point and/or the gradient of the function.

Algorithm	Regret bound	Running time	Citations
OGD	$O(GD\sqrt{T})$	$\tilde{O}(n) + T_{proj}$	[Zin03]
MW	$O(G_\infty\sqrt{T\log n})$	$\tilde{O}(n)$	[FS99]
perturbed FTL	$O(G_\infty\sqrt{T\log n})$	$\tilde{O}(n)$	[Han57, KV05]
Universal	$O(n\log T)$	$\tilde{O}(T^n), poly(T,n)$	[Cov91, KV03]
convex OGD	$O(\frac{G^2}{H}\log T)$	$\tilde{O}(n) + T_{proj}$	[HKKA06]
ONS	$O((\frac{1}{\alpha} + GD)n\log T)$	$\tilde{O}(n^2) + T^g_{proj}$	[HKKA06]
FTAL	$O((\frac{1}{\alpha} + GD)n\log T)$	$poly(n)$	[AH05, HKKA06]
EWOO	$O(\frac{n}{\alpha}\log T)$	$poly(T,n)$	[HKKA06]

Figure 3.1: Previous and New algorithms for online convex optimization.

The upper part of the figure details previous results whereas the lower part our contributions. The main advantage of our algorithms is the logarithmic dependence of the regret on the number of iterations. The only previous algorithm which attained a logarithmic regret bound for a setting general enough to include portfolio management was Cover's algorithm. However, as noted in the previous chapters, for more restricted settings logarithmic regret was attained using variants of Follow The Leader [MF92, GS00, CBL06]. The computational complexity of these algorithms is $poly(n, T)$, whereas the running time of most of our algorithms is independent from the number of iterations.

Remarks: The Multiplicative Weights algorithm does not apply to the full online convex optimization framework, but rather to the special case where the underlying convex set is the simplex. Cover's algorithm, Universal, only applies to the case of portfolio management. The Perturbed FTL algorithm only applies to the case where the payoff functions are linear.

3.2 Online Newton Step

If previous algorithms such as Online Gradient Descent and Multiplicative Weights are the analogues of the Gradient Descent optimization method for the online setting, then Online Newton Step is the online analogue of the Newton-Raphson method. The main difference is that Online Newton Step is based on **second order information**, i.e. the second derivatives of the cost functions, whereas previous algorithms only exploit first order information. Surprisingly, the information about the second derivatives is used only in the analysis, the algorithm itself uses only the gradients.

The Online Newton Step algorithm detailed in figure 3.2. The point chosen by the algorithm for a given iteration is the point chosen in the previous iteration added an additional vector. Whereas for the Online Gradient Descent algorithm this added vector is the gradient of the previous cost function, for Online Newton Step this vector is different: it is reminiscent to the direction in which the Newton-Raphson method would proceed if it were an offline optimization problem for the previous cost function. The Newton-Raphson algorithm would move in the direction of the vector which is the inverse Hessian multiplied by the gradient. In our case this direction is $A_t^{-1}\nabla_t$, and the matrix A_t is related to the Hessian as will be shown in the analysis.

Since just adding a multiple of the Newton vector to the current point may result in a point outside the convex set, we project back into the set to obtain $\mathbf{x}_t$. This projection is somewhat different than the standard projection used by Online Gradient Descent in section 2.3.2. It is the projection according to the norm defined by the matrix A_t, rather than the Euclidean norm. The reason for using this projection is technical, and will be pointed out in the analysis.

ONS

- In iteration 1, use an arbitrary point $\mathbf{x}_1 \in \mathcal{P}$.
- Let $\beta = \frac{1}{2}\min\{\frac{1}{4GD}, \alpha\}$. In iteration $t > 1$, use point:

$$\mathbf{x}_t = \Pi_{\mathcal{P}}^{\mathbf{A}_{t-1}}\left(\mathbf{x}_{t-1} + \frac{1}{\beta}\mathbf{A}_{t-1}^{-1}\nabla_{t-1}\right)$$

where $\nabla_\tau = \nabla f_\tau(\mathbf{x}_\tau)$, $\mathbf{A}_t = \sum_{i=1}^{t}\nabla_i\nabla_i^\top + \varepsilon\mathbf{I}_n$, $\varepsilon = \frac{1}{\beta^2 D^2}$, and $\Pi_{\mathcal{P}}^{\mathbf{A}_{t-1}}$ is the projection in the norm induced by $\mathbf{A}_{t-1}$, viz.,

$$\Pi_{\mathcal{P}}^{\mathbf{A}_{t-1}}(\mathbf{y}) = \arg\min_{\mathbf{x}\in\mathcal{P}} \; (\mathbf{y}-\mathbf{x})^\top\mathbf{A}_{t-1}(\mathbf{y}-\mathbf{x})$$

Figure 3.2: The Online Newton Step algorithm.

The following theorem bounds the regret of Online Newton Step. The intuition which led to this theorem appears in the next section on follow-the-leader and its surprising connection to the Newton method. The Online Newton Step algorithm and the theorem itself appeared in Hazan et al. [HKKA06]. However, the proof we present hereby was not published elsewhere.

Theorem 3.1.

$$\text{Regret}_T(\text{ONS}) \;\le\; 5\left(\frac{1}{\alpha} + GD\right) n \log T.$$

We begin with a few lemmas. The first lemma is used to approximate the cost functions up to the second order. Using the Taylor series we have $f(\mathbf{x}) = f(\mathbf{y}) + \nabla f(\mathbf{y})(\mathbf{x}-\mathbf{y}) +$

$\frac{1}{2}(\mathbf{x}-\mathbf{y})^\top \nabla^2 f(\zeta)(\mathbf{x}-\mathbf{y})$ for some ζ on the line between $\mathbf{x}$ and $\mathbf{y}$. Instead of using this approximation, we use a somewhat stronger approximation in which the Hessian of the cost function is not used, but rather only the gradient. Such an approximation is possible because we assume that the cost functions are α-exp-concave.

Lemma 3.2. *For a function $f : \mathcal{P} \to \mathbb{R}$, where $\mathcal{P}$ has diameter D, such that $\forall \mathbf{x} \in \mathcal{P}$. $\|\nabla f(\mathbf{x})\| \le G$ and $\exp(-\alpha f(\mathbf{x}))$ is concave, the following holds for $\gamma \le \frac{1}{2}\min\{\frac{1}{4GD}, \alpha\}$:*

$$\forall x, y \in \mathcal{P} : \; f(\mathbf{x}) \ge f(\mathbf{y}) + \nabla f(\mathbf{y})^\top(\mathbf{x}-\mathbf{y}) + \frac{\gamma}{2}(\mathbf{x}-\mathbf{y})^\top \nabla f(\mathbf{y}) \nabla f(\mathbf{y})^\top (\mathbf{x}-\mathbf{y})$$

Proof. Since $\exp(-\alpha f(\mathbf{x}))$ is concave and $2\gamma \le \alpha$, the function $h(\mathbf{x}) \triangleq \exp(-2\gamma f(\mathbf{x}))$ is also concave. Then by the concavity of $h(\mathbf{x})$,

$$h(\mathbf{x}) \;\le\; h(\mathbf{y}) + \nabla h(\mathbf{y})^\top (\mathbf{x}-\mathbf{y}).$$

Plugging in $\nabla h(\mathbf{y}) = -2\gamma \exp(-2\gamma f(\mathbf{y}))\nabla f(\mathbf{y})$ gives,

$$\exp(-2\gamma f(\mathbf{x})) \;\le\; \exp(-2\gamma f(\mathbf{y}))[1 - 2\gamma \nabla f(\mathbf{y})^\top (\mathbf{x}-\mathbf{y})].$$

Simplifying

$$f(\mathbf{x}) \ge f(\mathbf{y}) - \frac{1}{2\gamma}\log[1 - 2\gamma \nabla f(\mathbf{y})^\top(\mathbf{x}-\mathbf{y})].$$

Next, note that $|2\gamma \nabla f(\mathbf{y})^\top(\mathbf{x}-\mathbf{y})| \;\le\; 2\gamma GD \le \frac{1}{4}$ and that for $|z| \le \frac{1}{4}$, $-\log(1-z) \ge z + \frac{1}{4}z^2$. Applying the inequality for $z = 2\gamma \nabla f(\mathbf{y})^\top(\mathbf{x}-\mathbf{y})$ implies the lemma. □

Using this lemma we can bound the regret of Online Newton Step by the following expression

Lemma 3.3. *The regret of* Online Newton Step *is bounded by*

$$\text{Regret}_T(\text{ONS}) \;\le\; 4\left(\frac{1}{\alpha} + GD\right)(\sum_{t=1}^{T} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + 2).$$

Proof. Let $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T f_t(\mathbf{x})$ be the best decision in hindsight. By Lemma 3.2, we have

$$f_t(\mathbf{x}^*) - f_t(\mathbf{x}_t) \;\le\; R_t \triangleq \nabla_t^\top(\mathbf{x}^* - \mathbf{x}_t) - \frac{\beta}{2}(\mathbf{x}^* - \mathbf{x}_t)^\top \nabla_t \nabla_t^\top (\mathbf{x}^* - \mathbf{x}_t) \tag{3.2}$$

for $\beta = \frac{1}{2}\min\{\frac{1}{4GD}, \alpha\}$. For convenience, define $\mathbf{y}_{t+1} = \mathbf{x}_t + \frac{1}{\beta}\mathbf{A}_t^{-1}\nabla_t$ so that according to the update rule of the algorithm $\mathbf{x}_{t+1} = \Pi_{S_n}^{\mathbf{A}_t}(\mathbf{y}_{t+1})$. Now, by the definition of $\mathbf{y}_{t+1}$:

$$\mathbf{y}_{t+1} - \mathbf{x}^* = \mathbf{x}_t - \mathbf{x}^* + \frac{1}{\beta}\mathbf{A}_t^{-1}\nabla_t, \text{ and} \tag{3.3}$$

$$\mathbf{A}_t(\mathbf{y}_{t+1} - \mathbf{x}^*) = \mathbf{A}_t(\mathbf{x}_t - \mathbf{x}^*) + \frac{1}{\beta}\nabla_t. \tag{3.4}$$

Multiplying the transpose of (3.3) by (3.4) we get

$$(\mathbf{y}_{t+1} - \mathbf{x}^*)^\top \mathbf{A}_t (\mathbf{y}_{t+1} - \mathbf{x}^*) =$$
$$(\mathbf{x}_t - \mathbf{x}^*)^\top \mathbf{A}_t (\mathbf{x}_t - \mathbf{x}^*) - \frac{2}{\beta} \nabla_t^\top (\mathbf{x}^* - \mathbf{x}_t) + \frac{1}{\beta^2} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t. \tag{3.5}$$

Since $\mathbf{x}_{t+1}$ is the projection of $\mathbf{y}_{t+1}$ in the norm induced by $\mathbf{A}_t$, it is a well known fact that (see section 3.5 lemma 3.9)

$$(\mathbf{y}_{t+1} - \mathbf{x}^*)^\top \mathbf{A}_t (\mathbf{y}_{t+1} - \mathbf{x}^*) \geq (\mathbf{x}_{t+1} - \mathbf{x}^*)^\top \mathbf{A}_t (\mathbf{x}_{t+1} - \mathbf{x}^*)$$

This inequality is the reason for using generalized projections as opposed to standard projections, which were used in the analysis of Online Gradient Descent (see section 2.3.2 equation (2.2)). This fact together with (3.5) gives

$$\nabla_t^\top (\mathbf{x}^* - \mathbf{x}_t) \leq \frac{1}{2\beta} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + \frac{\beta}{2} (\mathbf{x}_t - \mathbf{x}^*)^\top \mathbf{A}_t (\mathbf{x}_t - \mathbf{x}^*) - \frac{\beta}{2} (\mathbf{x}_{t+1} - \mathbf{x}^*)^\top \mathbf{A}_t (\mathbf{x}_{t+1} - \mathbf{x}^*).$$

Now, summing up over $t = 1$ to T we get that

$$\begin{aligned}
\sum_{t=1}^{T} \nabla_t^\top (\mathbf{x}^* - \mathbf{x}_t) &\leq \frac{1}{2\beta} \sum_{t=1}^{T} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + \frac{\beta}{2} (\mathbf{x}_1 - \mathbf{x}^*)^\top \mathbf{A}_1 (\mathbf{x}_1 - \mathbf{x}^*) \\
&+ \frac{\beta}{2} \sum_{t=2}^{T-1} (\mathbf{x}_t - \mathbf{x}^*)^\top (\mathbf{A}_t - \mathbf{A}_{t-1}) (\mathbf{x}_t - \mathbf{x}^*) - \frac{\beta}{2} (\mathbf{x}_{T+1} - \mathbf{x}^*)^\top \mathbf{A}_T (\mathbf{x}_{T+1} - \mathbf{x}^*) \\
&\leq \frac{1}{2\beta} \sum_{t=1}^{T} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + \frac{\beta}{2} \sum_{t=1}^{T} (\mathbf{x}_t - \mathbf{x}^*)^\top \nabla_t \nabla_t^\top (\mathbf{x}_t - \mathbf{x}^*) + \frac{\beta}{2} (\mathbf{x}_1 - \mathbf{x}^*)^\top (\mathbf{A}_1 - \nabla_1 \nabla_1^\top) (\mathbf{x}_1 - \mathbf{x}^*).
\end{aligned}$$

In the last inequality we use the fact that $A_t - A_{t-1} = \nabla_t \nabla_t^\top$. The matrix A_t was defined in the first place to give this equation, so now when looking at $\sum_t R_t$ the terms of the form $(\mathbf{x}_t - \mathbf{x}^*)^\top (A_t - A_{t-1}) (\mathbf{x}_t - \mathbf{x}^*)$ would cancel out:

$$\sum_{t=1}^{T} R_t \leq \frac{1}{2\beta} \sum_{t=1}^{T} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + \frac{\beta}{2} (\mathbf{x}_1 - \mathbf{x}^*)^\top (\mathbf{A}_1 - \nabla_1 \nabla_1^\top) (\mathbf{x}_1 - \mathbf{x}^*).$$

Using the facts that $\mathbf{A}_1 - \nabla_1 \nabla_1^\top = \varepsilon \mathbf{I}_n$ and $\|\mathbf{x}_1 - \mathbf{x}^*\|^2 \leq 2D^2$, and the choice of $\varepsilon = \frac{1}{\beta^2 D^2}$ we get

$$\begin{aligned}
\textit{Regret}_T(\textit{ONS}) &\leq \sum_{t=1}^{T} R_t \leq \frac{1}{2\beta} \sum_{t=1}^{T} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + \varepsilon D^2 \beta \\
&\leq \frac{1}{2\beta} \sum_{t=1}^{T} \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t + \frac{1}{\beta}
\end{aligned}$$

Since $\beta = \frac{1}{2} \min\{\frac{1}{4GD}, \alpha\}$, we have $\frac{1}{\beta} \leq 8(GD + \frac{1}{\alpha})$. This gives the lemma. □

We can now prove Theorem 3.1.

Proof of Theorem 3.1. First we show that the term $\sum_{t=1}^T \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t$ is upper bounded by a telescoping sum. Notice that

$$\nabla_t^\top \mathbf{A}_t^{-1} \nabla_t = A_t^{-1} \bullet \nabla_t \nabla_t^\top = A_t^{-1} \bullet (A_t - A_{t-1})$$

where for matrices $A, B \in \mathbb{R}^{n \times n}$ we denote by $A \bullet B = \sum_{i,j=1}^n A_{ij} B_{ij}$ the inner product of these matrices as vectors in $\mathbb{R}^{n^2}$.

For real numbers $a, b \in \mathbb{R}_+$, the Taylor expansion of the logarithm implies $a^{-1}(a-b) \leq \log \frac{a}{b}$. An analogous fact holds for PSD matrices, i.e. $A^{-1} \bullet (A - B) \leq \log \frac{|A|}{|B|}$, where $|A|$ denotes the determinant of the matrix A (this is proved in Lemma 3.4). Using this fact we have (for convenience we denote $A_0 = \varepsilon I_n$)

$$\begin{aligned} \sum_{t=1}^T \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t &= \sum_{t=1}^T A_t^{-1} \bullet \nabla_t \nabla_t^\top \\ &= \sum_{t=1}^T A_t^{-1} \bullet (A_t - A_{t-1}) \\ &\leq \sum_{t=1}^T \log \frac{|A_t|}{|A_{t-1}|} = \log \frac{|A_T|}{|A_0|} \end{aligned}$$

Since $A_T = \sum_{t=1}^T \nabla_t \nabla_t^\top + \varepsilon I$ and $\|\nabla_t\| \leq G$, the largest eigenvalue of A_T is at most $TG^2 + \varepsilon$. Hence the determinant of A_T can be bounded by $|A_T| \leq (TG^2 + \varepsilon)^n$. Hence (recall that $\varepsilon = \frac{1}{\beta^2 D^2}$ and $\beta = \frac{1}{2} \min\{\frac{1}{4GD}, \alpha\}$)

$$\sum_{t=1}^T \nabla_t^\top \mathbf{A}_t^{-1} \nabla_t \quad \leq \log \left(\tfrac{TG^2 + \varepsilon}{\varepsilon} \right)^n \leq n \log(TG^2 \beta^2 D^2 + 1) \leq n \log T$$

Plugging into lemma 3.3 we obtain

$$\text{Regret}_T(\text{ONS}) \ \leq \ 4 \left(\frac{1}{\alpha} + 4GD \right) (n \log T + 2)$$

which implies the theorem.

$\square$

It remains to prove the technical lemma for PSD matrices used above.

Lemma 3.4. *Let $A \succeq B \succ 0$ be positive definite matrices. Then*

$$A^{-1} \bullet (A - B) \ \leq \ \log \frac{|A|}{|B|}$$

where $|A|$ denotes the determinant of matrix A.

Proof. For any positive definite matrix C, denote by $\lambda_1(C), \lambda_2(C), \ldots, \lambda_n(C)$ its (positive) eigenvalues.

$$\begin{aligned}
A^{-1} \bullet (A - B) &= \mathbf{Tr}(A^{-1}(A-B)) \\
&= \mathbf{Tr}(A^{-1/2}(A-B)A^{-1/2}) \\
&= \mathbf{Tr}(I - A^{-1/2}BA^{-1/2}) \\
&= \sum_{i=1}^{n} \left[1 - \lambda_i(A^{-1/2}BA^{-1/2})\right] && \because \mathbf{Tr}(C) = \sum_{i=1}^{n} \lambda_i(C) \\
&\leq -\sum_{i=1}^{n} \log \left[\lambda_i(A^{-1/2}BA^{-1/2})\right] && \because 1 - x \leq -\log(\mathbf{x}) \\
&= -\log \left[\prod_{i=1}^{n} \lambda_i(A^{-1/2}BA^{-1/2})\right] \\
&= -\log |A^{-1/2}BA^{-1/2}| = \log \frac{|A|}{|B|} && \because |C| = \prod_{i=1}^{n} \lambda_i(C)
\end{aligned}$$

In the last equality we use the facts $|AB| = |A||B|$ and $|A^{-1}| = \frac{1}{|A|}$ for PSD matrices. □

3.2.1 Implementation and running time

The Online Newton Step algorithm requires $O(n^2)$ space to store the matrix A_t. Every iteration requires the computation of the matrix A_t^{-1}, the current gradient, a matrix-vector product and possibly a projection onto the undelying convex set $\mathcal{P}$.

A naïve implementation would require computing the inverse of the matrix A_t every iteration. However, in case A_t is invertible, the matrix inversion lemma [Bro05] states that for invertible matrix A and vector $\mathbf{x}$

$$(A + \mathbf{x}\mathbf{x}^\top)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{x}\mathbf{x}^\top A^{-1}}{1 + \mathbf{x}^\top A^{-1}\mathbf{x}}$$

Thus, given A_{t-1}^{-1} and ∇_t one can compute A_t^{-1} in time $O(n^2)$ using only matrix-vector and vector-vector products.

The Online Newton Step algorithm also needs to make projections onto $\mathcal{P}$, but of a slightly different nature than Online Gradient Descent and other online convex optimization algorithms. The required projection, denoted by $\Pi_{\mathcal{P}}^{A_t}$, is in the vector norm induced by the matrix A_t, viz. $\|\mathbf{x}\|_{A_t} = \sqrt{\mathbf{x}^\top A_t \mathbf{x}}$. It is equivalent to finding the point $\mathbf{x} \in \mathcal{P}$ which minimizes $(\mathbf{x} - \mathbf{y})^\top A_t(\mathbf{x} - \mathbf{y})$ where y is the point we are projecting. This is a convex program which can be solved up to any degree of accuracy in polynomial time, see section 3.5.

Modulo the computation of generalized projections, the Online Newton Step algorithm can be implemented in time and space $O(n^2)$. In addition, the only information required is the gradient at each step (and the exp-concavity constant of the payoff functions).

3.3 The connection to the Follow The Leader algorithm

The basic follow-the-leader (FTL) method simply chooses on period t the single fixed decision that would have been the best to use on the previous $t-1$ periods. This corresponds to choosing $\mathbf{x}_t = \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{\tau=1}^{t-1} f_\tau(\mathbf{x})$ (see figure 2.3). By itself, this basic method fails to provide sub-linear regret even for linear cost functions.

As explained in section 2.3.4, Hannan [Han57] proposed and analyzed a randomized variation on the basic approach, which does attain sublinear regret for linear payoff functions over the simplex. Merhav and Feder [MF92] show that FTL achieves logarithmic regret for strictly convex functions on the simplex (see also Cesa-Bianchi and Lugosi [CBL06], and Gaivoronski and Stella [GS00]). This left open the question of convex (not necessarily strictly convex) cost functions, such as for the portfolio management problem.

In this section, we show that a simple deterministic FTL variant called Follow The Approximate Leader (FTAL), described in figure 3.3 [1], does provide sublinear regret for convex cost functions, including logarithmic payoff functions as for portfolio management, over any convex set. In fact, it attains regret which is logarithmic in the number of iterations. This answers questions posed by Cover and Ordentlich [Cov91] and independently Kalai and Vempala [KV05]. The FTAL algorithm also has computational advantages over the standard FTL algorithm, which are detailed in the last part of this section.

Follow The Approximate Leader.
Inputs: convex set $\mathcal{P} \subset \mathbb{R}^n$, and the parameter β.

- In iteration 1, use an arbitrary point $\mathbf{x}_1 \in \mathcal{P}$.
- In iteration t, use the point $\mathbf{x}_t$ defined as

$$\mathbf{x}_t \triangleq \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{\tau=1}^{t-1} \tilde{f}_\tau(\mathbf{x})$$

  where for $\tau = 1, \ldots, t-1$, let $\nabla_\tau = \nabla f_\tau(\mathbf{x}_\tau)$ and

$$\tilde{f}_\tau(\mathbf{x}) \triangleq f_\tau(\mathbf{x}_\tau) + \nabla_\tau^\top(\mathbf{x}-\mathbf{x}_\tau) + \frac{\beta}{2}(\mathbf{x}-\mathbf{x}_\tau)^\top \nabla_\tau \nabla_\tau^\top (\mathbf{x}-\mathbf{x}_\tau)$$

Figure 3.3: The Follow The Approximate Leader algorithm

The standard approach to analyze FTL-type algorithms proceeds by inductively showing,

$$\text{Regret}_T(\mathcal{A}) = \sum_{t=1}^T f_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T f_t(\mathbf{x}) \leq \sum_{t=1}^T f_t(\mathbf{x}_t) - f_t(\mathbf{x}_{t+1}) \tag{3.6}$$

[1] In case of degenerate cost functions the best decision in hindsight may not unique. In this case we can define $\mathbf{x}_t \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{\tau=1}^{t-1} \tilde{f}_\tau(\mathbf{x})$. The results of this section extend with this definition appropriately.

The standard analysis then proceeds by showing that the leader doesn't change too much, i.e. $\mathbf{x}_t \approx \mathbf{x}_{t+1}$, which in turn implies low regret. Such is the analysis deployed by Hannan in his seminal paper [Han57], which was then simplified by Kalai and Vempala in [KV05]. The work of [MF92, GS00, CBL06] also proceeds along these lines. In fact, as observed in [HKKA06], the Online Gradient Descent algorithm can also be analyzed as a follow-the-leader variant in a similar fashion.

Our analysis does not follow the above paradigm directly, but rather shows *average stability*, i.e. that the leader doesn't change much ($\mathbf{x}_t \approx \mathbf{x}_{t+1}$) on average. This focus on the average case is necessary, since even for portfolio management one can show that $\|\mathbf{x}_t - \mathbf{x}_{t-1}\|$ may be large in the worst case, regardless of t. However, using amortized analysis with the potential function of [AH05], one can show that one the average $\|\mathbf{x}_t - \mathbf{x}_{t-1}\|$ is on the order of $\frac{1}{t}$.

The analysis in this section is based on Agarwal and Hazan [AH05] and Hazan et al. [HKKA06].

Theorem 3.5. *For $\beta = \frac{1}{2}\min\{\frac{1}{4GD}, \alpha\}$,*

$$\mathit{Regret}_T(FTAL) \leq 16\left(\frac{1}{\alpha} + GD\right) n \log T$$

Proof. Let $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T f_t(\mathbf{x})$. First, observe that by Lemma 3.2, the function $\tilde{f}_t(\mathbf{x})$ defined by the Follow The Approximate Leader algorithm satisfies $\tilde{f}_t(\mathbf{x}_t) = f_t(\mathbf{x}_t)$ and $\tilde{f}_t(\mathbf{x}) \leq f_t(\mathbf{x})$ for all $x \in \mathcal{P}$. This implies

$$\begin{aligned} \text{Regret}_T(FTAL) &= \sum_{t=1}^T f_t(\mathbf{x}_t) - \sum_{t=1}^T f_t(\mathbf{x}^*) \qquad (3.7)\\ &\leq \sum_{t=1}^T \tilde{f}_t(\mathbf{x}_t) - \sum_{t=1}^T \tilde{f}_t(\mathbf{x}^*) \\ &\leq \sum_{t=1}^T \tilde{f}_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T \tilde{f}_t(\mathbf{x}) \end{aligned}$$

We proceed according to the approach outlined above. As a first step, we prove inductively that

$$\sum_{t=1}^T \tilde{f}_t(\mathbf{x}_{t+1}) \leq \min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T \tilde{f}_t(\mathbf{x})$$

For $t = 1$ the two are equal by definition of Follow The Leader. Assume correctness for $T - 1$, and

$$\begin{aligned} \sum_{t=1}^T \tilde{f}_t(\mathbf{x}_{t+1}) &\leq \min_{\mathbf{x}\in\mathcal{P}} \textstyle\sum_{t=1}^{T-1} \tilde{f}_t(\mathbf{x}) + \tilde{f}_T(\mathbf{x}_{T+1}) && \text{by induction hyphthesis} \\ &\leq \textstyle\sum_{t=1}^{T-1} \tilde{f}_t(\mathbf{x}_{T+1}) + \tilde{f}_T(\mathbf{x}_{T+1}) \\ &= \min_{\mathbf{x}\in\mathcal{P}} \textstyle\sum_{t=1}^{T} \tilde{f}_t(\mathbf{x}) && \text{by definition of Follow The Leader} \end{aligned}$$

Combining (3.7) and the above we get,

$$\text{Regret}_T(FTAL) \;=\; \sum_{t=1}^{T} \tilde{f}_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^{T} \tilde{f}_t(\mathbf{x}) \;\le\; \sum_{t=1}^{T} \left[\tilde{f}_t(\mathbf{x}_t) - \tilde{f}_t(\mathbf{x}_{t+1})\right] \tag{3.8}$$

The inequality (3.8) implies that it suffices to bound the expression $\sum_{t=1}^{T}\left[\tilde{f}_t(\mathbf{x}_t) - \tilde{f}_t(\mathbf{x}_{t+1})\right]$, which we do in the lemma below.

□

Lemma 3.6.

$$\sum_{t=1}^{T} \tilde{f}_t(\mathbf{x}_t) - \tilde{f}_t(\mathbf{x}_{t+1}) \le 16\left(\frac{1}{\alpha} + GD\right) n \log T$$

Proof. For the sake of readability, we introduce some notation. Define the function $F_t \triangleq \sum_{\tau=1}^{t-1} \tilde{f}_\tau$. Note that $\nabla f_t(\mathbf{x}_t) = \nabla \tilde{f}_t(\mathbf{x}_t)$ by the definition of $\tilde{f}_t$, so we will use the same notation ∇_t to refer to both. Finally, let Δ be the forward difference operator, for example, $\Delta\mathbf{x}_t = (\mathbf{x}_{t+1} - \mathbf{x}_t)$ and $\Delta\nabla F_t(\mathbf{x}_t) = (\nabla F_{t+1}(\mathbf{x}_{t+1}) - \nabla F_t(\mathbf{x}_t))$.

We use the gradient bound, which follows from the convexity of $\tilde{f}_t$:

$$\tilde{f}_t(\mathbf{x}_t) - \tilde{f}_t(\mathbf{x}_{t+1}) \;\le\; -\nabla\tilde{f}_t(\mathbf{x}_t)^\top(\mathbf{x}_{t+1} - \mathbf{x}_t) \;=\; -\nabla_t^\top \Delta\mathbf{x}_t \tag{3.9}$$

The gradient of F_{t+1} can be written as:

$$\nabla F_{t+1}(\mathbf{x}) = \sum_{\tau=1}^{t} \left[\nabla_\tau + \beta\nabla_\tau\nabla_\tau^\top(\mathbf{x} - \mathbf{x}_\tau)\right] \tag{3.10}$$

Therefore, for $A_t \triangleq \sum_{\tau=1}^{t} \nabla_\tau\nabla_\tau^\top$

$$\nabla F_{t+1}(\mathbf{x}_{t+1}) - \nabla F_{t+1}(\mathbf{x}_t) \;=\; \beta\sum_{\tau=1}^{t} \nabla_\tau\nabla_\tau^\top\Delta\mathbf{x}_t \;=\; \beta A_t\Delta\mathbf{x}_t \tag{3.11}$$

The LHS of (3.11) is

$$\nabla F_{t+1}(\mathbf{x}_{t+1}) - \nabla F_{t+1}(\mathbf{x}_t) \;=\; \Delta\nabla F_t(\mathbf{x}_t) - \nabla_t \tag{3.12}$$

Putting (3.11) and (3.12) together, and adding $\varepsilon\beta\Delta\mathbf{x}_t$ we get

$$\beta(A_t + \varepsilon I_n)\Delta\mathbf{x}_t = \Delta\nabla F_t(\mathbf{x}_t) - \nabla_t + \varepsilon\beta\Delta\mathbf{x}_t \tag{3.13}$$

Pre-multiplying by $-\frac{1}{\beta}\nabla_t^\top(A_t + \varepsilon I_n)^{-1}$, we get an expression for the gradient bound (3.9):

$$\begin{aligned} -\nabla_t^\top\Delta\mathbf{x}_t \;&=\; -\frac{1}{\beta}\nabla_t^\top(A_t + \varepsilon I_n)^{-1}[\Delta\nabla F_t(\mathbf{x}_t) - \nabla_t + \varepsilon\beta\Delta\mathbf{x}_t] \\ &=\; -\frac{1}{\beta}\nabla_t^\top(A_t + \varepsilon I_n)^{-1}[\Delta\nabla F_t(\mathbf{x}_t) + \varepsilon\beta\Delta\mathbf{x}_t] + \frac{1}{\beta}\nabla_t^\top(A_t + \varepsilon I_n)^{-1}\nabla_t \end{aligned} \tag{3.14}$$

Claim 3.7. *The first term of (3.14) can be bounded as follows:*

$$-\frac{1}{\beta}\nabla_t^\top (A_t+\varepsilon I_n)^{-1}[\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t] \;\le\; \varepsilon\beta D^2$$

Proof. Since $\mathbf{x}_\tau$ minimizes F_τ over $\mathcal{P}$, the following holds for any point $x \in \mathcal{P}$ (see [BV04]).

$$\nabla F_\tau(\mathbf{x}_\tau)^\top(\mathbf{x}-\mathbf{x}_\tau) \;\ge\; 0 \tag{3.15}$$

Using (3.15) for $\tau = t$ and $\tau = t+1$, we get

$$0 \;\le\; \nabla F_{t+1}(\mathbf{x}_{t+1})^\top(\mathbf{x}_t-\mathbf{x}_{t+1}) + \nabla F_t(\mathbf{x}_t)^\top(\mathbf{x}_{t+1}-\mathbf{x}_t) \;=\; -[\Delta\nabla F_t(\mathbf{x}_t)]^\top\Delta\mathbf{x}_t$$

Reversing the inequality and adding $\varepsilon\beta\|\Delta\mathbf{x}_t\|^2 = \varepsilon\beta\Delta x_t^\top\Delta\mathbf{x}_t$, we get

$$\begin{aligned}
\varepsilon\beta\|\Delta\mathbf{x}_t\|^2 \;&\ge\; [\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t]^\top\Delta\mathbf{x}_t\\
&=\; \frac{1}{\beta}[\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t]^\top(A_t+\varepsilon I_n)^{-1}[\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t-\nabla_t]\\
&\qquad\text{(by solving for } \Delta\mathbf{x}_t \text{ in (3.13))}\\
&=\; \frac{1}{\beta}[\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t]^\top(A_t+\varepsilon I_n)^{-1}(\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t)\\
&-\; \frac{1}{\beta}[\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\Delta\mathbf{x}_t]^\top(A_t+\varepsilon I_n)^{-1}\nabla_t\\
&\ge\; -\frac{1}{\beta}[\Delta\nabla F_t(\mathbf{x}_t)+\varepsilon\beta\Delta\mathbf{x}_t]^\top(A_t+\varepsilon I_n)^{-1}\nabla_t\\
&\qquad\text{(since } (A_t+\varepsilon I_n)^{-1}\succeq 0 \Rightarrow \forall\mathbf{x}:\ \mathbf{x}^\top(A_t+\varepsilon I_n)^{-1}\mathbf{x}\ge 0\text{)}
\end{aligned}$$

Finally, since the diameter of $\mathcal{P}$ is D, we have $\varepsilon\beta\|\Delta\mathbf{x}_t\|^2 \le \varepsilon\beta D^2$. □

Applying this claim to (3.14), summing up from $t=1$ to T and using (3.9)

$$\sum_{t=1}^T \tilde{f}_t(\mathbf{x}_t)-\tilde{f}_t(\mathbf{x}_{t+1}) \le \frac{1}{\beta}\sum_{t=1}^T \nabla_t^\top(A_t+\varepsilon I_n)^{-1}\nabla_t + T\beta\varepsilon D^2 \tag{3.16}$$

We now bound (3.16) by applying Lemma 3.4 in a similar way as in the analysis of Online Newton Step.

$$\begin{aligned}
&\textstyle\sum_{t=1}^T \nabla_t^\top(A_t+\varepsilon I_n)^{-1}\nabla_t\\
&= \textstyle\sum_{t=1}^T \nabla_t^\top(\sum_{\tau=1}^t \nabla_\tau\nabla_\tau^\top+\varepsilon I_n)^{-1}\nabla_t\\
&= \textstyle\sum_{t=1}^T (\sum_{\tau=1}^t \nabla_\tau\nabla_\tau^\top+\varepsilon I_n)^{-1}\bullet\nabla_t\nabla_t^\top\\
&\le\; \textstyle\sum_{t=1}^T \log\frac{|\sum_{\tau=1}^t \nabla_\tau\nabla_\tau^\top+\varepsilon I_n|}{|\sum_{\tau=1}^{t-1} \nabla_\tau\nabla_\tau^\top+\varepsilon I_n|} \qquad\text{by lemma 3.4}\\
&=\; \log\frac{|\sum_{\tau=1}^T \nabla_\tau\nabla_\tau^\top+\varepsilon I_n|}{|\varepsilon I_n|}
\end{aligned}$$

Since $\|\nabla_t\| \leq G$, the eigenvalues of the matrix $\sum_{\tau=1}^{T} \nabla_\tau \nabla_\tau^\top + \varepsilon I_n$ are bounded by $TG^2 + \varepsilon$. Hence $|\sum_{t=1}^{T} \nabla_t \nabla_t^\top + \varepsilon I_n| \leq (TG^2 + \varepsilon)^n$. Plugging into the inequality above and setting $\varepsilon = \frac{1}{\beta^2 D^2 T}$ we get

$$\sum_{t=1}^{T} \nabla_t^\top (A_t + \varepsilon I_n)^{-1} \nabla_t \ \leq n \log \left(\frac{G^2 T + \varepsilon}{\varepsilon} \right) \leq 2n \log(\beta DGT + 1)$$

Combining this with (3.16), and since $\beta = \frac{1}{2} \min\{\frac{1}{4GD}, \alpha\}$, $\varepsilon = \frac{1}{\beta^2 D^2 T}$

$$\begin{aligned} \sum_{t=1}^{T} \tilde{f}_t(\mathbf{x}_t) - \tilde{f}_t(\mathbf{x}_{t+1}) &\leq 2\frac{1}{\beta} n \log(\beta DGT + 1) + T\beta\varepsilon D^2 \\ &\leq 2\frac{1}{\beta} n \log(\beta DGT + 1) + \frac{1}{\beta} \\ &\leq 16 \left(GD + \frac{1}{\alpha} \right) (n \log(T/4 + 1) + 1) \leq 16 \left(GD + \frac{1}{\alpha} \right) n \log T \end{aligned}$$

□

3.3.1 Implementation and running time

The implementation of Follow The Approximate Leader is straightforward: the point $\mathbf{x}_t$ chosen at iteration t is the optimum of the following mathematical program:

$$\mathbf{x}_t = \arg\min_{\mathbf{x} \in \mathcal{P}} \sum_{\tau=1}^{t-1} \tilde{f}_\tau(\mathbf{x})$$

Since the approximate cost functions $\tilde{f}_t$ as well as the underlying set $\mathcal{P}$ are convex, this is a convex program which any general convex optimization algorithm applied to (here is another justification for our assumption that the set $\mathcal{P}$ is convex, see section 2.1.1). Notice that since all $\tilde{f}_t$ are quadratic polynomials, only the sum of the coefficients of these polynomials are required. Hence, the algorithm requires $\tilde{O}(n^2)$ space to store the sum of all gradients and matrices of the form $\nabla_t \nabla_t^\top$. The time and space complexity is thus independent from the number of iterations, in contrast to other previous variants of Follow The Leader.

We note that in practice, we have an excellent starting point to compute $\mathbf{x}_t$ - the optimum of the convex program of the previous iteration $\mathbf{x}_{t-1}$. As shown in the analysis, on the average these two consecutive points are very close.

3.4 Other logarithmic regret algorithms

In this section we describe two simple algorithms which also attain logarithmic regret for certain scenarios of online convex optimization. The ideas behind these algorithms are not new: the first is a variant of Zinkevich's Online Gradient Descent, and the other is an exponential weighting algorithm reminiscent of Cover's algorithm.

Both algorithms have downsides: the Online Gradient Descent algorithm requires strict convexity, and does not apply to portfolio management (unlike Online Newton Step).

The exponential weighting algorithm is very general - it guarantees logarithmic regret in even more general scenarios than Online Newton Step. However, it is not as efficient to implement as Online Newton Step or Online Gradient Descent.

3.4.1 Online Gradient Descent

The first algorithm that achieves regret logarithmic in the number of iterations is a twist on the online gradient descent discussed in section 2.3.2, and defined in figure 2.1. The following theorem, proved by Hazan et al. in [HKKA06], establishes logarithmic bounds on the regret if the cost functions are strictly convex.

The condition that the cost functions are strictly convex is rather strong, and does not apply to many interesting problems with curved cost functions such as portfolio management.

Theorem 3.8. Online Gradient Descent *with step sizes* $\eta_t = \frac{1}{Ht}$ *achieves the following guarantee, for all* $T \geq 1$.

$$Regret_T(OGD) \;\leq\; \frac{G^2}{2H}(1 + \log T)$$

Proof. Let $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T f_t(\mathbf{x})$. Recall the definition of regret (see section 2.1.1)

$$\text{Regret}_T(OGD) \;= \sum_{t=1}^T f_t(\mathbf{x}_t) - \sum_{t=1}^T f_t(\mathbf{x}^*)$$

Define $\nabla_t \triangleq \nabla f_t(\mathbf{x}_t)$. By H-strong convexity, we have,

$$\begin{aligned} f_t(\mathbf{x}^*) &\geq f_t(\mathbf{x}_t) + \nabla_t^\top(\mathbf{x}^* - \mathbf{x}_t) + \frac{H}{2}\|\mathbf{x}^* - \mathbf{x}_t\|^2 \\ 2(f_t(\mathbf{x}_t) - f_t(\mathbf{x}^*)) &\leq 2\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^*) - H\|\mathbf{x}^* - \mathbf{x}_t\|^2 \end{aligned} \tag{3.17}$$

Following Zinkevich's analysis, we upper-bound $\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^*)$. Using the update rule for $\mathbf{x}_{t+1}$ and the properties of projections (see lemma 3.9), we get

$$\|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 \;=\; \|\Pi(\mathbf{x}_t - \eta_{t+1}\nabla_t) - \mathbf{x}^*\|^2 \leq \|\mathbf{x}_t - \eta_{t+1}\nabla_t - \mathbf{x}^*\|^2.$$

The inequality above follows from the properties of projection onto convex sets. Hence,

$$\begin{aligned} \|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2 &\leq \|\mathbf{x}_t - \mathbf{x}^*\|^2 + \eta_{t+1}^2\|\nabla_t\|^2 - 2\eta_{t+1}\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^*) \\ 2\nabla_t^\top(\mathbf{x}_t - \mathbf{x}^*) &\leq \frac{\|\mathbf{x}_t - \mathbf{x}^*\|^2 - \|\mathbf{x}_{t+1} - \mathbf{x}^*\|^2}{\eta_{t+1}} + \eta_{t+1}G^2 \end{aligned} \tag{3.18}$$

Sum up (3.18) from $t = 1$ to T. Set $\eta_{t+1} = 1/(Ht)$, and using (3.17), we have:

$$\begin{aligned} 2\sum_{t=1}^{T} f_t(\mathbf{x}_t) - f_t(\mathbf{x}^*) &\leq \sum_{t=1}^{T} \|\mathbf{x}_t - \mathbf{x}^*\|^2 \left(\frac{1}{\eta_{t+1}} - \frac{1}{\eta_t} - H\right) + G^2 \sum_{t=1}^{T} \eta_{t+1} \\ &= 0 + G^2 \sum_{t=1}^{T} \frac{1}{Ht} \leq \frac{G^2}{H}(1 + \log T) \end{aligned}$$

□

We remark that the Online Gradient Descent algorithm above can be analyzed as a follow-the-leader variant. The proof is somewhat more complicated than the one above, and hence omitted.

3.4.2 Exponentially Weighted Online Optimization

Exponentially Weighted Online Optimization.
Inputs: convex set $\mathcal{P} \subset \mathbb{R}^n$, and the parameter α.

- Define weights $w_t(\mathbf{x}) = \exp(-\alpha \sum_{\tau=1}^{t-1} f_\tau(\mathbf{x}))$
- On iteration t use point $\mathbf{x}_t = \frac{\int_{\mathcal{P}} \mathbf{x}\ w_t(\mathbf{x}) d\mathbf{x}}{\int_{\mathcal{P}} w_t(\mathbf{x}) d\mathbf{x}}$

Figure 3.4: The Exponentially Weighted Online Optimization Algorithm.

In this section we describe our Exponentially Weighted Online Optimization (EWOO) algorithm which gives logarithmic regret for a very general setting of online convex optimization. All that the algorithm requires is that the cost functions be α-exp-concave (not even a bound on the gradient required, as opposed to Online Newton Step). The algorithm does not seem to be directly related to follow-the-leader. Rather, it is related to Cover's algorithm for universal portfolio management.

The downside of this algorithm is its running time. A trivial implementation of Exponentially Weighted Online Optimization would give exponential running time. Kalai and Vempala [KV03] give a randomized polynomial time (polynomial both in n and in T) implementation of Cover's algorithm, based on random sampling techniques. The same techniques can be applied to the Exponentially Weighted Online Optimization algorithm as well. However, the polynomial in the running time is quite large and the overall implementation involved.

Remark: In the implementation of Exponentially Weighted Online Optimization, choosing $\mathbf{x}_t$ at random with density proportional to $w_t(\mathbf{x})$, instead of computing the integral, also guarantees our regret bounds on the expectation. This is the basis for the [KV03] polynomial time implementation.

Theorem 3.1.

$$\mathit{Regret}_T(\mathit{EWOO}) \ \leq \ \frac{1}{\alpha} n(1+\log(1+T)).$$

Proof. Let $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{P}} \sum_{t=1}^T f_t(\mathbf{x})$. Recall the definition of regret (see section 2.1.1)

$$\text{Regret}_T(OGD) \ = \sum_{t=1}^T f_t(\mathbf{x}_t) - \sum_{t=1}^T f_t(\mathbf{x}^*)$$

Let $h_t(\mathbf{x}) = e^{-\alpha f_t(\mathbf{x})}$. The algorithm can be viewed as taking a weighted average over points $\mathbf{x} \in \mathcal{P}$. Hence, by concavity of h_t,

$$h_t(\mathbf{x}_t) \geq \frac{\int_{\mathcal{P}} h_t(\mathbf{x}) \prod_{\tau=1}^{t-1} h_\tau(\mathbf{x}) \ d\mathbf{x}}{\int_{\mathcal{P}} \prod_{\tau=1}^{t-1} h_\tau(\mathbf{x}) \ d\mathbf{x}}.$$

Hence, we have by telescoping product,

$$\prod_{\tau=1}^{t} h_\tau(\mathbf{x}_\tau) \geq \frac{\int_{\mathcal{P}} \prod_{\tau=1}^{t} h_\tau(\mathbf{x}) \ d\mathbf{x}}{\int_{\mathcal{P}} 1 \ d\mathbf{x}} = \frac{\int_{\mathcal{P}} \prod_{\tau=1}^{t} h_\tau(\mathbf{x}) \ d\mathbf{x}}{\text{vol}(\mathcal{P})} \tag{3.19}$$

By definition of $\mathbf{x}^*$ we have $\mathbf{x}^* \in \arg\max_{\mathbf{x}\in\mathcal{P}} \prod_{t=1}^T h_t(\mathbf{x})$. Following [BK97], define nearby points $S \subset \mathcal{P}$ by,

$$S = \left\{ \mathbf{x} \in S \mid \mathbf{x} = \frac{T}{T+1}\mathbf{x}^* + \frac{1}{T+1}\mathbf{y} \ , \ \mathbf{y} \in \mathcal{P} \right\}.$$

By concavity of h_t and the fact that h_t is non-negative, we have that,

$$\forall \mathbf{x} \in S \quad h_t(\mathbf{x}) \geq \frac{T}{T+1} h_t(\mathbf{x}^*).$$

Hence,

$$\forall \mathbf{x} \in S \quad \prod_{\tau=1}^T h_\tau(\mathbf{x}) \geq \left(\frac{T}{T+1}\right)^T \prod_{\tau=1}^T h_\tau(\mathbf{x}^*) \geq \frac{1}{e} \prod_{\tau=1}^T h_\tau(\mathbf{x}^*)$$

Finally, since $S = \mathbf{x}^* + \frac{1}{T+1}\mathcal{P}$ is simply a rescaling of $\mathcal{P}$ by a factor of $1/(T+1)$ (followed by a translation), and we are in n dimensions, $\text{vol}(S) = \text{vol}(\mathcal{P})/(T+1)^n$. Putting this together with equation (3.19), we have

$$\prod_{\tau=1}^T h_\tau(\mathbf{x}_\tau) \geq \frac{\text{vol}(S)}{\text{vol}(\mathcal{P})} \frac{1}{e} \prod_{\tau=1}^T h_\tau(\mathbf{x}^*) \geq \frac{1}{e(T+1)^n} \prod_{\tau=1}^T h_\tau(\mathbf{x}^*).$$

The theorem is obtained by taking logarithms. □

3.5 Projections onto convex sets

Some of the algorithms for online convex optimization described in this and the previous chapters require to compute projections onto convex sets. This correspond to the following computational problem: given a convex set $\mathcal{P} \subseteq \mathbb{R}^n$, and a point $\mathbf{y} \in \mathbb{R}^n$, find the point in the convex set which is closest in Euclidean distance to the given vector, denoted

$$\Pi_{\mathcal{P}}[\mathbf{y}] \triangleq \min_{\mathbf{x} \in \mathcal{P}} \|\mathbf{x} - \mathbf{y}\|_2$$

The Online Newton Step algorithm computes *generalized projections*, which are projections with respect to a norm other than the Euclidean norm, given by a PSD matrix. For a given PSD matrix $A \succeq 0$, a generalized projection of $y \in \mathbb{R}^n$ onto the convex set $\mathcal{P}$ is defined as

$$\Pi_{\mathcal{P}}^{A}[\mathbf{y}] \triangleq \min_{\mathbf{x} \in \mathcal{P}} (\mathbf{x} - \mathbf{y})^\top A (\mathbf{x} - \mathbf{y})$$

Both type of projections satisfy the following well know fact

Lemma 3.9 (folklore). *Let $\mathcal{P} \subseteq \mathbb{R}^n$ be a convex set, $\mathbf{y} \in \mathbb{R}^n$ and $\mathbf{z} = \prod_{\mathcal{P}}^{A}[\mathbf{y}]$ be the generalized projection of $\mathbf{y}$ onto $\mathcal{P}$ according to PSD matrix $A \succeq 0$. Then for any point $\mathbf{a} \in \mathcal{P}$ it holds that*

$$(\mathbf{y} - \mathbf{a})^\top A(\mathbf{y} - \mathbf{a}) \geq (\mathbf{z} - \mathbf{a})^\top A(\mathbf{z} - \mathbf{a})$$

If A is the identity matrix, this lemma is standard and follows from the fact that for any $\mathbf{a} \in \mathcal{P}$ the angle $\measuredangle(\mathbf{y}, \prod_{\mathcal{P}}[\mathbf{y}], \mathbf{a})$ is obtuse. The latter is implied by the fact that for any point outside a convex body there exists a hyperplane which separates it from all points on the convex set.

For a general PSD matrix $A \succeq 0$, the lemma can be proved by reduction to the simple case, as A generates a natural norm $\forall \mathbf{x} \in \mathbb{R}^n \ . \ \|\mathbf{x}\|_A = \mathbf{x}^\top A \mathbf{x}$. [2] We include a proof for completeness.

Proof. By the definition of generalized projections, the point $\mathbf{z}$ minimizes the function $f(\mathbf{x}) = (\mathbf{x} - \mathbf{y})^\top A(\mathbf{x} - \mathbf{y})$ over the convex set. It is a well known fact in optimization (see [BV04]) that for $\mathbf{z}$ the following holds

$$\forall \mathbf{a} \in \mathcal{P} \ . \ \nabla f(\mathbf{z})(\mathbf{a} - \mathbf{z}) \geq 0$$

Which implies

$$2(\mathbf{z} - \mathbf{y})^\top A(\mathbf{a} - \mathbf{z}) \geq 0 \quad \Rightarrow \quad 2\mathbf{a}^\top A(\mathbf{z} - \mathbf{y}) \geq 2\mathbf{z}^\top A(\mathbf{z} - \mathbf{y})$$

Now by simple calculation:

$$\begin{aligned}
&(\mathbf{y} - \mathbf{a})^\top A(\mathbf{y} - \mathbf{a}) - (\mathbf{z} - \mathbf{a})^\top A(\mathbf{z} - \mathbf{a}) \\
&\quad = \mathbf{y}^\top A\mathbf{y} - \mathbf{z}^\top A\mathbf{z} + 2\mathbf{a}^\top A(\mathbf{z} - \mathbf{y}) \\
&\quad \geq \mathbf{y}^\top A\mathbf{y} - \mathbf{z}^\top A\mathbf{z} + 2\mathbf{z}^\top A(\mathbf{z} - \mathbf{y}) && \text{by previous observation} \\
&\quad = \mathbf{y}^\top A\mathbf{y} - 2\mathbf{z}^\top A\mathbf{y} + \mathbf{z}^\top A\mathbf{z} \\
&\quad = (\mathbf{y} - \mathbf{z})^\top A(\mathbf{y} - \mathbf{z}) \geq 0 && A \text{ is PSD}
\end{aligned}$$

[2]This fact also follows from what is referred to in the machine learning community as the "Pythagorean Theorem" [GW00]

$\square$

Both types of projections are essentially convex programs. For convex polytopes, a projection reduces to a convex quadratic program with linear constraints. These type of convex programs can be solved more efficiently than general convex programs using interior point methods [LVBL98]. Another option is to efficiently approximate these convex programs using the techniques presented in chapter 4.

Even more generally, $\mathcal{P}$ can be specified by a membership oracle $\chi_{\mathcal{P}}$, such that $\chi_{\mathcal{P}}(\mathbf{x}) = 1$ if $\mathbf{x} \in \mathcal{P}$ and 0 if $\mathbf{x} \notin \mathcal{P}$, along with a point $\mathbf{x}_0 \in \mathcal{P}$ as well as radii $R \geq r > 0$ such that the balls of radii R and r around $\mathbf{x}_0$ contain and are contained in $\mathcal{P}$, respectively. In this case $\Pi_{\mathcal{P}}^{A}$ can be computed (to ε accuracy) in time $\tilde{O}(n^4 \log(\frac{R}{r}))$ using Vaidya's algorithm [Vai96].

However, for many simple convex bodies which arise in practical applications (e.g. portfolio management and applications of Chapter 4), projections can be computed much more efficiently. For the n-dimensional unit sphere, cube and the simplex these projections can be computed combinatorially in $\tilde{O}(n)$ time, rendering the online algorithms much more efficient when applied to these convex bodies.

The unit sphere The simplest projection is over the unit n-dimensional sphere, which we denote by $\mathbb{B}^n = \{\mathbf{x} \in \mathbb{R}^n \ , \ \|\mathbf{x}\|_2 \leq 1\}$. Given a vector $\mathbf{y} \in \mathbb{R}^n$, it is easy to verify that its projection is

$$\Pi_{\mathcal{P}}[\mathbf{y}] = \begin{cases} \mathbf{y} & \|\mathbf{y}\| \leq 1 \\ \\ \frac{\mathbf{y}}{\|\mathbf{y}\|} & o/w \end{cases}$$

The unit cube Another body which is easy to project onto is the unit n-dimensional cube, which we denote by $\mathbb{C}^n = \{\mathbf{x} \in \mathbb{R}^n \ , \ \|\mathbf{x}\|_\infty \leq 1\}$ (i.e. each coordinate is less than or equal to one). Given a vector $\mathbf{y} \in \mathbb{R}^n$, it is easy to verify that its projection is

$$\forall i \in [n] \ . \ \Pi_{\mathcal{P}}[\mathbf{y}](i) = \begin{cases} \mathbf{y}(i) & \mathbf{y}(i) \in [-1, 1] \\ \\ 1 & \mathbf{y}(i) > 1 \\ \\ -1 & \mathbf{y}(i) < -1 \end{cases}$$

The Simplex The first non-trivial projection we encounter is over the n-dimensional simplex. The simplex is the set of all n-dimensional distributions, and hence is particularly interesting in many real-world problems such as portfolio management and haplotype frequency estimation. Surprisingly, given an arbitrary vector in Euclidean space, the closest distribution can be found in near linear time. A procedure for computing such a projection is given in figure 3.5.

Lemma 3.10. SimplexProject $(\mathbf{y})$ *is the projection of* $\mathbf{y} \in \mathbb{R}^n$ *to the* n*-dimensional simplex, and can be computed in time* $\tilde{O}(n)$.

SIMPLEXPROJECT **(y).**
Suppose w.l.o.g that $\mathbf{y}_1 \leq \mathbf{y}_2 ... \leq \mathbf{y}_n$ (otherwise sort indices of $\mathbf{y}$).

- Let $a \in \mathbb{R}$ be the number such that $\sum_{i=1}^n \max\{\mathbf{y}_i - a, 0\} = 1$. Set $\forall i \in [n]\ .\ \mathbf{x}_i = \max\{\mathbf{y}_i - a, 0\}$.
- **return x**

Figure 3.5: A Procedure for projecting onto the Simplex

Proof. First, note that the number a computed in SIMPLEXPROJECT exists and is unique. This follows since the function $f(a) = \sum_{i=1}^n \max\{\mathbf{y}_i - a, 0\}$ is continuous, monotone decreasing, and takes values in $[0, \infty)$.

Next, the vector returned $\mathbf{x} = \mathit{SimplexProject}(\mathbf{y})$ is in the simplex. All its coordinates are positive by definition, and $\sum_{i=1}^n \mathbf{x}_i = \sum_{i=1}^n \max\{\mathbf{y}_i - a, 0\} = 1$.

To show that $\mathbf{x}$ is indeed the projection we need to prove that it is the optimum of the mathematical program

$$\min_{\mathbf{x} \in S_n} \sum_{i=1}^n (\mathbf{y}_i - \mathbf{x}_i)^2$$

It suffices to show that x is a local optimum, since the program is convex. Let $c_i \triangleq \mathbf{y}_i - \mathbf{x}_i$. Then the values $\{c_i\}$ are decreasing and of the form

$$(c_1, ..., c_n) = (a, ..., a, \mathbf{y}_k, ..., \mathbf{y}_n)$$

An allowed local change is of the form $\mathbf{x}'_i \leftarrow \mathbf{x}_i - \varepsilon$ and $\mathbf{x}'_j \leftarrow \mathbf{x}_j + \varepsilon$ for $i < j$, since all coordinates larger than k have $\mathbf{x}_k = 0$. This would cause a change in the objective of the form

$$\sum_{i=1}^d (\mathbf{y}_i - \mathbf{x}_i)^2 - (\mathbf{y}_i - \mathbf{x}'_i)^2 = a^2 - (a+\varepsilon)^2 + c_j - (c_j - \varepsilon)^2 = -2(a - c_j)\varepsilon - 2\varepsilon^2 < 0$$

Hence would only reduce the objective. Therefore $\mathbf{x}$ is indeed the projection of $\mathbf{y}$.

The procedure SIMPLEXPROJECT requires sorting n elements, and finding the value a, which is standard to implement in $O(n \log n) = \tilde{O}(n)$ time.

□

The bounded positive semidefinite cone In later chapters, when designing approximation algorithms for semidefinite program, the bounded semidefinite cone $\mathcal{P} = \{X \succeq 0, \mathbf{Tr}(X) \leq 1\}$ will be a common underlying convex set for online algorithms. The following lemma establishes how to project a symmetric matrix onto $\mathcal{P}$ in $\tilde{O}(n^3)$ time, using techniques very similar to those of SIMPLEXPROJECT.

Lemma 3.11. *A symmetric matrix $Y \in \mathbb{R}^{n \times n}$ can be projected onto $\mathcal{P} = \{X \succeq 0, \mathbf{Tr}(X) \leq 1\}$ in time $\tilde{O}(n^3)$.*

Proof. This projection amounts to solving the following optimization problem:

Given a symmetric matrix $Y \in \mathbb{R}^{n\times n}$, find

$$\min_{X\in P} \|Y - X\|_2$$

Because for matrices the ℓ_2 norm is equal to the Frobenius norm, which depends only on the eigenvalues of the matrix, it is invariant to change of basis. Let $Y = U^\top DU$ be the orthogonal diagonalization of Y. Applying the same transformation to X we denote $X = U^\top ZU$. Note that Z is not necessarily diagonal, but since the transformation is orthogonal remains in $Z \in \mathcal{P}$. By the above considerations,

$$\min_{X\in P} \|Y - X\|_2 = \min_{Z\in\mathcal{P}} \|D - Z\|_2 = \min_{Z\in\mathcal{P}} \left\{ \sum_{i\in[n]} (\lambda_i(Y) - Z_{ii})^2 + \sum_{i\neq j} Z_{ij}^2 \right\}$$

Since $\mathbf{Tr}(Z) \leq 1$ we have $\sum_{ij} Z_{ij}^2 = \|Z\|_2 = \|Z\|_F = \sum_{i=1}^n \lambda_i(Z)^2 \leq 1$. This mathematical program can be solved in time $\tilde{O}(n^2)$ as shown by the claim below.

Altogether, to compute the projection we need to diagonalize Y, compute the matrix Z, and compute the inverse transformation. All this can be done in time $\tilde{O}(n^3)$. □

Claim 3.12. *Given $\lambda_1, ..., \lambda_n \in \mathbb{R}$, the optimization problem*

$$\min_{Z\succeq 0, \sum_{i=1}^n Z_{ii}\leq 1} \left\{ \sum_{i\in[n]} (\lambda_i - Z_{ii})^2 + \sum_{i\neq j} Z_{ij}^2 \right\}$$

can be solved in time $O(n^2)$.

Proof. Obviously for $i \neq j$, set $Z_{ij} = 0$. Let w.l.o.g $\lambda_1 \geq ... \geq \lambda_d \geq 0 \geq \lambda_{d+1} \geq \lambda_n$. Again it is obvious that $Z_{ii} = 0$ for all $i > d$. If $\sum_{i=1}^d \lambda_i \leq 1$, then the solution is obtained by setting $Z_{ii} = \lambda_i$ for all $i \leq d$.

Otherwise, let $a \in \mathbb{R}_+$ be the number such that

$$\sum_{i=1}^d \max\{\lambda_i - a, 0\} = 1$$

We claim that the solution is obtained by setting $\forall i \leq d$. $Z_{ii} = \max\{0, \lambda_i - a\}$. To see that, we show that this is a local optimum, and since we have a convex optimization instance, it is also a global solution. This solution implies that the numbers $c_i \triangleq \lambda_i - Z_{ii}$ are of the form

$$(c_1, ..., c_n) = (a, ..., a, \lambda_k, ..., \lambda_l, 0, ...0)$$

The allowed changes are of the form $Z'_{ii} \leftarrow Z_{ii} - \varepsilon$ and $Z'_{jj} \leftarrow Z_{jj} + \varepsilon$. Since the only $i \leq d$ which have non-zero Z_{ii} have value $\lambda_i - Z_{ii} = a$, this would cause a change in the objective of the form

$$\sum_{i=1}^d (\lambda_i - Z_{ii})^2 - (\lambda_i - Z'_{ii})^2 = a^2 - (a+\varepsilon)^2 + c_j - (c_j - \varepsilon)^2 = -2(a - c_j)\varepsilon - 2\varepsilon^2 < 0$$

Hence would only reduce the objective. The value a can computed in time $O(n \log n)$ by sorting and other elementary operations. $\square$

3.6 Experiments with Portfolio Management

In this section we briefly describe experiments with portfolio management on real market data. The results are taken from [AHKS06].

We implemented the algorithms of Cover [Cov91][3], the Multiplicative Weights algorithm of [HSSW96], the uniform CRP and a variant of Online Newton Step described below. We also applied the technique of [SL05] to the algorithms of [HSSW96] and Online Newton Step to get variants which minimize *internal regret*. Internal regret is an alternative (and stronger) performance measure to the "regular" regret defined previously. This notion is useful for certain game theoretic applications. A precise definition is beyond our scope and the reader is referred to [SL05]. Table 3.6 below summarizes the algorithms tested and provides abbreviations that will appears in the comparison charts henceforth.

Abbreviation	Algorithm	Citation
Universal	Cover's algorithm	[Cov91]
MW	Multiplicative Weights algorithm	[HSSW96]
IR-MW	Internal regret variant of MW	[SL05]
ONS	Online Newton Step	[HKKA06]
IR-ONS	Internal regret variant of ONS	[AHKS06]
BCRP	the best CRP in hindsight	-
UCRP	the uniform CRP	-

Figure 3.6: Algorithms used in the experiments.

The Online Newton Step variant implemented for the experiments is given in figure 3.7. Notice it is somewhat different than the one described in section 3.2. This variant is an instantiation of the Online Newton Step algorithm in [HKKA06]. Both algorithms achieve the same theoretical guaranties through different analysis (the analysis in [HKKA06] is similar to the one given in section 3.3). The algorithm takes parameters β and δ for heuristic tuning. We implemented Online Newton Step with parameters $\beta = 1$, and $\delta = \frac{1}{8}$. Unless otherwise noted, we omit the results for IR-ONS because it was inferior to ONS.

We performed tests on the historical stock market data from the New York Stock Exchange (NYSE) used by Cover and Helmbold et al. In addition we randomly selected

[3]Recall that Cover's algorithm has exponential running time. Following previous papers (e.g. [HSSW96]), we use a random sampling heuristic to approximate Cover's algorithm. This implies a small degree of variability in the measurements recorded here. We used 1000 samples, which as suggested by [SL05], is sufficient to get a good estimate of the behavior of that algorithm.

ONS(β, δ)

- On period 1, use the uniform portfolio $\mathbf{p}_1 = \frac{1}{n}\mathbf{1}$.
- On period $t > 1$: use portfolio:

$$\mathbf{p}_t = \Pi_{S_n}^{\mathbf{A}_{t-1}} \left(\delta \mathbf{A}_{t-1}^{-1} \mathbf{b}_{t-1}\right)$$

where $\mathbf{b}_{t-1} = (1 + \frac{1}{\beta}) \sum_{\tau=1}^{t-1} \nabla[\log_\tau(\mathbf{p}_\tau \cdot \mathbf{r}_\tau)]$, $\mathbf{A}_{t-1} = \sum_{\tau=1}^{t-1} -\nabla^2[\log(\mathbf{p}_\tau \cdot \mathbf{r}_\tau)] + \mathbf{I}_n$, and $\Pi_{S_n}^{\mathbf{A}_{t-1}}$ is the projection onto the simplex in the norm induced by $\mathbf{A}_{t-1}$, viz.,

$$\Pi_{S_n}^{\mathbf{A}_{t-1}}(\mathbf{q}) = \arg\min_{\mathbf{p} \in S_n} (\mathbf{q} - \mathbf{p})^\top \mathbf{A}_{t-1} (\mathbf{q} - \mathbf{p})$$

Figure 3.7: The Online Newton Step variant for portfolio management.

portfolios of various sizes from a set of 50 randomly chosen S&P 500 stocks[4] and performed experiments over the past 4 years data from 12th December, 2001 to 30th November, 2005 obtained from Yahoo! Finance.

3.6.1 Performance vs. Portfolio Size

To measure the dependence of the performance of various algorithms on portfolio size we picked 50 sets of n random stocks from the data set, for values of n ranging from 5 to 40. All algorithms were run on the data, trading once every two weeks. The choice of trading period was to permit completion of the Universal algorithm in reasonable time. The trading period did not seem to affect the relative performance of the algorithms. The results are shown in Figure 3.8.

The improvement in the performance of ONS with increasing number of stocks is quite stark. The reason for this seems to be that ONS does an extremely good job of tracking the best stock in a given portfolio. Adding more stocks causes some good stock to get added, which ONS proceeds to track. Other algorithms behave more like the uniform CRP and so average out the increase in wealth due to the addition of a good stock. Figure 3.9 shows how ONS tracks CMC, which out-performs Kin-Ark for the test period, in a dataset composed of Kin-Ark and CMC (also used by Cover) while other algorithms have a nearly uniform distribution on both the stocks. This is the reason ONS outperforms all other algorithms on this dataset, as can be seen in Figure 3.11.

[4]The set of stocks used was RTN, SLB, ABK, PEG, KMG, FITB, CL, PSA, DOV, NKE, AT, NEM, VMC, D, CPWR, NVDA, SRE, HPQ, CMX, LXK, GPC, ABI, PGL, QLGC, OMX, QCOM, KO, PMTC, SWK, CTXS, FSH, HON, COF, LH, KMG, BLL, WB, OMX, K, LUV, DIS, SFA, APOL, HUM, CVH, IR, SPG, WY, TYC, NKE.

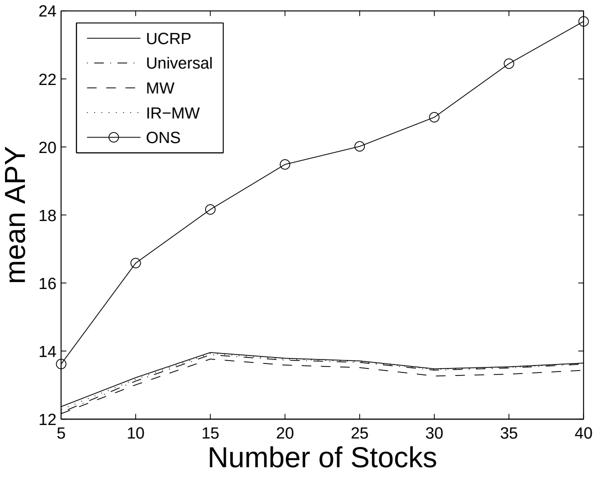

Figure 3.8: Performance vs. Portfolio Size

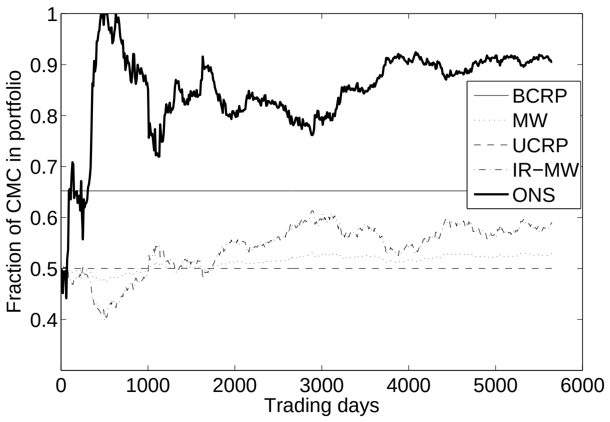

Figure 3.9: How ONS tracks CMC.

3.6.2 Random stocks from S&P 500

We tested the average APYs (over 50 trials of 10 random stocks from the S&P 500 list mentioned before) of the algorithms, for different frequencies of rebalancing, namely daily, weekly, fortnightly and monthly. As can be seen in figure 3.10 the performance of the ONS algorithm is superior to all other algorithms in all 4 cases. As is expected the performance of all algorithms degrades as trading frequency decreases, but not very significantly. The simple strategy of maintaining a uniform constant-rebalanced portfolio seems to outperform all previous algorithms. This rather surprising fact is consistent with the observation of [BEYG04].

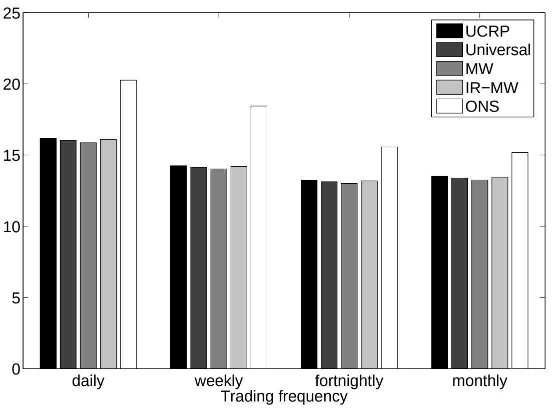

Figure 3.10: Performance vs. Trading Period.

3.6.3 Cover's Experiments

We replicated the experiments of Cover and Helmbold et al. on Iroquios Brands Ltd. and Kin Ark Corp., Commercial Metals (CMC) and Kin Ark, CMC and Meicco Corp., IBM and Coca Cola for the same 22 year period from 3rd July, 1962 to 31st December, 1984. As can be seen from Figure 3.11, ONS outperforms all other algorithms except on the Iroquios Brands Ltd. and Kin Ark Corp. dataset.

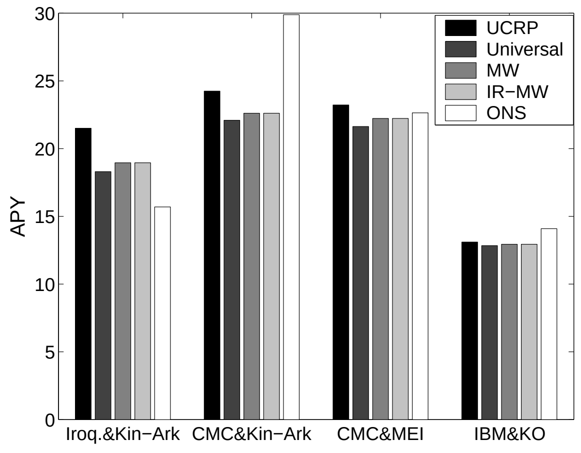

Figure 3.11: Four pairs of stocks tested by [Cov91] and [HSSW96].

Figure 3.12 shows how the total wealth (per dollar invested) varies over the entire period using the different algorithms for a portfolio of IBM and Coke. The ONS algorithm, and its internal regret variant IR-ONS, outperform even the best constant-rebalanced portfolio.

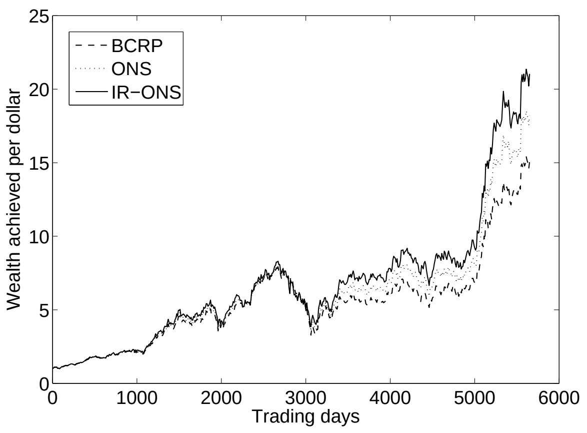

Figure 3.12: Wealth achieved by various algorithms on a portfolio consisting of IBM and Coke.

3.6.4 Stock volatility

We took the 50 stock data set used in previous experiments which had a history for 1000 days traded fortnightly and sorted them according to volatility and created two sets: the 10 stocks with largest and smallest price variance. Then we applied the different algorithms on the two different sets.

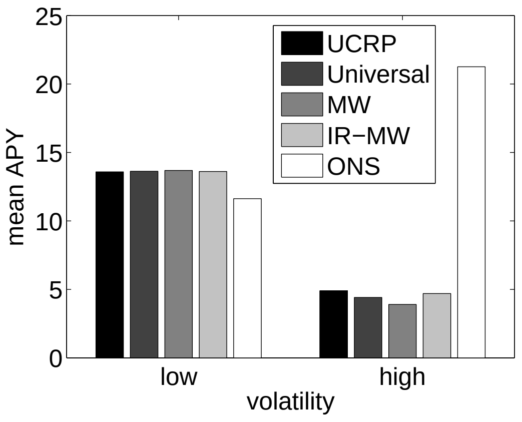

Figure 3.13: Performance of algorithms on high and low volatility datasets.

Figure 3.13 shows that the performance of ONS increases with market volatility whereas the performance of other algorithms decreases.

3.6.5 Margins loans

In line with [Cov91] and [HSSW96], we also tested the case where the portfolio can buy stocks on margin. The data set we tested on was the 22 year IBM and Coca Cola data mentioned earlier. Results for this case are given in Table 3.1. The margin purchases we incorporate are 50% down and 50% loan. The ONS algorithm seems to enhances its performance edge over other algorithms if margin loans are allowed.

Table 3.1: Incorporating margin loans.

Algorithm	APY, no margin	APY with margin
UCRP	12.73	14.84
Universal	12.46	14.40
MW	12.57	14.39
IR-MW	12.57	14.62
ONS	13.68	16.15

3.6.6 Sharpe Ratio and Mean-Variance Optimal CRPs

It is a well-known fact that one can achieve higher returns by investing in riskier assets [Lue98]. So it is important to rule out the possibility of the ONS algorithm achieving higher returns compared to other algorithms by trading more riskily. Parameters like the Sharpe ratio and the optimal mean-variance portfolio are used to measure this risk versus reward tradeoff. Sharpe ratio is defined as $\frac{R_p - R_f}{\sigma_p}$ where R_p is the average yearly return of the algorithm, which indicates reward, R_f is the risk-free rate (typically the average rate of return of Treasury bills), and σ_p is the standard deviation of the returns of the algorithm, which indicates its volatility risk. The higher the Sharpe Ratio the better is the algorithm at balancing high rewards with low risk.

The mean-variance optimal CRP for an algorithm is the CRP which achieves the same return as the algorithm but has minimum variance. This is the least risky CRP one could have used in hindsight to produce the same returns. The closer the volatility of the CRP to that of the algorithm, the better the algorithm is avoiding risk.

	Universal	**UCRP**	**MW**	**IR-MW**	**ONS**
Iro.&Kin-Ark	0.4986	0.5497	0.5390	0.5078	0.4578
CMC & Kin-Ark	0.5740	0.6020	0.5980	0.5812	0.7466
CMC & Meicco	0.3885	0.3834	0.3856	0.3854	0.5177
IBM & Coke	0.5246	0.5376	0.5356	0.5295	0.5824

Figure 3.14: Sharpe ratios for various algorithms on different datasets.

Figure 3.14 shows that ONS has either the best or slightly smaller Sharpe ratio among all algorithms. In figure 3.15, it can be seen that ONS has comparable volatility to the minimum variance CRP, implying that ONS does not take excessive risk in its portfolio selection. In the case of IBM & Coke and Kin-Ark & CMC, ONS beats the Best CRP in hindsight. Hence the concept of the optimal mean-variance CRP does not apply and the results for this case are omitted.

	Universal	**UCRP**	**MW**	**IR-MW**	**ONS**
Iro. & Kin-Ark	0.46/0.495	0.493/0.493	0.48/0.493	0.461/0.493	0.46/0.545
CMC & Meicco	0.191/0.273	0.191/0.272	0.191/0.272	0.191/0.272	0.207/0.351

Figure 3.15: Minimum variance CRPs for various algorithms on different datasets. The number to the left of the slash is the volatility of the minimum variance CRP and the number to the right is the volatility of the algorithm on the dataset.

Chapter 4

A Game Playing Framework for Offline Optimization

In this chapter we describe a general framework for approximate optimization via online convex optimization algorithms, which generalizes previous Lagrangian relaxation based methods.

We start by describing the main theorems that can be obtained. Then we give a few “meta-algorithms”, which transform online convex optimization algorithms with low regret to approximation algorithms. The new algorithms for online convex optimization, discussed in chapter 3, give rise to new approximate optimization algorithms which are more efficient in certain cases. We continue by exploring the limitations of the model, and relation to the von Neumann min-max theorem.

The results of this section are taken from [Haz06].

4.1 Introduction

The design of efficient approximation algorithms for certain convex and linear programs has received much attention in the previous two decades. Since interior point methods and other polynomial time algorithm are often too slow in practice [Bie01], researchers have tried to design approximation algorithms. Shahrokhi and Matula [SM90] developed the first approximation algorithm for the maximum concurrent flow problem. Their result spurred a great deal of research, which generalized the techniques to broader classes of problems (linear programming, semidefinite programming, packing and covering convex programs) and improved the running time [LSM$^+$91, KPST94, PST91, GK94, GK98, Fle00, GK95, KL96, AHK05b].

In this chapter we consider approximations to more general convex programs. The convex feasibility problem we consider is of the following form (the optimization version can be reduced to this feasibility problem by binary search),

$$\begin{aligned} f_j(x) &\le 0 \quad \forall j \in [m] \\ x &\in \mathbb{S}_n \end{aligned} \tag{4.1}$$

Here, $\{f_j, j \in [m]\}$ is a (possibly infinite) set of convex constraints and $\mathbb{S}_n = \{x \in \mathbb{R}^n, \sum_i x_i = 1, x_i \geq 0\}$ is the unit simplex. Our algorithm work almost without change if the simplex is replaced by other simple convex bodies such as the ball or hypercube. The more general version, where $\mathbb{S}_n$ is replaced by an arbitrary convex set in Euclidian space, can also be handled at the expense of slower running time (see section 4.3.1). We say that an algorithm gives an ε-approximate solution to the above program if it returns $x \in \mathcal{P}$ such that $\forall j \in [m]$. $f_j(x) \leq \varepsilon$, or returns proof that the program is infeasible. A common parameter in the analysis of lagrangian relaxation algorithms is the *width* - a measure of the size of the instance numbers. The *width* of program (4.1) is defined as $\omega = \max_{j \in [m]} \max_{x \in \mathbb{S}_n} |f_i(x)|$.

A common feature to all of the prior algorithms is that they can be viewed, sometimes implicitly, as Frank-Wolfe [FW56] algorithms, in that they iterate by solving an optimization problems over $\mathbb{S}_n$ (more generally over the underlying convex set), and take convex combinations of iterates. The optimization problem that is iteratively solved is of the following form.

$$\forall p \in \mathbb{S}_m \ . \ \textsc{Optimization Oracle (p)} \triangleq \begin{cases} x \in \mathbb{S}_n \text{ s.t } \sum_j p_j f_j(x) \leq 0 & \text{if exists such } x \\ \\ FAIL & \text{otherwise} \end{cases}$$

It is possible to extend the methods of PST [PST91] and others to problems such as (4.1) (see [Jan04, Kha04]) and obtain the following theorem

Theorem 4.1 (previous work)**.** *There exists an algorithm that for any $\varepsilon > 0$, returns a ε-approximation solution to mathematical program* (4.1)*. The algorithm makes at most $\tilde{O}(\frac{\omega^2}{\varepsilon^2})$ calls to* Optimization Oracle*, and requires $O(m)$ time between successive oracle calls.*

Remark 1: Much previous work focuses on reducing the dependance of the running time on the width. Linear dependence on ω was achieved for special cases such as *packing and covering problems* (see [You95]). For covering and packing problems the dependence on the width can be removed completely, albeit introducing another n factor into the running time [Jan04].

Remark 2: In case the constraint functions are linear, Optimization Oracle can be implemented in time $O(mn)$. Otherwise, the oracle reduces to optimization of a convex non-linear function over a convex set.

Klein and Young [KY99] proved an $\Omega(\varepsilon^{-2})$ lower bound for Frank-Wolfe type algorithms for covering and packing linear programs under appropriate conditions. This bound applies to all prior lagrangian relaxation algorithms till the recent result of Bienstock and Iyengar [BI04]. They give an algorithm for solving packing and covering linear programs in time linear in $\frac{1}{\varepsilon}$, proving

Theorem 4.2 ([BI04])**.** *There exists an algorithm that for any $\varepsilon > 0$, returns a ε-approximation solution to packing or covering linear programs with m constraints. The* *algorithm makes at most $\tilde{O}(\frac{n}{\varepsilon})$ iterations. Each iteration requires solving a convex separable quadratic program. The algorithm requires $O(mn)$ time between successive oracle calls.*

Their algorithm has a non-combinatorial component, viz., solving convex separable quadratic programs. To solve these convex programs one can use interior point methods, which have large polynomial running time. Another alternative is to apply lagrangian relaxation or other approximation methods, but these would introduce another factor polynomial in $\frac{1}{\varepsilon}$ to the total running time.

We give a simple approximation algorithms for convex programs whose running time is linear in $\frac{1}{\varepsilon}$. The algorithms requires just a separation oracle, as opposed to an optimization oracle.

The $\Omega(\varepsilon^{-2})$ lower bound of Klein and Young is circumvented by using the strict convexity of the constraints. The constraint functions are said to be strictly convex if there exists a positive real number $H > 0$ such that $\min_{j\in[m]} \min_{x\in\mathcal{P}} \nabla^2 f_j(x) \succeq H \cdot I$ [1] . In other words, the Hessian of the constraint function is positive definite (as opposed to positive semidefinite) with smallest eigenvalue at least $H > 0$.

Our running time bounds depend on the gradients of the constraint functions as well. Let $G = \max_{j\in[m]} \max_{x\in\mathcal{P}} \|\nabla f_j(x)\|_2$ be an upper bound on the norm of the gradients of the constraint functions. G is related to the width of the convex program: for linear constraints, the gradients are simply the coefficients of the constraints, and the width is the largest coefficient. Hence, G is at most $\sqrt{n}$ times the width. In section 4.3 we prove the following Theorem.

Theorem 4.3 (Main 1)**.** *There exists an algorithm that for any $\varepsilon > 0$, returns a ε-approximate solution to mathematical program* (4.1)*. The algorithm makes at most $\tilde{O}(\frac{G^2}{H} \cdot \frac{1}{\varepsilon})$ calls to* Separation Oracle*, and requires a single gradient computation and additional $O(n)$ time between successive oracle calls.*

Remark: Commonly the gradient of a given function can be computed in time which is linear in the function representation. Examples of functions which admit linear-time gradient computation include polynomials, logarithmic functions and exponentials.

The separation oracle which our algorithm invokes is defined as

$$\forall x \in \mathbb{S}_n \text{ . Separation Oracle (x)} \triangleq \begin{cases} j \in [m] \text{ s.t } f_j(x) > \varepsilon & \text{if exists such } f_j \\ FAIL & \text{otherwise} \end{cases}$$

If the constraints are given explicitly, often this oracle is easy to implement in time linear in the input size. Such constraints include linear functions, polynomials and logarithms. This oracle is also easy to implement in parallel: the constraints can be distributed amongst the available processors and evaluated in parallel.

For all cases in which H is zero or too small the theorem above cannot be applied. However, we can apply a simple reduction to strictly convex constraints and obtain the following corollary.

[1]we denote $A \succeq B$ if the matrix $A - B \succeq 0$ is positive semidefinite

Corollary 4.4. *For any $\varepsilon > 0$, there exists an algorithm that returns a ε-approximate solution to mathematical program* (4.1). *The algorithm makes at most $\tilde{O}(\frac{G^2}{\varepsilon^2})$ calls to* Separation Oracle *and requires additional $O(n)$ time and a single gradient computation between successive oracle calls.*

In comparison to Theorem 4.1, this corollary may require $O(n)$ more iterations. However, each iteration requires a call to Separation Oracle, as opposed to Optimization Oracle. A Separation Oracle requires only function evaluation, which can many times be implemented in linear time in the input size, whereas an Optimization Oracle could require expensive operations such as matrix inversions.

There is yet another alternative to deal with linear constraints and yet obtain linear dependence on ε. This is given by the following theorem. The approximation algorithm runs in time linear in $\frac{1}{\varepsilon}$, and yet does not require a lower bound on H. The downside of this algorithm is the computation of "generalized projections". A generalized projection of a vector $y \in \mathbb{R}^n$ onto a convex set $\mathcal{P}$ with respect to PSD matrix $A \succeq 0$ is defined to be the vector $\prod_{\mathcal{P}}^{A}(y) = \arg\min_{x \in \mathcal{P}} (x-y)^\top A(x-y)$. Generalized projections can be cast as convex mathematical programs. If the underlying set is simple, such as the ball or simplex, then the program reduces to a convex quadratic program (see section 3.5).

Theorem 4.5 (Main 2)**.** *There exists an algorithm that for any $\varepsilon > 0$ returns a ε-approximate solution to mathematical program* (4.1). *The algorithm makes at most $\tilde{O}(\frac{nG}{\varepsilon})$ calls to* Separation Oracle *and requires computation of a generalized projection onto $\mathbb{S}_n$, a single gradient computation and additional $\tilde{O}(n^2)$ time between successive oracle calls.*

An example of an application of the above theorem is the following linear program.

$$\forall j \in [m] \; . \; A_j \cdot x \geq 0, \quad x \in \mathbb{S}_n \tag{4.2}$$

It is shown in [DV04] that general linear programming can be reduced to this form, and that without loss of generality, $\forall j \in [m] \;\; \|A_j\| = 1$. This format is called the "perceptron" format for linear programs. As a corollary to Theorem 4.5, we obtain

Corollary 4.6. *There exists an algorithm that for any $\varepsilon > 0$ returns a ε-approximate solution to linear program (4.2). The algorithm makes $\tilde{O}(\frac{n}{\varepsilon})$ iterations. Each iteration requires $\tilde{O}(n(m+n))$ computing time plus computation of a generalized projection onto the simplex.*

Theorem 4.5 and Corollary 4.6 extend the result of Bienstock and Iyengar [BI04] to general convex programming [2]. The running time of the algorithm is very similar to theirs: the number of iterations is the same, and each iteration also requires to solve convex quadratic programs (generalized projections onto the simplex in our case). Our algorithm is very different from [BI04]. The analysis is simpler, and relies on recent results from online learning. We note that the algorithm of Bienstock and Iyengar allows improved running time for sparse instances, whereas our algorithm currently does not.

[2]Bienstock and Iyengar's techniques can also be extended to full linear programming by introducing dependence on the width which is similar to that of our algorithms [Bie06].

4.1.1 Lagrangian relaxation and solving zero sum games

The relation between lagrangian relaxation and solving zero sum games was implicit in the original PST work, and explicit in the work of Freund and Schapire on online game playing [FS99] (the general connection between zero sum games and linear programming goes back to von Neumann).

Most previous lagrangian relaxation algorithms can be viewed as reducing the optimization problem at hand to a zero sum game, and then applying a certain online game playing algorithm, the Multiplicative Weights algorithm, to solve the game.

Our main insight is that the Multiplicative Weights algorithm can be replaced by any *online convex optimization* (see chapter 2) algorithm. The recent developments in online convex optimization detailed in chapter 3, introduce algorithms with much better performance guaranties for online games with convex payoff functions. Our results are derived by reducing convex optimization problems to games with payoffs which stem from convex functions, and using the new algorithms to solve these games.

The online framework also provides an alternative explanation to the aforementioned Klein and Young $\Omega(\varepsilon^{-2})$ lower bound on the number of iterations required by Frank-Wolfe algorithms to produce an ε-approximate solution. Translated to the online framework, previous algorithm were based on online algorithms with $\Omega(\sqrt{T})$ *regret* (the standard performance measure for online algorithms, see chapter 2 for precise definition). Our linear dependance on $\frac{1}{\varepsilon}$ is the consequence of using of online algorithms with $O(\log T)$ regret. This is formalized in section 4.4.

4.2 The general scheme

We outline a general scheme for approximately solving convex programs using online convex optimization algorithms. This is a generalization of previous methods which also allows us to derive the results stated in the previous section.

For this section we consider the following general mathematical program, which generalizes (4.1) by allowing an arbitrary convex set $\mathcal{P}$.

$$\begin{aligned} f_j(x) \leq 0 \quad & \forall j \in [m] \\ x \in \mathcal{P} & \end{aligned} \tag{4.3}$$

In order to approximately solve (4.3), we reduce the mathematical problem to a game between two players: a *primal player* who tries to find a feasible point and the *dual player* who tries to disprove feasibility. This reduction is formalized in the following definition.

Definition 4.7. The associated game with mathematical program (4.3) is between a primal player that plays $x \in \mathcal{P}$ and a dual player which plays a distribution over the constraints $p \in S_m$. For a point played by the primal player and a distribution of the dual player, the loss that the primal player incurs (and the payoff gained by the dual player) is given by the following function

$$\forall\ x \in \mathcal{P}\ , p \in \mathbb{S}_m\ .\ \ g(x,p) \triangleq \sum_j p_j f_j(x)$$

The value of this game is defined to be $\lambda^* \triangleq \min_{x\in\mathcal{P}} \max_{p\in\mathbb{S}_m} g(x,p)$. Mathematical program (4.3) is feasible iff $\lambda^* \leq 0$.

By the above reduction, in order to check feasibility of mathematical program (4.3), it suffices to compute the value of the associated game λ^*. Notice that the game loss/payoff function g is smooth over the convex sets S_m and $\mathcal{P}$, linear with respect to p and convex with respect to x. For such functions, generalizations to the von Neumann minimax theorem, such as [Sio58] [3] imply that

$$\lambda^* = \min_{x\in\mathcal{P}} \max_{p\in\mathbb{S}_m} g(x,p) = \max_{p\in\mathbb{S}_m} \min_{x\in\mathcal{P}} g(x,p)$$

This suggests a natural approach to evaluate λ^*: simulate a repeated game between the primal and dual players such that in each iteration the game loss/payoff is determined according to the function g. In the simulation, the players play according to an online algorithm.

The online algorithms we consider fall into the online convex optimization framework (see chapter 2). Recall that in online convex optimization there is a fixed convex compact feasible set $\mathcal{P} \subset \mathbb{R}^n$ and an *arbitrary, unknown* sequence of convex cost functions $f_1, f_2, \ldots : \mathcal{P} \to \mathbb{R}$. The decision maker must make a sequence of decisions, where the t^{th} decision is a selection of a point $x_t \in \mathcal{P}$ and there is a cost of $f_t(x_t)$ on period t. However, x_t is chosen with only the knowledge of the set $\mathcal{P}$, previous points $x_1, \ldots, x_{t-1}$, and the previous functions $f_1, \ldots, f_{t-1}$. The standard performance measure for online convex optimization algorithms is called *regret* which is defined as:

$$\text{Regret}_T(\mathcal{A}) \triangleq \sup_{f_1,\ldots,f_T} \left\{ \sum_{t=1}^{T} f_t(x_t) - \min_{x^*\in\mathcal{P}} \sum_{t=1}^{T} f_t(x^*) \right\}$$

In certain cases we speak of an online player that wants to maximize payoff, rather than minimize cost. In this case the payoff functions $f_1, .., f_T$ are concave and regret is defined to be

$$\text{Regret}_T(\mathcal{A}) \triangleq \sup_{f_1,\ldots,f_T} \left\{ \max_{x^*\in\mathcal{P}} \sum_{t=1}^{T} f_t(x^*) - \sum_{t=1}^{T} f_t(x_t) \right\}$$

We say that an algorithm $\mathcal{A}$ has *low regret* if $\text{Regret}_T(\mathcal{A}) = o(T)$. Later, we use to the procedure OnlineAlg, by which we refer to any low regret algorithm for this setting.

Another crucial property of online convex optimization algorithms is their running time. The running time is the time it takes to produce the point $x_t \in \mathcal{P}$ given all prior game history.

The running time of our approximate optimization algorithms will depend on these two parameters of online game playing algorithms: regret and running time. In chapters 2,3 we survey some old and new online convex optimization algorithms and their properties.

[3] All algorithms and theorems in this paper can be proved without relying on this minimax theorem. In fact, our results provide a new algorithmic proof of the generalized min-max theorem which is included in Appendix 4.5.

We suggest three methods for approximating (4.3) using the approach outlined above. The first "meta algorithm" (it allows freedom in choice for the implementation of the online algorithm) is called PrimalGameOpt and depicted in figure 4.1. For this approach, the dual player is simulated by an optimal adversary: at iteration t it plays a dual strategy p_t that achieves at least the game value λ^* (this reduces exactly to Separation Oracle).

The implementation of the primal player is an online convex optimization algorithm with low regret, which we denote by OnlineAlg. This online convex optimization algorithm produces decisions which are points in the convex set $\mathcal{P}$. The cost functions $f_1, f_2, \ldots : \mathcal{P} \to \mathbb{R}$ are determined by the dual player's distributions. At iteration t, if the distribution output by the dual player us p_t, then the cost function to the online player is

$$\forall x \in \mathcal{P} \; . \; f_t(x) \triangleq g(x, p_t)$$

The low-regret property of the online algorithm used ensures that in the long run, the average strategy of the primal player will converge to the optimal strategy. Hence the average loss will converge to λ^*.

The "dual" version of this approach, in which the dual player is simulated by an online algorithm and the primal by an oracle, is called DualGameOpt. In this case, the adversarial implementation of the primal player reduces to Optimization Oracle. The dual player now plays according to an online algorithm OnlineAlg. This online algorithm produces points in the m-dimensional simplex - the set of all distributions over the constraints. The payoff functions are determined according to the decisions of the primal player: at iteration t, if primal player produced point $x_t \in \mathcal{P}$, the payoff function is

$$\forall p \in \mathbb{S}_m \; . \; f_t(p) \triangleq g(x_t, p)$$

We also explore a third option, in which both players are implemented by online algorithms. This is called the PrimalDualGameOpt meta-algorithm. Pseudo-code for all versions is given in figure 4.1.

The following theorem shows that all these approaches yield an ε-approximate solution when the online convex optimization algorithm used to implement OnlineAlg has low regret.

Theorem 4.8. *Suppose OnlineAlg is an online convex optimization algorithm with low regret. If a solution to mathematical program* (4.3) *exists, then meta-algorithms* PrimalGameOpt, DualGameOpt *and* PrimalDualGameOpt *return an ε-approximate solution. Otherwise,* PrimalGameOpt *and* DualGameOpt *return a dual solution proving that the mathematical program is infeasible, and* PrimalDualGameOpt *returns a dual solution proving the mathematical program to be ε-close to being infeasible.*

Further, a ε-approximate solution is returned in $O(\frac{R}{\varepsilon})$ iterations, where $R = R(OnlineAlg, \varepsilon)$ is the smallest number T which satisfies the inequality $Regret_T(OnlineAlg) \leq \varepsilon T$.

Proof. **Part 1: correctness of** PrimalGameOpt

```
PrimalGameOpt ($\varepsilon$)
Let $t \leftarrow 1$. While $\mathrm{Regret}_t(\textsc{OnlineAlg}) \geq \varepsilon t$ do
    • Let $x_t \leftarrow$ OnlineAlg $(p_1, ..., p_{t-1})$.
    • Let $j \leftarrow$ Separation Oracle $(x_t)$. If $FAIL$ return $x_t$. Let $p_t \leftarrow e_j$, where $e_j$ is
      the $j$'th standard basis vector of $\mathbb{R}^n$.
    • $t \leftarrow t + 1$
Return $\bar{p} = \frac{1}{T} \sum_{t=1}^{T} p_t$
```

```
DualGameOpt ($\varepsilon$)
Let $t \leftarrow 1$. While $\mathrm{Regret}_t(\textsc{OnlineAlg}) \geq \varepsilon t$ do
    • Let $p_t \leftarrow$ OnlineAlg $(x_1, ..., x_{t-1})$.
    • Let $x_t \leftarrow$ Optimization Oracle $(p_t)$. If $FAIL$ return $p_t$.
    • $t \leftarrow t + 1$
Return $\bar{x} \triangleq \frac{1}{T} \sum_{t=1}^{T} x_t$
```

```
PrimalDualGameOpt ($\varepsilon$)
Let $t \leftarrow 1$. While $\mathrm{Regret}_t(\textsc{OnlineAlg}) \geq \frac{\varepsilon}{2} t$ do
    • Let $x_t \leftarrow$ OnlineAlg $(p_1, ..., p_{t-1})$.
    • Let $p_t \leftarrow$ OnlineAlg $(x_1, ..., x_{t-1})$.
    • $t \leftarrow t + 1$
If $\bar{x} \triangleq \frac{1}{T} \sum_{t=1}^{T} x_t$ is $\varepsilon$-approximate return $\bar{x}$. Else, return $\bar{p} = \frac{1}{T} \sum_{t=1}^{T} p_t$.
```

Figure 4.1: meta algorithms for approximate optimization by online game playing

If at iteration t Separation Oracle returns $FAIL$, then by definition of Separation Oracle,

$$\forall p^* \ . \ g(x_t, p^*) \leq \varepsilon \ \Rightarrow \ \forall j \in [m] \ . \ f_j(x_t) \leq \varepsilon$$

implying that x_t is a ε-approximate solution.

Otherwise, for every iteration $g(x_t, p_t) > \varepsilon$, and we can construct a dual solution as follows. Since the online algorithm guaranties sub-linear regret, for some iteration T the regret will be $R \leq \varepsilon T$. By definition of regret we have for any strategy $x^* \in \mathcal{P}$,

$$\varepsilon < \frac{1}{T} \sum_{t=1}^{T} g(x_t, p_t) \leq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) + \frac{R}{T} \leq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) + \varepsilon \leq g(x^*, \bar{p}) + \varepsilon$$

The last inequality follows from the concavity (linearity) of $g(x,p)$ with respect to p Thus,

$$\forall x^* \ . \ g(x^*, \bar{p}) > 0$$

Hence $\bar{p}$ is a dual solution proving that the mathematical program is infeasible.

Part 2: correctness of DualGameOpt

If for some iteration t Optimization Oracle returns $FAIL$. According to the definition of Optimization Oracle,

$$\forall x \in \mathcal{P} \ . \ g(x, p_t) > 0$$

implying that p_t is a dual solution proving the mathematical program to be infeasible.

Else, in every iteration $g(x_t, p_t) \leq 0$. As before, for some iteration T the regret of the online algorithm will be $R \leq \varepsilon T$. By definition of regret we have (note that this time the online player wants to maximize his payoff)

$$\forall p^* \in P(\mathcal{F}) \ . \ 0 \geq \frac{1}{T}\sum_{t=1}^{T} g(x_t, p_t) \geq \frac{1}{T}\sum_{t=1}^{T} g(x_t, p^*) - \frac{R}{T} \geq \frac{1}{T}\sum_{t=1}^{T} g(x_t, p^*) - \varepsilon$$

Changing sides and using the convexity of the function $g(x,p)$ with respect to x (which follows from the convexity of the functions $f \in \mathcal{F}$) we obtain (for $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x_t$)

$$\forall p^* \in P(\mathcal{F}) \ . \ g(\bar{x}, p^*) \leq \frac{1}{T}\sum_{t=1}^{T} g(x_t, p^*) \leq \varepsilon$$

Which in turn implies that

$$\forall f \in \mathcal{F} \ . \ f(\bar{x}) \leq \varepsilon$$

Hence $\bar{x}$ is a ε-approximate solution.

Part 3: correctness of PrimalDualGameOpt

Denote R_1, R_2 the regrets attained by both online algorithms respectively. Using the low regret properties of the online algorithms we obtain for any x^*, p^*

$$\forall x^*, p^* \ . \ \sum_{t=1}^{T} g(x_t, p^*) - R_1 \leq \sum_{t=1}^{T} g(x_t, p_t) \leq \sum_{t=1}^{T} g(x^*, p_t) + R_2 \tag{4.4}$$

Let x^* be such that $\forall p \in P(\mathcal{F}) \ . \ g(x^*, p) \leq \lambda^*$. By convexity of $g(x,p)$ with respect to x,

$$\forall p^* \ . \ g(\bar{x}, p^*) \leq \frac{1}{T}\sum_{t=1}^{T} g(x_t, p^*) \leq \frac{1}{T}\sum_{t=1}^{T} g(x^*, p_t) + \frac{R_2 + R_1}{T} \leq \lambda^* + \varepsilon$$

Similarly, let p^* be such that $\forall x \in \mathcal{P} \ . \ g(x, p^*) \geq \lambda^*$. Then by concavity of g with respect to p and equation 4.4 we have

$$\forall x^* \ . \ g(x^*, \bar{p}) \geq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) \geq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p^*) - \frac{R_2 + R_1}{T} \geq \lambda^* - \varepsilon$$

Hence, if $\lambda^* \leq 0$, then $\bar{x}$ satisfies

$$\forall p^* \ . \ g(\bar{x}, p^*) \leq \varepsilon \ \Rightarrow \ \forall j \in [m] \ . \ f_j(\bar{x}) \leq \varepsilon$$

And hence is a ε-approximate solution. Else,

$$\forall x^* \ . \ g(x^*, \bar{p}) > -\varepsilon$$

And $\bar{p}$ is a dual solution proving that the following mathematical program is infeasible.

$$\begin{aligned} f_j(x) \leq -\varepsilon \quad & \forall j \in [m] \\ x \in \mathcal{P} \quad & \end{aligned}$$

□

4.3 Applications

4.3.1 Strictly convex programs

We start with the easiest and perhaps most surprising application of Theorem 4.8. Recall that the feasibility problem we are considering:

$$\begin{aligned} f_j(x) \leq 0 \quad & \forall j \in [m] \\ x \in \mathbb{S}_n \quad & \end{aligned}$$

where the functions $\{f_j\}$ are strictly convex such that $\forall x \in \mathbb{S}_n, j \in [m] \ . \ \nabla^2 f_j(x) \succeq H \cdot I_n$ and $\|\nabla f_j(x)\|_2 \leq G$

Proof of Theorem 4.3. Consider the associated game with value

$$\lambda^* \triangleq \min_{x \in \mathbb{S}_n} \max_{j \in [m]} f_j(x)$$

The convex problem is feasible iff $\lambda^* \leq 0$. To approximate λ^*, we apply the PRIMAL-GAMEOPT meta algorithm. In this case, the vectors x_t are points in the simplex, and p_t are distributions over the constraints.

The online algorithm used to implement ONLINEALG is ONLINE GRADIENT DESCENT (OGD). According to Theorem 3.8 in section 3.4.1, the regret of OGD is bounded by $\text{Regret}_T(OGD) = O(\frac{G^2}{H} \log T)$. Hence, the number of iterations till the regret drops to εT is $\tilde{O}(\frac{G^2}{H} \frac{1}{\varepsilon})$. According to Theorem 4.8, this is the number of iterations required to obtain an ε-approximation.

In each iteration, the OGD algorithm needs to update the current online strategy (the vector x_t) according to the gradient and project onto $\mathbb{S}_n$. This requires a single gradient computation. A projection of a vector $y \in \mathbb{R}^n$ onto $\mathbb{S}_n$ is defined to be $\prod_{\mathcal{P}}(y) = \arg\min_{x \in \mathbb{S}_n} \|x - y\|_2$, and can be computed in time $\tilde{O}(n)$ (see section 3.5). Other than the gradient computation and projection, the running time of OGD is $O(n)$ per iteration (see section 3.4.1). $\square$

Remark: It is clear that the above algorithm can be applied the more general version of convex program (4.3), where the simplex is replaced by an arbitrary convex set $\mathcal{P} \subseteq \mathbb{R}^n$. The only change required is in the projection step. For Theorem 4.3, we assumed the underlying convex set is the simplex, hence the projection can be computed in time $\tilde{O}(n)$. Projections can be computed in linear time also for the hypercube and ball. For convex sets which are intersections of hyperplanes (or convex paraboloids), computing a projection reduces to optimizing a convex quadratic function over linear (quadratic) constraints. These optimization problems allow for more efficient algorithms than general convex optimization [LVBL98].

As a concrete example of the application of Theorem 4.3, consider the case of strictly convex quadratic programming. In this case, there are m constraint functions of the form $f_j(x) = x^\top A_j x + b_j^\top x + c$, where the matrices A_j are positive-definite. If $\min_{j \in [m]} A_j \succeq H \cdot I$, and $\forall_{x \in \mathbb{S}_n} \|A_j x + b_j\|_2 \leq G$, then Theorem 4.3 implies that an ε-approximate solution can be found in $\tilde{O}(\frac{G^2}{H\varepsilon})$ iterations.

The implementation of Separation Oracle involves finding a constraint violated by more than ε. In the worst case all constrains need be evaluated in time $O(mn^2)$. The gradient of any constraint can be computed in time $O(n^2)$. We conclude that the total running time to obtain a ε-approximation solution is $\tilde{O}(\frac{G^2 mn^2}{H\varepsilon})$. Notice that the input size is mn^2 in this case.

We conclude this subsection with Corollary 4.4 as follows.

proof of Corollary 4.4. Given mathematical program (4.1), we consider the following program

$$\begin{gathered} f_j(x) + \delta\|x\|_2^2 - \delta \leq 0 \quad \forall j \in [m] \\ x \in \mathbb{S}_n \end{gathered} \tag{4.5}$$

This mathematical program has strictly convex constraints, as

$$\forall i \in [m] \ . \ \nabla^2(f_i(x) + \delta\|x\|_2^2 - \delta) = \nabla^2 f_i(x) + 2\delta I \succeq 2\delta I$$

The last inequality follows from our assumption that all constraints in (4.1) are convex and hence have positive semidefinite Hessian. Hence, to apply Theorem 4.3 we can use $H = 2\delta$. In addition, by the triangle inequality the gradients of the constraints of (4.5) satisfy

$$\|\nabla(f_i(x) + \delta\|x\|_2^2 - \delta)\|_2 \leq \|\nabla f_i(x)\|_2 + 2\delta\|x\|_1 \leq G + 2\delta = O(G)$$

where G is the upper bound on the norm of the gradients of the constraints of (4.1). Theorem 4.3 implies that a ε-approximate solution to (4.5) can be computed in $\tilde{O}(\frac{G^2}{\delta\varepsilon})$ iterations, each requiring a single gradient computation and additional $\tilde{O}(n)$ time.

Given a ε-approximate solution to (4.5) denoted y, it satisfies

$$\forall j \in [m] \ . \ f_j(y) + \delta\|y\|_2^2 - \delta \leq \varepsilon \ \Rightarrow \ f_j(y) \leq -\delta\|y\|_2^2 + \delta + \varepsilon \leq \delta + \varepsilon$$

Hence y is also a $(\varepsilon + \delta)$-approximate solution to (4.1). In addition, if (4.1) is feasible, i.e there exists $x^* \in S_n$ such that $\min_{j\in[m]} f_j(x^*) \leq 0$, then so is (4.5) since the same x^* satisfies $\min_{j\in[m]} f_j(x^*)+\delta\|x\|_2^2-\delta \leq \delta\|x\|_2^2-\delta \leq 0$. This implies that proof of infeasibility of (4.5) also proves infeasibility for (4.1).

Choosing $\delta = \varepsilon$, we conclude that a 2ε-approximate solution to (4.1) can be computed in $\tilde{O}(\frac{G^2}{\varepsilon^2})$ iterations. □

4.3.2 Linear and Convex Programs

In this section we prove Theorem 4.5, which gives an algorithm for convex programming that has running time proportional to $\frac{1}{\varepsilon}$. As a simple consequence we obtain corollary 4.6 for linear programs.

Since for general convex programs the constraints are not strictly convex, one cannot apply online algorithms with logarithmic regret directly as in the previous subsection. Instead, we first perform a reduction to a mathematical program with exp-concave constraints, and then approximate the reduced instance.

Proof of Theorem 4.5. In this proof it is easier for us to consider concave constraints rather than convex. Mathematical program (4.1) can be converted to the following by negating each constraint:

$$\begin{aligned} f_j(x) \geq 0 \quad \forall j \in [m] \\ x \in \mathcal{P} \end{aligned} \tag{4.6}$$

where the functions $\{f_j\}$ are all concave such that $\forall x \in \mathcal{P}, j \in [m] \ . \ \|\nabla f_j(x)\|_2 \leq G$ and $\forall x \in \mathcal{P}, j \in [m] \ . \ |f_j(x)| \leq \omega$. This program is even more general than (4.1) as it allows for an arbitrary convex set $\mathcal{P}$ rather than $\mathbb{S}_n$.

Let $\rho = \max_{x\in\mathcal{P}} \min_j\{f_j(x)\}$. The question to whether this convex program is feasible is equivalent to whether $\rho > 0$.

In order to approximately solve this convex program, we consider a different concave mathematical program,

$$\begin{aligned} \log(e + \omega^{-1} f_j(x)) \geq 1 \quad \forall j \in [m] \\ x \in \mathcal{P} \end{aligned} \tag{4.7}$$

It is a standard fact that concavity is preserved for the composition of a non-decreasing concave function with another concave function, i.e. the logarithm of positive concave functions is itself concave. To solve this program we consider the (non-linear) zero sum game defined by the following min-max formulation

$$\lambda^* \triangleq \max_{x\in\mathcal{P}} \min_{j\in[m]} \log(e+\omega^{-1}f_j(x)) \tag{4.8}$$

The following two claims show that program (4.7) is closely related to (4.6).

Claim 4.9. $\lambda^* = \log(e+\omega^{-1}\rho)$.

Proof. Let x be a solution to (4.6) which achieves the value ρ, that is $\forall j \in [m]$. $f_j(x) \geq \rho$. This implies that $\forall j \in [m]$. $\log(e+\omega^{-1}f_j(x)) \geq \log(e+\omega^{-1}\rho)$, and in particular $\forall q \;\; g(x,q) \geq \log(e+\omega^{-1}\rho)$ hence $\lambda^* \geq \log(e+\omega^{-1}\rho)$.

For the other direction, suppose that $\lambda^* = \log(e+z) > \log(e+\omega^{-1}\rho)$ for some $z > \omega^{-1}\rho$. Then there exists an x such that $\forall j \in [m]$. $\log(e+\omega^{-1}f_j(x)) \geq \lambda^* > \log(e+z)$ or equivalently $\forall j \in [m]$. $f_j(x) \geq z > \rho$ in contradiction to the definition of ρ. □

Claim 4.10. *An ε-approximate solution for* (4.7) *is a $3\omega\varepsilon$-approximate solution for* (4.6).

Proof. A ε-approximate solution to (4.7) satisfies $\forall j$. $\log(e+\omega^{-1}f_j(x)) \geq \lambda^* - \varepsilon = \log(e+\omega^{-1}\rho) - \varepsilon$. Therefore, by monotonicity of the logarithm we have

$$\begin{aligned}
\omega^{-1}f_j(x) \quad & \geq e^{\log(e+\omega^{-1}\rho)-\varepsilon} - e \\
& = (e+\omega^{-1}\rho)\cdot e^{-\varepsilon} - e \\
& \geq (e+\omega^{-1}\rho)(1-\varepsilon) - e \quad \text{since } e^{-x} \geq 1-x \\
& = \omega^{-1}\rho(1-\varepsilon) - e\varepsilon
\end{aligned}$$

Which implies

$$f_j(x) \geq \rho(1-\varepsilon) - 3\omega\varepsilon$$

□

We proceed to approximate λ^* using PrimalGameOpt and choose the Online Newton Step (ONS) algorithm (see chapter 3) as OnlineAlg. We note that here the primal player is maximizing payoff as opposed to the minimization version in the proof of Theorem 4.8. The maximization version of Theorem 4.8 can be proved analogously.

In order to analyze the number of iterations required, we calculate some parameters of the constraints of formulation (4.7). See chapter 3 for explanation on how the different parameters effect the regret and running time of Online Newton Step.

The constraint functions are 1-exp-concave, since their exponents are linear functions. Their gradients are bounded by

$$\tilde{G} \triangleq \max_{j\in m}\max_{x\in\mathcal{P}} \|\nabla \log(e+\omega^{-1}f_j(x))\| = \max_{j\in m}\max_{x\in\mathcal{P}} \|\frac{\omega^{-1}\nabla f_j(x)}{e+\omega^{-1}f_j(x)}\| \leq \omega^{-1}G$$

According to Theorem 3.1 in chapter 3, the regret of ONS (4.8) is $O((\frac{1}{\alpha}+GD)n\log T)$. In our setting, $\alpha = 1$ and G is replaced by $\tilde{G}$. Therefore, the regret becomes smaller then

εT after $O(\frac{nGD\omega^{-1}}{\varepsilon})$ iterations. By Theorem 4.8, after $T = \tilde{O}(\frac{nGD\omega^{-1}}{\delta})$ iterations we obtain an δ-approximate solution, i.e a solution x^* such that

$$\min_{j\in[m]} \log(e + \omega^{-1} f_j(x^*)) \geq \lambda^* - \delta$$

Which by claim 4.10 is a $3\omega\delta$-approximate solution to the original math program. Taking $\delta = O(\omega^{-1}\varepsilon)$ we obtain an ε-approximate solution to concave program (4.6) in $T = \tilde{O}(\frac{nGD}{\varepsilon})$ iterations.

We now analyze the running time per iteration. Each iteration requires a call to Separation Oracle in order to find an ε-violated constraint. The gradient of the constraint need be computed. According to the gradient the ONS algorithm takes $O(n^2)$ time to update its internal data structures. Finally ONS computes a generalized projection onto $\mathcal{P}$ (see section 3.5).

If $\mathcal{P} = \mathbb{S}_n$, then $D = 1$ and the bounds of Theorem 4.5 are met.

□

Given Theorem 4.5, it is straightforward to derive corollary 4.6 for linear programs:

proof of Corollary 4.6. For linear programs in format (4.2), the gradients of the constraints are bounded by $\max_{j\in[m]} \|A_j\| \leq 1$. In addition, Separation Oracle is easy to implement in time $O(mn)$ by evaluating all constraints.

Denote by T^S_{proj} the time to compute a generalized projection onto the simplex. A worst case bound is $T^S_{proj} = O(n^3)$, using interior point methods (this is an instance quadratically constrained convex quadratic program, see [LVBL98]).

Plugging these parameters into Theorem 4.5, the total running time comes to

$$\tilde{O}(\frac{n}{\varepsilon} \cdot (nm + n^2 + T_{A,proj}))$$

□

Remark: As is the case for strictly convex programming, our framework actually provides a more general algorithm that requires a Separation Oracle. Given such an oracle, the corresponding optimization problem can be solved in time $\tilde{O}(\frac{n}{\varepsilon} \cdot (n^2 + T_{A,proj} + T_{oracle}))$ where T_{oracle} is the running time of Separation Oracle.

4.3.3 Derivation of previous results

For completeness, we prove Theorem 4.1 using our framework. Even more generally, we prove the theorem for general convex program (4.3) rather than (4.1).

Proof of Theorem 4.1. Consider the associated game with value

$$\lambda^* \triangleq \min_{x\in\mathcal{P}} \max_{j\in[m]} f_j(x) = \max_{p\in S_m} \min_{x\in\mathcal{P}} \sum_{i=1}^{m} p_i f_i(x)$$

The convex problem is feasible iff $\lambda^* \leq 0$. To approximate λ^*, we apply the DualGameOpt meta algorithm. The vectors x_t are points in the convex set $\mathcal{P}$, and p_t are distributions over the constraints, i.e. points in the m dimensional simplex. The payoff functions for OnlineAlg in iteration t are of the form

$$\lambda p \,.\, g(x_t, p) = \sum_i p_i f_i(x_t)$$

The online algorithm used to implement OnlineAlg is the Multiplicative Weights algorithm (MW). According to Theorem 2.2 in section 2.3.3, the regret of MW is bounded by $\text{Regret}_T(MW) = O(G_\infty \sqrt{T \log m})$ (the dimension of the online player is m in this case). Hence, the number of iterations till the regret drops to εT is $\tilde{O}(\frac{G_\infty^2}{\varepsilon^2})$. According to Theorem 4.8, this is the number of iterations required to obtain an ε-approximation.

To bound G_∞, note that the payoff functions $\lambda p \,.\, g(x_t, p)$ are linear. Their gradients are m-dimensional vectors such that the i'th coordinate is the value of the i'th constraint on the point x_t, i.e. $f_i(x_t)$. Thus, the ℓ_∞ norm of the gradients can be bounded by

$$G_\infty = \max_{x \in \mathcal{P}} \max_{t \in [T]} \nabla(\lambda p \,.\, g(x_t, p)) \leq \max_{i \in [m]} \max_{x \in \mathcal{P}} f_i(x)$$

And the latter expression is bounded by the width $\omega = \max_{i \in [m]} \max_{x \in \mathcal{P}} |f_i(x)|$. Thus the number of iterations to obtain an ε-approximate solution is bounded by $\tilde{O}(\frac{\omega^2}{\varepsilon^2})$.

In each iteration, the MW algorithm needs to update the current online strategy (the vector p_t) according to the gradient in time $O(m)$. This requires a single gradient computation. $\square$

4.4 Lower bounds

The algorithmic scheme described hereby generalizes previous approaches, which are generally known as **Dantzig-Wolfe-type** algorithms. These algorithms are characterized by the way the constraints of mathematical program (4.1) are accessed: every iteration only a single Optimization Oracle call is allowed.

For the special case in which the constraints are linear, there is a long line of work leading to tight lower bounds on the number of iterations required for algorithms within the Dantzig-Wolfe framework to provide an ε-approximate solution. Already in 1977, Khachiyan proved an $\Omega(\frac{1}{\varepsilon})$ lower bound on the number of iterations to achieve an error of ε. This was tightened to $\Omega(\frac{1}{\varepsilon^2})$ by Klein and Young [KY99], and independently by Freund and Schapire [FS99]. Some parameters were tightened in [AHK05a].

For the game theoretic framework we consider, it is particularly simple and intuitive to derive tight lower bounds. These lower bounds do **not** hold for the more general Dantzig-Wolfe framework. However, virtually all lagrangian-relaxation-type algorithms known can be derived from our framework. Thus, for all these algorithms lower bounds on the running time in terms of ε can be derived from the following observation.

In our setting, the number of iterations depends on the regret achievable by the online game playing algorithm which is deployed. Tight lower bounds are known on regret achievable by online algorithms. Theorem 2.3 in section 2.5 shows that any online convex optimization algorithm incurs $\Omega(DG\sqrt{T})$ regret for linear cost functions. For linear cost functions over the real line segment $[-1, 1]$ the following weaker bounds hold

Lemma 4.11 (folklore)**.** *For linear cost functions over the real line segment* $[-1, 1]$ *any online convex optimization algorithm incurs* $\Omega(G_\infty \sqrt{T})$ *regret.*

Proof. This can be seen by a simple randomized example. Consider $\mathcal{P} = [-1, 1]$ and linear functions $f_t(x) = r_t x$, where $r_t = \pm 1$ are chosen in advance, independently with equal probability. $\mathbf{E}_{r_t}[f_t(x_t)] = 0$ for any t and x_t chosen online, by independence of x_t and r_t. However, $\mathbf{E}_{r_1,\ldots,r_T}[\min_{x \in K} \sum_1^T f_t(x)] = \mathbf{E}[-|\sum_1^T r_t|] = -\Omega(\sqrt{T})$. Multiplying r_t by any constant (which corresponds to G_∞) yields the result. □

The above simple lemma is essentially the reason why it took more than a decade to break the $\frac{1}{\varepsilon^2}$ running time. The reason why we obtain algorithms with linear dependance on ε is the use of strictly convex constraints (or, in case the original constraints are linear, apply a reduction to strictly convex constraints).

4.5 A general min-max theorem

In this section prove a generalized version of the von Neumann min-max theorem. The proof is algorithmic in nature, and differs from previous approaches which were based on fixed point theorems.

Freund and Schapire [FS99] provide an algorithmic proof of the (standard) min-max theorem, and this proof is an extension of their ideas to the more general case. The additional generality is in two parameters: first, we allow more general underlying convex sets, whereas the standard min-max theorem deals with the n-dimensional simplex $\mathbb{S}_n$. Second, we allow convex-concave functions as defined below rather than linear functions. Both generalities stems from the fact that we use general online convex optimization algorithms as the strategy for the two players, rather than specific "expert-type" algorithms which Freund and Schapire use. Other than this difference, the proof itself follows [FS99] almost exactly.

The original minimax theorem can be stated as follows.

Theorem 4.12 (von Neumann)**.** *If* X, Y *are finite dimensional simplices and* f *is a bilinear function on* $X \times Y$*, then* f *has a saddle point, i.e.*

$$\min_{x \in X} \max_{y \in Y} f(x, y) = \max_{y \in Y} \min_{x \in X} f(x, y)$$

Here we consider a more general setting, in which the two sets X, Y can be arbitrary closed, non-empty, bounded and convex sets in Euclidian space and the function f is convex-concave as defined by:

Definition 4.13. A function f on $X \times Y$ is convex-concave if for every $y \in Y$ the function $\forall x \in X \quad f_y(x) \triangleq f(x, y)$ is convex on X and for every $x \in X$ the function $\forall y \in Y \quad f_x(y) \triangleq f(x, y)$ is concave on Y.

Theorem 4.14. *If* X, Y *are closed non-empty bounded convex sets and* f *is a convex-concave function on* $X \times Y$*, then* f *has a saddle point, i.e.*

$$\max_{y \in Y} \min_{x \in X} f(x, y) = \min_{x \in X} \max_{y \in Y} f(x, y)$$

Proof. Let $\mu^* \triangleq \max_{y\in Y} \min_{x\in X} f(x, y)$ and $\lambda^* \triangleq \min_{x\in X} \max_{y\in Y} f(x, y)$. Obviously $\mu^* \le \lambda^*$ (this is called *weak duality*).

Apply the algorithm PrimalDualGameOpt with any low-regret online convex optimization algorithm. [4] Then by the regret guaranties we have for the first algorithm (let $\bar{y} = \frac{1}{T}\sum_{t=1}^{T} y_t$)

$$\begin{array}{rcll} \frac{1}{T}\sum_{t=1}^{T} f(x_t, y_t) & \le & \min_{x\in X} \frac{1}{T}\sum_{t=1}^{T} f(x, y_t) + \frac{R_1}{T} & \\ & \le & \min_{x\in X} f(x, \bar{y}) + \frac{R_1}{T} & \text{concavity of } f_x \\ & \le & \max_{y\in Y} \min_{x\in X} f(x, y) + \frac{R_1}{T} & \\ & = & \mu^* + \frac{R_1}{T} & \end{array}$$

Similarly for the second online algorithm we have (let $\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x_t$)

$$\begin{array}{rcll} \frac{1}{T}\sum_{t=1}^{T} f(x_t, y_t) & \ge & \max_{y\in Y} \frac{1}{T}\sum_{t=1}^{T} f(x_t, y) - \frac{R_1}{T} & \\ & \ge & \min_{x\in X} f(\bar{x}, y) + \frac{R_1}{T} & \text{convexity of } f_y \\ & \ge & \min_{x\in X} \max_{y\in Y} f(x, y) + \frac{R_1}{T} & \\ & = & \lambda^* + \frac{R_1}{T} & \end{array}$$

Combining both observations we obtain

$$\lambda^* - \frac{R_2}{T} \le \mu^* + \frac{R_1}{T}$$

As $T \mapsto \infty$ we obtain $\mu^* \ge \lambda^*$.

$\square$

[4] for a low-regret algorithm to exist, we need f to be convex-concave and the underlying sets X, Y to be convex, nonempty, closed and bounded.

Chapter 5

Estimating Haplotype Frequencies Efficiently

This chapter describes an algorithm for a computational biology application, which originally appeared in a joint paper with Eran Halperin [HH06]. The original motivation was completely biological and had no relation to online convex optimization. However, the mathematical program arising in this application is surprisingly similar to that from universal portfolio management discussed earlier. Given this similarity, it is tempting to apply our online convex optimization techniques, and indeed this results in simpler analysis, as described below.

The first two sections deals with describing the biological motivation, and how to model it mathematically. Some background in biology is assumed for this part. The remaining sections deal with optimization, and the reader who is only interested in these aspects can skip directly to section 5.3.

The original algorithm from [HH06], called HaploFreq, was implemented and benchmarked on actual biological data sets. As a final note, recently the HaploFreq software suite was made available from the National Center for Biotechnology Information (NCBI) web site, and is used by biologists around the world to analyze genetic data.

5.1 Introduction

The effort to characterize human variation is currently a major focus for the international research community [NIH02]. A central motivation is to associate specific portions of our genome with disease and traits. Most of the genetic variation among different people can be characterized by single nucleotide polymorphisms (SNPs), which are mutations at a single nucleotide position that occurred once in human history and were passed on through heredity. In order to understand the structure of this variation, we need to be able to determine the *haplotypes* of individuals, or which nucleotide base occurs at each position for each chromosome.

As opposed to haplotypes, the genotype gives the bases at each SNP for both copies of the chromosome, but loses the information as to the chromosome on which each base appears. Unfortunately, many sequencing techniques provide the genotypes and not the haplotypes. Haplotype analysis has become increasingly common in genetic studies of human disease. However, many of these methods rely on phase information, that is, the haplotype information vs. the genotype information. Phase can be inferred by genotyping family members of each subject, but this has its downsides because of logistic and budget issues. Alternatively, laboratory techniques such as long range PCR or chromosomal isolation have been also used [PBH$^+$01, MBTB$^+$96] but these are often costly and are not suitable for large scale polymorphism screening. As an alternative to those technologies, many computational methods have been developed for phasing the genotypes (e.g. [Cla90, Gus00, Gus01, LBI$^+$01, SSD01, NQXL01, Gus02, HE04, KS04].

Even though much of the attention was aimed at finding the haplotype phase, it is usually crucial to estimate correctly the haplotype frequencies in the population and not necessarily to phase the individual genotypes. For instance, in disease association studies, it is usually more informative to find the discrepancies between the control haplotype distribution and the cases haplotype distribution, than to find the phase of the haplotypes. The most likely estimation for the haplotype distribution in a population can be viewed as a weighted average over all possible phasing options. Therefore, finding the most likely phase and counting the number of occurrences of each haplotype could be used as a crude estimate for the haplotype distribution. In some cases this crude estimate may be inaccurate and more accurate frequency estimators are needed.

There are algorithms, based on the Expectation Maximization technique, that directly estimate the haplotype frequencies ([ES95, FS00, HK95, LWU95]). These methods use a likelihood function based on the underlying assumption that the Hardy-Weinberg equilibrium holds (that the two haplotypes of an individual are independently drawn from the haplotype distribution in the population). In particular, those methods try to find a *haplotype* distribution which maximizes the probability of observing genotypes in the given sample, under the assumption of Hardy-Weinberg equilibrium.

One of the main drawbacks in all previous methods is that there is no guarantee that the algorithm converges to a global maximum, or that the algorithm converges in polynomial time. Both the convergence of the EM algorithm to a global optimum and its running time are heavily affected by the starting point of the algorithm which is usually a 'reasonable' guess or a random point.

We present a method called HaploFreq which aims in overcoming the above limitations of previous approaches. Similarly to previous approaches, we use a likelihood function model. Our approach is different from previous approaches in the following aspects. First, we use an algorithm which is provably guaranteed to run efficiently and to find the haplotype distribution assuming that the number of samples is large enough and assuming a uniform error model. Second, we consider two different likelihood functions, one that assumes Hardy-Weinberg equilibrium and another that does not. The latter is used in order to find the *genotype* distribution given missing data, or the *haplotype* distribution given phased haplotypes with missing data. For instance, the phased haplotypes are given when sequencing chromosome X in men, or when sequencing the genome of certain organisms that are either haploid or have a short life span[1].

In the case where the Hardy-Weinberg equilibrium holds, the maximum likelihood function is a multinomial of high degree. In order to find the maximum value of this multinomial we relax the problem by allowing the variables to be n-dimensional vectors instead of real numbers. We then use convex optimization methods to find the maximum value of the relaxed problem. This relaxed objective function can be thought of as an alternative likelihood function since we show that the maximum value of the relaxed function approaches asymptotically to the haplotypes frequencies in the population.

We measured the performance of our algorithm over various data sets and compared it to the most widely used program PHASE [SSD01]. The promising experimental results appear in [HH06].

5.2 Estimating Haplotype Frequencies

One of the most natural tools in disease association studies is the search for discrepancies in the allele distribution between the cases and the controls. A natural extension of this tool is the search for discrepancies between the haplotype distribution in each of the populations. In particular, the haplotype frequency is calculated from the samples of each of the populations, and a statistical test (e.g. chi squared) is performed in order to assess whether the haplotype distributions of the two populations are identical. If the distributions are significantly different, then the region is likely to be correlated with the disease.

In order to estimate the haplotype frequencies in a population, a geneticist would sample a set of n individuals from the population. Throughout this section we assume that these n individuals are independently sampled from a large population. Each sample consists of a genotype, which is the information of the two copies of the chromosome in each base. The haplotype information therefore has to be derived from the genotype information. Furthermore, the sequenced data usually contains some missing data, which adds another complexity to the problem. In this chapter we focus on estimating the haplotype frequencies from the genotype data, with missing data. Both the model and algorithms have natural extensions for other types of noise, such as sequencing errors.

5.2.1 A maximum likelihood approach

In order to formalize the above scenario, we first need to set some notations and definitions. A *complete haplotype* is a binary string of length k. The values 0 and 1 correspond to the mutation and the wild type alleles. A *partial haplotype* is a string over $\{0, 1, *\}^k$. The character '*' corresponds to an unknown value (missing data).

We denote a genotype by a string over $\{0, 1, 2, *\}^k$, where 0,1 correspond to homozygous sites (i.e. the bases of the mother's chromosome and the father's chromosomes are the same), the value '2' corresponds to a heterozygous position, that is, a position where

[1]For example in Drosophila, the phased haplotypes can be obtained by breeding

the mother chromosome carries a different base than the father chromosome and '*' corresponds to unknown values for both haplotypes. For a given genotype g or haplotype h, we denote by $g(i)$ ($h(i)$ respectively) its value in the i-th coordinate.

We say that a genotype $g \in \{0,1,2,*\}^k$, and a pair of complete haplotypes $h^1, h^2 \in \{0,1\}^k$ are **compatible** if for every position i, if $g(i) \in \{0,1\}$ then $h^1(i) = h^2(i) = g(i)$ and if $g(i) = 2$ then $h^1(i) \neq h^2(i)$.

For a genotype g, we define $\mathcal{C}(g)$ to be the set of pairs of haplotypes that are compatible with g. We assume that the genotypes admit a Hardy-Weinberg equilibrium, that is, that the two haplotypes of each individual are independently picked from the distribution of haplotypes in the population.

Let $\mathcal{P}$ be a distribution over the set of all possible complete haplotypes of length k. We denote by $p(h)$ the probability assigned to the haplotype h by $\mathcal{P}$. We consider the following likelihood function [ES95] of a set of partial genotypes $\mathcal{G}$ and a distribution $\mathcal{P}$:

$$\mathcal{L}(\mathcal{G},\mathcal{P}) = \prod_{g \in \mathcal{G}} \sum_{(h_1,h_2) \in \mathcal{C}(g)} p(h_1)p(h_2). \tag{5.1}$$

The function $\mathcal{L}(\mathcal{G},\mathcal{P})$ is simply the probability of observing the genotypes $\mathcal{G}$ in a random sample of the population under Hardy-Weinberg equilibrium, given that the distribution of complete haplotypes in the population is $\mathcal{P}$ and that the distribution of missing data in a genotype g does not depend on the contents of g.

When the sample size approaches infinity, the maximum likelihood is attained when $\mathcal{P}$ is the actual distribution of haplotypes in the population[2]. Therefore, it is only natural to aim in finding the distribution $\mathcal{P}$ which maximizes the likelihood and to estimate the distribution of haplotypes in the population as $\mathcal{P}$. Previous methods [ES95, FS00, SSD01] use Expectation Maximization (EM) in order to find the maximum likelihood. When using EM, both the running time and the convergence to a global maximum depend on the starting point. In particular, these algorithms may be exponential, and they may give a non-optimal solution, even when the number of samples is large. In Section 5.4 we introduce an alternative approach which is guaranteed to converge to a global optimum of another likelihood function $\mathcal{L}_2(\mathcal{G},\mathcal{P})$. We further show in Section 5.4.1 that $\mathcal{L}$ and $\mathcal{L}_2$ have the same asymptotic behavior under Hardy-Weinberg.

5.2.2 Working with phased data

In some cases we are given the phased genotypes, possibly with missing data. For instance, some sequencing techniques provide the haplotypes and not the genotypes. In haploid organisms, or in diploid organisms with short life span such as drosophila[3] we can get the phased haplotypes. It is therefore interesting to estimate the haplotype frequencies given a sample of haplotypes with missing data. This approach may also be useful to estimate the *genotype* frequencies, given a sample of genotypes with missing data. The latter may be particularly important when there are departures from Hardy-Weinberg equilibrium in the underlying genotype distribution.

[2]This is true under some reasonable assumptions on the distribution of the missing data.

[3]In diploid organisms the haplotype data is found through breeding.

In order to formalize the above scenario, we need to introduce a few more notations and definitions. We say that a partial haplotype $h_1 \in \{0,1,*\}^k$ is **consistent** with a complete haplotype $h_2 \in \{0,1\}^k$ if they share the same values whenever $h_1(i) \neq *$. Given a partial haplotype h, we define $\mathcal{C}(h)$ to be the set of complete haplotypes that are consistent with h.

As before, let $\mathcal{P}$ be a distribution over the set of all possible complete haplotypes of length k. Given the set of partial haplotypes $\mathcal{H}$, the likelihood of $\mathcal{P}$ is given by

$$\mathcal{L}(\mathcal{H},\mathcal{P}) = \prod_{h\in\mathcal{H}} \sum_{h'\in\mathcal{C}(h)} p(h').$$

The function $\mathcal{L}(\mathcal{H},\mathcal{P})$ is simply the probability of observing the partial haplotypes $\mathcal{H}$ in a random sample of the population, given that the distribution of complete haplotypes in the population is $\mathcal{P}$ and that the distribution of missing data in a haplotype h does not depend on the contents of h.

Again, in order to estimate the haplotype frequencies, we find the distribution $\mathcal{P}$ that maximizes the likelihood $\mathcal{L}(\mathcal{H},\mathcal{P})$. In Section 5.3 we introduce an efficient polynomial time algorithm that finds the global maximum of $\mathcal{L}(\mathcal{H},\mathcal{P})$. This may seem surprising given that we essentially find a maximum point of a polynomial of potentially high degree. In general, finding an extremum of a polynomial is an intractable problem.

5.3 Estimating Haplotype Frequencies from a Phased Sample

In this section we introduce an algorithm which estimates the haplotype frequencies in a population given a sample of phased haplotypes with missing data.

Formally, given a set $\mathcal{H}$ of n partial haplotypes, we are interested in finding a distribution $\mathcal{P}$ which maximizes the function $\mathcal{L}(\mathcal{H},\mathcal{P})$ which is given in the previous section. Equivalently, are interested in maximizing the logarithm of the likelihood function $\mathcal{H}$ (taking the logarithm is useful to avoid numerical instabilities). Thus, finding the distribution of maximum likelihood can be done by solving the following mathematical programming problem:

$$\begin{array}{rll} \text{Maximize} & \sum_{h\in\mathcal{H}} \log(\sum_{h'\in\mathcal{C}(h)} p(h')) & \\ \text{s.t.} & \sum_{h\in\{0,1\}^k} p(h) = 1 & \\ & p(h) \geq 0 & , h \in \{0,1\}^k \end{array}$$

We will use the following definition in order to simplify the notations.

Definition 5.1. Given a partial haplotype $h \in \{0,1,*\}^k$ and a set of haplotypes $S = \{h_1, ..., h_n\} \subseteq \{0,1\}^k$, define the **compatibility vector** of h with respect to S as a vector $A_h \in \{0,1\}^n$ such that $A_h(i) = 1$ if $h_i \in \mathcal{C}(h)$ and $A_h(i) = 0$ otherwise.

Note that in practice the values of k are relatively small, and the set of possible haplotypes is limited to a reasonable size. A typical value for k is in the range of $10-50$, and there are typically at most a few hundreds of possible haplotypes, that is, haplotypes that are compatible with one of the genotypes.

Using this definition, the maximum likelihood formulation above is equivalent to solving the following problem. Note that we took the logarithm of the objective functions and scaled by the number of constraints. Maximizing the logarithm of the maximum likelihood is standard, to avoid numerical instabilities. We scale by the number of summands so that approximation of this mathematical program is independent of the problem dimension.

Definition 5.2 (FREQUENCY ESTIMATION OF PHASED GENOTYPES). .
Input: A matrix $A \in \{0,1\}^{m \times n}$ consisting of n row vectors $\{A_1, ..., A_m\} \in \{0,1\}^n$
Goal: Find a vector $\mathbf{p} \in \mathbb{R}_+^n$, such that:

1. $\mathbf{p} \in S_n = \{p \in \mathbb{R}^n | \sum_{i=1}^n p_i = 1 \;\; ; \;\; \forall i \; p_i \geq 0\}$
2. The following quantity is maximized: $f(\mathbf{p}) = \frac{1}{m} \sum_{i=1}^m \log(A_i^\top \cdot \mathbf{p})$

The above mathematical program concerns the maximization of a concave function (a sum of logarithms - which are concave functions - is also concave) over a convex set - the n dimensional simplex. Thus, FREQUENCY ESTIMATION OF PHASED GENOTYPES without assumption of Hardy-Weinberg equilibrium can be solved using the ellipsoid method or more efficient interior point methods. Both approaches have large (although polynomial) theoretical running time bounds suffers from poor performance in practice for our application.

5.3.1 Efficient Approximation Scheme

We proceed to provide an efficient combinatorial algorithm that approximates the solution to FREQUENCY ESTIMATION OF PHASED GENOTYPES to within any required (constant) precision parameter. The algorithm and analysis are simpler, albeit less efficient, than the ones presented in the original work [HH06], and are based on the machinery developed in the previous chapters. We remark that Helmbold et al [HSSW95] describe a very similar algorithm, that is less efficient in terms of the approximation guarantee.

In the next subsection we present the original HAPLOFREQ algorithm and its analysis, which are based on different techniques altogether.

As in previous chapters, we denote by $\mathbf{p}^*$ the optimum solution to a given instance of FREQUENCY ESTIMATION OF PHASED GENOTYPES. A solution $\mathbf{p} \in S_n$ is ε-approximate if $f(\mathbf{p}) \geq f(\mathbf{p}^*) - \varepsilon$.

As a first step, we describe a simple algorithm which follows from the most essential application of the techniques of Chapter 4. Notice that the optimal solution of FREQUENCY ESTIMATION OF PHASED GENOTYPES satisfies $\forall i \in [m] \; . \; A_i^\top \mathbf{p}^* \geq \tau > 0$. Hence, we can write the following mathematical program, where α is an estimate of the value of the optimal solution.

$$\begin{aligned} f_0(\mathbf{p}) &= \frac{1}{m}\sum_{i=1}^{m}\log(A_i^\top \mathbf{p}) - \alpha \geq 0 \\ f_i(\mathbf{p}) &= \frac{1}{\tau}\cdot(A_i^\top \mathbf{p} - \tau) \geq 0 \quad \forall i \in [m] \\ & \mathbf{p} \in S_n \end{aligned} \tag{5.2}$$

As detailed in Chapter 4, we reduce this optimization problem into a zero sum game with value

$$\lambda^* = \max_{\mathbf{p}\in S_n} \min_{i\in[0,m]} f_i(\mathbf{p})$$

The MW HaploFreq algorithm to solve this mathematical program, depicted in figure 5.1, is derived from meta algorithm PrimalGameOpt, while using the Multiplicative Weights algorithm as OnlineAlg. The theoretical guarantee we can prove for this algorithm is given in the following theorem

```
MW Haplofreq.
Inputs: matrix $A \in \{0,1\}^{n\times m}$, approximation parameter $\varepsilon$.
Guess $\alpha \in [-\log n, 0]$ for program 5.2 by binary search.
while $t < \frac{\log n}{\tau^2\varepsilon^2}$ repeat
    • Start with the uniform distribution $\mathbf{p}_1 = \frac{1}{n}\vec{1} \in S_n$. Let $\forall i \in [n]$ . $w_i^1 = 1$
    • At iteration $t$, let $i_t = \arg\min_{0=1}^{m} f_i(\mathbf{p}_{t-1})$ be the index of the most violated constraint.
      if $f_{i_t}(\mathbf{p}_{t-1}) \geq -\varepsilon$, return $\mathbf{p}_{t-1}$.
      else if $j = \arg\min_{i=1}^{m} f_i(\mathbf{p}_{t-1}) < -\varepsilon$ set $i_t \leftarrow j$.
    • update
          $w_i^t = w_i^{t-1}\cdot(1 + \frac{\tau}{2}\nabla f_{i_t}(\mathbf{p}_{t-1})[i])$
      and let $\mathbf{p}_t \triangleq \frac{w^t}{\|w^t\|_1}$
    • $t \leftarrow t+1$
return FAIL
```

Figure 5.1: The Multiplicative Weights algorithm applied to Frequency Estimation of Phased Genotypes

Theorem 5.3. *Algorithm* **MW HaploFreq** *returns a ε-approximate solution in time $\tilde{O}(\frac{mn}{\tau^2\varepsilon^2})$.*

Proof. The **MW HaploFreq** algorithm is an instantiation of the PrimalGameOpt meta algorithm, using the Multiplicative Weights online convex optimization algorithm as OnlineAlg.

Correctness (the fact that the algorithm returns a ε-approximate solution) follows immediately from Theorem 4.8. The number of iteration required to produce an ε-approximate solution, given the regret bounds for the Multiplicative Weights algorithm (see Chapter 2) is $O(\frac{G_\infty^2 \log n}{\varepsilon^2})$. Each iteration involves computing the gradient of the objective function, which takes $O(mn)$ time, and updating the current solution $\mathbf{p}_t$, which takes another $O(n)$ time.

The parameter α can be guessed by binary search over the solution range, which is $[\log \frac{1}{n}, 0]$. Hence, binary search over this space up to precision ε requires $O(\log(\frac{\log n}{\varepsilon})) = \tilde{O}(1)$ iterations.

G_∞ can be bounded as follows: for all constraint functions $\{f_i, i > 0\}$, the gradient is simply $\frac{1}{\tau} \cdot A_i$, and hence its infinity norm is bounded by $\frac{1}{\tau}$. Observe that if $\min_{i \in [m]} A_i^\top \mathbf{p} < \frac{1}{2}\tau$, then the value of the i'th constraint is at most $f_i(\mathbf{p}) \leq -\frac{1}{2} \leq -\varepsilon$, thus violated by more than ε.

Hence, the objective function constraint is the only constraint violated by ε only if $\min_{i \in [m]} A_i^\top \mathbf{p} \geq \frac{1}{2}\tau$. In this case, the infinity norm of the gradient is at most

$$\|\nabla f_0(\mathbf{p})\|\|_\infty = \left\| \frac{1}{m} \sum_{i=1}^{m} \frac{A_i}{A_i^\top \mathbf{p}} \right\|_\infty \leq \frac{1}{\min_i A_i^\top \mathbf{p}} \leq \frac{1}{\tau}$$

Since every row of the matrix A contains at least one non-zero entry (otherwise we could reduce the problem's dimension to begin with). Overall $G_\infty \leq O(\frac{1}{\tau})$, and the number of iterations to achieve an ε-approximation is bounded by $\tilde{O}(\frac{1}{\tau^2 \varepsilon^2})$.

□

The key to obtain a purely polynomial running time is the following lemma.

Lemma 5.4. *For the optimum* $\mathbf{p}^*$ *of an instance of* Frequency Estimation of Phased Genotypes, *it holds that* $\min_i A_i^\top \mathbf{p}^* = \Omega(\frac{1}{mn})$.

Proof. By the biological assumptions, each compatibility vector has at least one non-zero coordinate. Suppose that for $\mathbf{p}^*$ there exists some A_i, w.l.o.g A_1, for which $A_1^\top \mathbf{p}^* < \frac{c}{mn}$, for some small constant c to be determined later.

Let $g(\mathbf{p}) = e^{mf(\mathbf{p})} = \prod_{j=1}^{m} A_j^\top \mathbf{p}$. Obviously g and f have the same optimum over the simplex.

Consider $\mathbf{p'} = \mathbf{p}^*(1 - \frac{c}{m}) + \frac{c}{mn}\vec{1}$. Naturally, $\mathbf{p'}$ is also a distribution. We show that it has higher objective value, contradicting the optimality of $\mathbf{p}^*$. Notice that

$$A_1^\top \mathbf{p'} \geq A_1^\top \mathbf{p}^*(1 - \frac{c}{m}) + \frac{c}{mn} \geq A_1^\top \mathbf{p}^*(2 - \frac{c}{m})$$

In addition, for all other terms we have

$$A_i^\top \mathbf{p'} \geq A_i^\top \mathbf{p}^*(1 - \frac{c}{m})$$

Hence,

$$\begin{aligned} g(\mathbf{p}') &= \prod_{i=1}^{m} A_i^\top \mathbf{p}' \\ &\geq (2 - \tfrac{c}{m})(1 - \tfrac{c}{m})^{m-1} \prod_{j=1}^{m} A_j^\top \mathbf{p}^* \\ &\geq \tfrac{3}{2} e^{-2c} g(\mathbf{p}^*) > g(p^*) \qquad \text{for } c \leq \tfrac{1}{8} \end{aligned}$$

In contradiction to the optimality of $\mathbf{p}^*$.

□

Lemma 5.4 and Theorem 5.3 imply that the running time of **MW HaploFreq** is polynomial in the input size, as given in the following corollary.

Corollary 1. Setting $\tau = \Omega(\frac{1}{mn})$, algorithm **MW HaploFreq** returns a ε-approximate solution in time $\tilde{O}(\frac{m^3 n^3}{\varepsilon^2})$.

5.3.2 The HaploFreq algorithm

In this section we present the HaploFreq algorithm, given in Figure 5.2. The algorithm and analysis are not derived from the framework of chapter 4, although a very similar algorithm could be derived using Online Gradient Descent as the online algorithm. The running time bounds are better than those of the algorithm in the previous section, and any straightforward application of the framework. Another advantage of HaploFreq is its easy generalization to the semidefinite version of the problem we shall encounter in the next chapter.

```
HaploFreq
Inputs: matrix $A \in \{0,1\}^{n \times m}$, approximation parameter $\varepsilon$.
Set $t \leftarrow 1$ , $\mathbf{p}_1 \leftarrow \frac{1}{n}\vec{1}$.
Set $\delta_1 \leftarrow$ FindDelta $(\mathbf{p}_1, A)$.
while $\sum_{i=1}^{m} \frac{A_i^\top \delta_t}{A_i^\top \mathbf{p}_t} > \varepsilon$ repeat
    • $t \leftarrow t + 1$
    • Update $\mathbf{p}_t \leftarrow \mathbf{p}_{t-1} + \frac{\tau^2 \varepsilon}{16m} \delta_{t-1}$
    • if $\min_{i \in [m]} A_i^\top \mathbf{p}_t \leq \tau$ then $\mathbf{p}_t \leftarrow (1 - \frac{1}{8m})\mathbf{p}_t + \frac{1}{8mn}\mathbf{1}$
    • Set $\delta_t \leftarrow$ FindDelta $(\mathbf{p}_t, A)$
return $\mathbf{p}_t$
```

Figure 5.2: Algorithm HaploFreq

Our algorithm is a hill climbing algorithm. We start from the uniform distribution, and make a series of improvement steps until required performance guarantee is reached. In contrast to the algorithms of the previous subsection, or any of the algorithms we encountered so far, the solution is guarantied to improve in **every** iteration.

In the rest of this section we prove the following performance guarantee (notice this is a factor $\frac{m}{\varepsilon}$ improvement over the algorithm of the previous section).

Theorem 5.5. *For any constant $\varepsilon > 0$, the algorithm* HaploFreq *(ε) finds an ε-approximate solution in time $\tilde{O}(\frac{m^2 n^3}{\varepsilon})$.*

To prove this theorem, we show that HaploFreq makes a series of improvements and quantify by how much the objective value changes. Key in this scheme is the following definition.

Definition 5.6. Define a **ε-good** vector with respect to a current solution $\mathbf{p}$ as a vector δ that satisfies:

$$\sum_{i=1}^{n} \delta_i = 0 \quad ; \quad 0 \le \delta_i + \mathbf{p}(i) \le 1 \quad ; \quad |A_i \delta| \le 2 \quad ; \quad \sum_{i=1}^{m} \frac{A_i^\top \vec{\delta}}{A_i^\top \mathbf{p}} \ge \varepsilon$$

The procedure FindDelta returns an ε-good vector if one exists. Note that FindDelta is a linear optimization problem can be implemented using linear programming, but this is not very efficient. The following combinatorial method runs in $\tilde{O}(nm)$ time.

```
Procedure FindDelta (p,A)
Let $\vec{\alpha}$ such that $\forall i \;.\; \alpha_i = \sum_{j=1}^{m} \frac{A_{ji}}{A_i^\top \mathbf{p}}$
Suppose w.l.o.g that $\alpha_1 \le \alpha_2 \le ... \le \alpha_m$ (o/w sort $\vec{\alpha}$)
Set $\delta_m = 1 - p_m$
Set $\forall i < m \;.\; \delta_i = -p_i$
return $\vec{\delta}$
```

Lemma 5.7. *The procedure* FindDelta *finds a ε-good vector for the largest possible ε, and can be implemented in time $\tilde{O}(nm)$.*

Proof. The vector returned by FindDelta obviously satisfies the second and third of the conditions of a ε-good vector.

As for the first condition, note that:

$$\sum_{i=1}^{n} \delta_i = -\sum_{i<n} p_i + (1 - p_n) = 1 - \sum_{i=1}^{n} p_i = 0$$

In addition, δ maximizes $\sum_{i=1}^{m} \frac{A_i \vec{\delta}}{A_i^\top p}$ under the first two conditions. This follows from the fact $\sum_{i=1}^{m} \frac{A_i \vec{\delta}}{A_i^\top p} = \vec{\alpha}^T \cdot \vec{\delta}$ and the definition of $\vec{\delta}$. □

To prove Theorem 5.5, we first prove that we can always find an (εm)-good vector if our current solution is not a ε-approximate solution. Recall that our objective function is $f(\mathbf{p}) = \frac{1}{m} \sum_{i=1}^{m} A_i^\top \mathbf{p}$.

Lemma 5.8. *If $f(p^*) - f(p_t) \ge \varepsilon$, then there exists an (εm)-good vector δ_t.*

Proof. The optimal solution gives rise to a natural vector $\delta := \mathbf{p}^* - \mathbf{p}_t$. It obviously satisfies the first three conditions above, and as for the last:

$$\begin{aligned}\sum_{i=1}^{m} \frac{A_i^\top \delta}{A_i^\top \mathbf{p}_t} &= \sum_{i=1}^{m} \frac{A_i^\top \mathbf{p}^* - A_i^\top \mathbf{p}_t}{A_i^\top \mathbf{p}_t} = \sum_{i=1}^{m} \frac{A_i^\top \mathbf{p}^*}{A_i^\top \mathbf{p}_t} - m \\ &\geq m \cdot \sqrt[m]{\prod_{i=1}^{m} \frac{A_i^\top \mathbf{p}^*}{A_i^\top \mathbf{p}_t}} - m = m \cdot \sqrt[m]{e^{m(f(\mathbf{p}^*) - f(\mathbf{p}_t))}} - m \\ &\geq m \cdot \sqrt[m]{e^{\varepsilon m}} - m \;=\; m \cdot (e^{\varepsilon} - 1) \geq \varepsilon m\end{aligned}$$

where the first inequality follows from the arithmetic and geometric mean inequality and the last inequality is true since $e^x \geq x + 1$. □

Lemma 5.8 shows that if we are still far from the optimal solution then there is at least one (εm)-good vector. Since one such vector exists, FindDelta is guarantied to provide such a vector. We now show that the resulting improvement step brings us closer to the optimum.

Lemma 5.9. *Let $0 < \varepsilon \leq \frac{\sqrt{m}}{\tau}$ and let δ_t be a ε-good vector with respect to p_t. As before, let $\tau = \min_{i \in [m]} A_i^\top p_t$. Let $p_{t+1} := p_t + \sigma \delta_t$ where $\sigma = \frac{\tau^2 \varepsilon}{8m}$. Then $f(p_{t+1}) - f(p_t) \geq \dfrac{\epsilon^2 \tau^2}{16m}$.*

Proof. Denote $c_i := \frac{A_i^\top \delta_t}{A_i^\top \mathbf{p}_t}$. By Lemma 5.7 we know that $|A_i^\top \delta_t| \leq 2$, and therefore $|c_i| \leq \frac{2}{\tau}$. Hence:

$$\begin{aligned}m(f(p') - f(p)) &= \sum_{i=1}^{m} \log\left(\frac{A_i^\top (p + \sigma\delta)}{A_i^\top p}\right) = \sum_{i=1}^{m} \log(1 + \sigma c_i) \\ &\geq \sum_{i=1}^{m} \left[(\sigma c_i) - (\sigma c_i)^2\right] \geq \sigma\varepsilon - \sigma^2 \sum_{i=1}^{m} c_i^2 \geq \sigma\varepsilon - \frac{4m\sigma^2}{\tau^2} \geq \frac{\epsilon^2 \tau^2}{16m},\end{aligned}$$

where the approximation to the logarithm holds since $|\sigma c_i| \leq \frac{1}{2}$ (as long as $\varepsilon \leq \frac{\sqrt{m}}{\tau}$). □

Now we can prove Theorem 5.5:

Proof of Theorem 5.5. By Lemmas 5.8 and 5.7, at iteration t the procedure FindDelta will return a $(f(\mathbf{p}^*) - f(\mathbf{p}_t))m$-good vector. Thus by Lemma 5.9,

$$f(\mathbf{p}_{t+1}) - f(\mathbf{p}_t) \geq \frac{\tau^2 m^2 (f(\mathbf{p}^*) - f(\mathbf{p}_t))^2}{16m}$$

Therefore, as long as $f(\mathbf{p}^*) - f(\mathbf{p}_t) \geq \epsilon$,

$$f(\mathbf{p}^*) - f(\mathbf{p}_{t+1}) \leq (1 - \frac{\tau^2 m \epsilon}{16})(f(\mathbf{p}^*) - f(\mathbf{p}_t)).$$

Since we can start from a solution of value at least $m \cdot \frac{1}{m} \log \frac{1}{n} = \log \frac{1}{n}$ and the optimal solution is bounded by 1, we have that $f(\mathbf{p}^*) - f(\mathbf{p}_1) \leq \log n$. Therefore, after $\tilde{O}(\frac{1}{\tau^2 m \epsilon})$ iterations we find a solution such that $f(\mathbf{p}^*) - f(\mathbf{p}_t) \leq \epsilon$.

The HaploFreq algorithm updates $\mathbf{p}_t \leftarrow \mathbf{p}_t(1 - \frac{1}{8m}) + \frac{1}{8mn}\vec{1}$ in case $\min_{i \in [m]} A_i^\top \mathbf{p}_t < \tau$. By Lemma 5.4 this causes the objective value of $\mathbf{p}_t$ only to increase, and maintains the invariant that $\tau = \Omega(\frac{1}{mn})$. Therefore the total number of iterations is $\tilde{O}(\frac{1}{\tau^2 m \epsilon}) = \tilde{O}(\frac{mn^2}{\epsilon})$. □

5.4 Estimating Haplotype Frequencies from Unphased Genotypes

We now turn to the case where we have a set of genotypes and our goal is to find the frequencies of the underlying haplotypes. Recall that under the Hardy-Weinberg equilibrium, the likelihood function of a set of genotypes $\mathcal{G}$ and a distribution $\mathcal{P}$ is given by equation (5.1). Thus, finding the haplotype distribution with the maximum likelihood can be done by solving the following mathematical programming problem:

$$\begin{array}{lll} \text{Maximize} & \prod_{g \in \mathcal{G}} \sum_{(h_1,h_2) \in \mathcal{C}(g)} p(h_1)p(h_2) & \\ \text{s.t.} & \sum_{h \in \{0,1\}^k} p(h) = 1 & \\ & p(h) \geq 0 & , h \in \{0,1\}^k \end{array}$$

We follow the approach used for phased data, and try to solve this mathematical program by first abstracting it out. We first need the following definition, which is analogous to Definition 5.1.

Definition 5.10. Given a genotype $g \in \{0,1,2,*\}^k$ and a set of haplotypes $S = \{h_1, ..., h_n\} \subseteq \{0,1\}^k$, define the (symmetric) **compatibility matrix** of g with respect to S as a matrix $A^g \in \{0,1\}^{n \times n}$ such that $A^g_{ij} = 1$ if $(h_i, h_j) \in \mathcal{C}(g)$ and $A^g_{ij} = 0$ otherwise.

It is easy to verify that the maximum likelihood formulation given above can be solved if the following problem can be solved:

Definition 5.11 (Frequency Estimation of Unphased Genotypes)**.** .
Input: A set of matrices $\{A_1, ..., A_m\} \in \{0,1\}^{n \times n}$
Goal: Find a vector $\mathbf{p} \in \mathbb{R}^n_+$, such that:

1. $\mathbf{p} \in S_n = \{p \in \mathbb{R}^n | \sum_{i=1}^n p_i = 1 \;\; ; \;\; \forall i \; p_i \geq 0\}$
2. The following quantity is maximized: $f(\mathbf{p}) = \frac{1}{m}\sum_{i=1}^m \log(\mathbf{p}^\top A_i \mathbf{p})$

Unfortunately, the above mathematical program is NP-hard (see Section 5.5). We therefore suggest to use a different likelihood function $\mathcal{L}_2$ which can be thought of as a relaxation of $\mathcal{L}$. Instead of having a distribution $\mathcal{P}$ over the haplotypes, we assign to each haplotype h_i an n-dimensional vector $\vec{v_i}$, such that $\sum_{i=1}^{n} \vec{v_i} = v_0$ where $\|v_0\| = 1$. The likelihood $\mathcal{L}_2$ is now defined as a function of $\mathcal{G}$ and of $\mathcal{V} = \{\vec{v_1}, \ldots, \vec{v_m}\}$:

$$\mathcal{L}_2(\mathcal{G}, \mathcal{V}) = \prod_{g \in \mathcal{G}} \sum_{(h_i,h_j) \in \mathcal{C}(g)} \vec{v_i} \cdot \vec{v_j}.$$

We call $\mathcal{V}$ a vector distribution of the haplotypes. Note that if we restrict the vectors of $\mathcal{V}$ to be in one dimensional space then $\mathcal{V}$ is a probability distribution and $\mathcal{L}_2(\mathcal{G}, \mathcal{V}) = \mathcal{L}(\mathcal{G}, \mathcal{V})$. The vectors $\mathcal{V}$ can be represented by a positive semidefinite (PSD) matrix $\mathcal{P}$ such that $P_{ij} = \vec{v_i} \cdot \vec{v_j}$, i.e., P_{ij} is the scalar product of $\vec{v_i}$ and $\vec{v_j}$. Therefore, an analogous problem to Frequency Estimation of Unphased Genotypes is the following:

Definition 5.12 (Maximum Relaxed Unphased Likelihood). .
Input: A set of matrices $\{A_1, ..., A_m\} \in \{0,1\}^{n \times n}$
Goal: Find a PSD matrix $P \in \mathbb{R}^{n \times n}$ such that:

1. $\sum_{i,j} P_{ij} = 1$, $\forall i,j \; P_{ij} \geq 0$

2. The following quantity is maximized: $f(P) = \frac{1}{m} \sum_{i=1}^{m} \log(A_i \bullet P)$

If we can solve Maximum Relaxed Unphased Likelihood we could find the vector distribution $\vec{V}$ which maximizes $\mathcal{L}_2(G, V)$, by computing the Cholesky decomposition of the matrix P. In section 5.4.2 we introduce an efficient algorithm which solves Maximum Relaxed Unphased Likelihood in polynomial time.

5.4.1 Asymptotic Behavior of the Likelihood Function.

Finding the maximum likelihood of $\mathcal{L}_2$ does not ensure us that we will converge to the correct haplotype distribution when the number of samples is sufficiently large. We now turn to show that under Hardy-Weinberg equilibrium, and under the assumption that there is no missing data, if the sample size is large enough, the maximum of $\mathcal{L}_2(\mathcal{G}, \mathcal{V})$ is attained in a point which converges to the correct distribution.

Lemma 5.13. *Under the Hardy-Weinberg, the solution to* Maximum Relaxed Unphased Likelihood *converges to the haplotype frequencies in the population.*

Proof. Let the set of sampled genotypes be $\mathcal{G}$. Denote by $p(g)$ the frequency of genotype g in the population. Therefore, $p(g)$ is the probability to sample a genotype $g \in \mathcal{G}$. When the number of samples goes to infinity, the ratio of sampled genotypes g approaches $p(g)$. If the ratio is exactly $p(g)$, then maximizing $\mathcal{L}_2(\mathcal{G}, \mathcal{V})$ is equivalent to maximizing the function

$$\prod_{g \in \mathcal{G}} \left(\sum_{h_i,h_j \in \mathcal{C}(g)} \vec{v_i} \cdot \vec{v_j} \right)^{p_g}$$

It is easy to see that this objective is maximized when:

$$\forall_{g \in \mathcal{G}} \sum_{h_i,h_j \in \mathcal{C}(g)} \vec{v_i} \cdot \vec{v_j} = p_g.$$

As $|\mathcal{G}| \mapsto \infty$, we know that $p_g \mapsto \sum_{i,j\in\mathcal{C}(g)} p_i p_j$. Therefore, one optimal solution to this equation system is the solution $\vec{v_i} = p_i$. Observe that equations above imply that the homozygous genotype g_{ii} with haplotype h_i satisfies that $p_i^2 = \|\vec{v_i}\|^2$. These restrictions, together with the rest of the constraints, determine $\mathcal{V}$ uniquely. We can now use the fact that the function $\max_{\mathcal{V}} \mathcal{L}_2(\mathcal{G}, \mathcal{V})$ is a continuous function in order to complete the proof. $\square$

Now that we know that the solution of Maximum Relaxed Unphased Likelihood converges to the correct solution, in particular we know that for large enough sample the vectors $\vec{v_i}$ should be closed to one dimensional. We therefore define $p_i = \vec{v_i} \cdot \vec{1}$ as the suggested probability distribution. By Lemma 5.13, as the number of samples grow, the probabilities p_i get closer to the true frequencies in the population.

5.4.2 The HaploFreq2 algorithm

In this section we describe the SDP analogue of HaploFreq for the SDP version of the problem, which is given in Figure 5.4.2. Although algorithms with better theoretical running time bounds can be derived from the framework of Chapter 4, HaploFreq2 is the only algorithm which requires only eigenvalue computations and elementary operations. All other methods require Cholesky decomposition or matrix inversions, which are of the most time consuming operations to implement in practice.

The general framework for our algorithm is identical to the algorithm for the linear case. Starting from the uniform solution (the all-ones matrix **J**), the algorithm makes a series of local improvements up to the required performance guarantee is reached. However, for each "improvement step" we amend the current PSD matrix into another PSD matrix such that to improve the overall value of the solution. The improvement matrix, computed by the procedure FindPsdDelta, can be computed by a semidefinite program with linear constraints. A much more efficient way to implement FindPsdDelta is given in Chapter 6. We choose to describe the efficient implementation of FindPsdDelta in Chapter 6 as we feel the exposition of this procedure in the context of the entire chapter should be clearer.

In the rest of this section we prove the following performance guarantee.

Theorem 5.14. *For any constant $\varepsilon > 0$, the algorithm* HaploFreq2 *finds a ε-approximate solution in $\tilde{O}(\frac{mn^4}{\varepsilon})$ iterations.*

The proof is similar in nature to the linear variant proof, with several technical points that need attention. The improvement matrix Δ which is iteratively added to the current solution is defined as follows.

Definition 5.15. Define a (ε, σ)-**good** matrix with respect to a current solution P as a matrix Δ that satisfies:

1. $W \triangleq P + \Delta \succeq 0$
2. $\sum_{i,j} W_{ij} = 1$; $W_{ij} \geq 0$

```
HaploFreq2
Inputs: set of $m$ matrices $\{A_i \in \{0,1\}^{n\times n}, i \in [m]\}$, approximation parameter $\varepsilon$.
Set $t \leftarrow 1$ , $\mathbf{P}_1 \leftarrow \frac{1}{n^2}\mathbf{J}$.
Set $\Delta_1 \leftarrow$ FindPsdDelta $(\mathbf{P}_1, \{A_i\})$.
while $\sum_{i=1}^m \frac{A_i \bullet \Delta_t}{A_i \bullet \mathbf{P}_t} > \varepsilon$ repeat
    • $t \leftarrow t+1$
    • Update $\mathbf{P}_t \leftarrow \mathbf{P}_{t-1} + \frac{\tau^2\varepsilon}{16m}\Delta_{t-1}$
    • if $\min_{i\in[m]} A_i \bullet \mathbf{P}_t \leq \tau$ then $\mathbf{P}_t \leftarrow (1-\frac{1}{20m})\mathbf{P}_t + \frac{1}{20mn^2}\mathbf{J}$
    • Set $\Delta_t \leftarrow$ FindPsdDelta $(\mathbf{P}_t, \{A_i\})$
return $\mathbf{P}_t$
```

Figure 5.3: Algorithm HaploFreq2

3. $\sum_{i=1}^m \frac{A_i \bullet \Delta}{A_i \bullet \mathbf{P}} \geq \varepsilon$ or equivalently $(\sum_{i=1}^m \frac{A_i}{A_i \bullet \mathbf{P}}) \bullet W \geq \varepsilon - m$

The procedure FindPsdDelta is similar to the procedure FindDelta used in the linear variant. For its implementation, one can apply any solver to solve the corresponding SDP. A much more efficient implementation is given in the following lemma, which is proved in Chapter 6.

Lemma 5.16. *For any $\delta > 0$, the $(\varepsilon-\delta)$-good for the maximal $\varepsilon > 0$ can be found in time $\tilde{O}(\frac{n^{2.5}}{\delta^2})$. Further, the computation can be carried out using only approximate eigenvalue computations and elementary operation.*

We proceed to show that in case we're far from the optimum, there exists an improvement matrix.

Lemma 5.17. *If $f(P^*) - f(P) \geq \varepsilon$, then there exists a (εm)-good matrix with respect to P.*

Proof. We assume that the current solution $\mathbf{P}$ is a PSD matrix, and satisfies $\sum_{ij} \mathbf{P}_{ij} = 1$. The optimal PSD matrix, denoted $\mathbf{P}^*$, gives rise to a natural improvement matrix $\Delta := \mathbf{P}^* - \mathbf{P}$. Note that Δ satisfies the first two conditions of being (εm)-good, since $\mathbf{P}^*$ is the optimal solution. In addition:

$$\begin{aligned}
\sum_{i=1}^{m} \frac{A_i \bullet \Delta}{A_i \bullet \mathbf{P}} &= \sum_{i=1}^{m} \frac{A_i \bullet \mathbf{P}^* - A_i \bullet \mathbf{P}}{A_i \bullet \mathbf{P}} \\
&= \sum_{i=1}^{m} \frac{A_i \bullet \mathbf{P}^*}{A_i \bullet \mathbf{P}} - m \\
&\geq m \cdot (\sqrt[m]{\prod_{i=1}^{m} \frac{A_i \bullet \mathbf{P}^*}{A_i \bullet \mathbf{P}}} - 1) \quad \text{by the AMGM inequality} \\
&= m \cdot (\sqrt[m]{e^{m(f(\mathbf{P}^*) - f(\mathbf{P}))}} - 1) \geq m(\sqrt[m]{e^{m\varepsilon}} - 1) \\
&\geq m\varepsilon \quad \text{by Taylor series of } e^x
\end{aligned}$$

□

Lemma 5.18. *Let Δ be a (εm)-good PSD matrix with respect to $\mathbf{P}_t$. Define $\mathbf{P}_{t+1} := \mathbf{P}_t + \delta \cdot \Delta$ for $\delta = \frac{\tau^2 \varepsilon}{2}$. Then $\mathbf{P}_{t+1}$ is a feasible solution with value larger then the one obtained by $\mathbf{P}_t$ by at least:*

$$f(\mathbf{P}_{t+1}) - f(\mathbf{P}_t) \geq \frac{1}{2} m \tau^2 \varepsilon^2$$

Proof. First we prove that $\mathbf{P}_{t+1}$ is a feasible solution. Since Δ is a (εm)-good matrix, $W_t \triangleq \mathbf{P}_t + \Delta \succeq 0$. Since the PSD cone is convex we have $\mathbf{P}_{t+1} = \delta W_t + (1-\delta)\mathbf{P}_t \succeq 0$. Similarly, by definition of Δ we have $\mathbf{J} \bullet W_t = 1$ and $W_t(i,j) \geq 0$, thus

$$\mathbf{J} \bullet \mathbf{P}_{t+1} = \mathbf{J} \bullet (\delta W_t + (1-\delta)\mathbf{P}_t) = \delta \mathbf{J} \bullet W_t + (1-\delta)\mathbf{J} \bullet \mathbf{P}_t = 1$$

and

$$\mathbf{P}_{t+1}(i,j) = \delta W_t(i,j) + (1-\delta)\mathbf{P}_t(i,j) \geq 0$$

We proceed to show that the objective value increases. Denote

$$c_i := \delta \cdot \frac{A_i \bullet \Delta}{A_i \bullet \mathbf{P}_t}$$

Since the matrices $\mathbf{P}_t$ and W_t are PSD with trace at most one, $\forall_i |A_i \bullet (W_t - \mathbf{P}_t)| \leq 1$, and therefore $|c_i| = \delta |\frac{A_i \bullet (W_t - \mathbf{P}_t)}{A_i \bullet \mathbf{P}_t}| \leq \frac{\delta}{\tau} = \frac{1}{2}\tau\varepsilon \leq \frac{1}{2}$. Hence,

$$\begin{aligned}
f(\mathbf{P}_{t+1}) - f(\mathbf{P}_t) &= \sum_{i=1}^{m} \log\left(\frac{A_i \bullet (\mathbf{P}_t + \delta\Delta)}{A_i \bullet \mathbf{P}_t}\right) \\
&= \sum_{i=1}^{m} \log(1 + c_i) \\
&\geq \sum_{i=1}^{m} \left[c_i - (c_i)^2\right] \quad \text{holds when } |c_i| < \tfrac{1}{2} \\
&\geq \delta m \varepsilon - m\frac{\delta^2}{\tau^2} \geq \frac{1}{2} m \tau^2 \varepsilon^2 \quad \text{for } \delta = \tfrac{\tau^2 \varepsilon}{2}
\end{aligned}$$

□

Proof of Theorem 5.14. By Lemmas 5.17,5.16, at iteration t the procedure FindPsdDelta will return a $(m(f(\mathbf{P}^*) - f(\mathbf{P}_t)))$-good matrix. Thus by Lemma 5.18,

$$f(\mathbf{P}_{t+1}) - f(\mathbf{P}_t) \geq \frac{\tau^2 m(f(\mathbf{P}^*) - f(\mathbf{P}_t))^2}{2}$$

Therefore, as long as $f(\mathbf{P}^*) - f(\mathbf{P}_t) \geq \varepsilon$,

$$f(\mathbf{P}^*) - f(\mathbf{P}_{t+1}) \leq (1 - \frac{\tau^2 m\varepsilon}{2})(f(\mathbf{P}^*) - f(\mathbf{P}_t)).$$

Since we can start from a solution of value at least $m \cdot \frac{1}{m} \log \frac{1}{n^2} = \log \frac{1}{n^2}$ and the optimal solution is bounded by 1, we have that $f(\mathbf{P}^*) - f(\mathbf{P}_1) \leq 2 \log n$. Therefore, after $\tilde{O}(\frac{1}{\tau^2 m\varepsilon})$ iterations we find a solution such that $f(\mathbf{P}^*) - f(\mathbf{P}_t) \leq \varepsilon$.

The HaploFreq2 algorithm updates $\mathbf{P}_t \leftarrow (1-\frac{1}{20m})\mathbf{P}_t + \frac{1}{20mn^2}\mathbf{J}$ in case $\min_{i\in[m]} A_i^\top \mathbf{P}_t < \tau$. By Lemma 5.19 below this causes the objective value of $\mathbf{P}_t$ only to increase, and maintains the invariant that $\tau = \Omega(\frac{1}{mn^2})$. Therefore the total number of iterations is bounded by $\tilde{O}(\frac{1}{\tau^2 m\epsilon}) = \tilde{O}(\frac{mn^4}{\epsilon})$. □

Lemma 5.19. *For the optimum* $\mathbf{P}^*$ *of an instance of* Maximum Relaxed Unphased Likelihood*, it holds that* $\min_i A_i \bullet \mathbf{P}^* = \Omega(\frac{1}{mn^2})$.

Proof. Follows directly from Lemma 5.4 and the fact that if $\mathbf{P}^*$ is PSD then the matrix $\mathbf{P}' = (1 - \frac{c}{m})\mathbf{P}^* + \frac{c}{mn^2}\mathbf{J}$ is also PSD. □

5.5 Lower Bounds

In this section we show the following hardness result for the specific case of Frequency Estimation of Unphased Genotypes. In general, strong hardness results for optimizing over polynomials are known (see [BR92]). In some cases, as presented in this paper, the specific structure of a specific polynomial may be used to get a polynomial time algorithm. We show that for the special case of the polynomial introduced in Frequency Estimation of Unphased Genotypes, there is no such polynomial time algorithm if $P \neq NP$.

We denote the size of a certain an instance for the problem by N (that is, $N = m * n^2$ where m is the number of matrices and n their dimension). We now prove the following.

Theorem 5.20. *The* Frequency Estimation of Unphased Genotypes *Problem is NP-hard to approximate to within* $2^{N^{1-\varepsilon}}$ *for every constant* $\varepsilon > 0$.

To prove the theorem, we first show that Frequency Estimation of Unphased Genotypes is NP-hard, via a reduction from the Maximum Clique problem. The reduction is done using the following simple rule. An instance for the Maximum Clique problem is a graph $G = (V, E)$. We use the adjacency matrix A_G as an instance for Frequency Estimation of Unphased Genotypes. In particular, in this new instance, the set of matrices $\{A_1, ..., A_m\}$ consists of a single matrix, which is the adjacency matrix of G. We denote this new instance as I_G. Rewriting the objective function, we get that a probability p_i is assigned to every vertex in the graph, and we seek to maximize $\sum_{(i,j)\in E} p_i p_j$ where $\sum_{i\in V} p_i = 1$, and $p_i \geq 0$.

Claim 5.21. *For a given graph G, the objective of the* Frequency Estimation of Unphased Genotypes *instance I_G has value $\frac{1}{2}(1-\frac{1}{r})$ if and only if the maximal clique size of G is r.*

Proof. If the graph G contains a clique of size r, then by assigning $p_i = \frac{1}{r}$ for all vertices of this clique and 0 for all other vertices, we obtain an objective value of $\frac{1}{2}(1-\frac{1}{r})$ for I_G.

The other direction is a generalization of Turán's theorem (see [AS92]). Turán's theorem states that for a graph with n vertices, that does not contain a clique of size $r+1$, the maximum number of edges is $\frac{n^2}{2}(1-\frac{1}{r})$. Furthermore, the maximum is attained for the complement of the graph composed of r disjoint cliques of size $\frac{n}{r}$ each (these graphs are called Turán's graphs). When every vertex has a weight p_i, and the weight of an edge is $p_i p_j$, then Turán's theorem can be viewed as a special case of our claim, in which all p_i must be set to $\frac{1}{n}$, and thus all edges have the same weight. We now show that even when the vertices may have different weights, as long as the weights are non-negative, the objective function value for a graph without K^{r+1} is maximized for the Turán graph $T^r(n)$, in which it is precisely $\frac{1}{2}(1-\frac{1}{r})$.

We first consider the properties of an optimal solution to the instance I_G. Consider an optimal solution $p_1, \ldots, p_n$, and consider any two vertices v_i, v_j, such that $1 > p_i, p_j > 0$. One can verify, that by setting $p_i \mapsto p_i + \varepsilon$ and $p_j \mapsto p_j - \varepsilon$, the objective function will be changed in the following way:

$$\Delta(i,j,\varepsilon) \quad = \quad \pm\Theta(\varepsilon^2) + \varepsilon \sum_{t\in\Gamma_i \triangle \Gamma_j} x_t,$$

where Γ_i denotes set of neighbors of v_i, and $\Gamma_i \triangle \Gamma_j$ is the symmetric difference between the two sets. We thus get that for all p_i, p_j that are non-zero, the optimal solution satisfies $P_i = P_j$, where $P_i \triangleq \sum_{j\in\Gamma(i)} p_j$ is the sum of all variables corresponding to the vertices in $\Gamma(i)$. We denote $\forall i \,.\, P = P_i$.

Assume that for the optimal solution, $\sum_{(i,j)\in E} p_i p_j > \frac{1}{2}(1-\frac{1}{r})$. It suffices to show that the maximal clique size in G is $\omega(G) > r$. By the definition of P, we get

$$P = \sum_{i\in V} p_i P = \sum_{i\in V} p_i \sum_{j\in\Gamma_i} p_j = 2 \sum_{(i,j)\in E} p_i p_j > 1 - \frac{1}{r}.$$

Consider the subgraph of all vertices with $p_i > 0$, and consider the largest clique of this subgraph, say of size k. For simplicity of notations, assume that the variables of such a clique are $p_1, ..., p_k$. The set of neighbors of these vertices satisfy:

$$\sum_{i=1}^{k} P_i = \sum_{i=1}^{k} \sum_{j\in\Gamma(i)} p_j \leq (k-1) \sum_{j\in V} p_j \leq k-1,$$

where the first inequality holds since each vertex in the graph can be a neighbor of at most $k-1$ vertices of the clique. On the other hand, we have that $\sum_{i=1}^{k} P_i = k \cdot P > k(1-\frac{1}{r})$. Using both inequalities, we get that $k(1-\frac{1}{r}) < k-1$ and thus $k > r$ $\square$

In order to prove Theorem 5.20, we use the following theorem by Hastad (see [Has96]),

Theorem 5.22 (Hastad)**.** *It is NP-hard to decide whether a given graph on n vertices contains a clique of size $n^{99/100}$ or whether the maximal clique in the graph is of size at most $n^{1/100}$.*

Using this theorem we can now prove:

Theorem 5.20. According to Claim 5.21 and Hastad's theorem, FREQUENCY ESTIMATION OF UNPHASED GENOTYPES is NP-hard to approximate to within:

$$\frac{(1-N^{-99/100})}{(1-N^{-1/100})} = 1 + \frac{N^{-1/100} - N^{-99/100}}{1-N^{-1/100}} \geq 1 + \frac{1}{\sqrt{N}}$$

Now amplify this construction as follows: Given a graph G, create an instance of FREQUENCY ESTIMATION OF UNPHASED GENOTYPES as follows. In this instance, the set of matrices $\{A_1, ..., A_n\}$ consists of a M copies of the adjacency matrix of G. We denote this new instance as I'_G.

The size of the instance built is MN, where N is the size of the instance I_G described in Claim 5.21 (with a single copy of the adjacency matrix of G). Since I_G is hard to approximate within $1+\frac{1}{\sqrt{N}}$, it follows that I'_G is hard to approximate within

$$\left(1+\frac{1}{\sqrt{N}}\right)^M \geq e^{-M/\sqrt{N}}$$

Taking M to be a large polynomial in N, say $M = N^k$, we get an instance of size $T \triangleq N^{k+1}$ that is hard to approximate within $e^{-N^{k-1/2}} \leq e^{-T^{1-\frac{2}{k}}}$. Taking $k = \frac{1}{2\varepsilon}$ completes the proof. $\square$

Chapter 6

Fast Approximation Algorithms for Semidefinite Programming

This chapter describes a Lagrangian relaxation technique for approximately solving several families of semidefinite programs. The improvements in running time compared to previous approaches stem from a new method for reducing the width as well as improvements in approximate eigenvalue computations by using random sampling.

The results of this chapter are taken from a joint paper with Sanjeev Arora and Satyen Kale [AHK05b].

6.1 Introduction

Semidefinite programming (SDP) solves the following general problem:

$$\begin{aligned} \min c \bullet X & \\ A_i \bullet X &\geq b_i \qquad j = 1, 2, \ldots, m \\ X &\succeq 0 \end{aligned} \tag{6.1}$$

Here $X \in \mathbb{R}^{n \times n}$ is a matrix of variables and $A_1, A_2, \ldots, A_m \in \mathbb{R}^{n \times n}$. As appeared elsewhere in this thesis, for $n \times n$ matrices A and B, $A \bullet B$ is their inner product treating them as vectors in $\mathbb{R}^{n^2}$, and $A \succeq 0$ is notation for "A is positive semidefinite".

The first polynomial-time algorithm (strictly speaking, an approximation algorithm that computes the solution upto any desired accuracy ε) used the Ellipsoid method [GLS94] but faster interior-point methods were later given by Alizadeh [Ali95], and Nesterov and Nemirovskii [NN94]. The running time of Alizadeh's algorithm is $\tilde{O}(\sqrt{m}(m + n^3)L)$ where L is an input size parameter. In this section, the $\tilde{O}$ notation is used to suppress polylog$(\frac{mn}{\varepsilon})$ factors.

Much attention was focused on SDP as a result of the work of Goemans and Williamson [GW95], who used SDP to design new approximation algorithms for several NP-hard problems such as MaxCut, Max 2-Sat, and Max 3-Sat. In subsequent years, SDP-based approximation algorithms were designed for coloring k-colorable graphs, Max Dicut, etc.

Then progress halted for a few years, until recent work of Arora, Rao, Vazirani [ARV04] that gave a new $O(\sqrt{\log n})$-approximation for SPARSEST CUT. The ideas of this paper have been quickly extended to derive similar approximation algorithms for MIN 2CNF DELETION, MIN UNCUT, DIRECTED SPARSEST CUT, DIRECTED BALANCED SEPARATOR in [ACMM05] and NON-UNIFORM SPARSEST CUT in [CGR05, ALN05]. These new results rely on the so-called *triangle inequality* constraints, which impose a constraint for every *triple* of points. Thus the number of constraints $m = O(n^3)$, and the time to solve such SDPs is $\tilde{O}(n^{4.5})$.

In addition to these well-known approximation algorithms, SDP has also proved useful in a host of other settings. For instance, Linial, London, and Rabinovich [LLR95] observe that given an n-point metric space, finding its minimum-distortion embedding into ℓ_2 is a SDP with $m = n^2$ constraints, which takes n^4 time to solve. Recent approximation algorithms for *cut norm* of the matrix [AN04] and for certain subcases of correlation clustering [CW04] use a type of SDPs with $m = O(n)$, and hence require time $\tilde{O}(n^{3.5})$. (An intriguing aspect of this work is that the proof that the integrality gap of the SDP used in [AN04] is $O(1)$ uses the famous *Grothendieck inequality* from analysis.) Halperin and Hazan [HH06] showed that a biological *probability estimation problem*, which estimates the frequencies of haplotypes from a noisy sample, can be solved using SDP with $m = n^2$ (see chapter 5). Chazelle, Kingsford, and Singh [CKS04] use an SDP for *side chain positioning*, a problem in genomics.

Given the growing popularity of SDP, it would be extremely useful to develop alternative approaches that avoid the use of general-purpose interior point methods. Even problem-specific approaches would be very useful and seem hard to come by.

A similar situation developed in the past decade in case of linear programming, after LPs were used to design many approximation algorithms. Subsequent improvements to running times for these algorithms fall into two broad camps: (A) Eliminating use of LP in favor of a direct, combinatorial algorithm that uses the same intuition (in many cases, the same proof of the approximation ratio); (B) Solving the LP approximately instead of exactly. Typically this uses some version of the classical *Lagrangian relaxation* idea. Shahrokhi and Matula [SM90] gave the first approximation algorithm for flow problems. Plotkin, Shmoys, and Tardos [PST91] generalized the method to the family of *packing/covering* LPs. Later, Garg and Konemann [GK98] and Fleischer [Fle00] improved the running times further for flow LPs. A recent survey [AHK05a] by Arora, Hazan and Kale points out that all such algorithms are a subcase of a more general, widely useful, and older framework they called *multiplicative update rule* algorithms. From now on we refer to this as the *MW framework*. In chapter 4 we described an even more general framework for deriving approximation algorithms from online convex optimization algorithms. In this framework most previous lagrangian relaxation algorithms are special cases in which the online convex optimization algorithm used is the Multiplicative Weights algorithm (as given in section 2.3.3). However, for this chapter we concern only with the MW algorithm, and the additional generality is not required.

Speedups of type (A) and (B) for SDP-based algorithms are not easy. Speedups of type (A) have proved difficult because unlike LP-based approximation algorithms, the approximation ratio of SDP-based algorithms is proved by analysing a *rounding* algorithm rather than by comparing to the dual. The lone exception we are aware of is the notion of *expander flows* studied by Arora, Rao, and Vazirani (this was presented as an alternative to their more well-known rounding approach that was useful in most later papers). This duality-based framework allowed Arora,Hazan and Kale to design a $O(\sqrt{\log n})$-approximation algorithm for (uniform) Sparsest Cut that was combinatorial and ran in $\tilde{O}(n^2)$ time [AHK04], a significant improvement over the previously known running time of $\tilde{O}(n^{4.5})$. Interestingly, this algorithm was also derived in the MW framework. However, the duality-based framework from [ARV04] has it to be extended to problems other than uniform Sparsest Cut, though this may yet happen. Thus improvements of type (A) have not been forthcoming for the other problems.

Klein and Lu [KL96] initiated study of algorithms of type (B) for SDPs. They adapted the MW framework to approximately solve SDPs that arose in the algorithms of Goemans-Williamson and Karger, Motwani, and Sudan. The Klein-Lu approach reduces SDP solving to a sequence of approximate eigenvalue/eigenvector computations, which can be done efficiently using the well-known *power method*.

Our work. While the Klein-Lu work seemed promising, further progress then stalled. As we discuss in some detail later on, the main reason has to do with the *width* parameter, which is ρ such that the linear functions appearing in the constraints take values in the range $[-\rho, +\rho]$. Then the number of iterations in the MW framework is proportional to ρ^2. (Aside: In the PST packing-covering framework, the range of values was $[0, \rho]$, in which case the number of iterations is $O(\rho)$. This issue is discussed in [AHK05a].) Unfortunately, the width is large in most of the SDP relaxations mentioned above —the SDPs considered by Klein-Lu happened to be among the few where this problem is manageable.

Our first contribution is to modify the MW technique to handle some of these high-width SDPs. Our technique is a hybrid of the MW technique and an "exterior point" (i.e., Ellipsoid-like method) of Vaidya; this lowers the dependence on the width and is very efficient so long as the number of constraints with high width is "not too high." (Actually the Vaidya algorithm is overkill in most instances, where the number of high-width constraints is a small constant, and one can use simpler ideas, based on binary search, that are reminiscent of fixed-dimension LP algorithms.) Formally one needs a two-level implementation of the multiplicative update idea, that combines old constraints into new, smaller number of constraints. While this makes intuitive sense —MW methods excel at handling many low-width constraints and exterior point methods excel at handling a few, high-width constraints—this hybrid technique appears to be new. (It is related though to the observation in [PST91] that their packing-covering problems are solvable in polynomial time using the dual ellipsoid method.)

Our second contribution is to use a better technique for eigenvalue/eigenvector computations than the power method, namely, the Lanczos algorithm. This is the method of choice among numerical analysts, but has not been used in theory papers thus far because worst-case analysis for it is hard to find in the literature. We adapt an analysis for semidefinite matrices [KW92] to our needs (see Lemma 6.5). Then we suggest further speeding up the Lanczos algorithm for general matrices by first *sparsifying* the matrix with *random sampling*.

6.1.1 Overview of our results

Our algorithms assume a feasibility version of the SDP (6.1). Here, we implicitly perform a binary search on the optimum and the objective is converted to a constraint in the standard way. We also assume an additional constraint, a bound on the trace of the solution:

$$\begin{aligned} A_j \bullet X &\geq b_j \qquad j = 1, 2, \ldots, m \\ \sum_i X_{ii} &\leq R \\ X &\succeq 0 \end{aligned} \tag{6.2}$$

The upper bound on the trace, $\mathbf{Tr}(X) = \sum_i X_{ii}$, is usually absent in the textbooks, but is natural for relaxation SDPs. For instance, in combinatorial optimization, usually we have some unit vectors $v_1, v_2, \ldots, v_n$ associated with, say, the nodes in a graph, and $X_{ij} = v_i \cdot v_j$. Then $\mathbf{Tr}(X) = n$. In any case, $\mathbf{Tr}(X)$ for the optimum X can usually be "guessed" by binary search.

We wish to solve the SDP approximately up to a given tolerance ε, by which we mean that either we find a solution X which satisfies all the constraints up to an additive error of ε, i.e. $A_j \bullet X - b_j \geq -\varepsilon$ for $j = 1, 2, \ldots, m$, or conclude correctly that the SDP is infeasible.

As in the case of LP solving (PST, etc.), the Multiplicative Weights Update idea for solving SDPs is to associate at time t a non-negative weight $w_j^{(t)}$ with constraint j, where $\sum_j w_j^{(t)} = 1$. A high current weight for a constraint indicates that it was not "satisfied" too well in the past, and is therefore should receive higher priority in the next step. The optimization problem for the next step is to

$$\begin{aligned} \max \sum_j w_j^{(t)} (A_j \bullet X - b_j) \\ X &\succeq 0 \\ \sum_i X_{ii} &\leq R \end{aligned}$$

This is actually an eigenvalue problem in disguise, since the optimum is attained (wlog) at an X that has rank 1. Thus the Lagrangian relaxation idea would be to solve this eigenvalue problem, and update the weights w_i according to the usual multiplicative update rule

$$w_i^{(t+1)} \leftarrow w_i^{(t)} (1 - \alpha(A_j X_t - b_j))$$

where X_t was the solution to the eigenvalue problem (expressed as a rank 1 psd matrix) at time t. Then if α is small enough, this $\sum_t X_t$ is guaranteed to converge to a near-feasible solution to the original SDP (assuming a feasible solution exists). [1]

[1] In the terminology of chapter 4, this approach is the DualGameOpt meta-algorithm, where OnlineAlg is implemented using the Multiplicative Weights algorithm of section 2.3.3. The Optimization Oracle reduces to an eigenvalue computation.

Problem	Previous best	This paper	Improvement
MaxQP	$\tilde{O}(n^{3.5})$	$\tilde{O}\left(\frac{n^{1.5}}{\varepsilon^{2.5}} \cdot \min\left\{N,\ \frac{n^{1.5}}{\varepsilon\alpha^*}\right\}\right)$ $\alpha^* \in [\frac{1}{n}, 1]$	For $N = o(n^2)$ or $\alpha^* = \omega(\frac{1}{\sqrt{n}})$
HaploFreq	$\tilde{O}(n^4)$	$\tilde{O}\left(\frac{n^{2.5}}{\varepsilon^{2.5}}\right)$	$\Omega(n^{1.5})$
SCP	$\tilde{O}(n^4)$	$\tilde{O}\left(\frac{n^{1.5}N}{\varepsilon^{4.5}}\right)$	$\Omega(\frac{n^{3.5}}{N})$
Embedding	$\tilde{O}(n^4)$	$\tilde{O}\left(\frac{n^3}{d_{\min}^{2.5}\varepsilon^{3.5}}\right)$	For $d_{\min} = \omega(n^{-2/5})$
Sparsest Cut	$\tilde{O}(n^{4.5})$	$\tilde{O}(\frac{n^3}{\varepsilon^2})$	For $\varepsilon = \omega(n^{-0.75})$
Min Uncut Balanced Separator Min 2CNF deletion	$\tilde{O}(n^{4.5})$	$\tilde{O}\left(\frac{n^{3.5}}{\varepsilon^2}\right)$ $\varepsilon = O(\frac{\text{OPT}}{\lvert E\rvert})$	For $\varepsilon = \omega(\frac{1}{\sqrt{n}})$

Figure 6.1: Running time obtained for several applications.

Our new contributions to this approach, including the issue of managing the high-width constraints, and of fast eigenvalue computations, were already discussed earlier. Now we describe our main new results. The main point to stress is that in practice our algorithm may run even faster than the worst-case estimates summarized in table 6.1. Throughout the paper we carefully list times in terms of number of eigenvalue/eigevector computations required, and these tend run much faster than our worst-case estimate (which are formally given in Lemma 6.5 below). By contrast, each iteration of Alizadeh's SDP solver requires Cholesky decomposition, which is inherently a $\tilde{\Theta}(n^3)$ operation.

The worst-case running time is a function of the approximation guarantee ε. In some cases, there are also dependencies upon other problem parameters. For instance, in embedding, where one is seeking the minimum distortion embedding into ℓ_2, the ε is benign, say 0.1. However, there is a dependence on the aspect ratio; in other words, minimum squared internode distance d_{min}, where sum of squares of all $\binom{n}{2}$ internode distances is normalized to n^2. Our algorithm provides a speeedup when d_{min} is at least $n^{-2/5}$. (This is still an interesting set of metrics.) Likewise, our algorithm for MaxQP provides speedups when either of the following two conditions are true (a) the number of nonzero entries in the matrix A is $N = o(n^2)$ or (b) the optimum α is at least $\frac{1}{\sqrt{n}} \sum_{i,j} |A_{ij}|$. Again, this is an interesting class of matrices.

For problems such as (uniform) SPARSEST CUT, BALANCED SEPARATOR, and MIN-2-CNF-DELETION, the current approximation algorithms require a very small ε, namely, $OPT/|E|$. We improve upon existing SDP solvers when this is at least $1/\sqrt{n}$.

6.2 An illustration of the method

In this section, we give more details of the method by illustrating its application to the SDP (MaxQP) given below. This SDP arises in many algorithms such as approximating

MaxCut, maximizing the correlation in correlation clustering, approximating the Cut-Norm of a matrix, approximating the Grothendieck constant of a graph, etc. See [CW04] for a discussion.

$$\begin{aligned} \max\ A \bullet X & \\ X_{ii} &\le 1 \qquad i = 1, 2, \ldots, n \\ X &\succeq 0 \end{aligned} \tag{MaxQP}$$

We assume here that $\operatorname{diag}(A) \ge 0$, i.e. all diagonal entries of A are nonnegative. Let $N \ge n$ be the number of non-zero entries of A. We wish to get a multiplicative $1 - O(\varepsilon)$ approximation to the optimum value of the SDP. Note that Alizadeh's interior point method solves the SDP in $\tilde{O}(n^{3.5})$ time.

Step I: Bounding the optimum and trace. We compute bounds on the optimum of the SDP, denoted by α^*. For simplicity, assume $\sum_{ij} |A_{ij}| = 1$, this amounts to scaling the optimum by a fixed quantity. Let X^* be the optimum solution. Since X^* is positive semidefinite, for any i, j, $(X^*_{ij})^2 \le X^*_{ii} X^*_{jj} \le 1$, so $|X^*_{ij}| \le 1$. Thus, $\alpha^* = A \bullet X^* = \sum_{ij} A_{ij} X^*_{ij} \le \sum_{ij} |A_{ij}| = 1$. Conversely, the solution X specified by $X_{ij} = \frac{\operatorname{sgn}(A_{ij})}{n}$ and $X_{ii} = 1$ is positive semidefinite, and achieves an objective value of $\frac{1}{n}$. This gives a lower bound on α. We also compute a bound on $\mathbf{Tr}(X)$. In this case, this is simply $\sum_i X_{ii} \le n$.

Step II: Reduction to feasibility problem. We "guess" the value of α using binary search in the range computed in Step I. Let α be our current guess. Define a convex set $\mathcal{P} = \{X \succeq 0,\ \sum_i X_{ii} \le n\}$. We rewrite the SDP as a feasibility problem for the binary search as follows:

$$\begin{aligned} \frac{1}{\alpha} A \bullet X - 1 &\ge 0 \\ 1 - X_{ii} &\ge 0 \qquad i = 1, 2, \ldots, n \\ X &\in \mathcal{P} \end{aligned} \tag{6.3}$$

We now need to estimate the *width* of each constraint. This is defined as the maximum absolute value it can take for $X \in \mathcal{P}$. More specifically, for a constraint of the kind $A \bullet X - b \ge 0$ where $X \in \mathcal{P}$, assume that the range of values that $A \bullet X - b$ can take is $[-\ell, \rho]$ or $[-\rho, \ell]$ where $1 \le \ell \le \rho$. Then ρ is called the width of the constraint. In (6.3), the range of the constraint $\frac{1}{\alpha} A \bullet X - 1 \ge 0$ for $X \in \mathcal{P}$ is $[-\frac{n}{\alpha}, \frac{n}{\alpha}]$, so the width is $\frac{n}{\alpha}$. The range of the constraint $1 - X_{ii} \ge 0$ for $X \in \mathcal{P}$ is $[-n, 1]$, so the width is n.

Note that an additive error of ε translates to a multiplicative error of $1 - O(\varepsilon)$ to the objective, assuming the binary search guessed the value of the optimum to within a factor of $1 + \varepsilon$.

Step III: The Multiplicative Weights Update algorithm. Lagrangian relaxation algorithms assume that there is an algorithm, Oracle, to solve the following relaxed feasibility problem: given non-negative weights $w_0, w_1, w_2, \ldots, w_n$ on the constraints such that $\sum_{i=0}^{n} w_i = 1$, consider the weighted combination of constraints $w_0(\frac{1}{\alpha} A \bullet X - 1) + \sum_{i=1}^{n} w_i(1 - X_{ii})$. Then ORACLE either finds an $X \in \mathcal{P}$ which makes this combination $\geq -\frac{\varepsilon}{2}$ or declares correctly that no $X \in \mathcal{P}$ makes the combination non-negative.

In the latter case, we declare infeasibility of (6.3) since otherwise any feasible solution would make the weighted combination of constraints non-negative, a contradiction. If the former case holds whenever the ORACLE is presented a set of weights, then we can get an ε approximate solution to (6.3), as given in the following theorem:

Theorem 6.1. *Consider the general SDP (6.2). Let $\mathcal{P} = \{X \succeq 0, \sum_i X_{ii} \leq R\}$. Assume that for any j, $A_j \bullet X - b_j$ lies in one of the ranges $[-\ell, \rho]$, $[-\rho, \ell]$. Also, assume that there is an algorithm,* ORACLE, *which runs in time T_{oracle}, and given any set of non-negative weights $w_1, w_2, \ldots, w_m$ on the constraints summing to 1, either finds an $X \in \mathcal{P}$ which makes the weighted combination $\sum_{j=1}^{m} w_j(A_j \bullet X - b_j) \geq -\frac{\varepsilon}{2}$ or declares correctly that no $X \in \mathcal{P}$ makes this combination non-negative. Then there is an algorithm which runs in $O(\frac{\ell\rho}{\varepsilon^2}(T_{oracle} + m))$ time and either gets an ε approximate solution to (6.2) or concludes that it is infeasible.*

Step IV: ORACLE **from eigenvector computations.** Note that the ORACLE of Theorem 6.1 needs to maximize the weighted combination of constraints $\sum_{j=1}^{m} w_j(A_j \bullet X - b_j)$ over the set $\mathcal{P}$ of all positive semidefinite matrices X whose trace is bounded by R. We show in section 6.4 that this amounts to approximately computing the largest eigenvector of the matrix $C = \sum_{j=1}^{m} w_j(A_j - \frac{b_j}{R} I)$ up to tolerance $\delta = \frac{\varepsilon}{2R}$. Define $T_{\text{ev}}(C, \delta)$ to be the time needed for this. In Lemma 6.4, we show that $T_{\text{oracle}} = \tilde{O}(T_{\text{ev}}(C, \delta))$.

Getting back to our example, SDP (MAXQP), we have the following parameters: $\ell = \rho = \frac{n}{\alpha}$, $m = n+1$, $R = n$. The running time from Theorem 6.1 is $\tilde{O}(\frac{n^2}{\varepsilon^2 \alpha^2}(T_{\text{oracle}} + n))$ which is worse than Alizadeh's algorithm for $\alpha = o(n^{-0.25})$ even without factoring in T_{oracle}. We show how to improve the running time in the next step.

Step V: Inner and Outer SDPs. Now we indicate our width reduction technique. Observe that there is a single constraint, $\frac{1}{\alpha} A \bullet X - 1 \geq 0$, which has high width $(\frac{n}{\alpha})$. The other constraints have width bounded by n. This happens in all our applications: we will find a constant sized set of constraints of high width and the rest will have low width. We devise a hybrid algorithm, using the multiplicative update method to handle the low width constraints and an exterior point algorithm to handle the (few) high width constraints.

The idea is to push the high width constraint, $\frac{1}{\alpha} A \bullet X - 1 \geq 0$, into the convex set, to create a new convex set $\mathcal{Q} = \{X \in \mathcal{P}, \frac{1}{\alpha} A \bullet X - 1 \geq 0\}$, and run the Multiplicative Weights Update algorithm of Theorem 6.1 on the other constraints over $\mathcal{Q}$. We call this the *outer SDP*.

The ORACLE now gets a weighted combination of the constraints, $\sum_{i=1}^{n} w_i(1 - X_{ii})$, and needs to find an $X \in \mathcal{Q}$ which makes this $\geq -\frac{\varepsilon}{2}$ or declare that no such X makes the combination non-negative. This can be achieved by approximately solving the 2 constraint

SDP of the form

$$\begin{aligned}
\sum_{i=1}^{n} w_i(1 - X_{ii}) &\geq 0 \\
\frac{1}{\alpha} A \bullet X - 1 &\geq 0 \\
X &\in \mathcal{P}
\end{aligned}$$

We call this the *inner SDP*. The ORACLE for this SDP needs to optimize a weighted combination of *all* constraints over $\mathcal{P}$, this is identical to the one we had in Step IV.

We solve the inner SDP using an exterior point algorithm. The observation is that the ORACLE yields a separation hyperplane for the dual problem, and so we can apply Vaidya's algorithm. Recall that m is the number of constraints. Let $\mathcal{M}(m) = O(m^{2.36})$ be the time needed to multiply two $m \times m$ matrices.

Theorem 6.2. *With the setup as in Theorem 6.1, there is an algorithm which produces an ε approximate solution to the general SDP (6.2) or declares correctly its infeasibility in time*

$$\tilde{O}(m \log(\rho) \cdot T_{oracle} + m \log(\rho) \mathcal{M}(m \log(\rho)))$$

Note that this algorithm has poor dependence on the number of constraints but handles high width very well. In our example, the number of constraints in the inner SDP is just 2, and the width is poly(mn) from the trace bound. Thus, the algorithm of Theorem 6.2 solves it in $\tilde{O}(T_{\text{oracle}}) = \tilde{O}(T_{\text{ev}}(C, \frac{\varepsilon}{2n}))$ time.

In all our applications, we have a constant sized set of constraints with much higher (yet, polynomial) width than the rest. Let the other, low width, constraints take values in the range $[-\ell_L, \rho_L]$ or $[-\rho_L, \ell_L]$.

Corollary 2. With the given setup, the hybrid algorithm which composes an outer and inner SDP runs in time

$$\tilde{O}\left(\frac{\ell_L \rho_L}{\varepsilon^2}\left[T_{\text{ev}}\left(C, \frac{\varepsilon}{2R}\right) + m\right]\right)$$

For SDP (MAXQP), this yields the following theorem, which will be proved in section 6.5.1:

Theorem 6.3. *A multiplicative* $1-O(\varepsilon)$ *approximation to SDP* (MAXQP) *can be obtained in time*

$$\tilde{O}\left(\frac{n^{1.5}}{\varepsilon^{2.5}} \cdot \min\left\{N, \frac{n^{1.5}}{\varepsilon \alpha^*}\right\}\right)$$

This running time is always better than the $\tilde{O}(n^{3.5})$ running time of Alizadeh's interior point algorithm. It is asymptotically faster if the matrix A is not dense, i.e. $N = o(n^2)$, or if $\alpha^* = \omega(\frac{1}{\sqrt{n}})$.

We note here the special case of the MaxCut SDP. For this problem, the matrix A is the combinatorial Laplacian of the input graph, divided by $4m$. We note that $\alpha^* \geq \frac{1}{8}$ is easily obtained from the greedy algorithm. Thus, our algorithm runs in time $\tilde{O}(n^{1.5} \cdot \min\{N, n^{1.5}\})$.

The best algorithm for solving the MaxCut SDP is due to Klein and Lu [KL96], with running time $\tilde{O}(nN)$. Our algorithm is a $\sqrt{n}$ factor worse when $N = o(n^2)$. However, our algorithm solves the much more general problem (MaxQP) and the approach of [KL96] does not extend to this general problem.

6.3 Proofs of the Main Theorems

In this section we prove Theorem 6.2 and describe the proof of Theorem 6.1. For convenience of notation, we define the linear functions $f_j(X) = A_j \bullet X - b_j$ for $1 \leq j \leq m$. The problem is to check the feasibility of the system of inequalities $f_j(X) \geq 0$, for $1 \leq j \leq m$, with $X \in \mathcal{P}$. We assume that there is an Oracle which does the following task: given non-negative weights $p_1, p_2, \ldots, p_m$ summing to 1, it either finds an $X \in \mathcal{P}$ which makes the weighted combination $\sum_j p_j f_j(X) \geq -\frac{\varepsilon}{2}$ or declares correctly that no $X \in \mathcal{P}$ makes the combination non-negative.

Theorem 4.1 in chapter 4 applies to the setting of Theorem 6.1, and gives somewhat worse running time: $O(\frac{(\ell+\rho)^2}{\varepsilon^2}(T_{\text{oracle}} + m))$ instead of $O(\frac{\ell\rho}{\varepsilon^2}(T_{\text{oracle}} + m))$.

The improvement in running time is obtained by a more careful analysis of the regret if the MW algorithm for linear payoff functions. This analysis is given in the survey [AHK05a], and not detailed hereby due to its similarity to the analysis of section 2.3.3.

Proof of Theorem 6.2. To devise the algorithm, we will need the following version of Farkas' lemma:

Lemma 6.1 (Farkas)**.** *Given an $n \times m$ matrix A and a $1 \times m$ row vector c, both with real coefficients, one and only one of the following systems has a solution:*

1. *$Ap \leq 0$, $p \geq 0$ and $cp > 0$ for some m-column vector p;*
2. *$qA \geq c$ and $q \geq 0$ for some n-row vector q.*

Lemma 6.2. *Consider $X_1, ..., X_n \in \mathcal{P}$. Then exactly one of the following holds:*

1. *There exists a distribution $p_1, p_2, \ldots, p_m \geq 0$, $\sum_j p_j = 1$ such that such that: $\forall i \in [n]$, we have $\sum_j p_j f_j(X_i) \leq -\varepsilon$.*
2. *There exist a distribution $q_1, q_2, ..., q_n \geq 0$, $\sum_i q_i = 1$, such that for $y = \sum_i q_i X_i$ we have that $\forall j \in [m]$, y satisfies $f_j(y) \geq -\varepsilon$.*

Proof. In Farkas' Lemma, choose the matrix A to be $A_{ij} = f_j(X_i) + \varepsilon$, and the vector c as $c_j = 1$ for all j, $1 \leq j \leq m$. Since $c > 0$, we conclude that the vectors p and q in Farkas' Lemma are non-zero and non-negative. By scaling we may assume that p is a distributions as required in this lemma. Then the first case exactly corresponds to the first case of Farkas' Lemma. The second case of Farkas' Lemma translates to: there is a vector $\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_n \geq 0$ such that $\forall j \in [m]$, we have $\sum_i \tilde{q}_i(f_j(X_i) + \varepsilon) \geq 1$. Set $q_i = \tilde{q}_i / \sum_k \tilde{q}_k$ so that $q_1, q_2, \ldots, q_n$ is a distribution. Then $\forall j \in [m]$, we have $\sum_i q_i f_j(X_i) \geq \frac{1}{\sum_i \tilde{q}_i} - \varepsilon \geq -\varepsilon$ which implies that $f_j(\sum_i q_i X_i) \geq -\varepsilon$, since f_j is a linear function. Setting $y = \sum_i q_i X_i$ concludes the proof. □

Lemma 6.3. *Consider $X_1, ..., X_n \in \mathcal{P}$. Suppose the first case of Lemma 6.2 holds, i.e. there is a distribution $p_1^*, p_2^*, \ldots, p_m^*$ such that $\forall i \in [n]$, we have $\sum_j p_j^* f_j(X_i) \leq -\varepsilon$. Then the polytope $\mathcal{Q}$ of vectors $\langle p_1, p_2, \ldots, p_m \rangle \in \mathbb{R}^m$ defined by the linear inequalities*

$$\begin{aligned} \forall i \in [n] : \quad & \sum_j f_j(X_i) p_j \;\leq\; -\frac{\varepsilon}{2} \\ & \sum_j p_j \;\leq\; 1 + \frac{\varepsilon}{2\rho} \\ \forall j \in [m] : \quad & p_j \;\geq\; -\frac{\varepsilon}{2m\rho} \end{aligned} \tag{6.4}$$

has volume at least $(\frac{\varepsilon}{m\rho})^m$. Further, it is contained in an ℓ_∞ box of volume 2^m.

Proof. We show that the ℓ_∞ box around $p^* = \langle p_1^*, p_2^*, \ldots, p_m^* \rangle$ defined by $\mathcal{B} = \{p \in \mathbb{R}^m | \; ||p - p^*||_\infty \leq \frac{\varepsilon}{2m\rho}\}$ is contained in $\mathcal{Q}$. This box has volume $(\frac{\varepsilon}{m\rho})^m$. This is true because for any $p \in \mathcal{B}$, we have (considering the constraints in reverse order) $\forall j \in [m]$, $p_j \geq p_j^* - \frac{\varepsilon}{2m\rho} \geq -\frac{\varepsilon}{2m\rho}$, $\sum_j p_j \leq \sum_j p_j^* + m \cdot \frac{\varepsilon}{2m\rho} = 1 + \frac{\varepsilon}{2\rho}$, and finally, for any $i \in [n]$, we have

$$\sum_j f_j(X_i) p_j \;\leq\; \sum_j \left[f_j(X_i) p_j^* + |f_j(X_i)| \cdot \frac{\varepsilon}{2m\rho} \right] \;\leq\; -\varepsilon + m\rho \cdot \frac{\varepsilon}{2m\rho} \;=\; -\frac{\varepsilon}{2}$$

using the fact that $|f_j(X_i)| \leq \rho$.

Next, note that all p_j satisfy $-\frac{\varepsilon}{2m\rho} \leq p_j \leq (1 + \frac{\varepsilon}{\rho})$. So $\mathcal{Q}$ is contained in the box $\{p \in \mathbb{R}^m | \; \forall j \in [m] : -\frac{\varepsilon}{2m\rho} \leq p_j \leq (1 + \frac{\varepsilon}{2\rho})\}$, which has volume $((1 + \frac{\varepsilon}{\rho}) + \frac{\varepsilon}{2m\rho})^m \leq 2^m$. □

At this point, we are ready to present the algorithm. We run Vaidya's algorithm [Vai96] for deciding emptiness of a convex polytope equipped with a separation oracle. The algorithm is analogous to the Ellipsoid Algorithm but is more efficient in terms of iterations needed.

The polytope in question is defined adaptively as follows: in each iteration, a point $X \in \mathcal{P}$ may be generated by the separation oracle. Let $X_1, X_2, \ldots, X_n$ be all the points generated by the separation oracle over all iterations. Then the polytope is $\mathcal{Q}$, defined by the linear inequalities (6.4) in Lemma 6.3. The separation oracle works as follows: first it checks the last 2 constraints trivially, and returns one of them if it is violated. Otherwise, it calls Oracle on the test point p. If Oracle declares that there for all $X \in \mathcal{P}$, $\sum_j p_j f_j(X) < 0$, then the algorithm immediately aborts and declares infeasibility of the system. Otherwise, Oracle returns a point $X \in \mathcal{P}$ such that $\sum_j p_j f_j(X) \geq -\frac{\varepsilon}{2}$.

Then the constraint $\sum_j p_j f_j(X) \leq -\frac{\varepsilon}{2}$ serves as a separating hyperplane for Vaidya's algorithm and X becomes one of the X_i's mentioned above.

Vaidya's algorithm needs $n = O\left(\log\left(\frac{2^m}{(\frac{\varepsilon}{m\rho})^m}\right)\right) = \tilde{O}(m\log(\rho))$ iterations to decide emptiness of the polytope $\mathcal{Q}$. In this case, the first case of Lemma 6.2 doesn't hold, so the second must. Since the existence of the distribution $q_1, q_2, \ldots, q_n$ has been established, it can be found by solving the linear program:

$$\begin{aligned} \forall j \in [m]: \quad \sum_i f_j(X_i) q_i &\geq -\varepsilon \\ \sum_i q_i &= 1 \\ \forall i \in [n]: \quad q_i &\geq 0 \end{aligned} \tag{6.5}$$

Note that $n = \tilde{O}(m\log(\rho))$ so the linear program has $\tilde{O}(m\log(\rho))$ variables and $\tilde{O}(m\log(\rho))$ equations, and can be solved in $\tilde{O}((m\log(\rho))^3)$ time using Ye's algorithm [Ye91]. The time needed for Vaidya's algorithm is $\tilde{O}(m\log(\rho)\cdot T_{\text{oracle}} + m\log(\rho)\mathcal{M}(m\log(\rho))))$, so overall the time complexity is also $\tilde{O}(m\log(\rho)\cdot T_{\text{oracle}} + m\log(\rho)\mathcal{M}(m\log(\rho))))$.

□

6.4 Implementing Oracle using the approximate eigenvector computations

In this section, we present lemmas which describe how to efficiently implement the Oracle of Theorem 6.1 using approximate eigenvector computations.

Lemma 6.4. *Suppose we have a procedure, that given a matrix $C \in \mathbb{R}^{n\times n}$ and a tolerance $\delta > 0$, computes a unit vector x which satisfies $x^T C x \geq -\delta$, in time $T_{ev}(C, \delta)$, or declares correctly that C is negative definite. Then using this procedure once with $C = \sum_{j=1}^m w_i(A_j - \frac{b_j}{R}I)$ and $\delta = \frac{\varepsilon}{2R}$ we can implement* Oracle.

Proof. Recall that the Oracle from Theorem 6.1 is given non-negative weights $w_1, w_2, \ldots, w_m$ summing to 1 and needs to find an $X \in \mathcal{P}$ which makes the weighted combination $\sum_{j=1}^m w_j(A_j \bullet X - b_j) \geq -\frac{\varepsilon}{2}$ or prove that no $X \in \mathcal{P}$ makes this combination non-negative.

Suppose $\sum_{j=1}^m -w_j \cdot b_j \geq 0$. In this case, the Oracle can simply return $X = 0$. So assume that $\sum_{j=1}^m -w_j \cdot b_j < 0$, or equivalently, $\sum_{j=1}^m w_j \cdot b_j > 0$.

Then the Oracle constructs the matrix $C = \sum_{j=1}^m w_j(A_j - \frac{b_j}{R}I)$ and applies the eigenvector procedure on C with $\delta = \frac{\varepsilon}{2R}$. Suppose the procedure yields a unit vector x such that $x^T C x \geq -\delta$, then the Oracle returns the matrix $\tilde{X} = Rxx^T$. Note that $\mathbf{Tr}(\tilde{X}) = R$. For this matrix, we have

$$\begin{aligned} \sum_{j=1}^m w_j(A_j \bullet \tilde{X} - b_j) &= \sum_{j=1}^m w_j(A_j \bullet \tilde{X} - \frac{b_j}{R} I \bullet \tilde{X}) \\ = C \bullet \tilde{X} \;=\; C \bullet Rxx^T \;&=\; Rx^T C x \;\geq\; R \cdot -\frac{\varepsilon}{2R} \;=\; -\frac{\varepsilon}{2} \end{aligned}$$

as required.

Otherwise, the procedure declares correctly that the largest eigenvalue of C is negative, i.e. C is negative definite. In this case, the ORACLE declares no $X \in \mathcal{P}$ makes the combination non-negative. We show now that this is correct. Suppose that in fact there were an $X \in \mathcal{P}$ which satisfies $\sum_{j=1}^{m} w_j(A_j \bullet X - b_j) \geq 0$. Because $X \in \mathcal{P}$, $\mathbf{Tr}(X) \leq R$ and $X \succeq 0$. Since $C \prec 0$, we have $C \bullet X < 0$ (note that $X \neq 0$: we checked this solution). Then we have

$$\begin{aligned} 0 \ > \ C \bullet X \ &= \ \sum_{j=1}^{m} w_j \left(A_j \bullet X - \frac{b_j}{R} I \bullet X \right) \ = \ \sum_{j=1}^{m} w_j \left(A_j \bullet X - \frac{b_j}{R} \mathbf{Tr}(X) \right) \\ &= \ \sum_{j=1}^{m} w_j (A_j \bullet X - b_j) + \sum_{j=1}^{m} w_j b_j \left(1 - \frac{\mathbf{Tr}(X)}{R} \right) \ \geq \ 0 \end{aligned}$$

by the conditions on X given above. Thus, we have a contradiction. □

By Lemma 6.4, the ORACLE needed for our algorithms can be implemented by computing the eigenvector of the largest eigenvalue of the matrix which represents the weighted combination of the constraints. The Lanczos algorithm with a random starting vector is the most efficient algorithm for finding extreme eigenvectors. The running time for the Lanczos algorithm used in our context is the following:

Lemma 6.5. *Let $C \in \mathbb{R}^{n \times n}$ be a matrix with $N \geq n$ non-zero entries and eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Let $\delta > 0$ be a given error parameter. Let $\gamma = \max\{\frac{|\lambda_1|}{|\lambda_1| + |\lambda_n|}, \frac{\delta}{|\lambda_n|}\}$. Then the Lanczos algorithm with a random start applied to the matrix $C + \Lambda I$ yields with high probability a unit vector x which satisfies $x^T C x \geq -\delta$ or declares correctly that C is negative definite in time $T_{ev}(C, \delta) = \tilde{O}(\frac{N}{\sqrt{\gamma}})$.*

The parameter γ in Lemma 6.5 is not known a priori, but in applications we will derive lower bounds for it.

Proof. We need Theorem 3.2(a) of Kuczyński and Woźniakowski [KW92]:

Theorem 6.4 (KW)**.** *Let $M \in \mathbb{R}^{n \times n}$ be a positive semidefinite matrix with N non-zero entries. Then with high probability, the Lanczos algorithm produces in $O(\frac{\log(n)}{\sqrt{\gamma}})$ iterations a unit vector x such that $\frac{x^T M x}{\lambda(M)} \ \geq \ 1 - \gamma$.*

Note that each iteration of the Lanczos algorithm takes $\tilde{O}(N)$ time. Now let $M = C + |\lambda_n| I$. Notice that M is positive semidefinite, and $\lambda(M) = \lambda_1 + |\lambda_n|$. We apply Theorem 6.4 with $\gamma = \max\{\frac{|\lambda_1|}{|\lambda_1| + |\lambda_n|}, \frac{\delta}{|\lambda_n|}\}$ to obtain in time $\tilde{O}(\frac{N}{\sqrt{\gamma}})$ a unit vector x such that:

$$\frac{x^T M x}{\lambda(M)} \ = \ \frac{x^T (C + |\lambda_n| I) x}{\lambda_1 + |\lambda_n|} \ = \ \frac{x^T C x + |\lambda_n|}{\lambda_1 + |\lambda_n|} \ \geq \ 1 - \gamma$$

Simplifying, we get $x^T C x \geq (1 - \gamma)\lambda_1 - \gamma |\lambda_n|$. If $\lambda_1 \geq 0$, then for either choice of γ we get $x^T C x \geq -\delta$, so we return x. Otherwise, if $x^T C x < -\delta$, we conclude that $\lambda_1 < 0$, so we declare C to be negative definite. □

The previous lemma shows that the running time of the ORACLE depends on the sparsity of C, i.e. the number on non-zero entries in it. In section 6.6 we provide a randomized sparsification procedure with the following guarantee:

Lemma 6.6. *Let $C \in \mathbb{R}^{n\times n}$ be a symmetric matrix with N non-zero entries and let $S = \sum_{ij} |C_{ij}|$. Let $\delta > 0$ be a given error parameter. Then there is a randomized procedure which runs in $\tilde{O}(N)$ time and with high probability produces a symmetric matrix C' such that C' has $O(\frac{\sqrt{n}S}{\delta})$ non-zero entries and $||C - C'||_2 = O(\delta)$.*

Thus, C' can be used in place of C in the Lanczos algorithm, *if* it turns out to be sparser: the decision for specific applications depends on the smaller of the quantities N and $\frac{\sqrt{n}S}{\delta}$.

6.5 Applications

In this section, we describe several applications of the method outlined above. It should be noted that the method does not automatically yield faster algorithms; additional fine-tuning (mostly in terms of bounding large negative eigenvalues) is necessary for specific applications.

6.5.1 SDP relaxations of Quadratic Programs

Our first application is the SDP (MAXQP) that we used to illustrate the method, and we complete the proof of Theorem 6.3.

Proof. **[Theorem 6.3]**
We apply Corollary 2. The range of the constraints of the outer SDP, viz. $1 - X_{ii} \geq 0$ for $1 \leq i \leq n$, is $[1, -n]$ for $X \in \mathcal{Q}$. Thus $\ell_L = 1, \rho_L = n$. Now we bound the running time of the eigenvector computation procedure for the ORACLE.

Given non-negative weights $w_0, w_1, \ldots, w_n$ which sum to 1, the matrix C from Lemma 6.4 in this case is $w_0(\frac{1}{\alpha}A - \frac{1}{n}I) + \sum_{i=1}^n w_i(\frac{1}{n}I - e_ie_i^T)$, where e_i is the i^{th} standard basis vector, and $\delta = \frac{\varepsilon}{2n}$.

To apply Lemma 6.5, we need to bound the most negative eigenvalue, λ_n, of C. Observe that $\mathbf{Tr}(C) = w_0(\frac{1}{\alpha}\mathbf{Tr}(A) - 1) \geq -1$. Since the trace equals the sum of the eigenvalues, we conclude that $(n-1)\lambda_1 + \lambda_n \geq -1$. If $\lambda_1 \geq 0$. This implies that $\frac{|\lambda_1|}{|\lambda_1|+|\lambda_n|} \geq \frac{|\lambda_n|-1}{(n-1)(|\lambda_1|+|\lambda_n|)}$. If $|\lambda_n| \geq 2$, then $\frac{|\lambda_1|}{|\lambda_1|+|\lambda_n|} \geq \frac{1}{n}$. Otherwise, $\frac{\delta}{|\lambda_n|} \geq \frac{\varepsilon}{4n}$. Thus, $\gamma = \max\{\frac{|\lambda_1|}{|\lambda_1|+|\lambda_n|}, \frac{\delta}{|\lambda_n|}\} \geq \frac{\varepsilon}{4n}$, and by Lemma 6.5, the eigenvector procedure takes $\tilde{O}(N\frac{\sqrt{n}}{\sqrt{\varepsilon}})$ time.

If we apply the sparsification procedure of Lemma 6.6, then the relevant parameters are $S = \sum_{ij}|C_{ij}| = O(\frac{1}{\alpha}\sum_{ij}|A_{ij}|) = O(\frac{1}{\alpha})$ (recall $\sum_{ij}|A_{ij}| = 1$). Thus the sparsification procedure yields a matrix C' with $O(\frac{n^{1.5}}{\varepsilon\alpha})$ non-zero entries. Overall, the running time of the Lanczos algorithm becomes $\tilde{O}(\min\{N, \frac{n^{1.5}}{\varepsilon\alpha}\} \cdot \sqrt{\frac{n}{\varepsilon}})$ as stated.

Putting everything together, the final running time of the algorithm becomes

$$\tilde{O}\left(\frac{n^{1.5}}{\varepsilon^{2.5}} \cdot \min\left\{N,\ \frac{n^{1.5}}{\varepsilon\alpha^*}\right\}\right)$$

□

6.5.2 SDP relaxations of biological probability estimation problems

The following SDP arises in the context of the biologically-motivated problem of estimating haplotype frequencies. See chapter 5 for a more detailed description of the problem.

$$\begin{aligned} \max\ A \bullet X & \\ \sum_{ij} X_{ij} &= 1 \\ X_{ij} &\geq 0 \qquad 1 \leq i,j \leq n \\ X &\succeq 0 \end{aligned} \qquad (\textsc{HaploFreq})$$

where A is a non-negative matrix, i.e. all its entries are non-negative. This SDP is a natural relaxation in certain problems where a probability distribution is required. Intuitively, we want to find a probability distribution $\{p_1, p_2, \ldots, p_n\}$ which maximizes the objective $\sum_{ij} A_{ij} p_i p_j$. In the SDP relaxation, the X_{ij} variables represent $p_i p_j$.

We apply our method to this problem. Step I requires that we bound the optimum and the trace. Let the optimum to this SDP be denoted α^*. We claim that α^* is in the range $\max_{ij}\{A_{ij}\} \cdot [\frac{1}{4}, 1]$. The upper bound is trivial since the objective is a linear combination of the A_{ij} values. Let A_{kl} be the maximal A_{ij}. Then the lower bound is obtained by taking the unit vector $u = \frac{1}{2}(e_l + e_k)$, where e_i is the i^{th} standard basis vector, and letting X be the positive semidefinite matrix uu^T. Since all A_{ij} are non-negative, this solution has value at least $\frac{1}{4}A_{kl}$.

The trace of X is trivially bounded by 1 from the first constraint. Note also that w.l.o.g. we can relax the first constraint to be $\sum_{ij} X_{ij} \leq 1$, in the optimum the sum obviously equals 1 since all the quantities are non-negative.

Theorem 6.5. *SDP* (HaploFreq) *can be approximated up to a multiplicative error of* $1 - O(\varepsilon)$ *in* $\tilde{O}(\frac{n^{2.5}}{\varepsilon^{2.5}})$ *time.*

Proof. According to step II, we "guess" α using binary search and write the following feasibility SDP. Here, $\mathcal{P}$ is the convex set $\{X \succeq 0\ , \sum_i X_{ii} \leq 1\}$.

$$\begin{aligned} \frac{1}{\alpha} A \bullet X - 1 &\geq 0 \\ 1 - \sum_{ij} X_{ij} &\geq 0 \\ X_{ij} &\geq 0 \qquad 1 \leq i,j \leq n \\ X &\in \mathcal{P} \end{aligned}$$

The width of the first constraint is $(\frac{1}{\alpha}\sum_{ij}|A_{ij}|)^2 = O(n^4)$. This is the high width constraint which we will put into the inner SDP. The width for the rest of the constraints is $O(1)$, these constitute the outer SDP. Thus, $\ell_L = \rho_L = 1$, and $\delta = \frac{\varepsilon}{2}$. Let C represent the weighted combination of the constraints for the ORACLE. According to Corollary 2, the SDP can be ε approximately in $\tilde{O}(\frac{1}{\varepsilon^2}\cdot[T_{ev}(C,\frac{\varepsilon}{2})+n^2])$ time.

It remains to estimate $T_{ev}(C,\frac{\varepsilon}{2})$. The matrix C is of the form $w_0(\frac{1}{\alpha}A-\frac{1}{n}I)+w_1(\frac{1}{n}I-J)+\sum_{ij}w_{ij}E_{ij}$, where J is the all 1's matrix, and w_0, w_1, w_{ij} for $1 \le i,j \le n$ are non-negative weights summing to 1.

To bound the most negative eigenvalue, λ_n, of C, we use the Gershgorin circle theorem. This implies that $|\lambda_n| \le \max_i\{\sum_j |C_{ij}|\}$. For the matrix C, the dominant contributors to this maximum are the matrices $\frac{1}{\alpha}A$ and J. For any i, we have $\sum_j \frac{1}{\alpha}|A_{ij}| \le 4n$ since $\alpha \ge \frac{1}{4}\max_{ij} A_{ij}$. Also, for any i, $\sum_j |J_{ij}| = n$. Thus, the bound on $|\lambda_n|$ is $O(n)$.

Thus, the γ of Lemma 6.5 is $\ge \Omega(\frac{\varepsilon}{n})$, and hence $T_{ev}(C,\frac{\varepsilon}{2}) = \tilde{O}(\frac{n^{2.5}}{\sqrt{\varepsilon}})$ because C is a dense matrix. Since $\sum_{ij}|C_{ij}|$ can be as large as $\Omega(n^2)$, sparsification does not help here.

Finally, the total running time comes to $\tilde{O}(\frac{n^{2.5}}{\varepsilon^{2.5}})$. □

For comparison, the best known interior point algorithm solves this SDP in $\tilde{O}(n^4)$ time. As a corollary to this theorem we can prove Lemma 5.16 from section 5.4.2:

Proof of Lemma 5.16. By definition 5.15, an ε-good matrix X with respect to a PSD matrix $P \succeq 0$ satisfies:

$$\begin{aligned} \sum_{ij} X_{ij} &= 1 \\ X_{ij} &\ge 0 \qquad 1 \le i,j \le n \\ X &\succeq 0 \end{aligned}$$
$$\left(\sum_{i=1}^{m}\frac{A_i}{A_i \bullet P}\right)\bullet X \ge \varepsilon - m$$

where the matrices A_i are non-negative. Hence, the ε-good matrix for the largest possible ε can be found by solving SDP (HAPLOFREQ) where the matrix A is replaced by $(\sum_{i=1}^{m}\frac{A_i}{A_i\bullet P})$. □

6.5.3 SDP relaxations in Side Chain Positioning Problems

Consider the following SDP arising from computational problems having to do with side chain positioning problems in analyzing protein structure [CKS04]:

$$\begin{aligned} &\min\ E \bullet X \\ \sum_{i,j\in V_k} X_{ij} = \sum_{j\in V_k} X_{jj} &= 1 \qquad 1 \le k \le p \\ X_{ij} &\ge 0 \qquad 1 \le i,j \le n \\ X &\succeq 0 \qquad\qquad \text{(SCP)} \end{aligned}$$

Here, the V_k for $1 \leq k \leq p$ are disjoint sets of indices which together cover all the indices in $[n]$. In the protein folding problem, the sets V_k are bounded in size by a constant. This implies that the the ratio $\frac{n}{p}$ is also bounded by a constant.

Physically, the matrix E represents the *energy matrix* of the protein, and the entry E_{ij} specifies the energy between the positions i, j. Typically, for any position i, only a constant number (as n and p grow) of positions j have a significant value of E_{ij}. Thus, it may be assumed that for any i, the sum $\sum_j |E_{ij}|$ is bounded by a constant. This enables us to bound the eigenvalues of E by a constant, via the Gershgorin Circle Theorem.

Depending on the specific instance, the objective value of the SDP, denoted by α^*, above can be either negative or positive. Thus an additive ε approximation to α^* is appropriate here. An obvious bound on $|\alpha^*|$ is $\sum_{ij} |E_{ij}| = O(n)$. So the binary search needs only $\tilde{O}(1)$ steps. Finally, a bound on the trace of X is p, which follows from the second constraints. Let N denote the number of non-zero entries of the matrix E.

Theorem 6.6. *SDP* (SCP) *can be approximated up to an additive error ε in $\tilde{O}\left(\frac{n^{1.5}N}{\varepsilon^{4.5}}\right)$ time.*

Proof. As usual, guess the objective value α using binary search and write the following SDP. This time we choose the convex set to be $\mathcal{P} = \{X \succeq 0\ ,\ \sum_{i \in V_k} X_{ii} = 1,\ E \bullet X \leq \alpha\ \}$. For convenience of notation, the equality constraints below really stand for two inequalities in the standard manner.

$$\begin{aligned} 1 - \sum_{i,j \in V_k} X_{ij} &= 0 \qquad 1 \leq k \leq p \\ X_{ij} &\geq 0 \qquad 1 \leq i, j \leq k \\ X &\in \mathcal{P} \end{aligned}$$

Since $X \succeq 0$, we have $|X_{ij}| \leq \sqrt{X_{ii}X_{jj}} \leq \frac{X_{ii}+X_{jj}}{2} = O(1)$ for $X \in \mathcal{P}$. Thus, the range of the constraints $X_{ij} \geq 0$ is $[-O(1), O(1)]$. Since the sets V_k are of constant size, the constraints $1 - \sum_{i,j \in V_k} X_{ij} = 0$ contain $O(1)$ terms, each bounded by $O(1)$, so the range for these constraints is also $[-O(1), O(1)]$. Thus, by Theorem 6.1, this SDP can be approximated up to ε in time $\tilde{O}(\frac{1}{\varepsilon^2}(n^2 + T_{\text{oracle}}))$.

Here, T_{oracle} is the time required to approximate the following SDP. Let C_1 be the matrix obtained by the weighted combination of the constrains of the previous SDP as specified by Lemma 6.4. Let $\mathcal{Q} = \{X \succeq 0\ ,\ \sum_i X_{ii} = p\}$.

$$\begin{aligned} C_1 \bullet X &\geq 0 \\ \alpha - E \bullet X &\geq 0 \\ \sum_{i \in V_k} 1 - X_{ii} &= 0 \\ X &\in \mathcal{Q} \end{aligned}$$

The range of the constraints of the type $1 - \sum_{i \in V_k} X_{ii} = 0$ is $[-1, p]$ or $[-p, 1]$ (for the two inequalities) by the trace bound. The two other constraints have width bounded by

$O(n^2)$. We use the SDP composition of Corollary 2, and put the two high width constraints into the inner SDP and the rest into the outer SDP. Here, $\ell_L = 1,\ \rho_L = p,\ m = p+2$, and $\delta = \frac{\varepsilon}{2p}$. Let C_2 be the matrix representing the weighted combination of all the constraints.Thus, an ε approximate solution to the SDP can be found in time

$$T_{\text{oracle}} = \tilde{O}\left(\frac{p}{\varepsilon^2} \cdot \left[p + T_{\text{ev}}\left(C_2, \frac{\varepsilon}{2p}\right)\right]\right)$$

The most negative eigenvalue of C_2, λ_n, can be bounded in absolute value by $O(1)$. This is because the eigenvalues of E are bounded by a constant as discussed earlier. All the other constraints have only $O(1)$ terms so the same is true for the them. Thus, by Lemma 6.5, $T_{\text{ev}}(C_2, \frac{\varepsilon}{2p})$ can be bounded by $O(\frac{N\sqrt{p}}{\sqrt{\varepsilon}})$. Sparsification does not help here (and it is expected that the matrix E is sparse to begin with). Since $p = O(n)$, the total running time comes to $\tilde{O}(\frac{n^{1.5}N}{\varepsilon^{4.5}})$. $\square$

For comparison, the best known interior point algorithm solves this SDP in $\tilde{O}(n^4)$ time.

6.5.4 Embedding of finite metric spaces into ℓ_2

Given a finite metric space on n points specified by the pairwise distances $\{D_{ij}\}$, embedding into ℓ_2 with minimum distortion amounts to solving the following mathematical program. For convenience of notation, let $d_{ij} = D_{ij}^2$.

$$\begin{aligned} &\min \alpha \\ d_{ij} \ \le\ &||v_i - v_j||^2 \ \le\ \alpha \cdot d_{ij} \qquad 1 \le i < j \le n \\ &v_i \ \in\ \mathbb{R}^n \qquad\qquad\qquad 1 \le i \le n \end{aligned} \qquad (\textsc{Embedding})$$

By Bourgain's theorem [cite...] the minimum distortion is $O(\log n)$. Thus, the optimum value α^* of SDP Embedding is $O(\log^2 n)$. We assume that the distances are scaled so that $\sum_{ij} d_{ij} = n^2$. We claim that this implies that there is an optimal solution $v_1, v_2, \ldots, v_n$ which satisfies $\sum_i ||v_i||^2 \le \alpha^* n$: we may assume that the optimal solution satisfies $\sum_i v_i = 0$, otherwise we can shift the origin to the sum of the vectors; this does not change the pairwise distances $||v_i - v_j||^2$. Thus we have $\alpha^* \sum_{ij} d_{ij} \ge \sum_{ij} ||v_i - v_j||^2 = n \sum_i ||v_i||^2$. Since $\alpha^* = O(\log n)$, the claim follows.

Theorem 6.7. *SDP* (Embedding) *can be approximated up to a $1 + O(\varepsilon)$ multiplicative factor in $\tilde{O}(\frac{n^3}{d_{min}^{2.5}\varepsilon^{3.5}})$ time.*

Proof. Guess α using binary search, and formulate the following SDP. Define the convex set $\mathcal{P} = \{X \succeq 0 \mid \sum_i X_{ii} \le \alpha n\}$.

$$\begin{aligned} \frac{\alpha}{d_{ij}}(X_{ii} - 2X_{ij} + X_{jj}) - 1 \ &\ge\ 0 \qquad 1 \le i < j \le n \\ 1 - \frac{1}{d_{ij}}(X_{ii} - 2X_{ij} + X_{jj}) \ &\ge\ 0 \qquad 1 \le i < j \le n \\ X \ &\in\ \mathcal{P} \end{aligned} \qquad (6.6)$$

Since $X \succeq 0$, the expression $(X_{ii} - 2X_{ij} + X_{jj})$ is always positive. The bound on the trace implies that these terms are bounded by n. Hence the width of the constraints of SDP (6.6) is bounded by $\rho_L \leq \frac{n\alpha}{d_{ij}} = \tilde{O}(\frac{n}{d_{\min}})$, where $d_{\min} = \min_{ij}\{d_{ij}\}$. Also, $\ell_L = 1$, $m = 2n^2$, and $\delta = \frac{\varepsilon}{2\alpha n}$. Let C be the matrix representing the weighted combination of all the constraints. Thus, an ε approximate solution to the SDP can be found in time $\tilde{O}(\frac{n}{d_{\min}\varepsilon^2} \cdot \left[n^2 + T_{ev}\left(C, \frac{\varepsilon}{2\alpha n}\right)\right])$.

The most negative eigenvalue of C, λ_n, can be bounded in absolute value by $O(\frac{1}{d_{\min}})$. This is because all constraints have only $O(1)$ terms, each bounded in absolute value by $O(\frac{\alpha}{d_{\min}}) = \tilde{O}(\frac{1}{d_{\min}})$. Since C is a convex combination of the constraints, we conclude that $\sum_{ij} |C_{ij}| = \tilde{O}(\frac{1}{d_{\min}})$, and this bounds $|\lambda_n|$. Thus, by Lemma 6.5, $T_{\text{ev}}(C, \frac{\varepsilon}{n})$ can be bounded by $\tilde{O}(\frac{N\sqrt{n}}{\sqrt{\varepsilon d_{\min}}})$, where N is the number of non-zero entries in C.

Sparsification could potentially reduce the number of matrix entries. Since $S = \sum_{ij} |C_{ij}| = \tilde{O}(\frac{1}{d_{\min}})$, so by Lemma 6.6 the number of entries could be reduced to $\tilde{O}(\frac{n^{1.5}}{\varepsilon d_{\min}})$.

Thus the total running time comes to

$$\tilde{O}\left(\min\left\{\frac{n^3}{d_{\min}^{2.5}\varepsilon^{3.5}}, \frac{n^{3.5}}{d_{\min}^{1.5}\varepsilon^{2.5}}\right\}\right)$$

For comparison, interior point methods can solve this SDP in time $\tilde{O}(n^4)$. Note that in order to beat the running time of interior point methods, the first expression is always better.

□

6.5.5 SDP relaxation of Sparsest Cut

For a graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$, the following SDP arises as a relaxation for the Sparsest Cut problem in [ARV04]:

$$\begin{aligned} \min \sum_{\{i,j\} \in E} & ||v_i - v_j||^2 \\ ||v_i - v_j||^2 + ||v_i - v_k||^2 - ||v_j - v_k||^2 &\geq 0 \qquad 1 \leq i < j < k \leq n \\ \sum_{i<j} ||v_i - v_j||^2 &= n \qquad \text{(Sparsest Cut)} \end{aligned}$$

The range of the optimum, α^*, is clearly $[0, n]$. We desire an additive ε approximate solution.

Theorem 6.8. *SDP* (Sparsest Cut) *can be approximated up to an additive error ε in $\tilde{O}(\frac{n^3}{\varepsilon^2})$ time.*

Proof. As usual, we guess α using binary search, and write the following SDP, with the convex set $\mathcal{P} = \{X \succeq 0,\ \sum_i X_{ii} = 1\}$:

$$\begin{aligned}
\alpha - \sum_{ij \in E} (X_{ii} - 2X_{ij} + X_{jj}) &\geq 0 \\
(X_{ii} - 2X_{ij} + X_{jj}) + (X_{ii} - 2X_{ik} + X_{kk}) - (X_{jj} - 2X_{jk} + X_{kk}) &\geq 0 \qquad 1 \leq i < j < k \leq n \\
X &\in \mathcal{P}
\end{aligned}$$

Here, just as in the embeddings problem from the previous section, we make the assumption that $\sum_i v_i = 0$ w.l.o.g. Then the constraint $\sum_{ij} ||v_i - v_j||^2 \leq n$ is equivalent to $\sum_i ||v_i||^2 \leq 1$.

Since $\sum_i X_{ii} = 1$, for any i, j, $|X_{ij}| \leq \sqrt{X_{ii} X_{jj}} \leq 1$. Thus, the width of the first constraint is $O(n^2)$. The width for the rest of the constraints is $O(1)$. Since there are n^3 constraints anyway, for the Oracle we solve the eigenvector problem up to arbitrary precision using a standard algorithm such as QR, which runs in n^3 time as well. Finally, using the inner and outer SDPs of Corollary 2,the SDP can be approximated up to ε in time:

$$\tilde{O}(\frac{1}{\varepsilon^2} \cdot \left[T_{\text{oracle}} + n^3\right]) = \tilde{O}\left(\frac{n^3}{\varepsilon^2}\right)$$

□

6.5.6 SDP relaxations for Min Uncut and related problems

The approximation algorithm of Charikar *et al* [ACMM05] for the Min Uncut problem for a graph $G = (V, E)$ requires solving the following SDP formulation, for vectors $v_1, v_2, \ldots, v_n, v_{-1}, v_{-2}, \ldots, v_{-n} \in \mathbb{R}^{n \times n}$:

$$\begin{aligned}
\min \frac{1}{4} \sum_{\{i,j\} \in E} &||v_i - v_j||^2 \\
||v_i - v_j||^2 + ||v_i - v_k||^2 - ||v_j - v_k||^2 &\geq 0 \qquad 1 \leq i < j < k \leq n \\
||v_i||^2 &= 1 \qquad 1 \leq i \leq n \\
v_i \cdot v_{-i} &= -1 \qquad 1 \leq i \leq n \qquad \text{(Min Uncut)}
\end{aligned}$$

The range of the optimum is clearly bounded by $[0, |E|]$. We desire an additive ε approximate solution.

Theorem 6.9. *SDP* (Min Uncut) *can be approximated up to an additive error of* ε *in* $\tilde{O}(\frac{n^{3.5}}{\varepsilon^2})$ *time.*

Proof. Guess α by binary search. Note that the positive semidefinite matrix X has rows and columns indexed by $1, 2, \ldots, n, -1, -2, \ldots, -n$, unlike the other examples. We use a inner and outer SDP formulation as usual, except that this time we solve the inner SDP using Alizadeh's algorithm.

Thus, define the convex set

$$\mathcal{P} = \{X \succeq 0;\ \ 1 \leq i \leq n:\ X_{ii} = 1,\ X_{i,-i} = -1;\ \sum_{\{i,j\} \in E} (X_{ii} - 2X_{ij} + X_{jj}) \leq \alpha\ \}$$

We consider the following SDP:

$$\begin{array}{rcll} (X_{ii}-2X_{ij}+X_{jj})+(X_{ii}-2X_{ik}+X_{kk})-(X_{jj}-2X_{jk}+X_{kk}) & \geq & 0 & 1\leq i<j<k\leq n \\ X & \in & \mathcal{P} & \end{array} \tag{6.7}$$

Since all solutions we consider are in the polytope $\mathcal{P}$, the width is $O(1)$. According to Theorem 6.1, the SDP can be approximated up to ε in time $\tilde{O}(\frac{1}{\varepsilon^2}\cdot\left[n^3+T_{\text{oracle}}\right])$, where T_{oracle} is the time required to approximately solve the following SDP:

$$\begin{array}{rcll} C\bullet X & \geq & 0 & \\ X_{ii} & = & 1 & 1\leq i\leq n \\ X_{i,-i} & = & -1 & 1\leq i\leq n \\ \sum_{\{i,j\}\in E}(X_{ii}-2X_{ij}+X_{jj}) & \leq & \alpha & \\ X & \succeq & 0 & \end{array}$$

where C is the matrix obtained from a weighted combination of the constraints of SDP (6.7). This SDP can be solved using Alizadeh's algorithm in time $O(n^{3.5})$. The total running time comes to: $O(\frac{n^{3.5}}{\varepsilon^2})$. □

We note here that similar SDPs arise in relaxations of the other problems like Balanced Separator, Min 2CNF Deletion, etc. See [ACMM05] for a discussion. All of these can be solved in exactly the same manner.

The usefulness of this method for these problems is limited by the fact that the additive error needed for the approximation algorithms to work as desired is quite small, like $\tilde{O}(\frac{\text{OPT}}{|E|})$, where OPT is the particular value we are trying to approximate; such as the minimum number of edges not cut in the Min Uncut problem, the minimum number of edges cut in the Balanced Separator problem, and the minimum number of unsatisfied clauses in Min 2CNF Deletion. Provided these numbers are large enough in comparison to $|E|$, we get performance gains over Alizadeh's interior point algorithm, which takes $\tilde{O}(n^{4.5})$ time for all these problems.

6.6 Matrix sparsification

In this section, we prove the sparsification lemma (Lemma 6.6):

Proof. **[Lemma 6.6]**
The required procedure, Sparsify, is given in Figure 6.2. We now prove that it produces the desired result with high probability.

First, we prove that the number of non-zero entries in $\tilde{A}$ is with high probability $O(\frac{\sqrt{n}S}{\varepsilon})$.

Lemma 6.7. *With probability at least* $1-\exp(-\Omega(\frac{\sqrt{n}S}{\varepsilon}))$, *the matrix* $\tilde{A}$ *contains at most* $O(\frac{\sqrt{n}S}{\varepsilon})$ *non-zero entries.*

```
Procedure SPARSIFY($A, \varepsilon$)
for each $i \le j \in [n]$ do
if $|A_{ij}| > \frac{\varepsilon}{\sqrt{n}}$ then
    $\tilde{A}_{ji} = \tilde{A}_{ij} = A_{ij}$
else
    $\tilde{A}_{ji} = \tilde{A}_{ij} = \begin{cases} \mathrm{sgn}(A_{ij}) \cdot \frac{\varepsilon}{\sqrt{n}} & \text{with probability } p_{ij} = \frac{\sqrt{n}|A_{ij}|}{\varepsilon} \\ 0 & \text{with probability } 1 - p_{ij} \end{cases}$
return $\tilde{A}$
```

Figure 6.2: Procedure SPARSIFY

Proof. Since $\sum_{ij} |A_{ij}| = S$, the number of entries with magnitude larger than $\frac{\varepsilon}{\sqrt{n}}$ is at most $\frac{\sqrt{n}S}{\varepsilon}$. So without loss of generality, we may assume that all the entries have magnitude smaller than $\frac{\varepsilon}{\sqrt{n}}$.

The Chernoff bound [MR95] asserts that if $X_1, X_2, \ldots, X_n$ are indicator random variables and $X = \sum_i X_i$ with $\mathbf{E}[X] = \mu$, then

$$\mathbf{Pr}[X > (1+\delta)\mu] < \left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^\mu$$

In our case, we set up indicator random variables X_{ij} for $i \le j$ which are 0 or 1 depending on whether $\tilde{A}_{ij} = 0$ or not. Let $X = \sum_{i \le j} X_{ij}$. Then $2X$ is an upper bound on the number of non-zero entries of $\tilde{A}$. We have

$$\mathbf{E}[X] = \sum_{i \le j} p_{ij} = \sum_{i \le j} \frac{\sqrt{n}|A_{ij}|}{\varepsilon} \le \frac{\sqrt{n}S}{\varepsilon}.$$

The claim follows by using the Chernoff bound with $\delta = e - 1$. □

Next, define $M = A - \tilde{A}$. We will show that with high probability, for all unit vectors x, we have $|x^\top M x| \le O(\varepsilon)$, which implies $\|A - \tilde{A}\|_2 \le O(\varepsilon)$.

Notice that for all coordinates i, j such that $|A_{ij}| \ge \frac{\varepsilon}{\sqrt{n}}$, we have $M_{ij} = 0$. For the rest of the coordinates, since $\mathbf{E}[\tilde{A}_{ij}] = \mathrm{sgn}(A_{ij}) \cdot \frac{\varepsilon}{\sqrt{n}} \times \frac{\sqrt{n}|A_{ij}|}{\varepsilon} = A_{ij}$, we conclude that $\mathbf{E}[M_{ij}] = 0$. We will now consider a $\frac{\varepsilon_0}{\sqrt{n}}$-grid on the unit sphere (ε_0 is set to some constant, say $\frac{1}{2}$),

$$T = \left\{ x : \ x \in \frac{\varepsilon_0}{\sqrt{n}} \mathbb{Z}^n, \ \|x\|_2 \le 1 \right\}.$$

Feige and Ofek [FO05] give a bound on the size of T and show that it suffices to consider only vectors in T, which we reprove here for completeness.

Lemma 6.8. *The size of $|T|$ is at most $\exp(cn)$ for $c = (\frac{1}{\varepsilon_0} + 2)$. If for every $x, y \in T$ we have $|x^\top M y| \le \varepsilon$, then for every unit vector x, we have $|x^\top M x| \le \frac{\varepsilon}{(1-\varepsilon_0)^2}$.*

Proof. Map every point in $x \in T$ in a one-to-one correspondence with a n-dimensional hypercube of side length $\frac{\varepsilon_0}{\sqrt{n}}$ on the grid:

$$x \mapsto C_x = \left\{ x + u : \ u \geq \vec{0}, \ \|u\|_\infty \leq \frac{\varepsilon_0}{\sqrt{n}} \right\}.$$

The maximum length of any vector in C_x is bounded by $\|x\| + \varepsilon_0 \leq 1 + \varepsilon_0$, and thus the union of these cubes is contained in the n-dimensional ball B of radius $(1+\varepsilon_0)$. We conclude:

$$|T| \times \left(\frac{\varepsilon_0}{\sqrt{n}} \right)^n \ = \ \sum_{x \in T} \mathrm{vol}(C_x) \ \leq \ \mathrm{vol}(B) \ = \ \frac{\pi^{n/2}}{\Gamma(n/2+1)} (1+\varepsilon_0)^n.$$

And so:

$$|T| \ \leq \ \frac{\pi^{n/2}}{\Gamma(n/2+1)} \left(\frac{(1+\varepsilon_0)\sqrt{n}}{\varepsilon_0} \right)^n \ \leq \ \exp\left(\left(\frac{1}{\varepsilon_0} + 2 \right) n \right).$$

Next, given any unit vector, x, let $y = (1-\varepsilon_0)x$. By "rounding down" the coordinates of y to the nearest multiple of $\frac{\varepsilon_0}{\sqrt{n}}$, we get a grid point z such that $y \in C_z$. Thus, the maximum length of any vertex of C_z is bounded by $\|y\| + \varepsilon_0 = 1$, so all vertices of C_z are grid points in T. Express y as a convex combination of the vertices v_i of C_z; viz. $y = \sum_i \alpha_i v_i$ with $\alpha_i \geq 0$ and $\sum_i \alpha_i = 1$. Then we have

$$|y^\top M y| \ = \ |(\sum_i \alpha_i v_i)^\top M (\sum_i \alpha_i v_i)| \ \leq \ \sum_{i,j} \alpha_i \alpha_j |v_i^\top M v_j| \ \leq \sum_{i,j} \alpha_i \alpha_j \varepsilon \ = \ \varepsilon.$$

The second inequality above follows because we assumed that for all $x, y \in T$, $|x^\top M y| \leq \varepsilon$. Finally, since $y = (1-\varepsilon_0)x$, we have

$$|x^\top M x| \ = \ \frac{|y^\top M y|}{(1-\varepsilon_0)^2} \ \leq \ \frac{\varepsilon}{(1-\varepsilon_0)^2}.$$

□

Let $x, y \in T$. Since $\mathbf{E}[M_{ij}] = 0$, we conclude that $\mathbf{E}[x^\top M y] = 0$. We now a prove strong concentration bound:

Lemma 6.9. *With probability at least* $1 - \exp(-\Omega(n))$, *for every* $x, y \in T$ *it holds that* $|x^\top M y| \leq c\varepsilon$.

Proof. We use the following bound from Hoeffding's original paper [Hoe63]: let $X_1, ..., X_n$ be independent random variables, such that X_i takes values in the range $[a_i, b_i]$. Let $X = \sum_i X_i$, and $\mathbf{E}[X] = \mu$. Then for any $t > 0$

$$\Pr[|X - \mu| \geq t] \ \leq \ 2 \exp\left(-\frac{2t^2}{\sum_i (b_i - a_i)^2} \right).$$

Consider the random variables $Z_{ij} = M_{ij}x_iy_j$, then $x^\top My = \sum_{ij} M_{ij}x_iy_j = \sum_{ij} Z_{ij}$. Since $\tilde{A}_{ij}$ is either $\text{sgn}(A_{ij}) \cdot \frac{\varepsilon}{\sqrt{n}}$ or 0, the squared range of M_{ij} is $\frac{\varepsilon^2}{n}$. Thus, the sum of squared ranges for the variables $\{Z_{ij}, i \le j\}$ at most $\sum_{i\le j} \frac{\varepsilon^2}{n} x_i^2 y_j^2 \le \frac{\varepsilon^2}{n} \sum_i x_i^2 \sum_j y_j^2 \le \frac{\varepsilon^2}{n}$, and similarly the sum of squared ranges for the variables $\{Z_{ij}, i > j\}$ is bounded by $\frac{\varepsilon^2}{n}$. Since $\mathbf{E}[Z_{ij}] = 0$, by the Hoeffding bound we have:

$$\Pr\left[|\sum_{i\le j} Z_{ij}| \ \ge \ c\varepsilon\right] \ \le \ 2\exp\left(-\frac{2c^2\varepsilon^2}{\frac{\varepsilon^2}{n}}\right) \ = \ 2\exp(-2c^2n).$$

A similar bound holds for $\Pr[|\sum_{i>j} Z_{ij}| \ \ge \ c\varepsilon]$. Since $|x^\top My| = |\sum_{i\le j} Z_{ij} + \sum_{i>j} Z_{ij}|$, by the union bound we have

$$\Pr[|x^\top My| \ \ge \ 2c\varepsilon] \ \le \ 4\exp(-2c^2n).$$

Since there are $\exp(2cn)$ pairs of vectors $x, y \in T$, the union bound implies that with probability at least $1 - \exp(-\Omega(n))$, for all vectors $x, y \in T$, we have $|x^\top My| \le c\varepsilon$. □

This concludes the proof of Lemma 6.6. □

6.7 Discussion

In this chapter we have described hybrid Lagrangian relaxation algorithms for solving SDPs. The ideas are general though we customize them for some interesting SDPs. Each iteration step is an approximate eigenvector computation, which is very efficient in practice, even though the theoretical worst case bounds listed here do not show this. (Even so, in several cases the worst-case bounds provide speedups for specific SDPs over interior point methods.) The main benefit comes from avoiding expensive Cholesky decompositions which interior point methods require. Also, since the final solution is obtained as a convex combination of many rank 1 matrices, its Cholesky decomposition is automatically obtained and there is no extra work to be done. Typically, approximation algorithms require the Cholesky decomposition of the optimal solution.

The chief limitation of this method is chiefly from the polynomial dependence on $\frac{1}{\varepsilon}$. Some applications (such as general SPARSEST CUT) requires ε to be very tiny and then this method is rendered useless. The main goal of future work will be to reduce the dependence on $\frac{1}{\varepsilon}$. The framework of chapter 4 is clearly relevant for this goal.

Appendix A

A brief history of Newton's method

Isaac Newton first described his method for iteratively solving equations of the form $f(x) = 0$ in "*De metodis fluxionum et serierum infinitarum*", dating back to mid 1669 [1]. In modern terms, the algorithm iteratively updates an approximate solution according to the formula

$$x_{i+1} \leftarrow x_i - \frac{f(x_i)}{f'(x_i)}$$

The original description is quite far from this succinct formula, but rather proceeds by computing a sequence of polynomials and deriving the approximation only at the end. His original method applies only to polynomial functions (although later in "*Principia Mathematica*", Newton applied the method to a function which is not a polynomial).

Newton's method was probably based on the widely used (at the time) *Viete method*, published circa 1600, which is a less efficient iterative procedure for computing the solution for equations of the form $f(x) = 0$. The origins of Viete's method go back to the arabic mathematician of the 12'th century Sharaf al-Din al-Tusi, and possibly even earlier. The idea of an iterative procedure for computing solutions goes back further, to the arabic algebraist al-Khaayyam (active circa 1100) and earlier to Babylonian and Greek methods.

Joseph Raphson published his work on an iterative method for solving equations in 1690. The method is a significant simplification of Newton's original presentation, although like Newton, Raphson considered the application only to polynomial equations. Interestingly, Raphson did not notice the connection to Newton's method, and maintained it was of different origin. Lagrange observed in 1798 that the two methods are identical.

Surprisingly, it seems that both Newton and Raphson viewed their methods as purely algebraic, and failed to notice the intimate connection to calculus (whose development is attributed to Leibnitz and Newton himself). The connection to calculus was noted by Thomas Simpson (1710-1761), who also noted that the method can be used to solve systems of non-linear equations in several variables.

[1]the material in this section is taken from Ypma's comprehensive study on the history of Newton's method [Ypm95]

Simpson also observed that the Newton method can be used in multivariate unconstrained optimization: in order to maximize a function of several variables, he proposed to set the gradient equal to zero, and solve the system of nonlinear equations using Newton's method.

Bibliography

[AC88] P. Algoet and T.M. Cover. Asymptotic optimality and asymptotic equipartition properties of log-optimum investment. *Annals of Probability*, 2:876–898, 1988.

[ACMM05] Amit Agarwal, Moses Charikar, Konstantin Makarychev, and Yury Makarychev. O(sqrt(log n)) approximation algorithms for min uncut, min 2cnf deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

[AH05] Amit Agarwal and Elad Hazan. Efficient algorithms for online game playing and universal portfolio management. *ECCC TR06-033*, 2005.

[AHK04] Sanjeev Arora, Elad Hazan, and Satyen Kale. $O(\sqrt{\log n})$ approximation to sparsest cut in $\tilde{O}(n^2)$ time. In *45th IEEE FOCS*, pages 238–247, 2004.

[AHK05a] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. *manuscript*, 2005.

[AHK05b] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semide.nite programming using the multiplicative weights update method. In *46th IEEE FOCS*, pages 339–348, 2005.

[AHKS06] Amit Agarwal, Elad Hazan, Satyen Kale, and Robert E. Schapire. Algorithms for portfolio management based on the newton method. *19'th ICML*, 2006.

[Ali95] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1):13–51, 1995.

[ALN05] Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 553–562, 2005.

[AN04] Noga Alon and Assaf Naor. Approximating the cut-norm via grothendieck's inequality. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 72–80, New York, NY, USA, 2004. ACM Press.

[ARV04] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 222–231, 2004.

[AS92] N. Alon and J.H. Spencer. *The probabilistic method.* Wiley, 1992.

[BC80] R. Bell and T.M. Cover. Competitive optimality of logarithmic investment. *Mathematics of Operations Research*, 2:161–166, 1980.

[BC88] R. Bell and T.M. Cover. Game-theoretic optimal portfolios. *Management Science*, 6:724–733, 1988.

[BEYG04] A. Borodin, R. El-Yaniv, and V. Gogan. Can we learn to beat the best stock. *Journal of Artificial Intelligence Research*, 21:579–594, May 2004.

[BI04] D. Bienstock and G. Iyengar. Solving fractional packing problems in $o^{\mathrm{ast}}(1/\varepsilon)$ iterations. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 146–155, New York, NY, USA, 2004. ACM Press.

[Bie01] D. Bienstock. Potential function methods for approximately solving linear programs: Theory and practice, 2001.

[Bie06] Daniel Bienstock. personal communications, 2006.

[BK97] Avrim Blum and Adam Kalai. Universal portfolios with and without transaction costs. In *COLT '97: Proceedings of the tenth annual conference on Computational learning theory*, pages 309–313, New York, NY, USA, 1997. ACM Press.

[Bla56] D. Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6(1):1–8, 1956.

[Blu98] A. Blum. On-line algorithms in machine learning. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 306–325. LNCS 1442, 1998.

[BR92] M. Bellare and P. Rogaway. The complexity of approximating a quadratic program. *Journal of Mathematical Programming*, 409, 1992.

[Bro05] M. Brookes. The matrix reference manual. *[online] www.ee.ic.ac.uk/hp/staff/dmb/matrix/intro.html*, 2005.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[CBL06] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games.* Cambridge University Press, Cambridge, 2006.

[CGR05] S. Chawla, A. Gupta, and H. Räcke. Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005.

[CKS04] B. Chazelle, C. Kingsford, , and M. Singh. A semidefinite programming approach to side-chain positioning with new rounding strategies. In *INFORMS Journal on Computing, Special Issue on Computational Molecular Biology/Bioinformatics*, pages 380–392, 2004.

[Cla90] AG Clark. Inference of haplotypes from pcr-amplified samples of diploid populations. *Journal of Molecular Biology and Evolution*, 7(2):111–22, Mar 1990.

[Cov91] T. Cover. Universal portfolios. *Math. Finance*, 1:1–19, 1991.

[Cov96] Thomas M. Cover. Universal data compression and portfolio selection. In *37th Annual Symposium on Foundations of Computer Science*, pages 534–538, Burlington, Vermont, 1996. IEEE.

[CW04] Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending grothendieck's inequality. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 54–60, Washington, DC, USA, 2004. IEEE Computer Society.

[DV04] John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 315–320, New York, NY, USA, 2004. ACM Press.

[ES95] L Excoffier and M Slatkin. Maximum-likelihood estimation of molecular haplotype frequencies in a diploid population. *Molecular Biology and Evolution*, 12(5):921–7, Sept 1995.

[FL99] Drew Fudenberg and David K. Levine. An easier way to calibrate. *Games and Economic Behavior*, 29(1-2):131–137, October 1999. available at http://ideas.repec.org/a/eee/gamebe/v29y1999i1-2p131-137.html.

[Fle00] Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discret. Math.*, 13(4):505–520, 2000.

[FO05] U. Feige and E. Ofek. Spectral techniques applied to sparse random graphs. *To appear in Random Structures and Algorithms*, 2005.

[FS97] Yoav Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[FS99] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[FS00] D. Fallin and NJ. Schork. Accuracy of haplotype frequency estimation for biallelic loci, via the expectation-maximization algorithm for unphased diploid genotype data. *American Journal of Human Genetics*, 67:947–959, 2000.

[FV93] Dean P. Foster and Rakesh V. Vohra. A randomization rule for selecting forecasts. *Operations Research*, 41(4):704–709, 1993.

[FV98] Dean P. Foster and Rakesh V. Vohra. Asymptotic calibration. *Biometrika*, 85(2):379–390, 1998.

[FV99] Dean P. Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1-2):7–35, October 1999.

[FW56] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:149–154, 1956.

[GK94] Michael D. Grigoriadis and Leonid G. Khachiyan. Fast approximation schemes for convex programs with many block and coupling constraints. *SIAM Journal on Optimization*, 4:86–107, 1994.

[GK95] M. Grigoriadis and L. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. In *Operations Research Letters*, volume 18, pages 53–58, 1995.

[GK98] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science(FOCS-98)*, pages 300–309, Los Alamitos, CA, November8–11 1998. IEEE Computer Society.

[GLS94] Martin Grotschel, Laszlo Lovasz, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization (Algorithms and Combinatorics)*. Springer, December 1994.

[GS00] Alexei A. Gaivoronski and Fabio Stella. Stochastic nonstationary optimization for finding universal portfolios. *Annals of Operations Research*, 100:165–188, 2000.

[Gus00] D Gusfield. A practical algorithm for optimal inference of haplotypes from diploid populations. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, 2000.

[Gus01] D Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *Journal of Computational Biology*, 8(3):305–23, 2001.

[Gus02] Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In *Proceedings of the 6th Annual International Conference on Research in Computational Molecular Biology*, 2002.

[GW95] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[GW00] C. Gentile and M. K. Warmuth. Proving relative loss bounds for online learning algorithms by the bregman divergence. In *The 13th Annual Conference on Computational Learning Theory, Tutorial*, 2000.

[Han57] J. Hannan. Approximation to bayes risk in repeated plays. In M. Dresher, A Tucker, and P.Wolfe, editors, *Contributaions to the Theory of Games*, volume 3, pages 97–139. Princeton University Press, 1957.

[Has96] J. Hastad. Clique is hard to approximate within n1-. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, page 627, Washington, DC, USA, 1996. IEEE Computer Society.

[Haz06] Elad Hazan. Approximate optimization by online game playing. Technical report, Princeton University, 2006.

[HE04] E. Halperin and E. Eskin. Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, 2004.

[HH06] Eran Halperin and Elad Hazan. Haplofreq - estimating haplotype frequencies efficiently. *Journal of Computational Biology*, 13(2):481–500, 2006.

[HK95] ME Hawley and KK Kidd. Haplo: a program using the em algorithm to estimate the frequencies of multi-site haplotypes. *Journal of Heredity*, 86(5):409–11, Sep-Oct 1995.

[HKKA06] Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. Logarithmic regret algorithms for online convex optimization. *In 19'th COLT*, 2006.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HSSW95] David P. Helmbold, Yoram Singer, Robert E. Schapire, and Manfred K. Warmuth. A comparison of new and old algorithms for a mixture estimation problem. In *COLT '95: Proceedings of the eighth annual conference on Computational learning theory*, pages 69–78, New York, NY, USA, 1995. ACM Press.

[HSSW96] David P. Helmbold, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. On-line portfolio selection using multiplicative updates. In *ICML*, pages 243–251, 1996.

[Jan04] Klaus Jansen. Approximation algorithms for mixed fractional packing and covering problems. In Jean-Jacques Lévy, Ernst W. Mayr, and John C. Mitchell, editors, *IFIP TCS*, pages 223–236. Kluwer, 2004.

[Kel56] J.L. Kelly. A new interpretation of information rate. *Bell Systems Technical Journal*, pages 917–926, 1956.

[Kha04] Rohit Khandekar. *Lagrangian Relaxation based Algorithms for Convex Programming Problems.* PhD thesis, Indian Institute of Technology, Delhi, 2004. Available at `http://www.cse.iitd.ernet.in/~rohitk`.

[KL96] P. Klein and H-I. Lu. Efficient approximation algorithms semidefinite programs arising from max-cut and coloring. In *Proceedings of the 28th annual ACM symposium on Theory of computing*, pages 338–347, New York, NY, USA, 1996. ACM Press.

[KPST94] Philip Klein, Serge Plotkin, Clifford Stein, and Eva Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM J. Comput.*, 23(3):466–487, 1994.

[KS04] Gad Kimmel and Ron Shamir. Maximum likelihood resolution of multi-block genotypes. In *Proceedings of the eighth annual international conference on Computational molecular biology*, pages 2–9. ACM Press, 2004.

[KV03] Adam Kalai and Santosh Vempala. Efficient algorithms for universal portfolios. *J. Mach. Learn. Res.*, 3:423–440, 2003.

[KV05] Adam Kalai and Santosh Vempala. Efficient algorithms for on-line optimization. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[KW92] J. Kuczyn'ski and H. Woz'niakowski. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM Journal on Matrix Analysis and Applications*, 13(4):1094–1122, 1992.

[KW97] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Inf. Comput.*, 132(1):1–63, 1997.

[KY99] Philip Klein and Neal Young. On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. *Lecture Notes in Computer Science*, 1610:320–327, 1999.

[LBI$^+$01] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz. Snps problems, algorithms and complexity, european symposium on algorithms. In Springer-Verlag, editor, *Proceedings of the European Symposium on Algorithms (ESA-2001), Lecture Notes in Computer Science*, volume 2161, pages 182–193, 2001.

[LLR95] N. Linial, E. London, and Yu. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[LSM$^+$91] Tom Leighton, Clifford Stein, Fillia Makedon, &#201;va Tardos, Serge Plotkin, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 101–111, New York, NY, USA, 1991. ACM Press.

[Lue98] David G. Luenberger. *Investment Science*. Oxford University Press, Oxford, 1998.

[LV03a] László Lovász and Santosh Vempala. The geometry of logconcave functions and an $o^*(n^3)$ sampling algorithm. Technical Report MSR-TR-2003-04, Microsoft Research, 2003.

[LV03b] László Lovász and Santosh Vempala. Simulated annealing in convex bodies and an $0^*(n^4)$ volume algorithm. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS)*, pages 650–659, 2003.

[LVBL98] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Herve Lebret. Applications of second-order cone programming, 1998.

[LWU95] JC Long, RC Williams, and M Urbanek. An e-m algorithm and testing strategy for multiple-locus haplotypes. *American Journal of Human Genetics*, 56(3):799–810, Mar 1995.

[MBTB$^+$96] S. Michalatos-Beloin, SA. Tishkoff, KL. Bently, KK. Kidd, and G. Ruano. Molecular haplotyping of genetic markers 10 kb apart by allele-specific long-range pcr. *Nucleic Acids Res*, 24:4841–4843, 1996.

[MF92] Neri Merhav and Meir Feder. Universal sequential learning and decision from individual data sequences. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 413–427, New York, NY, USA, 1992. ACM Press.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge Univ. Press, 1995.

[NIH02] NIH. Large-scale genotyping for the haplotype map of the human genome. RFA: HG-02-005, 2002.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[NQXL01] Niu, Qin, Xu, and Liu. In silico haplotype determination of a vast set of single nucleotide polymorphisms. Technical report, Department of Statistics, Harvard University, 2001.

[PBH$^+$01] N Patil, AJ Berno, DA Hinds, WA Barrett, JM Doshi, CR Hacker, CR Kautzer, DH Lee, C Marjoribanks, DP McDonough, BT Nguyen, MC Norris, JB Sheehan, N Shen, D Stern, RP Stokowski, DJ Thomas, MO Trulson, KR Vyas, KA Frazer, SP Fodor, and DR Cox. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294(5547):1719–23, Nov 23 2001.

[PST91] Serge A. Plotkin, David B. Shmoys, and Tardos Tardos. Fast approximation algorithm for fractional packing and covering problems. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, FOCS'91 (San Juan, Puerto Rico, October 1-4, 1991)*, pages 495–504, Los Alamitos-Washington-Brussels-Tokyo, 1991. IEEE Computer Society Press.

[Sch03] R. E. Schapire. The boosting approach to machine learning: An overview. In D. D. Denison, M. H. Hansen, C. Holmes, B. Mallick, and B. Yu, editors, *Nonlinear Estimation and Classification*. Springer, 2003.

[Sio58] Maurice Sion. On general minimax theorems. *Pacific J. Math.*, 8:171–176, 1958.

[SL05] G. Stoltz and G. Lugosi. Internal regret in on-line portfolio selection. *Machine Learning*, 59:125–159, 2005.

[SM90] Farhad Shahrokhi and David W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.

[SSD01] M. Stephens, N. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction from population data. *American Journal of Human Genetics*, 68:978–989, 2001.

[Vai96] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Math. Program.*, 73(3):291–341, 1996.

[Ye91] Yinyu Ye. An $O(n^3L)$ potential reduction algorithm for linear programming. *Math. Program.*, 50:239–258, 1991.

[You95] Neal E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, San Francisco, California, 22–24 January 1995.

[Ypm95] Tjalling J. Ypma. Historical development of the newton-raphson method. *SIAM Rev.*, 37(4):531–551, 1995.

[Zin03] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference (ICML)*, pages 928–936, 2003.