Efficient Algorithms Using The Multiplicative Weights Update Method

Satyen Kale

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

November 2007

Abstract

Algorithms based on convex optimization, especially linear and semidefinite programming, are ubiquitous in Computer Science. While there are polynomial time algorithms known to solve such problems, quite often the running time of these algorithms is very high. Designing simpler and more efficient algorithms is important for practical impact.

In this thesis, we explore applications of the Multiplicative Weights method in the design of efficient algorithms for various optimization problems. This method, which was repeatedly discovered in quite diverse fields, is an algorithmic technique which maintains a distribution on a certain set of interest, and updates it iteratively by multiplying the probability mass of elements by suitably chosen factors based on feedback obtained by running another algorithm on the distribution.

We present a single meta-algorithm which unifies all known applications of this method in a common framework. Next, we generalize the method to the setting of symmetric matrices rather than real numbers. We derive the following applications of the resulting Matrix Multiplicative Weights algorithm:

1. The first truly general, combinatorial, primal-dual method for designing efficient algorithms for semidefinite programming. Using these techniques, we obtain significantly faster algorithms for obtaining $O(\sqrt{\log n})$ approximations to various graph partitioning problems, such as SPARSEST CUT, BALANCED SEPARATOR in both directed and undirected weighted graphs, and constraint satisfaction problems such as MIN UNCUT and MIN 2CNF DELETION.

2. An $\tilde{O}(n^3)$ time derandomization of the Alon-Roichman construction of expanders using Cayley graphs. The algorithm yields a set of $O(\log n)$ elements which generates an expanding Cayley graph in any group of n elements.

3. An $\tilde{O}(n^3)$ time deterministic $O(\log n)$ approximation algorithm for the quantum hypergraph covering problem.

4. An alternative proof of a result of Aaronson that the γ-fat-shattering dimension of quantum states on n qubits is $O(\frac{n}{\gamma^2})$.

Using our framework for the classical Multiplicative Weights Update method, we derive the following algorithmic applications:

1. Fast algorithms for approximately solving several families of semidefinite programs which beat interior point methods. Our algorithms rely on eigenvector computations, which are very efficient in practice compared to the Cholesky decompositions needed by interior point methods. We also give a matrix sparsification algorithm to speed up the eigenvector computation using the Lanczos iteration.

2. $O(\sqrt{\log n})$ approximation to the SPARSEST CUT and the BALANCED SEPARATOR problems in undirected weighted graphs in $\tilde{O}(n^2)$ time by embedding expander flows in the graph. This improves upon the previous $\tilde{O}(n^{4.5})$ time algorithm of Arora, Rao, and Vazirani, which was based on semidefinite programming.

Acknowledgments

This thesis, the culmination of a five year long journey, would not have been possible were it not for the constant support and encouragement of many people along the way. My biggest gratitude is to my advisor Sanjeev Arora: I couldn't have asked for a better advisor! His brilliance, creativity, keen intuitive thinking, and wide range of knowledge of different topics in theoretical Computer Science have helped me mature and grow as an independent researcher. Throughout the five years he has molded the way I think and given me a taste for working on important problems. The many dinners at his home with his family added that personal touch which made working with him so much more pleasant.

Special thanks are due to my friend and collaborator, Elad Hazan. Elad and I formed a deep friendship being incarcerated together in the basement of the CS department in our first year; thus being forced to work with each other, we continued the momentum all these years and most of the results that form this thesis are joint with Elad.

Princeton, with its stellar Theory faculty, not only in the CS department but also in the Math department and in the IAS, provided a perfect research environment. I owe thanks to the various members of the faculty: Boaz Barak, Moses Charikar, Bernard Chazelle, Rob Schapire, Avi Wigderson, and many others for numerous research discussions and helping create such a great place for Theory research. Thanks also to my friends in the CS department (there are too many to list, so I will sample some randomly): Nir Ailon, Eden Chlamtac, Miroslav Dudík, Adriana Karagiozova, Subhash Khot, Konstantin and Yury Makarychev, Diego Nehab, Iannis Tourlakis, Tony Wirth, C. Seshadhri, David Xiao and others. Many thanks as well to the friendly staff at Princeton, especially Melissa Lawson and Mitra Kelly. I gratefully acknowledge funding through Sanjeev Arora's grants: NSF grants MSPA-MCS 0528414, CCF 0514993, ITR 0205594, CCR 0098180 and the David and Lucile Packard Fellowship.

I had the privilege of two fruitful summer internships, and I owe many thanks to the people who made it possible. In IBM Almaden Research Center in 2005, I enjoyed working with Kamalika Chaudhuri, T. S. Jayram, Robi Krauthgamer, Nimrod Megiddo and Erik Vee. In Microsoft Research Silicon Valley Center in 2006, I had very fruitful collaboration with Kamalika Chaudhuri, Cynthia Dwork, Kunal Talwar and Frank McSherry. Thanks are also due to my co-authors on various papers: Amit Agarwal, Sanjeev Arora, Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Elad Hazan, T. S. Jayram, Adam Kalai Frank McSherry, Rob Schapire, Kunal Talwar and Erik Vee. I would also like to thank various members of the faculty in my undergraduate studies at IIT Bombay: my advisor, Abhiram Ranade, for inspiring me to get into Theory research, and Soumen Chakrabarti, Ajit Diwan, and Sundar Vishwanathan. I am especially grateful to M. Prakash for nurturing my interest in mathematics at an early age in the Mathematics Olympiads, and for being the driving force and inspiration behind my eventual decision to pursue a career in a mathematical discipline.

I warmly thank my friends who started the Ph.D. experience at Princeton with me, and have been my family away from home: Divya Arora, Nandita Gawade, Chaitanya Guttikar, Yogesh Mahajan, Anish Muttreja, Shivani Nayyar, and Sairam Subramanian.

Thanks also to countless other friends; if I had to include them all, this acknowledgements page would be longer than my thesis (so I will just mention some mailing lists that cover a big subset: `eightguyz` and `iitbcs`). Even so, I must thank Tejas Iyer, my friend and partner in much mischief and weird acoustics for twelveteen years. I warmly thank Anjali for being such a close friend all these years, and for the innumerable memorable experiences we shared.

Finally, it would be impossible for me to ever repay the debt of gratitude I owe my family: my parents Chandrakant and Jyoti Kale, and my sister Tanvee, for their love, their undying belief in me, and for their support in hard times. This thesis owes a lot to them, and I dedicate it to them.

To my family.

Contents

List of Figures

Chapter 1

Introduction

Convex optimization deals with the problem of minimizing a convex function on a convex domain in Euclidean space, and is a very well developed area (see [27]). Convex optimization techniques have found wide application in theoretical computer science, especially in the design of efficient algorithms for various fundamental combinatorial optimization problems. Typically, a relaxation to a given combinatorial optimization problem is obtained by embedding its solutions as vectors in Euclidean space, and optimizing the objective function over a convex body defined by constraints satisfied by any solution. This "convexification" of the problem allows the application of powerful tools from convex analysis. In addition, this relaxation endows a combinatorial problem with geometric structure which can be exploited to good effect for rounding the obtained solution to one of the desired characteristics, such as an integer solution.

In particular, various kinds of mathematical programming, such as linear programming (LP) and its more sophisticated cousin, semidefinite programming (SDP), have been of fundamental importance in the design of various exact and approximation algorithms, and numerous algorithms are based directly or indirectly on mathematical programming intuition. Similarly, convex optimization concepts such as duality and complementary slackness have inspired the design of many algorithms. The paradigm of obtaining approximation algorithms to **NP**-hard problems by using their linear or semidefinite programming relaxations has become standard; indeed, rarely does one find an algorithm which is not directly based on either LP or SDP, or for which an interpretation in the setting of an LP or an SDP does not yield additional intuition.

Thus, approximation algorithms for various optimization problems such as VERTEX COVER, SET COVER, and MAX-SAT have been developed inspired by their LP relaxations. Similarly, various network design problems such as k-MEDIAN, MULTICUT, and STEINER FOREST network flow problems such as Max Flow, Multicommodity Flow, etc. have algorithms based on LP. Many scheduling problems are also solved using LP. Even outside traditional Computer Science, linear programming has enjoyed spectacular success. Many practical problems arising in Operations Research and Business Management, such as network flow, inventory management, air traffic management, allocation for human and machine resources, food blending, portfolio and finance management, can be cast as linear programming problems.

Similarly, SDP has recently received much attention in Computer Science as a result of the work of Goemans and Williamson [45], 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 the work of Arora, Rao, Vazirani [18] that gave a new $O(\sqrt{\log n})$-approximation for the Sparsest Cut problem. The ideas of this paper were extended to derive similar approximation algorithms for Min 2CNF Deletion, Min Uncut, directed Sparsest Cut, directed Balanced Separator in [2] and Non-Uniform Sparsest Cut in [32, 17].

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 [75] observe that given an n-point metric space, its minimum-distortion embedding into Euclidean space can be found via SDP. Recent approximation algorithms for the *cut norm* of a matrix [8] and for certain subcases of correlation clustering [31] use SDPs. Halperin and Hazan [51] showed that a biological probability estimation problem, which estimates the frequencies of haplotypes from a noisy sample, can be solved using SDP. Outside Computer Science, semidefinite programming is used in Control Theory and for polynomial optimization.

Given the fundamental importance of linear and semidefinite programming, both of these have been intensively studied by algorithm designers and numerical analysts. Various general purpose methods have been developed which solve LPs and SDPs. Among these are the famous simplex algorithm of Dantzig [38], which, though not polynomial time, is nevertheless extremely efficient and widely used in practice to solve LPs; the ellipsoid method developed by Shor, Yudin and Nemirovskii in the 1970s and later adapted by Khachiyan [63] to give the first polynomial time algorithm for LP; and the interior point method introduced by Karmarkar [60] for LPs and adapted by Nesterov and Nemirovskii [83, 84] and Alizadeh [5] for SDPs. Appendix A gives more details of these algorithms.

Even though interior point methods for LPs are quite efficient in practice, the worst case running times one obtains for several important optimization problems mentioned in the beginning of this chapter are still quite high. This problem is exacerbated in the case of SDP, where interior point methods have even higher worst case running times. 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, and most SDP based algorithms use a general-purpose SDP solver as a first step.

A similar situation developed in the past decade in the 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). Various *primal-dual* algorithms that substitute (previously) LP-based algorithms fall in this category. Though these usually evolve out of (and use the same intuition as) earlier approximation algorithms that used LP as a black box, they do not solve the LP *per se.* Rather, the algorithm incrementally builds a dual solution together with an *integer* primal solution, updating them at each step using "combinatorial" methods. At the end, the candidate dual solution is feasible and the bound on the approximation ratio is derived by comparing the integer primal solution to the bound provided by this feasible dual. Usually the update rule is designed using intuition from the *rounding algorithm* used in the original LP-based algorithm.

Some canonical examples are network design problems [3] (or see the survey [46]) and $O(1)$-approximation for k-Median (LP-based algorithm in [30]; faster primal-dual algorithm in [58]).

(B) Solving the LP approximately instead of exactly. Typically this uses some version of the classical *Lagrangian relaxation* idea. Shahrokhi and Matula [80] gave the first approximation algorithm for multicommodity flow problems. Plotkin, Shmoys, and Tardos [85] generalized the method to the family of *packing/covering* LPs. Later, Garg and Könemann [44] and Fleischer [40] improved the running times further for flow LPs.

The technique of Lagrangian relaxation also finds applications in algorithms of type (A) above, such as in [58]. This technique solves a constrained optimization problem iteratively by combining the constraints into a single weighted constraint, and then solving the relaxed problem. The weights are then tuned based on the solution obtained. This technique has proved quite successful in designing very fast, combinatorial algorithms for various problems.

Such problem-specific algorithms for SDP are quite rare. The majority of this thesis describes how a specific Lagrangian relaxation technique, which we call the Multiplicative Weights algorithm, can be used to design fast algorithms for various optimization problems based on both LP and SDP. In the rest of this chapter, we formulate the linear and semidefinite programming problems, and then give a description of the results in this thesis.

1.1 Linear and Semidefinite Programming

In this section we formalize the linear and semidefinite programming problems. A linear program aims to optimize a linear objective function subject to linear equality and inequality constraints:

$$\begin{aligned} &\min \mathbf{c}\cdot\mathbf{x} \\ \mathbf{a}_1\cdot\mathbf{x} &\geq b_1 \\ \mathbf{a}_2\cdot\mathbf{x} &\geq b_2 \\ &\vdots \\ \mathbf{a}_m\cdot\mathbf{x} &\geq b_m \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

where $\mathbf{x} \in \mathbb{R}^n$ is a vector of variables, $\mathbf{c}, \mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m \in \mathbb{R}^n$, $b_1, b_2, \ldots, b_m \in \mathbb{R}$, and $\mathbf{x} \geq \mathbf{0}$ is notation for the condition that all coordinates of $\mathbf{x}$ are non-negative. Here, for vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, $\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^n u_i v_i$ is their inner product.

A semidefinite program has superficially the same form as a linear program, in that it has linear constraints and a linear objective function. The number of variables is $n + \binom{n}{2}$, and are written in the form of a square symmetric matrix $\mathbf{X}$. We have the additional stipulation that $\mathbf{X}$ is positive semidefinite, i.e. all its eigenvalues are non-negative. Thus, a general SDP can be written as follows:

$$\begin{aligned} \min \ \mathbf{C} \bullet \mathbf{X} & \\ \mathbf{A}_1 \bullet \mathbf{X} &\geq b_1 \\ \mathbf{A}_2 \bullet \mathbf{X} &\geq b_2 \\ &\vdots \\ \mathbf{A}_m \bullet \mathbf{X} &\geq b_m \\ \mathbf{X} &\succeq \mathbf{0} \end{aligned}$$

where $\mathbf{X} \in \mathbb{R}^{n \times n}$ is a square matrix of variables, $\mathbf{C}, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m \in \mathbb{R}^{n \times n}$, $b_1, b_2, \ldots, b_m \in \mathbb{R}$. We assume that all matrices are symmetric. Thus $\mathbf{X}$ has real eigenvalues, and $\mathbf{X} \succeq \mathbf{0}$ is notation for the condition that $\mathbf{X}$ is positive semidefinite. Here, for matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times n}$, $\mathbf{U} \bullet \mathbf{V} = \sum_{i=1}^n \sum_{j=1}^n U_{ij} V_{ij}$ is the inner product of the matrices thinking of them as vectors in $\mathbb{R}^{n^2}$.

A point of notation: in this thesis, bold lowercase letters such as $\mathbf{a}, \mathbf{b}, \mathbf{x}, \mathbf{v}$ represent vectors, and bold uppercase letters such as $\mathbf{A}, \mathbf{B}, \mathbf{X}, \mathbf{V}$ represent matrices. Variables written in normal weight font represent scalars, for e.g. x is a scalar variable, a_i is the i^{th} coordinate of the vector $\mathbf{a}$, and A_{ij} is the ij^{th} element of the matrix $\mathbf{A}$. Frequently, in summations where the index runs over the entire range, we will drop the specification of the range when it is clear from the context: for example, $\sum_i$ may be used to stand for $\sum_{i=1}^n$, and $\sum_{ij}$ may be used to stand for $\sum_{i=1}^n \sum_{j=1}^n$.

1.2 Results in this thesis

The focus of this thesis is the design of efficient algorithms using a specific Lagrangian relaxation technique which we call the Multiplicative Weights method. This method is certainly not new, and has been discovered and rediscovered in widely different areas. We now discuss the results of this thesis in some detail.

The Multiplicative Weights algorithm and Applications. In Chapter 2, we present a framework for the Multiplicative Weights algorithm that unifies all previously known applications of this method. This framework is based on an online player who needs to take one of several courses of actions in the face of uncertainty. Once the action is chosen, the cost of all the actions is revealed, and the player incurs the cost of her chosen action. The player's long term goal is to minimize her cost relative the cost of the best fixed action in hindsight. The Multiplicative Weights algorithm allows her to do that. She chooses her action based on some weights on the actions, and iteratively penalizes actions which have high cost by reducing their weight by a carefully chosen multiplicative factor. This simple idea is powerful enough to give efficient algorithms for a wide variety of problems. We present some of these applications, including solving feasibility problems with concave constraints on convex domains, and solving zero-sum games.

The Matrix Multiplicative Weights algorithm and Applications. In Chapter 3, we present a matrix generalization of the Multiplicative Weights algorithm. This algorithm makes use of the matrix exponential operation to replace the multiplicative weights update of the basic algorithm. We call this the Matrix Multiplicative Weights algorithm. This retains the original Multiplicative Weights algorithm as a special case when all the matrices involved are diagonal. In an analogous manner to the Multiplicative Weights algorithm, we present an application of this algorithm to solving a matrix generalization of a zero-sum game, and thus obtain a min-max theorem which is equivalent to semidefinite programming duality. This connection to semidefinite programming paves the way to the applications in the following chapter.

Combinatorial, Primal-Dual Approach to Semidefinite Programs. The first important application of the Matrix Multiplicative Weights algorithm appears in Chapter 4. SDPs satisfy a duality theorem just like LPs, and so in principle one should be able to solve them using primal-dual (type (A)) approaches. However, several conceptual difficulties arise, and no primal-dual algorithms for SDP were known so far. One issue is that the basic object in SDPs is a positive semidefinite matrix, rather than vectors as in LPs, and it is harder to reason about matrices than vectors. Another issue is that most rounding algorithms for SDPs use the global geometric structure of optimum or near-optimum solutions, and it is unclear how to use this geometric structure in the context of the grossly infeasible solutions one might encounter during a primal-dual algorithm. Finally, there is the issue of implementing matrix operations efficiently enough so that the running time is an improvement over interior point methods.

In Chapter 4, we show how the Matrix Multiplicative Weights algorithm gives us a generic primal-dual scheme for solving SDPs. This yields primal-dual algorithms for various SDPs. We also show how to implement our algorithms efficiently, by making a subtle use of the Johnson-Lindenstrauss lemma to reduce the expensive operation of computing matrix exponentials to matrix-vector products.

Thus, we obtain the fastest known algorithms for obtaining $O(\sqrt{\log n})$ and $O(\log n)$ approximations to the Sparsest Cut and minimum c-Balanced Separator problems in both undirected and directed, weighted graphs. These problems ask for a partition of the input graph into two large pieces while minimizing the size of the "interface" between them, as measured by the number of edges crossing the partition. Graph partitions or separators are central objects of study in the theory of Markov chains, geometric embeddings and are a natural algorithmic primitive in numerous settings, including clustering, divide and conquer algorithms, PRAM emulation, VLSI layout, and packet routing in distributed networks. Since finding optimal separators is **NP**-hard, one is forced to settle for approximation algorithms (see [89])

We also obtain the fastest known algorithms for obtaining an $O(\sqrt{\log n})$ approximation to the Min UnCut problem, and an $O(\sqrt{\log n})$ approximation to the Min 2CNF Deletion problem. The Min 2CNF Deletion problem is of particular interest since it is the hardest Min CSP problem that has nontrivial approximation guarantees. Khanna, Sudan, Trevisan and Williamson [66] classified the approximability of all Min CSP problems and both Min UnCut and Min 2CNF Deletion are complete problems for classes of Min CSP problems in their hierarchy. Our algorithms significantly improve upon previously known algorithms which are based on solving the SDP using interior point methods.

Derandomization and Quantum Algorithms. In Chapter 5, we describe some additional applications of the Matrix Multiplicative Weights algorithm to derandomization and quantum computing. The first result is a derandomization of the Alon-Roichman theorem [9], which says that the Cayley graph generated using $O(\log n)$ randomly chosen elements of an arbitrary group of order n is an expander with high probability. We show how to use the Matrix Multiplicative Weights algorithm to obtain an $\tilde{O}(n^3)$ time deterministic algorithm which chooses a set of $O(\log n)$ generators such that the Cayley graph thus formed is guaranteed to be an expander.

The second application is a deterministic $O(\log n)$ approximation to the Quantum Hypergraph Cover problem. This problem is a generalization of the classical Set Cover problem and it arises in quantum information theory. It was shown by Ahlswede and Winter [4] that a $O(\log n)$ approximation to the problem can be found by solving an associated SDP and randomly rounding the fractional solution thus obtained. We derandomize this process using the Matrix Multiplicative Weights algorithm, and obtain an $\tilde{O}(n^3)$ time algorithm to find an $O(\log n)$ approximate cover. This algorithm can be viewed as a generalization of the greedy algorithm for the Set Cover problem.

The third application is to a learning problem in quantum mechanics. An n-qubit quantum state is described by a positive semidefinite matrix of trace 1 of size $2^n \times 2^n$. Suppose we could repeatedly measure the quantum state, and now we want to build an approximate description of it (in terms of another quantum state). To learn the state exactly we may need to do exponentially many (in n) measurements, but if our goal is to learn the state well enough to have reasonable confidence about its behavior in future measurements, then we need to do only $O(n)$ measurements. This result was shown by Aaronson [1] by bounding a learning theoretic parameter called the fat-shattering dimension of quantum states by $O(n)$. Just like the VC-dimension, this parameter allows us to place bounds on the sample complexity of a learning problem in a relevant PAC-style framework. Using the Matrix Multiplicative Weights algorithm, we give an alternative proof of Aaronson's result.

Fast Algorithms for Approximate Semidefinite Programming. We then turn back to semidefinite programming in Chapter 6. This chapter describes how to apply the (basic) Multiplicative Weights algorithm to approximate SDPs. The algorithms thus obtained are of type (B). Our algorithm reduces SDP solving to a sequence of approximate eigenvalue/eigenvector computations, which can be done efficiently using the well-known iterative methods such as the power method or the more efficient Lanczos method. This generalizes the work of Klein and Lu [67] who had previously applied the Multiplicative Weights algorithm to approximately solve SDPs that arose in the algorithms of Goemans-Williamson for MaxCut and Karger, Motwani, and Sudan for Coloring. We make this approach more efficient by making judicious use of the ellipsoid algorithm in a hybrid two-level algorithm, and we further speed up the process of computing approximate eigenvectors by using a sparsification procedure for reducing the number of non-zero entries in the input matrix. Using this approach, we give significantly faster algorithms for approximately solving SDPs which arise in quadratic programming problem called MaxQP (which generalizes MaxCut), in a biological probability estimation problem called HaploFreq considered by Halperin and Hazan [51], and in computing the minimum distortion embedding of a given finite metric space into ℓ_2.

Fast Graph Partitioning algorithms using Expander Flows. Finally, in Chapter 7, we again consider the problem of approximating the Sparsest Cut and minimum c-Balanced Separator problems in undirected weighted graphs, which we first considered in Chapter 4. We give an alternative algorithm for approximating these problems up to a factor of $O(\sqrt{\log n})$ in $\tilde{O}(n^2)$ time. This approach is different from the one in Chapter 4 in that we do not solve the associated SDP; though the formulation in terms of expander flows is indeed inspired by the SDP dual. Instead, we use the Multiplicative Weights algorithm to embed an expander flow, which is a multicommodity flow whose demand graph is an expander, in the graph. The expander flow gives a lower bound on the expansion of the input graph. Then using the theorems of [18], we can find a cut of expansion within a factor of $O(\sqrt{\log n})$ of the lower bound thus obtained, which thus gives us an $O(\sqrt{\log n})$ factor approximation of the Sparsest Cut. The algorithm uses a combination of multicommodity flow computations, eigenvalue computations, and random sampling to achieve the $\tilde{O}(n^2)$ running time.

At the core of all these algorithms is the Multiplicative Weights algorithm, and this is the common theme which runs through all the results in this thesis.

Chapter 2

The Multiplicative Weights Update method

The *Multiplicative Weights method* is a simple idea which has been repeatedly discovered in fields as diverse as Machine Learning, Optimization, and Game Theory. The setting for this algorithm is the following. A decision maker has a choice of n decisions, and needs to repeatedly make a decision and obtain an associated payoff. The decision maker's goal, in the long run, is to achieve a total payoff which is comparable to the payoff of that fixed decision that maximizes the total payoff with the benefit of hindsight. While this best decision may not be known *a priori*, it is still possible to achieve this goal by maintaining weights on the decisions, and choosing the decisions randomly with probability proportional to the weights. In each successive round, the weights are updated by multiplying them with factors which depend on the payoff of the associated decision in that round. Intuitively, this scheme works because it tends to focus higher weight on higher payoff decisions in the long run.

This idea lies at the core of a variety of algorithms. Some examples include: Freund and Schapire's AdaBoost algorithm in machine learning [42]; algorithms for game playing studied in economics (see Section 2.4), the Plotkin-Shmoys-Tardos algorithm for packing and covering LPs [85], and its improvements in the case of flow problems by Garg-Könneman [44] and Fleischer [40]; etc. The analysis of the running time uses a potential function argument and the final running time is proportional to $1/\varepsilon^2$.

It has been clear to most researchers that these results are very similar, see for instance, Khandekar's PhD thesis [64]. In this chapter, we develop a unified framework for all these algorithms. This meta algorithm is a generalization of Littlestone and Warmuth's *Weighted Majority* algorithm from learning theory [78]. We call this the Multiplicative Weights algorithm (a similar algorithm, *Hedge*, was developed by Freund and Schapire [42]). This algorithmic framework, and the derivation of previously known algorithms using it, have been studied in much more detail in the survey paper [15]. We also present some applications of this framework in designing algorithms to approximately solve zero-sum games, feasibility problems with concave constraints over a convex domain, and fractional packing and covering linear programs.

2.1 The Weighted Majority Algorithm

Consider the following setting. We are trying to invest in a certain stock. For simplicity, think of its price movements as a sequence of binary events: up/down. (Below, this will be generalized to allow non-binary events.) Each morning we try to predict whether the price will go up or down that day; if our prediction happens to be wrong we lose a dollar that day.

In making our predictions, we are 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 algorithm we present will be able to limit its losses to roughly the same as the *best* of these experts. 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.

The algorithm does this by maintaining a *weighting* of the experts. Initially all have equal weight. As time goes on, some experts are seen as making better predictions than others, and the algorithm increases their weight proportionately. The Weighted Majority algorithm is given in Figure 2.1.

Weighted majority algorithm

Initialization: Fix an $\varepsilon \leq \frac{1}{2}$. For each expert i, associate the weight ${w_i}^{(1)} := 1$.
For $t = 1, 2, \ldots, T$:

1. Make the prediction that is the weighted majority of the experts' predictions based on the weights ${w_1}^{(t)}, \ldots, {w_n}^{(t)}$. That is, predict "up" or "down" depending on which prediction has a higher total weight of experts advising it (breaking ties arbitrarily).

2. For every expert i who predicts wrongly, decrease his weight for the next round by multiplying it by a factor of $(1-\varepsilon)$:

$$ {w_i}^{(t+1)} = (1-\varepsilon){w_i}^{(t)} \qquad \text{(update rule)}. \tag{2.1} $$

Figure 2.1: The Weighted Majority algorithm.

Theorem 1. *After T steps, let ${m_i}^{(T)}$ be the number of mistakes of expert i and $m^{(T)}$ be the number of mistakes our algorithm has made. Then we have the following bound for every i:*

$$ m^{(T)} \leq \frac{2\ln n}{\varepsilon} + 2(1+\varepsilon){m_i}^{(T)}. $$

In particular, this holds for i which is the best expert, i.e. having the least ${m_i}^{(T)}$.

Proof: A simple induction shows that ${w_i}^{(t+1)} = (1-\varepsilon)^{{m_i}^{(t)}}$. Let $\Phi^{(t)} = \sum_i {w_i}^{(t)}$ ("the potential function"). Thus $\Phi^{(1)} = n$. Each time we make a mistake, the weighted majority of experts also made a mistake, so at least half the total weight decreases by a factor $1-\varepsilon$. Thus, the potential function decreases by a factor of at least $(1-\varepsilon/2)$:

$$\Phi^{(t+1)} \;\leq\; \Phi^{(t)}\left(\frac{1}{2}+\frac{1}{2}(1-\varepsilon)\right) \;=\; \Phi^{(t)}(1-\varepsilon/2).$$

Thus another simple induction gives $\Phi^{(T+1)} \leq n(1-\varepsilon/2)^{m^{(T)}}$. Finally, since $\Phi_i{}^{(T+1)} \geq w_i{}^{(T+1)}$ for all i, the claimed bound follows by comparing the above two expressions and using the fact that $-\ln(1-\varepsilon) \leq \varepsilon+\varepsilon^2$ since $\varepsilon < \frac{1}{2}$. $\square$

The beauty of this analysis is that it makes no assumption about the sequence of events: they could be arbitrarily correlated and could even depend upon our current weighting of the experts. In this sense, this algorithm delivers more than initially promised, and this lies at the root of why (after generalization) it can give rise to the diverse algorithms mentioned earlier. In particular, the scenario where the events are chosen adversarially resembles a zero-sum game, which we consider later in Section 2.3.1.

2.2 The Multiplicative Weights algorithm

In the general setting, we still have n experts. The set of events/outcomes may not be necessarily binary and could even be infinite. To model this, we dispense with the notion of predictions altogether, and instead suppose that in each round, every expert recommends a course of action, and our task is to pick an expert and use his advice. At this point the costs of all actions recommended by the experts is revealed by nature. We suffer the cost of the action recommended by the expert we chose.

To motivate the Multiplicative Weights algorithm, consider the naïve strategy that, in each iteration, simply picks an expert at random. The expected penalty will be that of the "average" expert. Suppose now that a few experts clearly outperform their competitors. This is easy to spot as events unfold, and so it is sensible to reward them by increasing their probability of being picked in the next round (hence the multiplicative weight update rule).

Intuitively, being in complete ignorance about the experts at the outset, we select them uniformly at random for advice. This maximum entropy starting rule reflects our ignorance. As we learn who the hot experts are and who the duds are, we lower the entropy to reflect our increased knowledge. The multiplicative weight update is our means of skewing the distribution.

We now set up some notation. Let $t = 1, 2, \ldots, T$ denote the current round, and let i be a generic expert. In each round t, we select a distribution $\mathbf{p}^{(t)}$ over the set of experts, and select an expert i randomly from it (and use his advised course of action). At this point, the costs of all the actions recommended by the experts are revealed by nature in the form of the vector $\mathbf{m}^{(t)}$ such that expert i incurs cost $m_i{}^{(t)}$. We assume that the costs lie in the range $[-1, 1]$. This is the only assumption we make on the costs; nature is completely free to choose the cost vector as long as these bounds are respected, even with full knowledge of the actions recommended by the experts.

The expected cost to the algorithm for choosing the distribution $\mathbf{p}^{(t)}$ is

$$\mathop{\mathbb{E}}_{i \in \mathbf{p}^{(t)}}[m_i{}^{(t)}] \;=\; \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}.$$

The total expected cost over all rounds is therefore $\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. Just as before, our goal is to design an algorithm which achieves a total expected cost not too much more than the cost of the best expert, viz. $\min_i \sum_{t=1}^{T} m_i{}^{(t)}$.

Multiplicative Weights algorithm

Initialization: Fix an $\varepsilon \leq \frac{1}{2}$. For each expert i, associate the weight $w_i{}^{(t)} := 1$.
For $t = 1, 2, \ldots, T$:

1. Choose expert i with probability proportional to his weight $w_i{}^{(t)}$. I.e., use the distribution $\mathbf{p}^{(t)} = \{w_1{}^{(t)}/\Phi^{(t)}, \ldots, w_n{}^{(t)}/\Phi^{(t)}\}$ where $\Phi^{(t)} = \sum_i w_i{}^{(t)}$.
2. Observe the costs of the experts $\mathbf{m}^{(t)}$.
3. Penalize the costly experts by updating their weights as follows: for every expert i,

$$w_i{}^{(t+1)} = \begin{cases} w_i{}^{(t)}(1-\varepsilon)^{m_i{}^{(t)}} & \text{if } m_i{}^{(t)} \geq 0 \\ w_i{}^{(t)}(1+\varepsilon)^{-m_i{}^{(t)}} & \text{if } m_i{}^{(t)} < 0 \end{cases}$$

Figure 2.2: The Multiplicative Weights algorithm.

The following theorem —completely analogous to Theorem 1— bounds the total expected cost of the Multiplicative Weights algorithm (given in Figure 2.2) in terms of the total cost of the best expert:

Theorem 2. *In the given setup, the Multiplicative Weights algorithm guarantees that after T rounds, for any expert i, we have*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; \sum_{t=1}^{T} m_i{}^{(t)} + \varepsilon \sum_{t=1}^{T} |m_i{}^{(t)}| + \frac{\ln n}{\varepsilon}.$$

Proof: We use the following facts, which follow immediately from the convexity of the exponential function:

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

The proof is along the lines of the earlier one, using the potential function $\Phi^{(t)} = \sum_i w_i{}^{(t)}$.

Since ${m_i}^{(t)} \in [-1, 1]$, using the facts above we have,

$$\begin{aligned}
\Phi^{(t+1)} &= \sum_i {w_i}^{(t+1)} \\
&= \sum_{i\colon\, {m_i}^{(t)} \geq 0} {w_i}^{(t)} (1-\varepsilon)^{{m_i}^{(t)}} + \sum_{i\colon\, {m_i}^{(t)} < 0} {w_i}^{(t)} (1+\varepsilon)^{-{m_i}^{(t)}} \\
&\leq \sum_i {w_i}^{(t)} (1 - \varepsilon {m_i}^{(t)}) \\
&= \Phi^{(t)} - \varepsilon \Phi^{(t)} \sum_i {m_i}^{(t)} {p_i}^{(t)} \\
&= \Phi^{(t)} (1 - \varepsilon \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \\
&\leq \Phi^{(t)} \exp(-\varepsilon \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).
\end{aligned}$$

Here, we used the fact that ${p_i}^{(t)} = {w_i}^{(t)} / \Phi^{(t)}$. Thus, by induction, after T rounds, we have

$$\Phi^{(T+1)} \;\leq\; \Phi^{(1)} \exp(-\varepsilon \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}) \;=\; n \cdot \exp(-\varepsilon \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}).$$

Furthermore, for every expert i,

$$\Phi^{(T+1)} \geq {w_i}^{(T+1)} = (1-\varepsilon)^{\sum_{\geq 0} {m_i}^{(t)}} \cdot (1+\varepsilon)^{-\sum_{<0} {m_i}^{(t)}},$$

where the subscripts "≥ 0" and "< 0" in the summations refer to the rounds t where ${m_i}^{(t)}$ is ≥ 0 and < 0 respectively. Now we get the desired bound by taking logarithms and simplifying as before. We used the facts that $\ln(\frac{1}{1-\varepsilon}) \leq \varepsilon + \varepsilon^2$ and $\ln(1+\varepsilon) \geq \varepsilon - \varepsilon^2$ for $\varepsilon \leq \frac{1}{2}$. □

Remark: From the proof, it can be seen that the following multiplicative update rule:

$${w_i}^{(t+1)} \;=\; {w_i}^{(t)} (1 - \varepsilon {m_i}^{(t)})$$

regardless of the sign of ${m_i}^{(t)}$, would also give the same bounds. Such a rule may be easier to implement.

Corollary 1. *If the costs of all experts lie in the range* $[0, 1]$*, then the Multiplicative Weights algorithm also guarantees that after* T *rounds, for any distribution* $\mathbf{p}$ *on the experts,*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;\leq\; (1+\varepsilon) \sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p} + \frac{\ln n}{\varepsilon}.$$

Proof: This corollary follows immediately from Theorem 2, by taking a convex combination of the inequalities for all experts i with the distribution $\mathbf{p}$. □

2.2.1 Gains instead of losses

There are situations where it makes more sense for the vector $\mathbf{m}^{(t)}$ to specify *gains* for each expert rather than losses. Now our goal is to get as much total expected payoff as possible in comparison to the total payoff of the best expert. We can get an algorithm for this case simply by running the Multiplicative Weights algorithm using the loss vector $-\mathbf{m}^{(t)}$.

The algorithm that results updates the weight of expert i by a factor of $(1+\varepsilon)^{m_i{}^{(t)}}$ when $m_i{}^{(t)} \geq 0$, and $(1-\varepsilon)^{-m_i{}^{(t)}}$ when $m_i{}^{(t)} < 0$. The following theorem follows directly from Theorem 2 by simply negating the quantities:

Theorem 3. *In the given setup, the Multiplicative Weights algorithm (for gains) guarantees that after T rounds, for any expert i, we have*

$$\sum_{t=1}^{T} \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \ \geq \ \sum_{t=1}^{T} m_i{}^{(t)} - \varepsilon \sum_{t=1}^{T} |m_i{}^{(t)}| - \frac{\ln n}{\varepsilon}.$$

2.3 Applications

Typically, the Multiplicative Weights method is applied in the following manner. A prototypical example is to solve a constrained optimization problem. We then let an expert represent each constraint in the problem, and the events correspond to points in the domain of interest. The penalty of the expert is made proportional to *how well* the corresponding constraint is satisfied on the point represented by an event. This might seem counterintuitive, but recall that we *reduce* an expert's weight depending on his penalty, and if an expert's constraint is well satisfied on events so far we would like his weight to be smaller, so that the algorithm focuses on experts whose constraints are poorly satisfied. With these weights, the algorithm generates a *maximally adversarial* event, i.e. the event whose corresponding point maximizes the expected penalty, i.e. the weighted sum of penalties. With this intuition, we can describe the following applications.

2.3.1 Solving zero-sum games approximately

We show how the general algorithm above can be used to approximately solve zero-sum games. This is a duplication of the results of Freund and Schapire [43], who gave the same algorithm but a different proof of convergence that used KL-divergence.

Let $\mathbf{A}$ be the payoff matrix of a finite 2-player zero-sum game, with n rows (the number of columns will play no role). When the row player plays strategy i and the column player plays strategy j, then the payoff to the column player is $A(i,j) := A_{ij}$. We assume that $A(i,j) \in [0,1]$. If the row player chooses his strategy i from a distribution $\mathbf{p}$ over the rows, then the expected payoff to the column player for choosing a strategy j is $A(\mathbf{p},j) := \mathbb{E}_{i\in\mathbf{p}}[A(i,j)]$. Thus, the best response for the column player is the strategy j which maximizes this payoff. Similarly, if the column player chooses his strategy j from a distribution $\mathbf{q}$ over the columns, then the expected payoff he gets if the row player chooses the strategy i is $A(i, \mathbf{q}) := \mathbb{E}_{j \in \mathbf{q}}[A(i,j)]$. Thus, the best response for the row player is the strategy i which minimizes this payoff. John von Neumann's min-max theorem says that if each of the players chooses a distribution over their strategies to optimize their worst case payoff (or payout), then the value they obtain is the same:

$$\min_{\mathbf{p}} \max_{j} A(\mathbf{p}, j) \;=\; \max_{\mathbf{q}} \min_{i} A(i, \mathbf{q}) \tag{2.2}$$

where $\mathbf{p}$ (resp., $\mathbf{q}$) varies over all distributions over rows (resp., columns). Also, i (resp., j) varies over all rows (resp., columns). The common value of these two quantities, denoted λ^*, is known as the value of the game.

Let $\delta > 0$ be an error parameter. We wish to approximately solve the zero-sum game up to additive error of δ, namely, find mixed row and column strategies $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ such that

$$\lambda^* - \delta \;\le\; \min_{i} A(i, \tilde{\mathbf{q}}) \tag{2.3}$$

$$\max_{j} A(\tilde{\mathbf{p}}, j) \;\le\; \lambda^* + \delta. \tag{2.4}$$

The algorithmic assumption about the game is that given any distribution $\mathbf{p}$ on experts, we have an efficient way to pick the best event, namely, the pure column strategy j that maximizes $A(\mathbf{p}, j)$. This quantity is at least λ^* from the definition above. Call this algorithm the Oracle.

Theorem 4. *Given an error parameter $\delta > 0$, there is an algorithm which solves the zero-sum game up to an additive factor of δ using $O(\frac{\log n}{\delta^2})$ calls to* Oracle*, with an additional processing time of $O(n)$ per call.*

Proof: We map our general algorithm from Section 2.2 to this setting by considering (2.3) as specifying n linear constraints on the probability vector $\tilde{\mathbf{q}}$: viz., for all rows i, $A(i, \tilde{\mathbf{q}}) \ge \lambda^* - \delta$. Now, following the intuition given in the beginning of this section, we make our "experts" to correspond to pure strategies of the row player. Thus a distribution on the experts corresponds to a mixed row strategy. "Events" correspond to pure strategies of the column player. The penalty paid by an expert i when an event j happens is $A(i, j)$.

In each round, given a distribution $\mathbf{p}^{(t)}$ on the rows, we will set the event $j^{(t)}$ to be the best response strategy to $\mathbf{p}^{(t)}$ for the column player, by calling Oracle. Thus, the cost vector $\mathbf{m}^{(t)}$ is the $j^{(t)}$-th column of the matrix $\mathbf{A}$.

Since all $A(i, j) \in [0, 1]$, we can apply Corollary 1 to get that after T rounds, for any distribution on the rows $\mathbf{p}$, we have

$$\sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\le\; (1+\varepsilon) \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \frac{\ln n}{\varepsilon}.$$

Dividing by T, and using the fact that $A(\mathbf{p}, j^{(t)}) \le 1$ and that for all t, $A(\mathbf{p}^{(t)}, j^{(t)}) \ge \lambda^*$, we get

$$\lambda^* \;\le\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \;\le\; \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \varepsilon + \frac{\ln n}{\varepsilon T}$$

Setting $\mathbf{p} = \mathbf{p}^*$, the optimal row strategy, we have $A(\mathbf{p}, j) \leq \lambda^*$ for any j. By setting $\varepsilon = \frac{\delta}{2}$ and $T = \lceil \frac{4 \ln n}{\delta^2} \rceil$, we get that

$$\lambda^* \ \leq \ \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \ \leq \ \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}, j^{(t)}) + \delta \ \leq \ \lambda^* + \delta. \tag{2.5}$$

Thus, $\frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)})$ is an (additive) δ-approximation to λ^*.

Let $\tilde{t}$ be the round t with the minimum value of $A(\mathbf{p}^{(t)}, j^{(t)})$. We have, from (2.5),

$$A(\mathbf{p}^{(\tilde{t})}, j^{(\tilde{t})}) \ \leq \ \frac{1}{T} \sum_{t=1}^{T} A(\mathbf{p}^{(t)}, j^{(t)}) \ \leq \ \lambda^* + \delta.$$

Since $j^{(\tilde{t})}$ maximizes $A(\mathbf{p}^{(\tilde{t})}, j)$ over all j, we conclude that $\mathbf{p}^{(\tilde{t})}$ is an approximately optimal mixed row strategy, and thus we can set $\mathbf{p}^* := \mathbf{p}^{(\tilde{t})}$. [1]

We set $\mathbf{q}^*$ to be the distribution which assigns to column j the probability $\frac{|\{t:\ j^{(t)} = j\}|}{T}$. From (2.5), for any row strategy i, by setting $\mathbf{p}$ to be concentrated on the pure strategy i, we have

$$\lambda^* - \delta \ \leq \ \frac{1}{T} \sum_{t=1}^{T} A(i, j^{(t)}) \ = \ A(i, \mathbf{q}^*)$$

which shows that $\mathbf{q}^*$ is an approximately optimal mixed column strategy. $\square$

2.3.2 Approximating Linear Feasibility Programs on Convex Domains

Plotkin, Shmoys, and Tardos [85] generalized some known flow algorithms to a framework for approximately solving *fractional packing and covering* problems. Their algorithm is a quantitative version of the classical *Lagrangian relaxation* idea, and applies also to general linear programs. Below, we derive the algorithm for convex programs which can be stated as trying to find a point in a convex domain satisfying a number of linear inequalities. We will then mention the slight modification that yields better running time for fractional packing and covering LPs.

The basic problem is to check the feasibility of the following convex program:

$$\exists ? \mathbf{x} \ \in \ \mathcal{P} : \quad \mathbf{A}\mathbf{x} \ \geq \ \mathbf{b} \tag{2.6}$$

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x} \in \mathbb{R}^n$, and $\mathcal{P}$ is a convex set in $\mathbb{R}^n$. Intuitively, the set $\mathcal{P}$ represents the "easy" constraints to satisfy, such as non-negativity, and $\mathbf{A}$ represents the "hard" constraints to satisfy.

We wish to design an algorithm that given an error parameter $\delta > 0$, either solves the problem to an additive error of δ, i.e., finds an $\mathbf{x} \in P$ such that for all i, $\mathbf{A}_i \mathbf{x} \geq b_i - \delta$, or failing that, proves that the system is infeasible. Here, $\mathbf{A}_i$ is the i^{th} row of $\mathbf{A}$.

[1] Alternatively, we can set $\mathbf{p}^* = \frac{1}{T} \sum_t \mathbf{p}^{(t)}$. For let j^* be the optimal column player response to $\mathbf{p}^*$. Then we have $A(\mathbf{p}^*, j^*) = \frac{1}{T} \sum_t A(\mathbf{p}^{(t)}, j^*) \leq \frac{1}{T} \sum_t A(\mathbf{p}^{(t)}, j^{(t)}) \leq \lambda^* + \delta$.

We assume the existence of an algorithm, called ORACLE, which, given a probability vector $\mathbf{p}$ on the m constraints, solves the following feasibility problem:

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad \mathbf{p}^\top \mathbf{A}\mathbf{x} \ \geq \ \mathbf{p}^\top \mathbf{b} \tag{2.7}$$

It is reasonable to expect such an optimization procedure to exist (indeed, such is the case for many applications) since we only need to check the feasibility of one constraint rather than m. If the feasibility problem (2.6) has a solution $\mathbf{x}^*$, then the same solution also satisfies (2.7) for *any* probability vector $\mathbf{p}$ over the constraints. Thus, if there is a probability vector $\mathbf{p}$ over the constraints such that no $\mathbf{x} \in \mathcal{P}$ satisfies (2.7), then it is proof that the original problem is infeasible.

We assume that the ORACLE satisfies the following technical condition, which is necessary for deriving running time bounds:

Definition 1. *An (ℓ, ρ)*-**bounded** ORACLE, *for parameters $0 \leq \ell \leq \rho$, is an algorithm which given a probability vector $\mathbf{p}$ over the constraints, solves the feasibility problem (2.7). Furthermore, there is a fixed subset $I \subseteq [m]$ of constraints such that whenever the* ORACLE *manages to find a point $\mathbf{x} \in \mathcal{P}$ satisfying (2.7), the following holds:*

$$\begin{aligned} \forall i \in I: \quad \mathbf{A}_i\mathbf{x} - b_i \ &\in \ [-\ell, \rho] \\ \forall i \notin I: \quad \mathbf{A}_i\mathbf{x} - b_i \ &\in \ [-\rho, \ell] \end{aligned}$$

The value ρ is called the **width** *of the problem.*

In previous work, such as [85], only (ρ, ρ)-bounded ORACLEs are considered. We separate out the upper and lower bounds in order to obtain tighter guarantees on the running time. The results of [85] can be recovered simply by setting $\ell = \rho$.

Theorem 5. *Let $\delta > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded* ORACLE *for the feasibility problem (2.6). Assume that $\ell \geq \frac{\delta}{2}$. Then there is an algorithm which either solves the problem up to an additive error of δ, or correctly concludes that the system is infeasible, making only $O(\frac{\ell\rho\log(m)}{\delta^2})$ calls to the* ORACLE, *with an additional processing time of $O(m)$ per call.*

PROOF: The condition $\ell \geq \frac{\delta}{2}$ is only technical, and if it is not met we can just redefine ℓ to be $\frac{\delta}{2}$. To map our general framework to this situation, we have an expert representing each of the m constraints. Events correspond to vectors $\mathbf{x} \in \mathcal{P}$. The loss of the expert corresponding to constraint i for event $\mathbf{x}$ is $\frac{1}{\rho}[\mathbf{A}_i\mathbf{x} - b_i]$ (so that the costs lie in the range $[-1, 1]$).

In each round t, given a distribution over the experts (i.e. the constraints) $\mathbf{p}^{(t)}$, we run the ORACLE with $\mathbf{p}^{(t)}$. If the ORACLE declares that there is no $\mathbf{x} \in \mathcal{P}$ such that ${\mathbf{p}^{(t)}}^\top \mathbf{A}\mathbf{x} \geq {\mathbf{p}^{(t)}}^\top \mathbf{b}$, then we stop, because now $\mathbf{p}^{(t)}$ is proof that the problem (2.6) is infeasible.

So let us assume that this doesn't happen, i.e. in all rounds t, the ORACLE manages to find a solution $\mathbf{x}^{(t)}$ such ${\mathbf{p}^{(t)}}^\top \mathbf{A}\mathbf{x} \geq {\mathbf{p}^{(t)}}^\top \mathbf{b}$. Since the cost vector to the Multiplicative

Weights algorithm is specified to be $\mathbf{m}^{(t)} := \frac{1}{\rho}[\mathbf{A}\mathbf{x}^{(t)} - \mathbf{b}]$, we conclude that the expected cost in each round is non-negative:

$$\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \;=\; \frac{1}{\rho}[\mathbf{A}\mathbf{x}^{(t)} - \mathbf{b}] \cdot \mathbf{p}^{(t)} \;=\; \frac{1}{\rho}[\mathbf{p}^{(t)\top}\mathbf{A}\mathbf{x} - \mathbf{p}^{(t)\top}\mathbf{b}] \;\geq\; 0.$$

Let $i \in I$. Then Theorem 2 tells us that after T rounds,

$$\begin{aligned} 0 \;&\leq\; \sum_{t=1}^{T} \frac{1}{\rho}[\mathbf{A}_i\mathbf{x}^{(t)} - b_i] + \varepsilon \sum_{t=1}^{T} \frac{1}{\rho}|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| + \frac{\ln m}{\varepsilon} \\ &=\; (1+\varepsilon)\sum_{t=1}^{T} \frac{1}{\rho}[\mathbf{A}_i\mathbf{x}^{(t)} - b_i] + 2\varepsilon \sum_{<0} \frac{1}{\rho}|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| + \frac{\ln m}{\varepsilon} \\ &\leq\; (1+\varepsilon)\sum_{t=1}^{T} \frac{1}{\rho}[\mathbf{A}_i\mathbf{x}^{(t)} - b_i] + \frac{2\varepsilon\ell}{\rho}T + \frac{\ln m}{\varepsilon} \end{aligned}$$

Here, the subscript "< 0" refers to the rounds t when $\mathbf{A}_i\mathbf{x}^{(t)} - b_i < 0$. The last inequality follows because if $\mathbf{A}_i\mathbf{x}^{(t)} - b_i < 0$, then $|\mathbf{A}_i\mathbf{x}^{(t)} - b_i| \leq \ell$. Dividing by T, multiplying by ρ, and letting $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^{(t)}$ (note that $\bar{\mathbf{x}} \in \mathcal{P}$ since $\mathcal{P}$ is a convex set), we get that

$$0 \;\leq\; (1+\varepsilon)[\mathbf{A}_i\bar{\mathbf{x}} - b_i] + 2\varepsilon\ell + \frac{\rho\ln(m)}{\varepsilon T}.$$

Now, if we choose $\varepsilon = \frac{\delta}{4\ell}$ (note that $\varepsilon \leq \frac{1}{2}$ since $\ell \geq \frac{\delta}{2}$), and $T = \lceil\frac{8\ell\rho\ln(m)}{\delta^2}\rceil$, we get that

$$0 \;\leq\; (1+\varepsilon)[\mathbf{A}_i\bar{\mathbf{x}} - b_i] + \delta \quad\Longrightarrow\quad \mathbf{A}_i\bar{\mathbf{x}} \;\geq\; b_i - \delta.$$

Reasoning similarly for $i \notin I$, we get the same inequality. Putting both together, we conclude that $\bar{\mathbf{x}}$ satisfies the feasibility problem (2.6) up to an additive δ factor, as desired. $\square$

Concave constraints

The algorithm of Section 2.3.2 works not just for linear constraints over a convex domain, but also for concave constraints. Imagine that we have the following feasibility problem:

$$\exists ? \mathbf{x} \in \mathcal{P} : \quad \forall i \in [m] : \; f_i(\mathbf{x}) \geq 0 \tag{2.8}$$

where, as before, $\mathcal{P} \in \mathbb{R}^n$ is a convex domain, and for $i \in [m]$, $f_i : \mathcal{P} \to \mathbb{R}$ are concave functions. We wish to satisfy this system approximately, up to an additive error of δ. Again, we assume the existence of an ORACLE, which, when given a probability distribution $\mathbf{p} = \langle p_1, p_2, \ldots, p_m\rangle^\top$, solves the following feasibility problem:

$$\exists ? \mathbf{x} \in \mathcal{P} : \quad \sum_i p_i f_i(\mathbf{x}) \geq 0 \tag{2.9}$$

An ORACLE would be called (ℓ, ρ)-bounded there is a fixed subset of constraints $I \subseteq [m]$ such that whenever it returns a feasible solution $\mathbf{x}$ to (2.9), all constraints $i \in I$ take values in the range $[-\ell, \rho]$ on the point $\mathbf{x}$, and all the rest take values in $[-\rho, \ell]$.

Theorem 6. *Let $\delta > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded* Oracle *for the feasibility problem (2.8). Assume that $\ell \geq \frac{\delta}{2}$. Then there is an algorithm which either solves the problem up to an additive error of δ, or correctly concludes that the system is infeasible, making only $O(\frac{\ell\rho\log(m)}{\delta^2})$ calls to the* Oracle, *with an additional processing time of $O(m)$ per call.*

Proof: Just as before, we have an expert for every constraint, and events correspond to $\mathbf{x} \in \mathcal{P}$. The loss of the expert corresponding to constraint i for event $\mathbf{x}$ is $\frac{1}{\rho} f_i(\mathbf{x})$.

Now we run the Multiplicative Weights algorithm with this setup. Again, if at any point the Oracle declares that (2.9) is infeasible, we immediately halt and declare the system (2.8) infeasible. So assume this never happens. Then as before, the expected cost in each round is $\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \geq 0$. Now, applying Theorem 2 as before, we conclude that for any $i \in I$, we have

$$0 \;\leq\; (1+\varepsilon)\sum_{t=1}^{T} \frac{1}{\rho} f_i(\mathbf{x}^{(t)}) + \frac{2\varepsilon\ell}{\rho}T + \frac{\ln m}{\varepsilon}.$$

Dividing by T, multiplying by ρ, and letting $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^{(t)}$ (note that $\bar{\mathbf{x}} \in \mathcal{P}$ since $\mathcal{P}$ is a convex set), we get that

$$0 \;\leq\; (1+\varepsilon) f_i(\bar{\mathbf{x}}) + 2\varepsilon\ell + \frac{\rho\ln(m)}{\varepsilon T},$$

since $\frac{1}{T}\sum_{t=1}^{T} f_i(\mathbf{x}^{(t)}) \leq f_i(\frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^{(t)})$, by Jensen's inequality, since all the f_i are concave.

Now, if we choose $\varepsilon = \frac{\delta}{4\ell}$ (note that $\varepsilon \leq \frac{1}{2}$ since $\ell \geq \frac{\delta}{2}$), and $T = \lceil \frac{8\ell\rho\ln(m)}{\delta^2} \rceil$, we get that

$$0 \;\leq\; (1+\varepsilon) f_i(\bar{\mathbf{x}}) + \delta \quad\Longrightarrow\quad f_i(\bar{\mathbf{x}}) \;\geq\; -\delta.$$

Reasoning similarly for $i \notin I$, we get the same inequality. Putting both together, we conclude that $\bar{\mathbf{x}}$ satisfies the feasibility problem (2.8) up to an additive δ factor, as desired. $\square$

Approximate Oracles

The algorithm described in the previous section allows some slack for the implementation of the Oracle. This slack is very useful in designing efficient implementations for the Oracle.

Define a δ-approximate Oracle for the feasibility problem (2.6) to be one that solves the feasibility problem (2.7) up to an additive error of δ. That is, given a probability vector $\mathbf{p}$ on the constraints, either it finds an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{p}^\top \mathbf{A}\mathbf{x} \geq \mathbf{p}^\top \mathbf{b} - \delta$, or it declares correctly that (2.7) is infeasible.

Theorem 7. *Let $\delta > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded $\frac{\delta}{3}$-approximate* Oracle *for the feasibility problem (2.6). Assume that $\ell \geq \frac{\delta}{3}$. Then there is an algorithm which either solves the problem up to an additive error of δ, or correctly concludes that the system is infeasible, making only $O(\frac{\ell\rho\log(m)}{\delta^2})$ calls to the* Oracle, *with an additional processing time of $O(m)$ per call.*

PROOF: We run the algorithm of the previous section with the given ORACLE, setting $\varepsilon = \frac{\delta}{6\ell}$. Now, in every round, the expected payoff is at least $-\frac{\delta}{3\rho}$. Simplifying as before, we get that after T rounds, we have, the average point $\bar{\mathbf{x}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}^{(t)}$ returned by the ORACLE satisfies

$$-\frac{\delta}{3} \;\leq\; (1+\varepsilon)[\mathbf{A}_i\bar{\mathbf{x}} - b_i] + 2\varepsilon\ell + \frac{\rho\ln(m)}{\varepsilon T}.$$

Now, if $T = \lceil\frac{18\ell\rho\ln(m)}{\delta^2}\rceil$, then we get that for all i, $\mathbf{A}_i\bar{\mathbf{x}} \geq b_i - \delta$, as required. □

Fractional Covering Problems

In fractional covering problems, the framework is the same as above, with the crucial difference that the coefficient matrix $\mathbf{A}$ is such that $\mathbf{Ax} \geq 0$ for all $\mathbf{x} \in \mathcal{P}$, and $\mathbf{b} > 0$. A δ-approximation solution to this system is an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{Ax} \geq (1-\delta)\mathbf{b}$.

We assume without loss of generality (by appropriately scaling the inequalities) that $b_i = 1$ for all rows, so that now we desire to find an $\mathbf{x} \in \mathcal{P}$ which satisfies the system within an additive δ factor. Since for all $\mathbf{x} \in \mathcal{P}$, we have $\mathbf{Ax} \geq 0$, and since all $b_i = 1$, we conclude that for any i, $\mathbf{A}_i\mathbf{x} - b_i \geq -1$. Thus, we assume that there is a $(1,\rho)$-bounded ORACLE for this problem. Now, applying Theorem 5, we get the following:

Theorem 8. *Suppose there exists a $(1,\rho)$-bounded* ORACLE *for the program $\mathbf{Ax} \geq \mathbf{b}$ with $\mathbf{x} \in \mathcal{P}$. Given an error parameter $\delta > 0$, there is an algorithm which computes a δ-approximate solution to the program, or correctly concludes that it is infeasible, using $O(\frac{\rho\log(m)}{\delta^2})$ calls to the* ORACLE*, plus an additional processing time of $O(m)$ per call.*

Fractional Packing Problems

A fractional packing problem can be written as

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad \mathbf{Ax} \;\leq\; \mathbf{b}$$

where $\mathcal{P}$ is a convex domain such that $\mathbf{Ax} \geq 0$ for all $\mathbf{x} \in \mathcal{P}$, and $\mathbf{b} > 0$. A δ-approximate solution to this system is an $\mathbf{x} \in \mathcal{P}$ such that $\mathbf{Ax} \leq (1+\delta)\mathbf{b}$.

Again, we assume that $b_i = 1$ for all i, scaling the constraints if necessary. Now by rewriting this system as

$$\exists ?\mathbf{x} \in \mathcal{P}: \quad -\mathbf{Ax} \;\geq\; -\mathbf{b}$$

we cast it in our general framework, and a solution $\mathbf{x} \in \mathcal{P}$ which satisfies this up to an additive δ is a δ-approximate solution to the original system. Since for all $\mathbf{x} \in \mathcal{P}$, we have $\mathbf{Ax} \geq 0$, and since all $b_i = 1$, we conclude that for any i, $-\mathbf{A}_i\mathbf{x} + b_i \leq 1$. Thus, we assume that there is a $(1,\rho)$-bounded ORACLE for this problem. Now, applying Theorem 5, we get the following:

Theorem 9. *Suppose there exists a $(1,\rho)$-bounded* ORACLE *for the program $-\mathbf{Ax} \geq -\mathbf{b}$ with $\mathbf{x} \in \mathcal{P}$. Given an error parameter $\delta > 0$, there is an algorithm which computes a δ-approximate solution to the program, or correctly concludes that it is infeasible, using $O(\frac{\rho\log(m)}{\delta^2})$ calls to the* ORACLE*, plus an additional processing time of $O(m)$ per call.*

2.4 A brief history of various applications of the Multiplicative Weights method

An algorithm similar in flavor to the Multiplicative Weights algorithm were proposed in game theory in the early fifties [29, 28, 86]. Following Brown [28], this algorithm was called "Fictitious Play": at each step each player observes actions taken by his opponent in previous stages, updates his beliefs about his opponents' strategies, and chooses myopic pure best responses against these beliefs. In the simplest case, the player simply assumes that the opponent is playing form an stationary distribution and sets his current belief of the opponent's distribution to be the empirical frequency of the strategies played by the opponent. This simple idea (which was shown to lead to optimal solutions in the limit in various cases) led to many subfields of economics, including Arrow-Debreu General Equilibrium theory and more recently, evolutionary game theory. Grigoriadis and Khachiyan [49] showed how a randomized variant of "Fictitious Play" can solve two player zero-sum games efficiently. This algorithm is precisely the multiplicative weights algorithm. It can be viewed as a soft version of fictitious play, when the player gives higher weight to the strategies which pay off better, and chooses her strategy using these weights rather than choosing the myopic best response strategy.

In Machine Learning, the earliest form of the multiplicative weights update rule was used by Littlestone in his well-known Winnow algorithm [76, 77]. This algorithm was generalized by Littlestone and Warmuth [78] in the form of the Weighted Majority algorithm.

The multiplicative update rule (and the exponential potential function) was also discovered in Computational Geometry in the late 1980s [34] and several applications in geometry are described in Chazelle [33] (p. 6, and p. 124).

The weighted majority algorithm as well as more sophisticated versions have been independently discovered in operations research and statistical decision making in the context of the *On-line decision problem*; see the surveys of Cover [37], Foster and Vohra [41], and also Blum [23] who includes applications of weighted majority to machine learning. A notable algorithm, which is different from but related to our framework, was developed by Hannan in the fifties [52]. Kalai and Vempala showed how to derive efficient algorithms via similar methods [59].

Within computer science, several researchers have previously noted the close relationships between multiplicative update algorithms used in different contexts. Young [103] notes the connection between fast LP algorithms and Raghavan's method of pessimistic estimators for derandomization of randomized rounding algorithms. Klivans and Servedio [70] relate boosting algorithms in learning theory to proofs of Yao's XOR Lemma. Garg and Khandekar describe a common framework for convex optimization problems that contains Garg-Könemann [44] and Plotkin-Shmoys-Tardos [85] as subcases.

In the survey paper [15], we use the framework developed in this chapter to unify previously known applications of the Multiplicative Weights method. In the same paper, we also give lower bounds (inspired by the work of Klein and Young [69]) that show that the analysis of the Multiplicative Weights algorithm is tight in the various parameters involved.

Chapter 3

The Matrix Multiplicative Weights Algorithm

In the preceding chapter, we considered an online decision making problem with experts. We refer to that setting as the *basic* or *scalar* case. In the present chapter, we consider a different online decision making problem with experts, which is seemingly quite different from the previous one, but has enough structure that we can obtain an analogous algorithm for it. We move from loss vectors to loss matrices, and from probability vectors to density matrices. For this reason, we refer to the current setting as the *matrix* case. We call the algorithm presented in this setting the *Matrix Multiplicative Weights algorithm.*

The matrix case has the basic case as a subcase when all the matrices involved are diagonal, and in this sense the results of this chapter are a generalization of the ones in the last. The advantage of this matrix formulation is that it allows us to algorithmically obtain good bounds on the extreme eigenvalues of matrices of interest. This immediately leads to applications in:

- solving semidefinite programs, which have non-negativity conditions on the eigenvalues of a matrix of variables (see Chapter 4),
- derandomizing constructions of expander graphs, which are characterized by having second largest eigenvalues of their normalized adjacency matrix bounded away from 1 (see Chapter 5), and
- a learning problem in quantum computing concerning density matrices (see Chapter 5).

We expect there to be many more applications, just as the basic case does.

3.1 Preliminaries

The domain of interest is $\mathbb{R}^n$, and we will deal with symmetric matrices in $\mathbb{R}^{n\times n}$ throughout (or, in general, Hermitian matrices in $\mathbb{C}^{n\times n}$: all results stated here follow without change). We review a few basic matrix facts which can be found in [55, 56]. By the spectral theorem, any symmetric matrix $\mathbf{A}$ can be diagonalized and has real eigenvalues, i.e. there exists an orthogonal matrix $\mathbf{U}$ and a diagonal matrix $\mathbf{D}$ with the eigenvalues of $\mathbf{A}$ on the diagonal such that $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^\top$. We let $\lambda_1(\mathbf{A}), \lambda_2(\mathbf{A}), \ldots, \lambda_n(\mathbf{A})$ be the eigenvalues of $\mathbf{A}$ in decreasing order.

The trace of a matrix $\mathbf{A}$, denoted by $\mathbf{Tr}(\mathbf{A})$, is the sum of its diagonal entries, i.e. $\mathbf{Tr}(\mathbf{A}) = \sum_{i=1}^n A_{ii}$, and is also equal to the sum of all its eigenvalues, $\sum_{i=1}^n \lambda_i(\mathbf{A})$. A matrix $\mathbf{A}$ will be called positive semidefinite, denoted by $\mathbf{A} \succeq \mathbf{0}$, if all its eigenvalues are non-negative, i.e. for all $i \in [n]$, $\lambda_i(\mathbf{A}) \geq 0$. The positive semidefiniteness property induces a partial order on all symmetric matrices: we say that $\mathbf{A} \succeq \mathbf{B}$ if $\mathbf{A} - \mathbf{B} \succeq \mathbf{0}$.

For any function $f : \mathbb{R} \to \mathbb{R}$, extend the definition of f to symmetric matrices as follows. For a diagonal matrix $\mathbf{D}$, $f(\mathbf{D})$ is the diagonal matrix with f applied to each diagonal entry of $\mathbf{D}$. For an arbitrary symmetric matrix $\mathbf{A}$, we first diagonalize $\mathbf{A}$ as $\mathbf{U}\mathbf{D}\mathbf{U}^\top$ where $\mathbf{U}$ is an orthogonal matrix and $\mathbf{D}$ is a diagonal matrix of the eigenvalues of $\mathbf{A}$. Then $f(\mathbf{A}) = \mathbf{U}f(\mathbf{D})\mathbf{U}^\top$.

Lemma 1. *Let $f, g : \mathbb{R} \to \mathbb{R}$, and suppose that the inequality $f(x) \geq g(x)$ holds for $x \in D$ for some $D \subseteq \mathbb{R}$. Then for any symmetric matrix $\mathbf{A}$ all of whose eigenvalues lie in D, we have $f(\mathbf{A}) \succeq g(\mathbf{A})$.*

Proof: Let $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^\top$ be the diagonalization of $\mathbf{A}$. Then $f(\mathbf{A}) - g(\mathbf{A}) = \mathbf{U}(f(\mathbf{D}) - g(\mathbf{D}))\mathbf{U}^\top$, and since all eigenvalues of $\mathbf{A}$ are in D, $f(\mathbf{D}) - g(\mathbf{D})$ is diagonal matrix with non-negative diagonal. Thus, $f(\mathbf{A}) - g(\mathbf{A})$ is a positive semidefinite matrix, and hence $f(\mathbf{A}) \succeq g(\mathbf{A})$. $\Box$

In particular, we will need the notion of the matrix exponential, $\exp(\mathbf{A})$, which can be equivalently defined as

$$\exp(\mathbf{A}) \ = \ \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}.$$

This power series converges for all $\mathbf{A}$. We will need the following matrix inequalities:

Corollary 2. *For any $\varepsilon \leq 1$, let $\varepsilon_1 = 1 - e^{-\varepsilon}$ and $\varepsilon_2 = e^{\varepsilon} - 1$. Then we have the following matrix inequalities:*

1. *If all eigenvalues of a symmetric matrix $\mathbf{A}$ lie in $[0, 1]$, then $\exp(-\varepsilon\mathbf{A}) \preceq \mathbf{I} - \varepsilon_1\mathbf{A}$.*

2. *If all eigenvalues of a symmetric matrix $\mathbf{A}$ lie in $[-1, 0]$, then $\exp(-\varepsilon\mathbf{A}) \preceq \mathbf{I} - \varepsilon_2\mathbf{A}$.*

Proof: The inequalities follow immediately from Lemma 1 using the following inequalities over real numbers, which follow from the convexity of the exponential function:

$$\begin{aligned} \exp(-\varepsilon x) &\leq 1 - \varepsilon_1 x && \text{if } x \in [0,1] \\ \exp(-\varepsilon x) &\leq 1 - \varepsilon_2 x && \text{if } x \in [-1,0] \end{aligned}$$

$\Box$

We also need the Golden-Thompson inequality:

Lemma 2 (Golden [48], Thompson [91])**.** *Let $\mathbf{A}$ and $\mathbf{B}$ be any two real symmetric matrices. Then,*

$$\mathbf{Tr}(\exp(\mathbf{A} + \mathbf{B})) \ \leq \ \mathbf{Tr}(\exp(\mathbf{A})\exp(\mathbf{B})).$$

3.2 A Matrix Game

We associate an expert with every unit vector $\mathbf{v}$ in $\mathbb{S}^{n-1}$, the unit sphere in $\mathbb{R}^n$. As in the basic case, in every round, each expert recommends an action, and our task is to pick an expert $\mathbf{v} \in \mathbb{S}^{n-1}$ and follow his suggested course of action. At this point, the losses of all actions recommended by the experts are revealed by nature. These losses are not arbitrary, but they are correlated in the following way. A *loss matrix* $\mathbf{M} \in \mathbb{R}^{n\times n}$ is revealed, and the loss of an expert $\mathbf{v}$ is then $\mathbf{v}^\top \mathbf{M}\mathbf{v}$. We assume that all these losses are either in the range $[0,1]$ or in $[-1,0]$. Again, as in the basic case, this is the only assumption we make on the way nature chooses the costs; indeed, the costs could even be chosen adversarially. Equivalently, we assume that the matrix $\mathbf{M}$ satisfies the bounds $\mathbf{0} \preceq \mathbf{M} \preceq \mathbf{I}$ or $-\mathbf{I} \preceq \mathbf{M} \preceq \mathbf{0}$.

This game is repeated over a number of rounds. Let $t = 1, 2, \ldots, T$ denote the current round. In each round t, we select a distribution $\mathcal{D}^{(t)}$ over the set of experts $\mathbb{S}^{n-1}$, and select an expert $\mathbf{v}$ randomly from it (and use his advised course of action). At this point, the losses of all the experts are revealed by nature in the form of the loss matrix $\mathbf{M}^{(t)}$. The expected cost to the algorithm for choosing the distribution $\mathcal{D}^{(t)}$ is

$$\mathop{\mathbb{E}}_{\mathbf{v}\in\mathcal{D}^{(t)}}[\mathbf{v}^\top \mathbf{M}^{(t)}\mathbf{v}] \;=\; \mathop{\mathbb{E}}_{\mathbf{v}\in\mathcal{D}^{(t)}}[\mathbf{M}^{(t)} \bullet \mathbf{v}\mathbf{v}^\top] \;=\; \mathbf{M}^{(t)} \bullet \mathop{\mathbb{E}}_{\mathbf{v}\in\mathcal{D}^{(t)}}[\mathbf{v}\mathbf{v}^\top].$$

Define the matrix $\mathbf{P}^{(t)} := \mathbb{E}_{\mathbf{v}\in\mathcal{D}^{(t)}}[\mathbf{v}\mathbf{v}^\top]$. Note that $\mathbf{P}^{(t)}$ is positive semidefinite: this is because it is a convex combination of the elementary positive semidefinite matrices $\mathbf{v}\mathbf{v}^\top$. Also, $\mathbf{Tr}(\mathbf{P}^{(t)}) = 1$, again because for all $\mathbf{v}$, we have $\mathbf{Tr}(\mathbf{v}\mathbf{v}^\top) = \|\mathbf{v}\|^2 = 1$. A matrix $\mathbf{P}$ which is positive semidefinite and has trace 1 is called a *density matrix*.

We will only be interested in the expected loss to the algorithm, and all the information required for computing the expected loss for a given distribution $\mathcal{D}$ over $\mathbb{S}^{n-1}$ is contained in the associated density matrix. Conversely, given a density matrix $\mathbf{P}$, there is a canonical way to associate a distribution $\mathcal{D}$ over $\mathbb{S}^{n-1}$ such that its density matrix is $\mathbf{P}$: let $\mathcal{P} = \sum_{i=1}^n \lambda_i(\mathbf{P})\mathbf{v}_i\mathbf{v}_i^\top$ be the eigen decomposition of $\mathbf{P}$, where for $i = 1, 2, \ldots, n$, $\mathbf{v}_i$ is a (unit) eigenvector belonging to the eigenvalue $\lambda_i(\mathbf{P})$. Then the canonical distribution $\mathcal{D}$ is the one that chooses the vector $\mathbf{v}_i$ with probability $\lambda_i(\mathbf{P})$. Note that for all i, $\lambda_i(\mathbf{P}) \geq 0$ (since $\mathbf{P}$ is positive semidefinite), and $\sum_{i=1}^n \lambda_i(\mathbf{P}) = \mathbf{Tr}(\mathbf{P}) = 1$, so the eigenvalues do form a valid probability distribution. Note that $\mathcal{D}$ is a discrete distribution.

With this intuition, in each round t, we require our online algorithm to choose a density matrix $\mathbf{P}^{(t)}$, rather than a distribution $\mathcal{D}^{(t)}$ over $\mathbb{S}^{n-1}$ (the distribution is implicit in the choice of $\mathbf{P}^{(t)}$). We then observe the loss matrix $\mathbf{M}^{(t)}$ revealed by nature, and suffer the expected loss $\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$. After T rounds, the total expected loss is $\sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$, while the best fixed expert in hindsight corresponds to the unit vector $\mathbf{v}$ which minimizes $\sum_{t=1}^T \mathbf{v}^\top \mathbf{M}^{(t)}\mathbf{v}$. Since we minimize this quantity over all unit vectors $\mathbf{v}$, the variational characterization of eigenvalues implies that this minimum loss is exactly $\lambda_n(\sum_{t=1}^T \mathbf{M}^{(t)})$. Our goal is to design an online algorithm whose total expected loss over the T rounds is not much more than the loss of the best expert.

The following theorem bounds the total expected loss of the Matrix Multiplicative Weights algorithm (given in Figure 3.1) in terms of the loss of the best fixed expert:

Matrix Multiplicative Weights algorithm

Initialization: Fix an $\varepsilon \le \frac{1}{2}$. Initialize the weight matrix $\mathbf{W}^{(1)} = \mathbf{I}_n$.
For $t = 1, 2, \ldots, T$:

1. Use the density matrix $\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\mathbf{Tr}(\mathbf{W}^{(t)})}$.
2. Observe the loss matrix $\mathbf{M}^{(t)}$.
3. Update the weight matrix as follows:

$$\mathbf{W}^{(t+1)} \;=\; \exp(-\varepsilon \textstyle\sum_{\tau=1}^{t} \mathbf{M}^{(\tau)}).$$

Figure 3.1: The Matrix Multiplicative Weights algorithm.

Theorem 10. *In the given setup, the Matrix Multiplicative Weights algorithm guarantees that after T rounds, for any expert $\mathbf{v}$, we have*

$$(1-\varepsilon)\sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1+\varepsilon)\sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\le\; \sum_{t=1}^{T} \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v} + \frac{\ln n}{\varepsilon}. \tag{3.1}$$

Here, the subscripts "$\succeq \mathbf{0}$" and "$\preceq \mathbf{0}$" in the summations are used to refer to the rounds t when $\mathbf{M}^{(t)} \succeq \mathbf{0}$ and $\mathbf{M}^{(t)} \preceq \mathbf{0}$ respectively.

Proof: The proof is exactly on the lines of the proof of Theorem 2, and is based on the potential function $\Phi^{(t)} = \mathbf{Tr}(\mathbf{W}^{(t)})$. We track the changes in $\Phi^{(t)}$ over time. The analysis is complicated by the fact that matrix multiplication is non-commutative, so $\exp(\mathbf{A}+\mathbf{B}) \neq \exp(\mathbf{A})\exp(\mathbf{B})$ in general. However, we can use the Golden-Thompson inequality (Lemma 2): $\mathbf{Tr}(\exp(\mathbf{A}+\mathbf{B})) \le \mathbf{Tr}(\exp(\mathbf{A})\exp(\mathbf{B}))$. Define $\varepsilon_1 = 1 - e^{-\varepsilon}$ and $\varepsilon_2 = e^{\varepsilon} - 1$. We have:

$$\begin{aligned}
\Phi^{(t+1)} &= \mathbf{Tr}(\mathbf{W}^{(t+1)}) \\
&= \mathbf{Tr}(\exp(-\varepsilon \textstyle\sum_{\tau=1}^{t} \mathbf{M}^{(\tau)})) \\
&\le \mathbf{Tr}(\exp(-\varepsilon \textstyle\sum_{\tau=1}^{t-1} \mathbf{M}^{(\tau)}) \exp(-\varepsilon \mathbf{M}^{(t)})) && \because \text{ Golden-Thompson inequality} \\
&= \mathbf{W}^{(t)} \bullet \exp(-\varepsilon \mathbf{M}^{(t)}) && \because \mathbf{Tr}(\mathbf{AB}) = \mathbf{A} \bullet \mathbf{B} \\
&\le \begin{cases} \mathbf{W}^{(t)} \bullet (\mathbf{I} - \varepsilon_1 \mathbf{M}^{(t)}) & \text{if } \mathbf{M}^{(t)} \succeq \mathbf{0} \\ \mathbf{W}^{(t)} \bullet (\mathbf{I} - \varepsilon_2 \mathbf{M}^{(t)}) & \text{if } \mathbf{M}^{(t)} \preceq \mathbf{0} \end{cases} && \text{by Corollary 2} \\
&= \begin{cases} \mathbf{Tr}(\mathbf{W}^{(t)}) \cdot (1 - \varepsilon_1 \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}) & \text{if } \mathbf{M}^{(t)} \succeq \mathbf{0} \\ \mathbf{Tr}(\mathbf{W}^{(t)}) \cdot (1 - \varepsilon_2 \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}) & \text{if } \mathbf{M}^{(t)} \preceq \mathbf{0} \end{cases} \\
&\le \begin{cases} \Phi^{(t)} \cdot \exp(-\varepsilon_1 \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}) & \text{if } \mathbf{M}^{(t)} \succeq \mathbf{0} \\ \Phi^{(t)} \cdot \exp(-\varepsilon_2 \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}) & \text{if } \mathbf{M}^{(t)} \preceq \mathbf{0} \end{cases}
\end{aligned}$$

By induction, since $\Phi^{(1)} = \mathbf{Tr}(\mathbf{W}^{(1)}) = \mathbf{Tr}(\mathbf{I}_n) = n$, we get that

$$\Phi^{(T+1)} \;\leq\; n \exp\left(-\varepsilon_1 \textstyle\sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} - \varepsilon_2 \textstyle\sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}\right).$$

On the other hand, we have:

$$\Phi^{(T+1)} \;=\; \mathbf{Tr}(\mathbf{W}^{(T+1)}) \;=\; \mathbf{Tr}(\exp(-\varepsilon \textstyle\sum_{t=1}^{T} \mathbf{M}^{(t)})) \;\geq\; \exp(-\varepsilon \lambda_n (\textstyle\sum_{t=1}^{T} \mathbf{M}^{(t)})).$$

The last inequality follows because $\mathbf{Tr}(e^{\mathbf{A}}) = \sum_{k=1}^{n} e^{\lambda_k(\mathbf{A})} \geq e^{\lambda_n(\mathbf{A})}$. Thus, we conclude that

$$\exp(-\varepsilon \lambda_n (\textstyle\sum_{t=1}^{T} \mathbf{M}^{(t)})) \;\leq\; n \exp\left(-\varepsilon_1 \textstyle\sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} - \varepsilon_2 \textstyle\sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}\right).$$

Now, for any unit vector $\mathbf{v}$, $\lambda_n(\sum_{t=1}^{T} \mathbf{M}^{(t)}) \leq \sum_{t=1}^{T} \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v}$. Using this fact, and by taking logarithms and simplifying, we get the required inequality. We also need the facts that $\varepsilon_1 = 1 - e^{-\varepsilon} \geq \varepsilon(1-\varepsilon)$ and $\varepsilon_2 = e^{\varepsilon} - 1 \leq \varepsilon(1+\varepsilon)$ for $\varepsilon \leq 1/2$. □

Corollary 3. *In the given setup, the Matrix Multiplicative Weights algorithm guarantees that after T rounds, for any density matrix $\mathbf{P}$, we have*

$$(1-\varepsilon) \sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1+\varepsilon) \sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\leq\; \sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P} + \frac{\ln n}{\varepsilon}. \tag{3.2}$$

Furthermore, we have

$$(1-\varepsilon) \sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1+\varepsilon) \sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\leq\; \lambda_n\left(\sum_{t=1}^{T} \mathbf{M}^{(t)}\right) + \frac{\ln n}{\varepsilon}. \tag{3.3}$$

Proof: The inequality (3.3) follows from (3.1) because $\min_{\mathbf{v} \in \mathbb{S}^{n-1}} \sum_{t=1}^{T} \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v} = \lambda_n(\sum_{t=1}^{T} \mathbf{M}^{(t)})$. As for (3.2), let $\mathbf{P} = \sum_{i=1}^{n} \lambda_i(\mathbf{P}) \mathbf{v}_i \mathbf{v}_i^\top$ be the eigen decomposition of $\mathbf{P}$, where $\mathbf{v}_i$ are unit vectors. Then

$$\sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P} \;=\; \sum_{t=1}^{T} \sum_{i=1}^{n} \lambda_i(\mathbf{P}) \mathbf{v}_i^\top \mathbf{M}^{(t)} \mathbf{v}_i \;\geq\; \sum_{i=1}^{n} \lambda_i(\mathbf{P}) \cdot \lambda_n\left(\sum_{t=1}^{T} \mathbf{M}^{(t)}\right) \;=\; \lambda_n\left(\sum_{t=1}^{T} \mathbf{M}^{(t)}\right),$$

because $\sum_{i=1}^{n} \lambda_i(\mathbf{P}) = \mathbf{Tr}(\mathbf{P}) = 1$. □

Remark. While we have restricted our matrices to be real and symmetric, the algorithm and the theorem hold without change even for complex Hermitian matrices. The analysis is identical because the Golden-Thompson inequality and Corollary 2 hold for complex Hermitian matrices as well. This consideration is important when we apply the algorithm to quantum computing problems where the matrices are complex Hermitian (see Chapter 5).

3.2.1 Gains instead of losses

Just as in Section 2.2.1, we can consider a version of the matrix game where the matrices $\mathbf{M}^{(t)}$ specify gains for the experts instead of losses. We get an algorithm for this case by using the Matrix Multiplicative Weights algorithm with the loss matrices $-\mathbf{M}^{(t)}$. The weight matrix in round $t+1$ is now

$$\mathbf{W}^{(t+1)} \;=\; \exp(\varepsilon \textstyle\sum_{\tau=1}^{t} \mathbf{M}^{(t)}).$$

Now Theorem 10 immediately implies the following theorem:

Theorem 11. *In the given setup, the Matrix Multiplicative Weights algorithm (for gains) guarantees that after T rounds, for any expert $\mathbf{v}$, we have*

$$(1-\varepsilon)\sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1+\varepsilon)\sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\geq\; \sum_{t=1}^{T} \mathbf{v}^\top \mathbf{M}^{(t)} \mathbf{v} - \frac{\ln n}{\varepsilon}. \tag{3.4}$$

We also have the following corollary analogous to Corollary 3:

Corollary 4. *In the given setup, the Matrix Multiplicative Weights algorithm (for gains) guarantees that after T rounds, for any density matrix $\mathbf{P}$, we have*

$$(1-\varepsilon)\sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1+\varepsilon)\sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\geq\; \sum_{t=1}^{T} \mathbf{M}^{(t)} \bullet \mathbf{P} - \frac{\ln n}{\varepsilon}. \tag{3.5}$$

Furthermore, we have

$$(1-\varepsilon)\sum_{\preceq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} + (1+\varepsilon)\sum_{\succeq \mathbf{0}} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\geq\; \lambda_1\left(\sum_{t=1}^{T} \mathbf{M}^{(t)}\right) - \frac{\ln n}{\varepsilon}. \tag{3.6}$$

3.2.2 Connection to the Basic Setting

The algorithm presented here can be viewed as a generalization of the basic Multiplicative Weights algorithm, when all matrices involved are diagonal. To avoid complications due to positive and negative losses, we only consider the setting where all losses in the basic setting are non-negative, and all loss matrices in the matrix setting are positive semidefinite.

We associate the n experts in the basic setting with the n standard basis vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$, where $\mathbf{e}_i$ has 1 in the i^{th} coordinate, and zeros elsewhere. In round t, the loss vector $\mathbf{m}^{(t)}$ is specified in the form of a diagonal positive semidefinite matrix $\mathbf{M}^{(t)} = \operatorname{diag}(\mathbf{m}^{(t)})$. Then the loss of expert i is exactly $\mathbf{e}_i^\top \mathbf{M}^{(t)} \mathbf{e}_i$. A distribution $\mathbf{p}^{(t)}$ over the experts can also be specified as the diagonal density matrix $\mathbf{P}^{(t)} = \operatorname{diag}(\mathbf{p}^{(t)})$. Then the expected loss for using distribution $\mathbf{p}^{(t)}$ in round t is exactly $\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$.

With these transformations, we see that the Matrix Multiplicative Weights algorithm in this setting reduces exactly to the basic Multiplicative Weights algorithm (with the minor change that the parameter ε in the Matrix Multiplicative Weights algorithm is actually $-\ln(1-\varepsilon')$ where ε' is the parameter used in the basic Multiplicative Weights algorithm). The loss bound of Theorem 10 in this case also translates directly to the loss bound of Theorem 2.

3.3 A Min-Max Theorem

Freund and Schapire [43] observed that the basic Multiplicative Weights algorithm can be used to construct approximately optimal strategies for two player zero-sum games (see Section 2.3.1). They also noted that this gives an alternative way to prove von Neumann's min-max theorem for two player zero-sum games. In the previous section, we saw that the Matrix Multiplicative Weights algorithm reduces to the basic Multiplicative Weights algorithm if all the matrices involved are diagonal. This indicates that there should be an analogous min-max theorem arising out of the Matrix Multiplicative Weights algorithm.

We describe such a min-max theorem now. We imagine a two player zero-sum game where the first player's strategy set is $\mathbb{S}^{n-1}$, and the second player's strategy set is a subset $\mathcal{M}$ of symmetric matrices in $\mathbb{R}^{n\times n}$ with eigenvalues in $[0,1]$. In each round of the game, the first player chooses a unit vector $\mathbf{v} \in \mathbb{S}^{n-1}$, and the second player chooses a matrix $\mathbf{M} \in \mathcal{M}$. The first player then suffers a loss of $\mathbf{v}^\top \mathbf{M}\mathbf{v}$ to the second player. We allow both players to randomize over their strategy sets, i.e. the first player chooses his strategy from a distribution $\mathcal{P}$ over $\mathbb{S}^{n-1}$ and the second player chooses his from a distribution $\mathcal{Q}$ over $\mathcal{M}$. Then the expected payoff to the second player is denoted by $\mathbb{E}_{\mathcal{P},\mathcal{Q}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}]$. The following min-max theorem shows that if both players choose their respective distributions to optimize their worst case payoffs, then the expected payoff in either case is the same:

Theorem 12. *With the given setup, the following equality holds:*

$$\min_{\mathcal{P}}\max_{\mathcal{Q}}\mathop{\mathbb{E}}_{\mathcal{P},\mathcal{Q}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] \;=\; \max_{\mathcal{Q}}\min_{\mathcal{P}}\mathop{\mathbb{E}}_{\mathcal{P},\mathcal{Q}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}].$$

Proof: First, we observe that the expected payoff $\mathbb{E}_{\mathcal{P},\mathcal{Q}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] = \bar{\mathbf{M}} \bullet \bar{\mathbf{P}}$ where $\bar{\mathbf{M}} = \mathbb{E}_{\mathcal{Q}}[\mathbf{M}]$ and $\bar{\mathbf{P}} = \mathbb{E}_{\mathcal{P}}[\mathbf{v}\mathbf{v}^\top]$. Note that $\mathbf{0} \preceq \bar{\mathbf{M}} \preceq \mathbf{I}$, and $\mathbf{P}$ is a density matrix. Now define $\lambda_1 = \min_{\mathcal{P}}\max_{\mathcal{Q}}\mathbb{E}_{\mathcal{P},\mathcal{Q}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}]$ and $\lambda_2 = \max_{\mathcal{Q}}\min_{\mathcal{P}}\mathbb{E}_{\mathcal{P},\mathcal{Q}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}]$.

Standard calculations show that $\lambda_1 \geq \lambda_2$: we have, for any $\mathcal{P}_0, \mathcal{Q}_0$,

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathcal{P}_0,\mathcal{Q}_0}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] &\geq \min_{\mathcal{P}}\mathop{\mathbb{E}}_{\mathcal{P},\mathcal{Q}_0}[\mathbf{v}^\top\mathbf{M}\mathbf{v}]\\
\therefore \max_{\mathcal{Q}_0}\mathop{\mathbb{E}}_{\mathcal{P}_0,\mathcal{Q}_0}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] &\geq \max_{\mathcal{Q}_0}\min_{\mathcal{P}}\mathop{\mathbb{E}}_{\mathcal{P},\mathcal{Q}_0}[\mathbf{v}^\top\mathbf{M}\mathbf{v}]\\
\therefore \min_{\mathcal{P}_0}\max_{\mathcal{Q}_0}\mathop{\mathbb{E}}_{\mathcal{P}_0,\mathcal{Q}_0}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] &\geq \max_{\mathcal{Q}_0}\min_{\mathcal{P}}\mathop{\mathbb{E}}_{\mathcal{P},\mathcal{Q}_0}[\mathbf{v}^\top\mathbf{M}\mathbf{v}].
\end{aligned}$$

To show $\lambda_1 \leq \lambda_2$, we imagine a matrix game between a player, who iteratively chooses and updates distributions on $\mathbb{S}^{n-1}$, $\mathcal{P}^{(t)}$, expressed in the form of density matrices $\mathbf{P}^{(t)}$, using the Matrix Multiplicative Weights algorithm. Nature chooses the most adversarial distribution on $\mathcal{M}$, i.e. $\mathcal{Q}^{(t)} = \arg\max_{\mathcal{Q}}\mathbb{E}_{\mathcal{Q}}[\mathbf{M}\bullet\mathbf{P}^{(t)}]$, in response to the distribution $\mathcal{P}^{(t)}$, and plays the matrix $\mathbf{M}^{(t)} = \mathbb{E}_{\mathcal{Q}^{(t)}}[\mathbf{M}]$. This implies that for all rounds t,

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;=\; \mathop{\mathbb{E}}_{\mathcal{P}^{(t)},\mathcal{Q}^{(t)}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] \;\geq\; \lambda_1.$$

Now, applying Theorem 10 and using the fact that in all rounds t we have $\mathbf{M}^{(t)} \succeq \mathbf{0}$, we get that for any density matrix $\mathbf{P}$,

$$(1-\varepsilon)T\lambda_1 \;\leq\; (1-\varepsilon)\sum_{t=1}^{T}\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;\leq\; \sum_{t=1}^{T}\mathbf{M}^{(t)} \bullet \mathbf{P} + \frac{\ln n}{\varepsilon}.$$

Divide by T, and define $\bar{\mathcal{Q}}^{(T)} = \frac{1}{T}\sum_{t=1}^{T}\mathcal{Q}^{(t)}$. Since $\mathbf{P}$ can represent any distribution $\mathcal{P}$ over $\mathbb{S}^{n-1}$, we have

$$(1-\varepsilon)\lambda_1 \;\leq\; \mathop{\mathbb{E}}_{\mathcal{P},\bar{\mathcal{Q}}^{(T)}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] + \frac{\ln(n)}{\varepsilon T}.$$

If we choose $\mathcal{P}$ to be the most adversarial response of the online player to nature's distribution $\bar{\mathcal{Q}}^{(T)}$, i.e. $\mathcal{P} = \arg\min_{\mathcal{P}_0}\mathbb{E}_{\mathcal{P}_0,\bar{\mathcal{Q}}^{(T)}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}]$, and using the definition of λ_2, we get

$$\mathop{\mathbb{E}}_{\mathcal{P},\bar{\mathcal{Q}}^{(T)}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] \;\leq\; \lambda_2.$$

Putting these together, and choosing $\varepsilon = \sqrt{\frac{\ln(n)}{T}}$, we get that

$$\left(1-\sqrt{\frac{\ln n}{T}}\right)\lambda_1 \;\leq\; \lambda_2 + \sqrt{\frac{\ln(n)}{T}}.$$

Letting $T \to \infty$, we get that $\lambda_1 \leq \lambda_2$ as desired. $\square$

Remark. From the proof of Corollary 3, it can be seen that this theorem is equivalent to the following:

$$\min_{\mathcal{P}}\max_{\mathbf{M}\in\mathcal{M}}\mathop{\mathbb{E}}_{\mathcal{P}}[\mathbf{v}^\top\mathbf{M}\mathbf{v}] \;=\; \max_{\mathcal{Q}}\lambda_n\!\left(\mathop{\mathbb{E}}_{\mathcal{Q}}[\mathbf{M}]\right)$$

This fact also follows from the (strong) semidefinite programming (SDP) duality. Alternatively, this can be seen as a proof of the strong SDP duality, in the same way that von Neumann's min-max theorem is equivalent to LP duality. This fact is the basis of the primal-dual algorithms for SDP of the next chapter.

3.4 Related work

A form of the Matrix Multiplicative Weights algorithm for learning problems had been previously discovered in Machine Learning by Tsuda, Rätsch and Warmuth [92], as the *Matrix Exponentiated Gradient* algorithm. Warmuth and Kuzmin extended these ideas to independently discover the same Matrix Multiplicative Weights algorithm as ours and gave further applications to online variance minimization [98] and online Principal Component Analysis [99, 100]. They also developed a related Bayesian probability calculus for density matrices [97]. Warmuth [96] applies the algorithm to online Winnowing Subspaces. All these papers have a different proof of convergence using the quantum relative entropy as a potential function.

Finally, Wigderson and Xiao [101] independently obtained some more applications to derandomization and covering SDPs (see Chapter 5 for some of these applications). Their algorithms are derived by derandomizing the Ahlswede-Winter [4] Chernoff bound for matrix valued random variables using the technique of pessimistic estimators, and in fact implicitly use the Matrix Multiplicative Weights algorithm. This is directly analogous to Young's [103] algorithms for fractional packing and covering linear programs which are based on derandomizing the standard Chernoff bounds using pessimistic estimators. In fact, it was this very connection to Chernoff bounds discovered by Young that led us to wonder whether a matrix analogue of the basic Multiplicative Weights algorithm can be designed, using the matrix Chernoff bounds of Ahlswede and Winter. The result was the Matrix Multiplicative Weights algorithm.

Chapter 4

Combinatorial, Primal-Dual Algorithms for Semidefinite Programming

As discussed in the introductory chapter, semidefinite programming (SDP) has proved useful in design of approximation algorithms for NP-hard problems, and often (as in the case of MaxCut, Sparsest Cut, Min UnCut, Min 2CNF Deletion, etc.) yields better approximation ratios than known LP-based methods.

But in several ways, our understanding of SDPs seriously lags our understanding of LPs. One is running time: though LP and SDP are syntactically similar when viewed as subcases of cone optimization and can theoretically be solved in similar amounts of time [5, 84], in practice SDP solvers are slower. Another is conceptual: LP-inspired notions such as duality are ubiquitous in algorithm design whereas corresponding SDP-inspired concepts are rarely used. Thus, primal-dual algorithms which achieve fast running time by circumventing the necessity of solving an LP using a generic method (such as the ellipsoid algorithm, interior point methods, etc.) abound, whereas similar algorithms for SDPs are quite rare.

In this chapter, we describe how to use the Matrix Multiplicative Weights algorithm to obtain a general scheme to design primal-dual algorithms for SDPs. We apply this method to various graph partitioning and constraint satisfaction problems, and obtain the fastest known algorithms for them, beating the running times of previous algorithms based on interior point methods. Furthermore, these algorithms are *combinatorial*, and are able to exploit the special structure of these problems. Thus, we obtain $O(\sqrt{\log n})$ and $O(\log n)$ approximation algorithms to the Sparsest Cut and Balanced Separator problems in undirected and directed weighted graphs, and $O(\sqrt{\log n})$ approximation algorithms to the Min UnCut and Min 2CNF Deletion problems. The design of our primal-dual algorithms is guided by a robust analysis of rounding algorithms used to obtain integer solutions from fractional ones.

The results of this chapter first appeared in a joint paper with Sanjeev Arora [87].

4.1 Primal-Dual algorithms

Primal-dual algorithms in the LP world can be typically divided into two classes. The first compute $(1+\varepsilon)$-approximation to special families of LPs, such as multicommodity flow. They eschew interior point methods in favor of more efficient (and combinatorial) Lagrangian relaxation methods; see Plotkin, Shmoys, Tardos [85], Young [103], Garg, Könemann [44], etc. (these are the type (B) algorithms considered in Chapter 1).

The second class consists of primal-dual approximation algorithms for **NP**-hard problems. These are the type (A) algorithms considered in in Chapter 1. Though these usually evolve out of (and use the same intuition as) earlier approximation algorithms that used LP as a black box, they do not solve the LP *per se*. Rather, the algorithm incrementally builds a dual solution together with an *integer* primal solution, updating them at each step using "combinatorial" methods. At the end, the candidate dual solution is feasible and the bound on the approximation ratio is derived by comparing the integer primal solution to the bound provided by this feasible dual. Usually the update rule is designed using intuition from the *rounding algorithm* used in the original LP-based algorithm. Some canonical examples are network design problems [3] (or see the survey [46]) and $O(1)$-approximation for k-median (LP-based algorithm in [30]; faster primal-dual algorithm in [58]). Arguably, a primal-dual algorithm gives *more* insight than an algorithm that uses LP as a black box. For instance, the primal-dual algorithm for k-median problem inspired the discovery of algorithms for many related problems, as well as algorithms in the online and streaming models.

Since SDPs also satisfy a duality theorem, in principle one should be able to solve them using primal-dual approaches. But several conceptual difficulties arise. First, the basic object in SDPs is a positive semidefinite matrix, whereas it is a half-space (equivalently, a vector) in LPs, and matrix operations are just harder to visualize than vector operations. Second, the recent spate of rounding algorithms for SDPs use the global structure of optimum or near-optimum solutions (e.g., the Arora, Rao, Vazirani (ARV)-style rounding depends upon the geometry of ℓ_2^2 spaces), and it is unclear how to use those rounding ideas in context of the grossly infeasible solutions one might encounter during a primal-dual algorithm. Finally, even if one surmounts the previous two difficulties, there is the issue of implementing matrix operations efficiently enough so that the running time is an improvement over interior point methods.

We note that an *ad hoc* primal-dual approach did prove useful for the SPARSEST CUT problem, resulting in an $O(\sqrt{\log n})$-approximation in $\tilde{O}(n^2)$ time [12], improving upon the $\tilde{O}(n^{4.5})$ time using SDPs [18]. A related paper gives an even more efficient $\tilde{O}(m+n^{1.5})$ time algorithm for SPARSEST CUT, albeit with a worse approximation ratio of $O(\log^2 n)$ [65]. But there is no obvious way to generalize these *ad hoc* approaches to other SDPs, especially as both rely upon the connection between eigenvalues and expansion, which does not extend to problems other than SPARSEST CUT.

4.2 Results of this chapter

In this chapter, we overcome the difficulties mentioned in the previous section and present general techniques that lead to fast primal-dual approximation algorithms for a number of problems.

We give a general primal-dual approximation algorithm for *any* SDP that uses the Matrix Multiplicative Weights algorithm introduced in Chapter 3. The matrix version is useful in an SDP context because $\exp(\mathbf{A})$ is positive semidefinite for all symmetric $\mathbf{A}$. The algorithm is similar to the one described in Section 2.3.2, and requires an appropriate ORACLE which computes loss matrices as feedback to the density matrices generated by the Matrix Multiplicative Weights algorithm.

For general SDPs the implementation of the ORACLE amounts to solving a very simple LP (see Section 4.4), but the convergence time of the algorithm depends upon the "width" of the problem as in the corresponding LP algorithms (see Section 2.3.2). We give very simple and combinatorial implementations of the ORACLE for several problems and prove that the width is low, resulting in (very fast) polynomial running times for the algorithms. Sometimes (as in our algorithm for MAXCUT) computing the feedback is as simple as sorting; at other times it may involve multicommodity flows and shortest paths (as in our algorithms for SPARSEST CUT, MIN UNCUT, and all related problems). Since the goal is an approximation algorithm for an **NP**-hard problem, one can terminate the above process far before the primal SDP solution is $(1 + \varepsilon)$-approximate. Instead, one rounds the current primal candidate and proves its goodness by comparing to the dual. This is the basis for all approximation algorithms in this paper.

Not surprisingly, the computation of the feedback function is inspired by SDP rounding algorithms from [18] and subsequent papers such as [2] (though we had to modify the rounding algorithms in various places). This is the SDP analog of what Young [103] called "Randomized Rounding without solving the LP." We use the following observation about ARV-style rounding techniques: if the rounding fails to yield a good integer solution when applied to a candidate primal solution, then it actually uncovers gross deviations from feasibility in the candidate solution, which can be used as "feedback" (the vector $\mathbf{y}$ in the generic algorithm of Section 4.4) to improve the primal.

We observe that in our context it suffices to compute matrix exponentials only approximately, for which we give efficient algorithms that can make use of sparsity; see Section 4.7. (By contrast, exact matrix exponentiation is tricky and inefficient because of accuracy issues; see [81].) This relies upon a subtle use of the Johnson-Lindenstrauss lemma on random projections. We show that for certain well-conditioned implementations of the ORACLE, it is possible to compute a good enough approximation to the matrix exponential by computing instead its product with several randomly chosen vectors. This primitive is commonly available in packages for solving linear differential equations, raising hope that our approach may be practical.

Table 4.1 lists the running times of our algorithms for approximating various problems on a *weighted* graph with n vertices and m edges, and compares the running time to those of the previously known algorithms from [12, 2]. For problems other than undirected SPARSEST CUT and BALANCED SEPARATOR, the previous best algorithms needed to solve

Problem	Previous: $O(\sqrt{\log n})$ apx	Current: $O(\sqrt{\log n})$ apx	Current: $O(\log n)$ apx
Undir. Sparsest Cut	$\tilde{O}(n^2)$ [12]	$\tilde{O}(n^2)$	$\tilde{O}(m + n^{1.5})$
Undir. Balanced Separator	$\tilde{O}(n^2)$ [12]	$\tilde{O}(n^2)$	$\tilde{O}(m + n^{1.5})$
Dir. Sparsest Cut	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(m^{1.5} + n^{2+\mu})$	$\tilde{O}(m^{1.5})$
Dir. Balanced Separator	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(m^{1.5} + n^{2+\mu})$	$\tilde{O}(m^{1.5})$
Min UnCut	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(n^3)$	–
Min 2CNF Deletion	$\tilde{O}(n^{4.5})$ [2]	$\tilde{O}(nm^{1.5} + n^3)$	–

Table 4.1: Running times obtained for various approximation algorithms.

an SDP with $m = O(n^3)$ triangle inequality constraints. Thus, an interior point algorithm needs $O(\sqrt{m}(n^3 + m)L) = \tilde{O}(n^{4.5})$ time to solve them, which gives the reported running time in the table. Throughout this chapter, the $\tilde{O}(\cdot)$ notation suppresses polylog(n) and poly$(\frac{1}{\varepsilon})$ factors. In the case of directed Balanced Separator and directed Sparsest Cut, the running time depends on a parameter μ which is any given constant. The $\tilde{O}$ notation hides poly$(\frac{1}{\mu})$ dependence on μ in this case.

A recent paper [65] suggested that the gold standard for approximation algorithms in this area should be T_{flow}, the time to compute single commodity max flows. Even though methods based upon LP duality can not yet attain this gold standard, that paper gave algorithms that attain it while computing $O(\log^2 n)$-approximation to Sparsest Cut and Balanced Separator in undirected graphs. We can attain this gold standard for four of the problems even with $O(\log n)$-approximation (T_{flow} is $\tilde{O}(m^{1.5})$ for directed graphs and $\tilde{O}(n^{1.5})$ undirected graphs). For the last three problems, we do not know of any published primal-dual algorithms (even in the LP world).

Related work. Our primal-dual algorithm should not be confused with earlier *primal-only* methods for approximate solutions to SDPs, such as the ones in Arora, Hazan and Kale [13] (described in Chapter 6), and an earlier paper by Klein and Lu [67].

They depend upon the Multiplicative Weights algorithm described in Section 2.3.2 and have a significant drawback – they find primal solutions which satisfy every constraint up to an *additive* error ε, and the running time is proportional to $1/\varepsilon^2$. Since the recent wave of SDP-based approximation algorithms for minimization problems require ε to be quite small, it is difficult to get significant running time improvement (though the algorithms of Chapter 6 get around this hurdle with "hybrid" approaches). This problem is exacerbated if the graph is weighted, when ε may depend upon the largest weight in the graph. By contrast, our algorithm can round the current primal candidate at each step and stop as soon as the gap with the dual is small enough. All our algorithms are strongly polynomial so long as the algorithm for max flow is.

4.3 Preliminaries

4.3.1 Positive Semidefinite Matrices

We recall some matrix notation and facts which can be found in [55, 56]. We will deal with symmetric matrices in $\mathbb{R}^{n\times n}$, unless specified otherwise. As usual, $\mathbf{Tr}(\mathbf{A})$ is the trace of matrix $\mathbf{A}$, which is the sum of the diagonal entries (equivalently, the sum of the eigenvalues) of $\mathbf{A}$. Matrix $\mathbf{A}$ is positive semidefinite, or *PSD*, if there is a matrix $\mathbf{V}$ such that $\mathbf{A} = \mathbf{V}\mathbf{V}^\top$ (equivalently, if every eigenvalue of $\mathbf{A}$ is nonnegative). Such a $\mathbf{V}$ is called the *Cholesky* decomposition of $\mathbf{A}$; note that $A_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$ where $\mathbf{v}_i$ is the i^{th} row of $\mathbf{V}$, and $\mathbf{A}$ is known as the *Gram matrix* of the vectors $\mathbf{v}_i$. For matrices $\mathbf{A}$ and $\mathbf{B}$, define $\mathbf{A} \bullet \mathbf{B} := \mathbf{Tr}(\mathbf{A}\mathbf{B}) = \sum_{ij} A_{ij} B_{ij}$. Notice, this is just the usual inner product if we think of $\mathbf{A}, \mathbf{B}$ as n^2-dimensional vectors. It is easily checked that $\mathbf{A}$ is *PSD* if and only if $\mathbf{A} \bullet \mathbf{B} \geq 0$ for all *PSD* $\mathbf{B}$. We say $\mathbf{A} \succeq \mathbf{B}$ if $\mathbf{A} - \mathbf{B}$ is *PSD*. We use the notation $\mathbf{A} \in [a, b]$, for real numbers a, b, if $a\mathbf{I} \preceq \mathbf{A} \preceq b\mathbf{I}$. We will use the ℓ_2 norm of matrices: $\|\mathbf{A}\|$ is the largest eigenvalue of $\mathbf{A}$ in absolute value, i.e. $\min\{\lambda \geq 0 : \ \mathbf{A} \in [-\lambda, \lambda]\}$. Note that since $\|\cdot\|$ is a norm, the triangle inequality $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$ holds. We will often use the following inequality: for any two vectors $\mathbf{v}$ and $\mathbf{w}$, we have $2\|\mathbf{v}\|^2 + 2\|\mathbf{w}\|^2 \geq \|\mathbf{v} - \mathbf{w}\|^2$.

4.3.2 Laplacians

We often use the following special matrix. If $G = (V, E)$ is a undirected graph with weight $c_{\{i,j\}}$ on edge $\{i, j\}$ then its *combinatorial Laplacian* is a matrix $\mathbf{C}$, with rows and columns indexed by the nodes of G such that $C_{ii} = \sum_{j \neq i} c_{\{i,j\}}$, i.e. the weighted degree of node i, and C_{ij} is $-c_{\{i,j\}}$. We will use two important properties of Laplacians. First, for any positive semidefinite matrix $\mathbf{X}$, if $\mathbf{v}_i$ are the vectors obtained from its Cholesky decomposition, then

$$\mathbf{C} \bullet \mathbf{X} \ = \ \sum_{\{i,j\} \in E} c_{\{i,j\}} \|\mathbf{v}_i - \mathbf{v}_j\|^2.$$

This final expression should be familiar to readers who have encountered SDP relaxations of problems such as MaxCut and Sparsest Cut. Second, we have

$$\mathbf{C} \bullet \mathbf{X} \ = \ \sum_{\{i,j\} \in E} c_{\{i,j\}} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ \leq \ \sum_{\{i,j\} \in E} 2c_{\{i,j\}} (\|\mathbf{v}_i\|^2 + \|\mathbf{v}_j\|^2) \ \leq \ \sum_i 2d\|\mathbf{v}_i\|^2 \ = \ 2d\mathbf{I} \bullet \mathbf{X},$$

where d is the maximum weighted degree in the graph. Thus, we conclude that $\mathbf{C} \preceq 2d\mathbf{I}$. Also, since $\mathbf{C} \bullet \mathbf{X} \geq 0$ for all *PSD* $\mathbf{X}$, we conclude that $\mathbf{C}$ is *PSD*.

Some problems deal with directed graphs, and we need to use the *directed Laplacian*. Let $G = (V, E)$ be a directed graph with weights c_{ij} on directed edge (i, j). Depending on the application, we will actually have two different kinds of directed Laplacians. In the first setting, we associate vectors $\mathbf{v}_i$ and $\mathbf{v}_{n+i}$ with every node i. Note that now the dimension is $2n$ rather than n. The directed Laplacian of the first kind is the matrix $\mathbf{D} \in \mathbb{R}^{2n \times 2n}$ such that if $\mathbf{X} \in \mathbb{R}^{2n \times 2n}$ is any *PSD* matrix with vectors $\mathbf{v}_i$ obtained from its Cholesky decomposition, then

$$\mathbf{D} \bullet \mathbf{X} \;=\; \sum_{(i,j)\in E} c_{ij}(\|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{v}_{n+i} - \mathbf{v}_i\|^2 + \|\mathbf{v}_{n+j} - \mathbf{v}_j\|^2).$$

More explicitly, we have for all $i \in V$, $D_{ii} = 2\sum_{(j,i)\in E} c_{ji}$ and $D_{n+i,n+i} = \sum_{(j,i)\in E} c_{ji} - \sum_{(i,j)\in E} c_{ij}$, and for all $i \neq j \in V$, if $(i,j) \in E$ then $D_{ij} = D_{ji} = -c_{ij}$, $D_{i,n+i} = D_{n+i,i} = \sum_{(i,j)\in E} c_{ij} - \sum_{(j,i)\in E} c_{ji}$, and all other entries are 0. Arguing as before, if d is the maximum degree in the graph, then $-2d\mathbf{I} \preceq \mathbf{D} \preceq 4d\mathbf{I}$.

In the second setting, we associate vectors $\mathbf{v}_i$ with every node i. We also have a special vector $\mathbf{v}_0$. Note that now the dimension is $n+1$ rather than n. The directed Laplacian of the second kind is the matrix $\mathbf{B} \in \mathbb{R}^{(n+1)\times(n+1)}$ such that if $\mathbf{X} \in \mathbb{R}^{(n+1)\times(n+1)}$ is any *PSD* matrix with vectors $\mathbf{v}_i$ obtained from its Cholesky decomposition, then

$$\mathbf{B} \bullet \mathbf{X} \;=\; \sum_{(i,j)\in E} c_{ij}(\|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{v}_0 - \mathbf{v}_i\|^2 + \|\mathbf{v}_0 - \mathbf{v}_j\|^2).$$

More explicitly, we have for all $i \in V$, $B_{ii} = 2\sum_{(j,i)\in E} c_{ji}$ and $B_{00} = 0$, and for all $i \neq j \in V$, if $(i,j) \in E$ then $D_{ij} = D_{ji} = -c_{ij}$, $D_{i0} = D_{0i} = \sum_{(i,j)\in E} c_{ij} - \sum_{(j,i)\in E} c_{ji}$, and all other entries are 0. Again, if $M = \sum_{(i,j)\in E} c_{ij}$ is the total weight of edges in the graph, then $-M\mathbf{I} \preceq \mathbf{K}_2 \preceq 2M\mathbf{I}$.

4.3.3 Matrix Exponentials

Finally, we discuss matrix exponentials. If $\mathbf{A}$ is a matrix, then the exponential is $\exp(\mathbf{A}) = \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}$. Notice, since $\mathbf{AB} \neq \mathbf{BA}$ in general, $\exp(\mathbf{A}+\mathbf{B}) \neq \exp(\mathbf{A})\exp(\mathbf{B})$. Furthermore, $\exp(\mathbf{A})$ is *PSD* for all symmetric $\mathbf{A}$ since $\exp(\mathbf{A}) = \exp(\frac{1}{2}\mathbf{A})^\top \exp(\frac{1}{2}\mathbf{A})$. In particular, this allows us to assume without loss of generality that algorithms for matrix exponentiation can also output the Cholesky decomposition of the output matrix.

Matrix exponentiation itself is tricky because of accuracy issues; see the classic article [81]. However, in context of solving SDPs we need the Cholesky decomposition of $\exp(\mathbf{A})$, namely $\exp(\frac{1}{2}\mathbf{A})$. Furthermore, in all the SDPs of interest in this chapter, the constraints involve squared lengths of vectors $\|\mathbf{v}_i\|^2$ or $\|\mathbf{v}_i - \mathbf{v}_j\|^2$, rather than inner products $\mathbf{v}_i \cdot \mathbf{v}_j$. Thus it actually suffices to compute the Cholesky decomposition approximately, in a way that these lengths are preserved upon a multiplicative factor $(1+\varepsilon)$. By the well-known Johnson-Lindenstrauss lemma, the lengths of n vectors in ℓ_2 are essentially determined by their projections on $O(\frac{\log n}{\varepsilon^2})$ random directions, so it suffices to compute $\exp(\frac{1}{2}\mathbf{A}) \cdot \mathbf{u}$ for a random unit vector $\mathbf{u}$. This operation is extremely efficient (even for arbitrary vector $\mathbf{u}$) since it is at the heart of software packages to solve systems of linear differential equations. The algorithms are also provably efficient and run in $\tilde{O}(m)$ time if $\mathbf{A}$ is "well-conditioned," where m is the number of nonzero entries in $\mathbf{A}$. All our matrices are well-conditioned or can be easily made well-conditioned. Section 4.7 describes these algorithms in greater detail, as well as a subtlety that needs addressing.

4.4 Primal-Dual Approach for Approximately Solving SDPs

This section describes a general primal-dual algorithm to compute a near-optimal solution to any SDP (and not just SDPs used in approximation algorithms). As an illustrative example we also describe its use for the SDP relaxation for MAXCUT.

A general SDP with n^2 variables (thought of as an $n \times n$ matrix variable $\mathbf{X}$) and m constraints, and its dual can be written as follows:

$$\begin{array}{rrcl}
 & \max\ \mathbf{C} \bullet \mathbf{X} & & \\
\forall j \in [m]: & \mathbf{A}_j \bullet \mathbf{X} & \leq & b_j \\
 & \mathbf{X} & \succeq & \mathbf{0}
\end{array}
\qquad\qquad
\begin{array}{rcl}
\min\ \mathbf{b} \cdot \mathbf{y} & & \\
\sum_{j=1}^{m} \mathbf{A}_j y_j & \succeq & \mathbf{C} \\
\mathbf{y} & \geq & \mathbf{0}
\end{array}$$

Here, $\mathbf{y} = \langle y_1, y_2, \ldots, y_m \rangle^\top$ are the dual variables and $\mathbf{b} = \langle b_1, b_2, \ldots, b_m \rangle^\top$. Also, $[m]$ is notation for the set $\{1, 2, \ldots, m\}$. Just as in the case of LPs, strong duality holds for SDPs under very mild conditions (always satisfied by the SDPs considered here) and the optima of the two programs coincide.

Note that a linear program is the special case whereby all the matrices involved are diagonal. (Aside: The recipe for writing SDP duals is syntactically similar to the one for LP dual, except instead of vector inequalities such as $\mathbf{a} \geq \mathbf{b}$, one uses matrix inequalities $\mathbf{A} \succeq \mathbf{B}$.)

For notational ease assume that $\mathbf{A}_1 = \mathbf{I}$ and $b_1 = R$. This serves to bound the trace of the solution: $\mathbf{Tr}(\mathbf{X}) \leq R$ and is thus a simple scaling constraint. It is very naturally present in SDP relaxations for combinatorial optimization problems.

We assume that our algorithm uses binary search to reduce optimization to feasibility. Let α be the algorithm's current guess for the optimum value of the SDP. It is trying to either construct a *PSD* matrix that is primal feasible and has value $> \alpha$, or a dual feasible solution whose value is at most $(1 + \delta)\alpha$ for some arbitrarily small $\delta > 0$.

As is usual in primal-dual algorithms, the algorithm starts with a trivial candidate for a primal solution, in this case the trivial *PSD* matrix (possibly infeasible) of trace R, viz. $\mathbf{X}^{(1)} = \frac{R}{n}\mathbf{I}$. Then it iteratively generates candidate primal solutions $\mathbf{X}^{(2)}, \mathbf{X}^{(3)}, \ldots$. At every step it tries to improve $\mathbf{X}^{(t)}$ to obtain $\mathbf{X}^{(t+1)}$, and in this it has help from an auxiliary algorithm, called the ORACLE, that tries to certify the validity of the current $\mathbf{X}^{(t)}$ as follows. ORACLE searches for a vector $\mathbf{y}$ from the polytope $\mathcal{D}_\alpha = \{\mathbf{y} :\ \mathbf{y} \geq 0,\ \mathbf{b} \cdot \mathbf{y} \leq \alpha\}$ such that

$$\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \ \geq\ 0. \tag{4.1}$$

If ORACLE succeeds in finding such a $\mathbf{y}$ then we claim $\mathbf{X}^{(t)}$ is either primal infeasible or has value $\mathbf{C} \bullet \mathbf{X}^{(t)} \leq \alpha$. The reason is that otherwise

$$\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \ \leq\ \sum_{j=1}^{m} b_j y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \ <\ \alpha - \alpha \ =\ 0,$$

which would contradict (4.1). Thus $\mathbf{y}$ implicitly contains some useful information to improve the candidate primal $\mathbf{X}^{(t)}$, and we use $\mathbf{y}$ to update $\mathbf{X}^{(t)}$ using a familiar-looking *matrix exponential update* rule (step 5 in the algorithm). Our observation about matrix exponentials ensures that the new matrix $\mathbf{X}^{(t+1)}$ is also *PSD*.

Lemma 3. *If there is no vector* $\mathbf{y} \in \mathcal{D}_\alpha$ *which satisfies (4.1), then a suitably scaled version of* $\mathbf{X}^{(t)}$ *is a primal feasible solution of objective value at least* α.

Proof: Consider the following linear program, and its dual:

$$\begin{array}{rcl} \max \sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j & & \\ \mathbf{b} \cdot \mathbf{y} & \leq & \alpha \\ \mathbf{y} & \geq & 0 \end{array} \qquad\qquad \begin{array}{rcl} \min \ \alpha\phi & & \\ \forall j: \ b_j \phi & \geq & (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) \\ \phi & \geq & 0 \end{array}$$

Now, since there is no vector $\mathbf{y} \in \mathcal{D}_\alpha$ which satisfies (4.1), we conclude that the optimum of the primal is less than $\mathbf{C} \bullet \mathbf{X}^{(t)}$. Thus, the optimum is finite, and so the dual is feasible and has the same optimum. Let ϕ^* be the optimum solution of the dual. Then $\alpha\phi^* \leq \mathbf{C} \bullet \mathbf{X}^{(t)}$. Furthermore, since $b_1 = R$ and $\mathbf{A}_1 \bullet \mathbf{X}^{(t)} = \mathbf{I} \bullet \mathbf{X}^{(t)} = \mathbf{Tr}(\mathbf{X}^{(t)}) = R$, we conclude that $\phi^* \geq 1$. Let $\mathbf{X}^* = \frac{1}{\phi^*}\mathbf{X}^{(t)}$. Then for all j, $\mathbf{A}_j \bullet \mathbf{X}^* \leq b_j$, and $\mathbf{C} \bullet \mathbf{X}^* \geq \alpha$. So $\mathbf{X}^*$ is a primal feasible solution of objective value at least α. $\Box$

The important point here is that the desired $\mathbf{y}$ is not dual feasible: in fact the Oracle can ignore the *PSD* constraint and its task consists of solving an LP with *just one non-trivial constraint* (the others are just non-negativity constraints)! Thus, one may hope to implement Oracle efficiently, and furthermore, even find $\mathbf{y}$ with *nice* properties, so that the algorithm makes fast progress towards feasibility. Now we formalize one aspect of *nice*-ness, the *width.* (Another aspect of niceness concerns a subtle condition that allows quick matrix exponentiation; see the discussion in the Section 4.7.)

Definition 2. *An* (ℓ, ρ)**-bounded** Oracle, *for parameters* $0 \leq \ell \leq \rho$, *is an algorithm that finds a vector* $\mathbf{y} \in \mathcal{D}_\alpha$ *that satisfies (4.1) such that either* $\sum_j \mathbf{A}_j y_j - \mathbf{C} \in [-\ell, \rho]$ *or* $\sum_j \mathbf{A}_j y_j - \mathbf{C} \in [-\rho, \ell]$ *holds. The value* ρ *is called the width of the* Oracle.

The constraint on width may seem like a backdoor way to bring in the semidefiniteness constraint into the oracle but it is more correctly viewed as a measure of the Oracle's effectiveness in helping the algorithm make progress (high width equals slow progress). This is analogous to the bounds on the responses of the Oracle when solving linear feasibility problems (Section 2.3.2). In all our applications the vector $\mathbf{y}$ will be computed using combinatorial ideas, such as simple case analysis or multicommodity flow. Thus our algorithms do adhere to the primal-dual philosophy. The algorithm is shown in Figure 4.1.

The following theorem bounds the number of iterations needed in the algorithm.

Theorem 13. *In the Primal-Dual SDP algorithm, assume that the* Oracle *never fails for* T *iterations. Let* $\bar{\mathbf{y}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{y}^{(t)}$, *and define* $\mathbf{y}^*$ *as* $y_1^* = \bar{y}_1 + \frac{\delta\alpha}{R}$, *and* $y_j^* = \bar{y}_j$ *for* $j \geq 2$. *Then* $\mathbf{y}^*$ *is a feasible dual solution with objective value at most* $(1+\delta)\alpha$.

Proof: In the Matrix Multiplicative Weights algorithm, the expected loss in round t is

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \ = \ \frac{1}{\ell + \rho}\left(\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \ell^{(t)}\mathbf{I}\right) \bullet \frac{1}{R}\mathbf{X}^{(t)} \ \geq \ \frac{\ell^{(t)}}{\ell + \rho}.$$

Primal-Dual Algorithm for Maximization SDPs
Given: Access to (ℓ, ρ)-bounded ORACLE, where $\ell \geq \frac{\delta\alpha}{4R}$.

1. Run the Matrix Multiplicative Weights algorithm with $\varepsilon = \frac{\delta\alpha}{2\ell R}$ up to $T = \frac{8\ell\rho R^2 \ln(n)}{\delta^2\alpha^2}$ rounds.
2. In each round t, run the ORACLE with candidate solution $\mathbf{X}^{(t)} = R\mathbf{P}^{(t)}$, where $\mathbf{P}^{(t)}$ is the density matrix generated by the Matrix Multiplicative Weights algorithm.
3. If the ORACLE fails, stop and output $\mathbf{X}^{(t)}$.
4. Else, let $\mathbf{y}^{(t)}$ be the vector generated by ORACLE.
5. Provide the loss matrix to the Matrix Multiplicative Weights algorithm:

$$\mathbf{M}^{(t)} \;=\; \left[\frac{1}{\ell+\rho}\sum_{j=1}^{m}\mathbf{A}_j y_j^{(t)} - \mathbf{C} + \ell^{(t)}\mathbf{I}\right], \tag{4.2}$$

where

$$\ell^{(t)} \;=\; \begin{cases} \ell & \text{if } \sum_{j=1}^{m}\mathbf{A}_j y_j^{(t)} - \mathbf{C} \in [-\ell, \rho] \\ -\ell & \text{if } \sum_{j=1}^{m}\mathbf{A}_j y_j^{(t)} - \mathbf{C} \in [-\rho, \ell] \end{cases}$$

Figure 4.1: The Primal-Dual SDP algorithm.

because the ORACLE finds a vector $\mathbf{y}^{(t)}$ such that $\sum_{j=1}^{m}(\mathbf{A}_j y_j^{(t)} \bullet \mathbf{X}^{(t)}) - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \geq 0$, and $\mathbf{I} \bullet \mathbf{X}^{(t)} = \mathbf{Tr}(\mathbf{X}^{(t)}) = R$. Plugging this bound in the inequality (3.3), we get

$$\sum_{t:\ \ell^{(t)}\geq 0} \frac{(1-\varepsilon)\ell^{(t)}}{\ell+\rho} + \sum_{t:\ \ell^{(t)}\leq 0} \frac{(1+\varepsilon)\ell^{(t)}}{\ell+\rho} \;\leq\; \frac{1}{\ell+\rho}\lambda_n\left(\sum_{t=1}^{T}\sum_{j=1}^{m}\mathbf{A}_j y_j^{(t)} - \mathbf{C} + \ell^{(t)}\mathbf{I}\right) + \frac{\ln n}{\varepsilon}.$$

Simplifying, and dividing by T, we get

$$0 \;\leq\; \lambda_n\left(\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{m}\mathbf{A}_j y_j^{(t)} - \mathbf{C}\right) + \varepsilon\ell + \frac{(\ell+\rho)\ln(n)}{\varepsilon T}.$$

Then using the specified values of ε and T, and the fact that $\bar{\mathbf{y}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{y}^{(t)}$, we get

$$-\frac{\delta\alpha}{R}\mathbf{I} \;\preceq\; \sum_{j=1}^{m}\mathbf{A}_j\bar{y}_j - \mathbf{C}. \tag{4.3}$$

Now, since $\mathbf{A}_1 = \mathbf{I}$, we get that $\mathbf{A}_1(\bar{y}_1 + \frac{\delta\alpha}{R}) + \sum_{j=2}^{m}\mathbf{A}_j\bar{y}_j \succeq \mathbf{C}$. Thus, the vector $\mathbf{y}^*$ as defined in the statement of the theorem is dual feasible, and has dual objective value

$$\frac{\delta\alpha}{R} \cdot b_1 + \mathbf{b} \cdot \bar{\mathbf{y}} \;\leq\; \delta\alpha + \frac{1}{T}\sum_{t=1}^{T} \mathbf{b} \cdot \mathbf{y}^{(t)} \;\leq\; (1+\delta)\alpha,$$

since $\mathbf{y}^{(t)} \in \mathcal{D}_\alpha$, and so $\mathbf{b} \cdot \mathbf{y}^{(t)} \leq \alpha$. $\Box$

As a warmup, we illustrate the use of Theorem 13 in the following simple application to the MaxCut SDP on a graph with n vertices and m edges. For comparison, the previous best algorithm for approximating the MaxCut SDP, by Klein and Lu [67], runs in time $\tilde{O}(mn)$. On graphs where the maximum degree is within a constant factor of the average degree, our algorithm runs in $\tilde{O}(m)$ time.

Theorem 14. *Let $G = (V, E)$ be a weighted graph with n nodes and m edges with maximum weighted degree Δ, and average weighted degree d. Then the* MaxCut *SDP can be approximated in $\tilde{O}(\frac{\Delta^2}{d^2}m)$ time.*

Proof: Let c_{ij} be the weight of edge $\{i,j\}$. Then the MaxCut SDP in vector and matrix form is as follows (the standard SDP has a factor of $\frac{1}{4}$ in the objective, but we disregard it since the optimum solution is the same):

$$\begin{aligned} \max \sum_{\{i,j\}\in E} c_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 & \\ \forall i \in [n]:\ \|\mathbf{v}_i\|^2 \;\leq\; 1 & \end{aligned} \qquad\qquad \begin{aligned} \max\ \mathbf{C} \bullet \mathbf{X} & \\ \forall i \in [n]:\ X_{ii} \;\leq\; 1 & \\ \mathbf{X} \;\succeq\; \mathbf{0} & \end{aligned}$$

The dual SDP is the following:

$$\begin{aligned} \min\ \textstyle\sum_{i=1}^{n} x_i & \\ \mathrm{diag}(\mathbf{x}) \;\succeq\; \mathbf{C} & \\ \forall i \in [n]:\ x_i \;\geq\; 0 & \end{aligned}$$

Here, $\mathbf{C}$ is the combinatorial Laplacian of the graph, and $\mathrm{diag}(\mathbf{x})$ is the diagonal matrix with the vector $\mathbf{x}$ on the diagonal. Since the maximum degree in the graph is Δ, we have $\mathbf{0} \preceq \mathbf{C} \preceq 2\Delta\mathbf{I}$.

We use the Primal-Dual SDP algorithm to solve this within a factor of $(1-\delta)$. Since $\sum_{\{i,j\}\in E} c_{ij} = \frac{1}{2}nd$, the MaxCut lies in the range $[\frac{1}{4}nd, \frac{1}{2}nd]$, since a random cut has expected size at least $\frac{1}{4}nd$. Let α^* be the SDP optimum. The Goemans-Williamson analysis indicates that $\alpha^* \in [4 \cdot \frac{1}{4}nd, \frac{4}{0.878} \cdot \frac{1}{2}nd] \subseteq [nd, 3nd]$. We perform a binary search for α^* in this range. Let α be our current estimate of α^*.

Now, if $\mathbf{X}$ is any primal feasible solution, then $\mathbf{Tr}(\mathbf{X}) \leq n$, so $R = n$. We now show how to implement an $(O(\Delta), O(\Delta))$-bounded Oracle, which ensures by Theorem 13 that the number of iterations is $O(\log n)$. Each invocation of Oracle and the matrix exponentiation step will take $\tilde{O}(m)$ time (the latter uses Lemma 24 and the fact that the number of non-zero matrix entries in $\mathbf{C}$ is $O(m)$). This yields the desired running time.

It remains to describe ORACLE. For every node i, let d_i denote the weighted degree, i.e. $d_i = \sum_j c_{ij}$. Given a candidate solution $\mathbf{X}$, it needs to find a vector $\mathbf{x} \geq \mathbf{0}$ such that $\sum_i x_i \leq \alpha$ and $\sum_i x_i X_{ii} - \mathbf{C} \bullet \mathbf{X} \geq 0$. Intuitively, to make $\sum_i x_i X_{ii}$ as large as possible, we should make x_i large for all i where X_{ii} is large.

1. If $\mathbf{C} \bullet \mathbf{X} \leq \alpha$, then set all $x_i = \frac{\alpha}{n}$. Then since $\sum_i X_{ii} = \mathbf{Tr}(\mathbf{X}) = n$ we have $\sum_i x_i X_{ii} - \mathbf{C} \bullet \mathbf{X} \geq \frac{\alpha}{n} \sum_i X_{ii} - \alpha = 0$. Note that $-2\Delta\mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) - \mathbf{C} \preceq O(d)\mathbf{I}$.

2. So assume $\mathbf{C} \bullet \mathbf{X} \geq \alpha$. Let $\mathbf{C} \bullet \mathbf{X} = \lambda\alpha$ for some $\lambda \geq 1$. Since $\mathbf{C} \preceq 4\Delta\mathbf{I}$, we have $\lambda\alpha = \mathbf{C} \bullet \mathbf{X} \leq 4n\Delta$. Since $\alpha \geq nd$, we have that $\lambda \leq \frac{4\Delta}{d}$. Let $S := \{i : X_{ii} \geq \lambda\}$, and let E_S be the set of edges with at least one endpoint in S. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the vectors obtained from the Cholesky decomposition of $\mathbf{X}$.

   Let $w := \sum_{i \in S} 4 d_i X_{ii}$. If $w \geq \delta\lambda\alpha$, then let $k = \frac{\lambda\alpha}{w}$. Note that $k \leq \frac{1}{\delta} = O(1)$. We set $x_i = 4kd_i$ for all $i \in S$, and $x_i = 0$ for all $i \notin S$. Then

$$\sum_i x_i \;=\; \sum_{i \in S} 4kd_i \;=\; \frac{\lambda\alpha \sum_{i \in S} 4d_i}{\sum_{i \in S} 4d_i \|\mathbf{v}_i\|^2} \;\leq\; \alpha$$

   since $\|\mathbf{v}_i\|^2 \geq \lambda$ for all $i \in S$. Then $\sum_i x_i X_{ii} - \mathbf{C} \bullet \mathbf{X} = \frac{\lambda\alpha}{w} \sum_{i \in S} 4d_i X_{ii} - \lambda\alpha = 0$. Furthermore, $-2\Delta\mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) - \mathbf{C} \preceq O(\Delta)\mathbf{I}$.

3. In every other case we show that we can easily construct a feasible primal solution from $\mathbf{X}$ with objective value at least $(1-\delta)\alpha$, which is therefore approximately optimum.

   Construct new vectors $\mathbf{v}'_i$ such that $\mathbf{v}'_i = \mathbf{v}_i$ for $i \notin S$, and $\mathbf{v}'_i = \mathbf{v}_0$ for $i \in S$, for an arbitrary fixed unit vector $\mathbf{v}_0$. Let $\tilde{\mathbf{X}}$ be the resulting Gram matrix of the $\mathbf{v}'_i$ vectors. Now we have $\mathbf{C} \bullet (\tilde{\mathbf{X}} - \mathbf{X}) \geq -\sum_{\{i,j\} \in E_S} c_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2$. We can lower bound the RHS by $-\delta\lambda\alpha$ as follows. Using the fact that for any i, j we have $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq 4\max\{\|\mathbf{v}_i\|^2, \|\mathbf{v}_j\|^2\}$, we get that

$$\begin{aligned}
\sum_{\{i,j\} \in E_S} c_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 &\leq \sum_{\{i,j\} \in E_S} 4c_{ij} \cdot \max\{\|\mathbf{v}_i\|^2, \|\mathbf{v}_j\|^2\} \\
&\leq \sum_{i \in S} \sum_j 4c_{ij} \|\mathbf{v}_i\|^2 \\
&= \sum_{i \in S} 4d_i \|\mathbf{v}_i\|^2 \\
&\leq \delta\lambda\alpha,
\end{aligned}$$

   since we assumed that $w \leq \delta\lambda\alpha$. The second inequality above follows because for every edge $\{i, j\} \in E_S$ such that $i \in S$ and $j \notin S$, we have $\max\{\|\mathbf{v}_j\|^2, \|\mathbf{v}_j\|^2\} = \|\mathbf{v}_i\|^2$.

   Thus, $\mathbf{C} \bullet \tilde{\mathbf{X}} \geq (1-\delta)\lambda\alpha$. Furthermore, for all i, $\tilde{\mathbf{X}}_{ii} = \|\mathbf{v}'_i\|^2 \leq \lambda$. So the matrix $\mathbf{X}^* = \frac{1}{\lambda}\tilde{\mathbf{X}}$ is a feasible primal solution with objective value at least $(1-\delta)\alpha$.

□

4.4.1 Extension to minimization problems

Since the rest of this chapter concerns minimization problems, we extend the above framework to it. A general minimization SDP can be written as follows:

$$\begin{array}{rrcl}
 & \min\ \mathbf{C} \bullet \mathbf{X} & & \\
\forall j \in [m]: & \mathbf{A}_j \bullet \mathbf{X} & \geq & b_j \\
 & \mathbf{X} & \succeq & \mathbf{0}
\end{array}
\qquad\qquad
\begin{array}{rcl}
\max\ \mathbf{b} \cdot \mathbf{y} & & \\
\sum_{j=1}^{m} \mathbf{A}_j y_j & \preceq & \mathbf{C} \\
\mathbf{y} & \geq & \mathbf{0}
\end{array}$$

Here, we assume that $\mathbf{A}_1 = -\mathbf{I}$ and $b_1 = -R$, which translates to the trace bound $\mathbf{Tr}(\mathbf{X}) \leq R$.

The Primal-Dual algorithm in this case is essentially the same as before, with a few changes. First, given a candidate solution $\mathbf{X}$, the Oracle needs to find a vector $\mathbf{y}$ from the polytope $\mathcal{D}_\alpha = \{\mathbf{y} : \ \mathbf{y} \geq 0,\ \mathbf{b} \cdot \mathbf{y} \geq \alpha\}$ such that $\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}) y_j - (\mathbf{C} \bullet \mathbf{X}) \leq 0$. The algorithm now is the same as the one described before, except that the loss matrix in (4.2) is negated, i.e.

$$\mathbf{M}^{(t)} \ = \ \frac{-1}{\ell + \rho} \left[\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{C} + \ell^{(t)} \mathbf{I} \right].$$

Finally, and most important, we allow the Oracle to find a matrix $\mathbf{F}^{(t)}$ such that for all primal feasible $\mathbf{X}$, we have $\mathbf{F}^{(t)} \bullet \mathbf{X} \leq \mathbf{C} \bullet \mathbf{X}$, and a vector $\mathbf{y}^{(t)} \in \mathcal{D}_\alpha$ such that

$$\textstyle\sum_{j=1}^{m} (\mathbf{A}_j \bullet \mathbf{X}^{(t)}) y_j^{(t)} - (\mathbf{F}^{(t)} \bullet \mathbf{X}) \ \leq \ 0. \tag{4.4}$$

In this case, we use

$$\mathbf{M}^{(t)} \ = \ \frac{-1}{\ell + \rho} \left[\sum_{j=1}^{m} \mathbf{A}_j y_j^{(t)} - \mathbf{F}^{(t)} + \ell^{(t)} \mathbf{I} \right].$$

In other words, we can replace $\mathbf{C}$ by $\mathbf{F}^{(t)}$, which is under our control. Note that if $\mathbf{F}^{(t)} \preceq \mathbf{C}$, then because any primal feasible $\mathbf{X}$ is *PSD*, we have $\mathbf{F}^{(t)} \bullet \mathbf{X} \leq \mathbf{C} \bullet \mathbf{X}$. So it suffices to find $\mathbf{F}^{(t)} \preceq \mathbf{C}$. (The reason for allowing $\mathbf{F}$ in the framework is to reduce width; see Section 4.5.)

We now present a theorem analogous to Theorem 13.

Theorem 15. *In the modified Primal-Dual algorithm for a minimization SDP as described above, suppose the* Oracle *never fails for T iterations. Let $\bar{\mathbf{y}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{y}^{(t)}$, and define $\mathbf{y}^*$ as $y_1^* = \bar{y}_1 + \frac{\delta\alpha}{R}$, and $y_j^* = \bar{y}_j$ for $j \geq 2$. Then $\mathbf{y}^*$ is a dual solution that proves that the primal optimum is at least $(1 - \delta)\alpha$.*

Proof: The proof is on the same lines as the proof of Theorem 13. Following the proof till (4.3), we obtain the following inequality

$$-\frac{\delta\alpha}{R} \mathbf{I} \ \preceq \ -\sum_{j=1}^{m} \mathbf{A}_j y_j + \bar{\mathbf{F}},$$

where $\bar{\mathbf{F}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{F}^{(t)}$. Now, since $\mathbf{A}_1 = -\mathbf{I}$, we get that $\mathbf{A}_1(\bar{y}_1 + \frac{\delta\alpha}{R}) + \sum_{j=2}^{m}\mathbf{A}_j\bar{y}_j \preceq \bar{\mathbf{F}}$, i.e. $\sum_{j=1}^{m}\mathbf{A}_j y_j^* \preceq \bar{\mathbf{F}}$, for the vector $\mathbf{y}^*$ defined in the statement of the theorem. Now let $\mathbf{X}^*$ be the optimal solution to the primal, and let the primal optimum be α^*. Then we have

$$\bar{\mathbf{F}} \bullet \mathbf{X}^* \;=\; \frac{1}{T}\sum_{t=1}^{T}\mathbf{F}^{(t)} \bullet \mathbf{X}^* \;\leq \frac{1}{T}\sum_{t=1}^{T}\mathbf{C} \bullet \mathbf{X}^* \;=\; \mathbf{C} \bullet \mathbf{X}^* \;=\; \alpha^*.$$

On the other hand, since $\mathbf{X}^* \succeq \mathbf{0}$, we have

$$\bar{\mathbf{F}} \bullet \mathbf{X}^* \;\geq\; \sum_{j=1}^{m} y_j^*(\mathbf{A}_j \bullet \mathbf{X}^*) \;\geq\; \sum_{j=1}^{m} y_j^* b_j \;=\; \frac{\delta\alpha}{R}\cdot b_1 + \frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{T} {y_j}^{(t)} b_j \;\geq\; (1-\delta)\alpha,$$

because $\mathbf{y}^{(t)} \in \mathcal{D}_\alpha$. This shows that $\alpha^* \geq (1-\delta)\alpha^*$, as required. □

4.4.2 Approximate Oracles

Just as in the case of linear programs (Section 2.3.2), the Primal-Dual algorithms discussed here also work with approximate ORACLEs. We will discuss this only in the context of maximization SDPs, the extension to minimization SDPs is analogous.

If α is the current estimate of the optimum, define a δ-approximate ORACLE to be an algorithm that, given a candidate solution $\mathbf{X}$, finds a vector $\mathbf{y} \in \mathcal{D}_\alpha$ such that $\sum_{j=1}^{m} y_j(\mathbf{A}_j \bullet \mathbf{X}) - (\mathbf{C} \bullet \mathbf{X}) \geq -\delta\alpha$. We now have the following theorem:

Theorem 16. *Suppose that there exists an (ℓ,ρ)-bounded, $\frac{\delta}{3}$-approximate* ORACLE. *Assume that the Primal-Dual algorithm is run with this* ORACLE *with the parameters $\ell = \frac{\delta\alpha}{3\ell R}$ and $T = \frac{18\ell\rho R^2\ln(n)}{\delta^2\alpha^2}$, and that the* ORACLE *never fails for T iterations. Let $\bar{\mathbf{y}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{y}^{(t)}$, and define $\mathbf{y}^*$ as $y_1^* = \bar{y}_1 + \frac{\delta\alpha}{R}$, and $y_j^* = \bar{y}_j$ for $j \geq 2$. Then $\mathbf{y}^*$ is a feasible dual solution with objective value at most $(1+\delta)\alpha$.*

PROOF: The proof is identical to that of Theorem 13. The only difference is that the expected loss in round t is

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \;=\; \frac{1}{\ell+\rho}\left(\sum_{j=1}^{m}\mathbf{A}_j y_j^{(t)} - \mathbf{C} + \ell^{(t)}\mathbf{I}\right) \bullet \frac{1}{R}\mathbf{X}^{(t)} \;\geq\; \frac{-\frac{\delta\alpha}{3R} + \ell^{(t)}}{\ell+\rho},$$

since the ORACLE finds a vector $\mathbf{y}^{(t)}$ such that $\sum_{j=1}^{m} {y_j}^{(t)}(\mathbf{A}_j \bullet \mathbf{X}^{(t)}) - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \geq -\frac{\delta\alpha}{3}$. The rest of the proof is the same as before. □

4.5 Primal-dual approximation algorithms via SDPs

In this section, we apply our general framework of Section 4.4.1 to design faster approximation algorithms for a host of NP-hard problems for which thus far we needed to solve SDPs. The important difference from Section 4.4 is that we do not try to solve the SDP to near-optimality as that would take too long. Instead, we use the framework to produce a dual solution of a certain value together with an *integer* primal solution whose cost is $O(\log n)$ or $O(\sqrt{\log n})$ factor higher than the value of the dual solution.

We now outline how to implement the Oracle, which, as mentioned in the Section 4.2, uses the known SDP rounding techniques (stemming from the Arora, Rao, Vazirani paper and subsequent work) for the problem in question. At each step the Oracle starts by applying the rounding algorithm on the current primal solution. If the rounding succeeds, then it yields a good integer solution. Otherwise, it actually uncovers gross deviations from feasibility in the candidate primal solution, which can be used as "feedback" (the vector $\mathbf{y}$ in the generic algorithm of Section 4.4.1) to improve the primal. To aid the Oracle in finding deviations from feasibility, we augment most SDPs for the problems we consider with extra constraints, all of which are easily implied by linear combinations of the original constraints. This gives us extra dual variables to play with, and the added flexibility makes the description of the Oracle cleaner.

The main point is that the rounding could potentially succeed even though the primal is quite far from feasibility, which is why the algorithm may end only with a feasible dual solution. (Of course, the general framework of Section 4.4 could be used to continue the algorithm until it also finds a feasible primal, but the Oracle's width parameter increases, raising the running time a lot.) Thus the running time of the Oracle (and the algorithm) depends upon how efficiently it can compute the "feedback" when the rounding algorithm fails, and often this running time is much less for $O(\log n)$-approximation as compared to a $O(\sqrt{\log n})$-approximation.

The key insights in the implementation of the Oracle are that: (a) the feedback, in the form of dual weights, can be viewed as *Laplacians* of certain weighted graphs, whose spectral behavior is easy to understand, and this allows us to bound the width, and (b) the spectral behavior (in other words, the width bound) is improved by careful choice of these weights, and this was the main reason for allowing $\mathbf{F}$ instead of $\mathbf{C}$ in Theorem 15 in the first place.

4.5.1 Undirected Balanced Separator

We are given a capacitated graph $G = (V, E)$ with $|V| = n$, $|E| = m$, and capacity c_e on edge $e \in E$. For a subset $S \subseteq V$, let $\bar{S} = V \setminus S$. A cut $(S, \bar{S})$ is called *c-balanced* if $|S| \geq cn$, and $|\bar{S}| \geq cn$. The minimum c-Balanced Separator problem is to find the c-balanced cut with minimum capacity. A t pseudo-approximation to the minimum c-Balanced Separator is a c'-balanced cut for some other constant c' whose expansion is within a factor t of that of the minimum c-Balanced Separator.

Before we run the algorithm, we pre-process the graph using the sparsification algorithm of Benczúr and Karger [22]. This algorithm randomly chooses edges using a non-uniform sampling approach that leaves all cuts in that leaves very few (weighted) edges while approximately preserving the value of any cut in the graph:

Theorem 17 ([22])**.** *There is an algorithm which given a weighted graph with n nodes and m edges with non-zero weight, produces in $O(m \log^3 n)$ time a new graph on the same set of nodes which has only $O(n \log(n)/\varepsilon^2)$ edges with non-zero weight, such that the value of any cut in the original graph is preserved within a multiplicative factor of $1 \pm \varepsilon$.*

Since we want to approximate the minimum c-Balanced Separator, we can run this algorithm (with $\varepsilon = 0.5$, say) to reduce the number of non-zero edges to $\tilde{O}(n)$. From now on, we assume the graph has already been pre-processed in this manner.

Theorem 18.

1. *An $O(\log n)$ pseudo-approximation to the minimum c-Balanced Separator can be computed in $\tilde{O}(m + n^{1.5})$ time using $O(\log^2(n))$ single commodity flow computations.*

2. *An $O(\sqrt{\log n})$ pseudo-approximation to the minimum c-Balanced Separator can be computed in $\tilde{O}(n^2)$ time using $O(\log n)$ multicommodity flow computations.*

Proof: First, we consider the well-known c-Balanced Separator SDP (see [18]). To increase the flexibility in handling the candidate solution $\mathbf{X}$ we throw in additional constraints, which are not part of the standard SDP specification, but which are implied by it. (The reason is that the candidate primal solutions at intermediate steps are not feasible, so these constraints are actually helpful to Oracle.) We assign vectors $\mathbf{v}_i$ to the nodes in G. Let $\mathbf{X}$ be the Gram matrix of these vectors. Below, we write the SDP in vector and its corresponding matrix form. Let $\mathbf{C}$ be the combinatorial Laplacian of the graph, and for any subset S of the nodes, let $\mathbf{K}_S$ be the Laplacian of the graph where all nodes in S are connected by edges, and all other edges are absent. We will only consider sets S of size at least $(1-\varepsilon)n$ (for some ε to be fixed later), and we use the notation "$\forall S$" to mean only such sets. Let $p = (i_1, i_2, \ldots, i_k)$ be a generic path of nodes in the complete graph, and let $\mathbf{T}_p$ be the difference of the Laplacian of p and that of a single edge connecting its endpoints.

$$\begin{array}{rrcl}
 & \min \sum_{e=\{i,j\}\in E} c_e \|\mathbf{v}_i - \mathbf{v}_j\|^2 & & \\
\forall i: & \|\mathbf{v}_i\|^2 & = & 1 \\
\forall p: & \sum_{j=1}^{k-1} \|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \\
\forall S: & \sum_{i,j\in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 & \geq & an^2
\end{array}
\qquad
\begin{array}{rrcl}
 & \min \mathbf{C} \bullet \mathbf{X} & & \\
\forall i: & X_{ii} & = & 1 \\
\forall p: & \mathbf{T}_p \bullet \mathbf{X} & \geq & 0 \\
\forall S: & \mathbf{K}_S \bullet \mathbf{X} & \geq & an^2 \\
 & \mathbf{X} & \succeq & \mathbf{0}
\end{array}$$

The path inequalities, $\mathbf{T}_p \bullet \mathbf{X} \geq 0$, are usually omitted from the standard c-Balanced Separator SDP, which only involves triangle inequalities. However, they are implied by the triangle inequalities, so we retain them. Similarly, the original SDP only has the single spreading constraint $\mathbf{K}_V \bullet \mathbf{X} \geq 4c(1-c)n^2$. If $S \subseteq V$ of size at least $(1-\varepsilon)n$, then $(\mathbf{K}_V - \mathbf{K}_S) \bullet \mathbf{X} = \sum_{i \in S, j \in V} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq 4\varepsilon n^2$ since $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq 2\|\mathbf{v}_i\|^2 + 2\|\mathbf{v}_j\|^2 = 4$. Thus, the constraints $\mathbf{K}_S \bullet \mathbf{X} \geq an^2$ where $a = 4[c(1-c) - \varepsilon]$ are also implied, and we retain them. We set $\varepsilon = \frac{1}{2}c(1-c)$, so that $a = 2c(1-c)$.

The optimum of this SDP divided by 4 is a lower bound on the minimum c-Balanced Separator. This can be seen as follows. For a cut $(S, \bar{S})$, select an arbitrary unit vector $\mathbf{v}_0$, and set $\mathbf{v}_i = \mathbf{v}_0$ for all $i \in S$, and $\mathbf{v}_j = -\mathbf{v}_0$ for all $j \in \bar{S}$. Then $\|\mathbf{v}_i - \mathbf{v}_j\|^2 = 4$ if

$i \in S$ and $j \in -S$ (or *vice-versa*), and 0 otherwise. Thus, the objective function is

$$\sum_{e=\{i,j\}\in E} c_e \|\mathbf{v}_i - \mathbf{v}_j\|^2 \; = \; 4E(S, \bar{S}).$$

The dual SDP is the following. It has variables x_i for every node i, f_p for every path p, and z_S for every set S of nodes of size at least $(1-\varepsilon)n$. Let $\mathrm{diag}(\mathbf{x})$ be the diagonal matrix with the vector $\mathbf{x}$ on the diagonal.

$$\begin{aligned} \max \; \textstyle\sum_i x_i + an^2 \textstyle\sum_S z_S & \\ \mathrm{diag}(\mathbf{x}) + \textstyle\sum_p f_p \mathbf{T}_p + \textstyle\sum_S z_S \mathbf{K}_S \; &\preceq \; \mathbf{C} \\ \forall p, S: \quad f_p, z_S \; &\geq \; 0 \end{aligned}$$

Let the current guess for the optimum in the Primal-Dual algorithm be α. Let $\mathbf{X}$ be a candidate solution generated by the Primal-Dual algorithm. Note that $\mathbf{Tr}(\mathbf{X}) = n$. The Oracle needs to find variables x_i, $f_p \geq 0$, $z_S \geq 0$ and a matrix variable $\mathbf{F} \preceq \mathbf{C}$ (c.f. Theorem 15) such that $\sum_i x_i + an^2 \sum_S z_S \geq \alpha$ which satisfies

$$\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} + \textstyle\sum_p f_p (\mathbf{T}_p \bullet \mathbf{X}) + \textstyle\sum_S z_S (\mathbf{K}_S \bullet \mathbf{X}) - (\mathbf{F} \bullet \mathbf{X}) \; \leq \; 0.$$

If it succeeds in doing this, then the matrix returned as feedback is $\mathrm{diag}(\mathbf{x}) + \sum_p f_p \mathbf{T}_p + \sum_S z_S \mathbf{K}_S - \mathbf{F}$.

Our implementation of Oracle works as follows. Given a candidate solution $\mathbf{X}$, the Oracle checks first whether all the X_{ii} are $O(1)$. If a significant fraction of them aren't, then the Oracle can punish the solution $\mathbf{X}$ by setting the x_i's appropriately, in a similar way as done for MaxCut. Next, it checks whether $\mathbf{K}_S \bullet \mathbf{X} \geq \Omega(n^2)$ for some set S of nodes. If it isn't, then by setting the corresponding z_S appropriately, it can punish $\mathbf{X}$.

The most complicated case is when almost all X_{ii} are $O(1)$, and $\mathbf{K}_S \bullet \mathbf{X} \geq \Omega(n^2)$. In this case, we perform a flow computation, and interpret the f_p variables as a multicommodity flow in the graph.

Now, we describe in detail the implementation of a $(O(\frac{\alpha}{n}), \tilde{O}(\frac{\alpha}{n}))$-bounded Oracle. Given a candidate solution $\mathbf{X}$, the Oracle runs the following steps (all unspecified variables, including $\mathbf{F}$, are set to 0):

1. Assume, without loss of generality, that $X_{11} \leq X_{22} \leq \cdots \leq X_{nn}$. Let $h = (1-\varepsilon)n + 1$. If $X_{hh} \geq 2$, then set $x_i = -\frac{\alpha}{\varepsilon n}$ for $i \geq k$, and $x_i = \frac{2\alpha}{(1-\varepsilon)n}$ for $i < k$. Then

   $$\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} \; = \; \sum_{i \geq k} -\frac{\alpha}{\varepsilon n} X_{ii} + \sum_{i \geq k} \frac{2\alpha}{(1-\varepsilon)n} X_{ii} \; \leq \; -\frac{\alpha}{\varepsilon n} \cdot 2 \cdot \varepsilon n + \frac{2\alpha}{(1-\varepsilon)n} \cdot (n - 2\varepsilon n) \; \leq \; 0.$$

   Finally, since all $x_i = O(\frac{\alpha}{n})$, $\|\mathrm{diag}(\mathbf{x})\| \leq O(\frac{\alpha}{n})$.

2. Now we assume that for all but εn exceptional nodes i, $X_{ii} \leq 2$. Let W be the set of all the exceptional nodes, and let $S := V \setminus W$. Note that $|S| \geq (1-\varepsilon)n$, so we have

the constraint $\mathbf{K}_S \bullet \mathbf{X} \geq an^2$ in the SDP. If $\mathbf{K}_S \bullet \mathbf{X} \leq \frac{an^2}{2}$, then choose $z_S = \frac{2\alpha}{an^2}$ all $x_i = -\frac{\alpha}{n}$. Then

$$\left(-\frac{\alpha}{n}\mathbf{I} + \frac{2\alpha}{an^2}\mathbf{K}_S\right) \bullet \mathbf{X} \ \leq \ \alpha - \alpha \ = \ 0.$$

Also, since $\mathbf{0} \preceq \mathbf{K}_S \preceq n\mathbf{I}$, we have $\| -\frac{\alpha}{n}\mathbf{I} + \frac{2\alpha}{an^2}\mathbf{K}_S\| \leq O(\frac{\alpha}{n})$.

3. Now assume that $\mathbf{K}_S \bullet \mathbf{X} \geq \frac{an^2}{2}$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}$. Note that for all nodes $i \in S$, we have $\|\mathbf{v}_i\|^2 \leq 2$. Also, $\mathbf{K}_S \bullet \mathbf{X} \geq \frac{an^2}{2}$ implies that $\sum_{i,j \in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{an^2}{2}$.

   Now, the algorithm nudges $\mathbf{X}$ towards satisfying the path inequality constraints. At first, it is even unclear how to *check* at any time that the path inequalities are satisfied, since there are so many of them. For this we use multicommodity flow. First, some notation. For a flow which assigns value f_p to path p define f_e to be the flow on edge e, i.e. $f_e := \sum_{p \ni e} f_p$. Define f_i to be the total flow from node i, i.e. $f_i = \sum_{p \in \mathcal{P}_i} f_p$ where $\mathcal{P}_i$ is the set of paths starting from i. Finally, define f_{ij} to be the total flow between nodes i, j, i.e. $f_{ij} = \sum_{p \in \mathcal{P}_{ij}} f_p$, where $\mathcal{P}_{ij}$ is the set of paths from i to j. A *valid d-regular flow* is one that satisfies the capacity constraints: $\forall e: \ f_e \leq c_e$, and $\forall i: \ f_i \leq d$.

   Our main tool is the following lemma, which shows that either we can find a nice flow to make progress (i.e., give substantial "feedback"), or a cut with the desired expansion (i.e., a near-optimal integer solution). This proof of this lemma appears after the current proof.

   Lemma 4. *Let $S \subseteq V$ be a set of nodes of size $\Omega(n)$. Suppose we are given, for all $i \in S$, vectors $\mathbf{v}_i$ of length $O(1)$, such that $\sum_{i,j \in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$, and a quantity α. Then:*

   (a) There is an algorithm, which, using a single max-flow computation, either outputs a valid $O(\frac{\log(n)\alpha}{n})$-regular flow $\mathbf{f} = \langle f_p \rangle_p$ such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or a c'-balanced cut of expansion $O(\log(n)\frac{\alpha}{n})$.

   (b) There is an algorithm, which, using a single multicommodity flow computation, either outputs a valid $O(\frac{\alpha}{n})$-regular flow $\mathbf{f} = \langle f_p \rangle_p$ such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or a c'-balanced cut of expansion $O(\sqrt{\log(n)}\frac{\alpha}{n})$.

   We apply the two algorithms of Lemma 4 to the set S, corresponding to the two cases of Theorem 18. In case we find a cut with the desired expansion, then we stop. Otherwise, we get a valid d-regular flow which satisfies $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, where $d = O(\frac{\log(n)\alpha}{n})$ or $O(\frac{\alpha}{n})$ depending on the two cases.

   Now, we set $\mathbf{F}$ to be the Laplacian of the weighted graph with edge weights f_e. The capacity constraints $f_e \leq c_e$ imply that $\mathbf{F} \preceq \mathbf{C}$, because $\mathbf{C} - \mathbf{F}$ is the Laplacian of the graph with edge weights $c_e - f_e$, and is hence *PSD*. Let $\mathbf{D}$ be the Laplacian of the demand graph, i.e. the complete weighted graph where only edges $\{i, j\}$ with $i \in S$ and $j \in T$ have weight f_{ij}, and the rest have 0 weight.

Now, we have that $\mathbf{D} \bullet \mathbf{X} = \sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$. Then we set all $x_i = \frac{\alpha}{n}$, and all $z_S = 0$. It can be checked easily that $\sum_p f_p \mathbf{T}_p = \mathbf{F} - \mathbf{D}$. Thus, the "feedback" matrix becomes

$$\mathrm{diag}(\mathbf{x}) + \mathbf{F} - \mathbf{D} - \mathbf{F} \;=\; \mathrm{diag}(\mathbf{x}) - \mathbf{D}.$$

Then $\left(\frac{\alpha}{n}\mathbf{I} - \mathbf{D}\right) \bullet \mathbf{X} \leq \alpha - \alpha = 0$. Also, since the flow is d-regular, we have $\mathbf{0} \preceq \mathbf{D} \preceq 2d\mathbf{I}$. Hence, $-2d\mathbf{I} \;\preceq\; \frac{\alpha}{n}\mathbf{I} - \mathbf{D} \;\preceq\; \frac{\alpha}{n}\mathbf{I}$.

We now estimate the running time for each algorithm.

1. In this case, we have $\ell = O(\frac{\alpha}{n}$, $\rho = O(\frac{\log(n)\alpha}{n})$ and $R = n$. Thus, the number of iterations, from Theorem 13 is $O(\log^2(n))$. Each iteration involves at most one max-flow computation, which can be done in $\tilde{O}(n^{1.5})$ time since there are $\tilde{O}(n)$ edges, using the algorithm of Goldberg and Rao [47].

   In each iteration, we compute an approximation to the Cholesky decomposition of the matrix exponential by projecting on a random $O(\log n)$ dimensional subspace. Furthermore, since there are only $O(\log^2(n))$ iterations and each iteration adds at most $\tilde{O}(n^{1.5})$ demand pairs in the max-flow computation, the matrix to be exponentiated has only $\tilde{O}(n^{1.5})$ non-zero entries, and thus the product of the matrix with a given vector can be computed in $\tilde{O}(n^{1.5})$ time (we are disregarding the $\mathbf{K}_S$ matrices here, but it is easy to compute the product $\mathbf{K}_S\mathbf{u}$ for any vector $\mathbf{u}$ in $O(n)$ time, since $\mathbf{K}_S\mathbf{u} = \sum_{i,j \in S}(u_i - u_j)^2 = |S| \sum_i u_i^2 - (\sum_{i \in S} u_i)^2)$. Overall, Lemma 23 shows that the matrix exponentiation step can be done in $\tilde{O}(n^{1.5})$ time.

   Overall, the running time, accounting for the initial graph sparsification, becomes $\tilde{O}(m + n^{1.5})$.

2. In this case, we have $\ell = \rho = O(\frac{\alpha}{n})$ and $R = n$. Thus, the number of iterations, from Theorem 13 is $O(\log(n))$. Each iteration involves at most one maximum multicommodity flow computation, which can be done in $\tilde{O}(n^{1.5})$ time since there are $\tilde{O}(n)$ edges, using the algorithm of Fleischer [40].

   In each iteration, Fleischer's multicommodity flow algorithm adds at most $\tilde{O}(n)$ demand pairs. Since there are only $O(\log n)$ iterations, the matrix to be exponentiated has only $\tilde{O}(n)$ non-zero entries (again disregarding the $\mathbf{K}_S$ matrices). Overall, Lemma 23 shows that the matrix exponentiation step can be done in $\tilde{O}(n)$ time.

   Overall, the running time becomes $\tilde{O}(n^2)$.

□

Finally, we turn to the proof of Lemma 4.

Proof:[Lemma 4]

Part 1. We seek a valid d-regular flow $\mathbf{f} = \langle f_p \rangle_p$ for $d := \frac{\beta \log(n) \cdot \alpha}{n}$ where β is a sufficiently large constant to be chosen later. For this, we choose a direction represented by a unit vector $\mathbf{u}$ at random from the uniform distribution over the unit sphere (i.e. according to the Haar measure).

Since $\mathbf{K}_S \bullet \mathbf{X} \geq \Omega(n^2)$, we have that $\sum_{ij \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$. Then Lemma 14 in Section 4.6 shows that we can find sets L and R of size cn each, for some constant $c > 0$, such that for all $i \in L$ and $j \in R$, we have $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$ for some constant $\sigma > 0$.

Now, we use the Lemma 13 regarding the Gaussian nature of random projections for $t = \Theta(\sqrt{\log n})$, to conclude that with very high probability, for any pair of nodes i, j, we have that $|(\mathbf{v}_i - \mathbf{v}_j) \cdot \mathbf{u}| \leq O(\sqrt{\log(n)}) \cdot \frac{\|\mathbf{v}_i - \mathbf{v}_j\|}{\sqrt{n}}$. Since with constant probability, we have $|(\mathbf{v}_i - \mathbf{v}_j) \cdot \mathbf{u}| \geq \frac{\sigma}{\sqrt{n}}$ for all $i \in L$, $j \in R$, we conclude that the bound $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{\gamma}{\log(n)}$ for some constant γ holds with constant probability.

Assuming this is the case, we connect all nodes in L to a single source and connect all nodes in R to a single sink with edges of capacity d each. Let $\mathbf{f} = \langle f_p \rangle_p$ be the max flow in this network, where every edge e has its original capacity c_e.

Suppose the total flow obtained is at least $\frac{c\beta}{2} \log(n) \cdot \alpha$. We may assume that all the flow originates from some node $i \in L$ and ends at some node $j \in R$. Then we have

$$\textstyle\sum_{i \in L, j \in R} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\geq\; \frac{c\beta}{2} \log(n) \cdot \alpha \times \frac{\gamma}{\log(n)} \;=\; \alpha$$

if we choose $\beta = \frac{2}{c\gamma}$.

Now suppose that the total flow obtained in the previous step is less than $\frac{c\beta}{2} \log(n) \cdot \alpha$. By the max-flow-min-cut theorem, the cut obtained also has capacity at most $O(\log(n) \cdot \alpha)$. Note that this cut will be $c/2$-balanced, since at most $\frac{c\beta}{2} \log(n) \cdot \alpha / d = cn/2$ source (and sink) edges can be cut. Thus, the expansion of the cut is $O(\log(n) \cdot \frac{\alpha}{n})$.

Part 2: We seek a valid d-regular flow $\mathbf{f} = \langle f_p \rangle_p$ for $d := \frac{\beta\alpha}{n}$ where β is a sufficiently large constant to be chosen later. Now we consider a maximum multicommodity flow problem, where the demand pairs are nodes $i, j \in S$ such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s to be specified later. Let D be the set of such pairs of nodes. We seek a valid d-regular flow for $d := \frac{\beta\alpha}{n}$, which maximizes the total flow $\sum_{ij \in D} f_{ij}$. The d-regularity condition can be equivalently obtained by adding an artificial edge $\{i, i'\}$ of capacity $c_{ii'} = d$ to every node i, where i' is a new node, and considering the new set of demand pairs $D' = \{\{i', j'\} : \{i, j\} \in D\}$. Let the new graph be G'. The multicommodity flow problem can be expressed by the following LP, and its dual. Here, p refers to a generic path in G' between some node pair $\{i', j'\} \in D'$. Any such path consists of the edge $\{i', i\}$, followed by a path connecting i to j in G, and then the edge $\{j, j'\}$.

$$\begin{aligned} &\max \sum_p f_p \\ \forall e :\; &\sum_{p \ni e} f_p \;\leq\; c_e \\ \forall p :\; &f_p \;\geq\; 0 \end{aligned} \qquad\qquad \begin{aligned} &\min \sum_e c_e w_e \\ \forall p :\; &\sum_{e \in p} w_e \;\geq\; 1 \\ \forall e :\; &w_e \;\geq\; 0 \end{aligned}$$

Using Fleischer's algorithm [40], this problem can be solved up to any given constant factor, say $\frac{1}{2}$. Now we check if the flow obtained satisfies $\sum_{ij \in D} f_{ij} \geq \frac{\alpha}{s}$. If it does, then we have $\sum_{ij \in D} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$.

Assume now that the total flow obtained $\sum_{ij} f_{ij} < \frac{\alpha}{s}$. Since we chose an approximation factor of $\frac{1}{2}$ in Fleischer's algorithm, this means that the optimum of the multicommodity flow problem is less than $\frac{2\alpha}{s}$. Let w_e be the optimal set of edge weights to the dual of the LP. For every node i, set $s_i := w_{ii'}$. Since the optimum value is less than $\frac{2\alpha}{s}$, we get that these edge weights satisfy the following conditions:

$$\sum_e c_e w_e + \sum_i \frac{\beta\alpha}{n} s_i \;\leq\; \frac{2\alpha}{s},$$

and for any demand pair $i, j \in D$ and a path p connecting them,

$$s_i + s_j + \sum_{e\in p} w_e \;\geq\; 1.$$

Now we apply the algorithm from the following theorem (with the vectors $\mathbf{v}_i$ scaled down by 2 to ensure their length is at most 1). The proof, which appears in Section 4.6, can be derived using the techniques of Arora, Rao and Vazirani [18] and Lee [73]. Note that the number edges, m, is $\tilde{O}(n)$ here since we sparsified the graph first.

Theorem 19. *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors of length at most 1, such that $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq an^2$. Let w_e be weights on edges and nodes and let $\alpha = \sum_e c_e w_e$. Then there is an algorithm which runs in $\tilde{O}(m^{1.5})$ time and finds a cut of value C which is c-balanced for some constant c, such that there exists a pair of nodes i, j with the property that the graph distance between i and j is at most $O(\sqrt{\log n} \cdot \frac{\alpha}{C})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ where s is a constant which only depends on a. Furthermore, this is true even if any fixed set of τn nodes are prohibited from being i or j, for some small constant τ.*

We have $\sum_e c_e w_e \leq \frac{2\alpha}{s} = O(\alpha)$. At most τn nodes have $s_i \geq \frac{2}{\beta\tau s}$. Let these nodes form the forbidden set in the theorem. We run the algorithm from the theorem to obtain a cut of value C. Let i, j be the pair of nodes whose existence is guaranteed by the theorem. The value s is defined to be the lower bound on $\|\mathbf{v}_i - \mathbf{v}_j\|^2$ from the theorem; thus, i, j is a demand pair in D. Choose $\beta = \frac{8}{\tau s}$. Then $s_i, s_j \leq \frac{1}{4}$, and we have

$$s_i + s_j + O\left(\sqrt{\log n}\frac{\alpha}{C}\right) \;\geq\; 1 \quad\Rightarrow\quad C \;=\; O(\sqrt{\log n} \cdot \alpha).$$

Since the cut is $\Omega(1)$-balanced, its expansion is at most $O(\sqrt{\log n}\frac{\alpha}{n})$. □

4.5.2 Undirected Sparsest Cut

We have the same setup as for undirected Balanced Separator. The Sparsest Cut in a graph $G = (V, E)$ is the cut $(S, \bar{S})$ with minimum expansion, $\frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}$. As before, we assume that the graph has been pre-processed using the algorithm of Benczúr and Karger [22] to have only $\tilde{O}(n)$ edges with non-zero weight, with all cuts having capacities within a constant factor of their original capacities.

Theorem 20.

1. *An $O(\log n)$ pseudo-approximation to the* Sparsest Cut *can be computed in $\tilde{O}(m + n^{1.5})$ time using $O(\log^2(n))$ single commodity flow computations.*

2. *An $O(\sqrt{\log n})$ pseudo-approximation to the* SPARSEST CUT *can be computed in $\tilde{O}(n^2)$ time using $O(\log n)$ multicommodity flow computations.*

PROOF: The SPARSEST CUT SDP, in vector and matrix form, is the following:

$$
\begin{array}{rrcl}
 & \min \sum_{e=\{i,j\}\in E} c_e\|\mathbf{v}_i - \mathbf{v}_j\|^2 & & \\
\forall p: & \sum_{j=1}^{k-1}\|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \\
 & \|\sum_i \mathbf{v}_i\|^2 & = & 0 \\
 & \sum_i \|\mathbf{v}_i\|^2 & = & n
\end{array}
\qquad
\begin{array}{rrcl}
 & \min \mathbf{C}\bullet\mathbf{X} & & \\
\forall p: & \mathbf{T}_p \bullet \mathbf{X} & \geq & 0 \\
 & \mathbf{J}\bullet\mathbf{X} & = & 0 \\
 & \mathbf{Tr}(\mathbf{X}) & = & n \\
 & \mathbf{X} & \succeq & \mathbf{0}
\end{array}
$$

Here, $\mathbf{J}$ is the all ones matrix. We note that the last two constraints are not standard. Typically, we only have the constraint $\sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 = n^2$. Note that the SDP optimum doesn't change if we throw in the constraint that $\sum_i \mathbf{v}_i = \mathbf{0}$, or equivalently, $\|\sum_i \mathbf{v}_i\|^2 = 0$. With this additional constraint, the constraint $\sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 = n^2$ is equivalent to $\sum_i \|\mathbf{v}_i\|^2 = n$, precisely the kind of trace bound we need.

The optimum divided by $2n$ is a lower bound on the expansion of the SPARSEST CUT. This can be seen as follows. Given a cut $(S, \bar{S})$, select an arbitrary unit vector $\mathbf{v}_0$, and set $\mathbf{v}_i = \frac{n}{2\sqrt{|S||\bar{S}|}}\mathbf{v}_0$ for all $i \in S$, and $\mathbf{v}_i = -\frac{n}{2\sqrt{|S||\bar{S}|}}\mathbf{v}_0$ for all $i \in -S$. Then $\|\mathbf{v}_i - \mathbf{v}_j\|^2 = \frac{n^2}{|S||\bar{S}|}$ if $i \in S$ and $j \in -S$ (or *vice-versa*), and 0 otherwise. Thus, the objective function is

$$\sum_{e=\{i,j\}\in E} c_e\|\mathbf{v}_i - \mathbf{v}_j\|^2 = \frac{n^2 E(S,\bar{S})}{|S||\bar{S}|} \leq 2n\frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}.$$

The dual SDP is the following:

$$
\begin{array}{rcl}
 & \max\ nx & \\
x\mathbf{I} + \sum_p f_p \mathbf{T}_p + z\mathbf{J} & \preceq & \mathbf{C} \\
\forall p: \quad f_p & \geq & 0
\end{array}
$$

Given a candidate solution $\mathbf{X}$, the ORACLE always sets $x = \frac{\alpha}{n}$. Since $x\mathbf{I}\bullet\mathbf{X} = \alpha$, it now needs to find f_p, z and $\mathbf{F} \preceq \mathbf{C}$ such that

$$\alpha + \textstyle\sum_p f_p(\mathbf{T}_p \bullet \mathbf{X}) + z(\mathbf{J}\bullet\mathbf{X}) - (\mathbf{F}\bullet\mathbf{X}) \leq 0.$$

It runs the following steps (as before, all unspecified variables, including $\mathbf{F}$, are set to 0):

1. If $\mathbf{J}\bullet\mathbf{X} \geq n^2/5$, then set $z = -\frac{5\alpha}{n^2}$, so that $z(\mathbf{J}\bullet\mathbf{X}) \leq -\alpha$. Furthermore, $\|\frac{\alpha}{n}\mathbf{I} - z\mathbf{J}\| \leq O(\frac{\alpha}{n})$, since $\mathbf{0} \preceq \mathbf{J} \preceq n\mathbf{I}$.

2. Assume now that $\mathbf{J}\bullet\mathbf{X} \leq n^2/5$. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}$. Then since $\mathbf{I}\bullet\mathbf{X} = \mathbf{Tr}(\mathbf{X}) = n$, the condition $\mathbf{J}\bullet\mathbf{X} \leq n^2/5$ implies that $n^2 \geq \sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{4}{5}n^2$, since the Laplacian of the complete graph is exactly $\mathbf{K}_V = n\mathbf{I} - \mathbf{J}$, and $\mathbf{K}_V \bullet \mathbf{X} = \sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2$. Now, we can apply the algorithm of the following lemma (proved after the current proof):

Lemma 5. *Suppose we are given, for all $i \in V$, vectors $\mathbf{v}_i$, such that $n^2 \geq \sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{4}{5}n^2$, and a quantity α. Then there is an algorithm, which, using a single max-flow computation, outputs either*

(a) a valid $O(\frac{\alpha}{n})$-regular flow $\mathbf{f} = \langle f_p \rangle_p$, such that $\sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or,

(b) a cut of expansion $O(\frac{\alpha}{n})$, or

(c) a set of nodes $S \subseteq V$ of size $\Omega(n)$, such that there is a node $i_0 \in S$ such that for all $i \in S$, $\|\mathbf{v}_i - \mathbf{v}_{i_0}\|^2 = O(1)$, and $\sum_{i,j \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$.

If we get a cut of expansion, $O(\frac{\alpha}{n})$, we output it. If we get a flow $\mathbf{f} = \langle f_p \rangle_p$ such that $\sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, then just as in step 3 of the Oracle for undirected Balanced Separator, we can set $\mathbf{F}$ and $\mathbf{D}$ to be the flow and demand graph Laplacians respectively, and make progress, since

$$\alpha + \sum_p f_p(\mathbf{T}_p \bullet \mathbf{X}) - (\mathbf{F} \bullet \mathbf{X}) \;=\; \alpha - \mathbf{D} \bullet \mathbf{X} \;=\; \alpha - \textstyle\sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\leq\; 0.$$

Finally, if we get a set of nodes $S \subseteq V$ of size $\Omega(n)$, such that there is a node i_0 such that for all $i \in S$, $\|\mathbf{v}_i - \mathbf{v}_{i_0}\|^2 = O(1)$, and $\sum_{i,j \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$, then we can apply Lemma 4 to S with the vector $\mathbf{v}_i - \mathbf{v}_{i_0}$ associated to node $i \in S$, with the current value of α. This will again yield either a cut of small expansion, in which case we stop, or a $\tilde{O}(\frac{\alpha}{n})$-regular flow such that $\sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, in which case we again make progress by setting $\mathbf{F}$ and $\mathbf{D}$ to be the flow and demand graph Laplacians respectively of this new flow.

The running time is bounded in the same way as in the case of minimum c-Balanced Separator, noting that the most expensive additional step required here is a max-flow computation, which can be done in $\tilde{O}(n^{1.5})$ time using the algorithm of Goldberg and Rao [47]. □

Now, we prove Lemma 5.

Proof:[Lemma 5] Given vectors $\mathbf{v}_i$ such that $n^2 \geq \sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{4}{5}n^2$, we run the following steps:

1. For a node i, and radius r, let $B(i, r) = \{j : \|\mathbf{v}_i - \mathbf{v}_j\| \leq r\}$. If there is a node i such that $|B(i, \frac{1}{2\sqrt{10}})| \geq n/4$, then any $i_0 \in B(i, \frac{1}{2\sqrt{10}})$ satisfies $|B(i_0, \frac{1}{\sqrt{10}})| \geq n/4$. So we can find such an i_0 by simple random sampling. Let $L = B(i_0, \frac{1}{\sqrt{10}})$, and let $R = V \setminus L$. For $j \in R$, define $\Delta(j, L) = \min_{i \in L} \|\mathbf{v}_i - \mathbf{v}_j\|^2$ (doesn't need to be computed). We have

$$\begin{aligned}\frac{4}{5}n^2 &\le \sum_{ij}\|\mathbf{v}_i-\mathbf{v}_j\|^2\\ &\le \sum_{ij}2\|\mathbf{v}_i-\mathbf{v}_{i_0}\|^2+2\|\mathbf{v}_{i_0}-\mathbf{v}_j\|^2\\ &\le 2n\sum_i\|\mathbf{v}_i-\mathbf{v}_{i_0}\|^2\\ &\le 2n\sum_i[2\Delta(i,L)+\frac{2}{10}].\end{aligned}$$

Thus, $\sum_i d(i,L)\ge[\frac{1}{5}-\frac{1}{10}]n=\frac{n}{10}$. Since for $i\in L$, we have $d(i,L)=0$, we conclude that $\sum_{j\in R}d(j,L)\ge\frac{n}{10}$. Let $k:=\frac{|R|}{|L|}$. Note that $k\le 4$.

Now, we connect all nodes in L to a single source with edges of capacity $\frac{10k\alpha}{n}$ and all nodes in R to a single sink with edges of capacity $\frac{10\alpha}{n}$, and compute the max-flow $\mathbf{f}$ in the graph. Though the flow is between a single source and sink, we ignore the artificial edges added and associate the flow to paths between node pairs $i\in L$, $j\in R$ in the natural way. For such a node pair i,j, let f_{ij} be the total flow from i to j. If the flow saturates all source and sink nodes, then we have

$$\sum_{i\in L,j\in R}f_{ij}\|\mathbf{v}_i-\mathbf{v}_j\|^2 \;\ge\; \sum_{j\in R}\frac{10\alpha}{n}\cdot\Delta(j,L)\;\ge\;\alpha.$$

2. If the flow doesn't saturate all source and sink edges, then in the resulting cut, let the number of nodes in L connected to the source be n_s and the number of nodes in R connected to the sink be n_t. Then the capacity of the graph edges cut is at most $\frac{10\alpha}{n}(|R|-kn_s-n_t)$, and the smaller side of the cut has at least $\min\{|L|-n_s,|R|-n_t\}$ nodes. Thus, the expansion of the cut obtained is at most $\frac{10k\alpha}{n}=O(\frac{\alpha}{n})$.

3. Now assume that for all nodes i we have $|B(i,\frac{1}{2\sqrt{10}})|<n/4$. Then we claim that there is a node i such that $|B(i,2)|\ge n/2$. Otherwise, for all nodes i, there are more than $n/2$ nodes j such that $\|\mathbf{v}_i-\mathbf{v}_j\|^2\ge 2^2=4$. This is a contradiction since this would imply that
$$\sum_{ij}\|\mathbf{v}_i-\mathbf{v}_j\|^2\;>\;n\cdot\frac{1}{2}n\cdot 4\cdot\frac{1}{2}\;=\;n^2.$$
Again, by random sampling, we can find an i_0 such that $|B(i_0,4)|\ge n/2$. Let $S=B(i_0,4)$. Since for every $i\in S$, $|B(i,\frac{1}{2\sqrt{10}})|<n/4$, we conclude that there are at least $n/2-n/4=n/4$ nodes $j\in S$ such that $\|\mathbf{v}_i-\mathbf{v}_j\|^2\ge\frac{1}{40}$. Thus, we have
$$\sum_{i,j\in S}\|\mathbf{v}_i-\mathbf{v}_j\|^2\;\ge\;\frac{1}{2}n\cdot\frac{1}{4}n\cdot\frac{1}{40}\cdot\frac{1}{2}\;=\;\Omega(n^2).$$
We return this set S.

□

Connection to KRV's SPARSEST CUT algorithm. Khandekar, Rao and Vazirani [65] obtain an $O(\log^2 n)$ approximation to the BALANCED SEPARATOR and SPARSEST CUT in undirected graphs in $\tilde{O}(m + n^{1.5})$ time. We obtain $O(\log n)$-approximation in the same time. Here we note that despite superficial differences, their algorithm is a close cousin of ours. At each iteration, their algorithm maintains a (multi)graph that is a union of perfect matchings. It uses spectral methods (specifically, a random walk) to identify sparse cuts in this graph. Then it computes a single-commodity max flow across this sparse cut, finds a new perfect matching via flow decomposition, and adds it to the current multigraph before proceeding to the next iteration. The analysis of convergence uses an *ad hoc* potential function, and the main theorem says that the union of matchings converges in $O(\log^2 n)$ iterations to an *expander*.

Our algorithm is somewhat similar except we use matrix exponentiation at each iteration instead of random walks. However, the two are related. If $\mathbf{L}$ is a graph Laplacian, and $\beta < 1/(\text{max degree})$ is any constant, then $\exp(-\beta\mathbf{L})$ is the transition matrix after 1 time unit for the following continuous random walk: in each time interval δt, every node sends out a $\beta\delta t$ fraction of its probability mass to each of its neighbors. The algorithm of [65] simulates such a random walk, where $\mathbf{L}$ is the Laplacian of the union of the perfect matchings found so far. In our case, $\mathbf{L}$ is the Laplacian of the union of the flows found so far. Both algorithms compute a projection of the rows of the transition matrix on a random vector and then compute a max-flow based on the projections.

Of course, their *ad hoc* analysis does not apply to the other problems considered in this paper.

4.5.3 Directed BALANCED SEPARATOR

We have a directed graph $G = (V, E)$ with capacity c_e on edge $e \in E$. For a cut $(S, \bar{S})$ in the graph, define $E(S, \bar{S})$ to be the total capacity of arcs going from S to $\bar{S}$. The minimum c-BALANCED SEPARATOR is the c-balanced cut $(S, \bar{S})$ with minimum value of $E(S, \bar{S})$.

Theorem 21. *1. An $O(\log n)$ pseudo-approximation to the minimum c-*BALANCED SEPARATOR *in directed graphs can be computed in $\tilde{O}(m^{1.5})$ time using polylog(n) single-commodity flow computations.*

*2. An $O(\sqrt{\log n})$ pseudo-approximation to the minimum c-*BALANCED SEPARATOR *in directed graphs can be computed in $\tilde{O}(m^{1.5} + n^{2+\mu})$ time using polylog(n) single-commodity flow computations, for any specified constant $\mu > 0$. The $\tilde{O}$ notation hides polynomial dependence on $\frac{1}{\mu}$.*

PROOF: Consider the minimum c-BALANCED SEPARATOR SDP for a directed graph (see [2]). As usual, we have a vector $\mathbf{v}_i$ corresponding to every node. In addition, we also have a vector $\mathbf{w}_i$. For notational convenience, we will also denote this vector by $\mathbf{v}_{n+i}$. Thus, we now have matrices in $\mathbb{R}^{2n\times 2n}$. For a directed edge (i, j), we define its directed length $d(i, j) := \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2$. With the same notation as in the case of undirected BALANCED SEPARATOR, the SDP, in vector and matrix form, is given below:

$$\begin{array}{rrcl} & \min \sum_{e=(i,j)\in E} c_e d(i,j) & & \\ \forall i \in [2n]: & \|\mathbf{v}_i\|^2 & = & 1 \\ \forall i,j \in [n]: & \|\mathbf{w}_i - \mathbf{w}_j\|^2 & = & 0 \\ \forall p: & \sum_{j=1}^{k-1} \|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \\ \forall S: & \sum_{i,j\in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 & \geq & an^2 \end{array} \qquad \begin{array}{rrcl} & \min \mathbf{C} \bullet \mathbf{X} & & \\ \forall i \in [2n]: & X_{ii} & = & 1 \\ \forall i,j \in [n]: & \mathbf{E}_{ij} \bullet \mathbf{X} & = & 0 \\ \forall p: & \mathbf{T}_p \bullet \mathbf{X} & \geq & 0 \\ \forall S: & \mathbf{K}_S \bullet \mathbf{X} & \geq & an^2 \\ & \mathbf{X} & \succeq & \mathbf{0} \end{array}$$

Note that $\mathbf{C}$ is the directed Laplacian of the first kind (c.f. Section 4.3.2) of the graph. In the standard SDP for this problem [2], it is intended that all the $\mathbf{w}_i$ vectors are the same, and equal to some unit vector $\mathbf{v}_0$, and this is enforced by the constraint $\|\mathbf{w}_i - \mathbf{w}_j\|^2 = 0$. The matrix $\mathbf{E}_{ij}$ corresponds to this constraint, and is the (undirected) Laplacian of a single unit capacity edge joining $n+i$ to $n+j$. We introduce these additional vectors for the purpose of keeping the width bounded.

The optimum of this SDP divided by 8 is a lower bound on the capacity of the minimum c-Balanced Separator. This can be seen as follows. For a directed cut $(S, \bar{S})$, select an arbitrary unit vector $\mathbf{v}_0$, and set $\mathbf{w}_i = \mathbf{v}_0$ for all $i \in V$, and set $\mathbf{v}_i = \mathbf{v}_0$ for all $i \in S$, and $\mathbf{v}_i = -\mathbf{v}_0$ for all $i \in -S$. Then $d(i,j) = 8$ if $i \in S$ and $j \in -S$, and 0 otherwise. Thus, the objective function is

$$\sum_{e=(i,j)\in E} c_e d(i,j) \;=\; 8E(S, \bar{S}).$$

The dual to the SDP is as follows:

$$\begin{array}{rcl} \max \sum_i x_i + an^2 \sum_S z_S & & \\ \operatorname{diag}(\mathbf{x}) + \sum_{i,j\in[n]} y_{ij}\mathbf{E}_{ij} + \sum_p f_p \mathbf{T}_p + \sum_S z_S \mathbf{K}_S & \preceq & \mathbf{C} \\ \forall p, S: \quad f_p, z_S & \geq & 0 \end{array}$$

Now we turn to the implementation of the Oracle. Most of the steps are the same as in the undirected case. In the most complicated case, we need to compute a directed flow. If we have a flow in the graph which satisfies the capacity constraints $f_e \leq c_e$, then if we set $\mathbf{F}$ to be the directed Laplacian (of the first kind) of the flow, then for any primal feasible $\mathbf{X}$, with vectors $\mathbf{v}_i, \mathbf{w}_i$ obtained from its Cholesky decomposition, we have that

$$\mathbf{F} \bullet \mathbf{X} \;=\; \sum_{e=(i,j)\in E} f_e d(i,j) \;\leq\; \sum_{e=(i,j)\in E} c_e d(i,j) \;=\; \mathbf{C} \bullet \mathbf{X}.$$

because the triangle inequalities on the primal feasible matrix $\mathbf{X}$ imply the directed distance $d(i,j) \geq 0$. This matrix $\mathbf{F}$ can thus be used in the Primal-Dual framework for minimization SDPs.

The Oracle is implemented as follows:

1. This step is identical to Step 1 in the undirected case. Let $H := \{i \in [2n] : X_{ii} \geq 2\}$. If $|H| \geq \frac{\varepsilon}{2}n$, then we set $x_i = -\frac{\alpha}{|H|}$ for all $i \in H$, and $x_i = \frac{2\alpha}{2n-|H|}$ for all $i \notin H$. Then
$$\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} \;\leq\; -\frac{\alpha}{|H|} \cdot 2|H| + \frac{2\alpha}{2n-|H|} \cdot (2n - 2|H|) \;\leq\; 0.$$
Also, $\|\mathrm{diag}(\mathbf{x})\| \leq O(\frac{\alpha}{n})$.

2. Now, assume that $|H| < \frac{\varepsilon}{2}n$. Now we try to find $\Omega(n)$ disjoint pairs $i, j \in [n]$ such that $\|\mathbf{w}_i - \mathbf{w}_j\| \geq \delta$ for some $\delta \geq \Omega(\frac{1}{\log n})$ to be fixed later. We restrict attention to vectors $\mathbf{w}_i$ for $i \in W := \{i \in [n] : \; i, n+i \notin H\}$. Note that $|W| \geq (1 - \frac{\varepsilon}{2})n$.

   For a node $i \in W$, and radius r, let $C(i, r) = \{j \in W : \; \|\mathbf{w}_i - \mathbf{w}_j\| \leq r\}$. Suppose for all nodes $i \in W$ we have $|C(i, \delta)| \leq (1 - \varepsilon)n$. Equivalently, for every node $i \in W$, there are at least $\frac{\varepsilon}{2}n$ nodes j such that $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta^2$. Thus by randomly sampling, in expected $O(n)$ time, we can greedily remove $k := \frac{\varepsilon n}{8}$ disjoint pairs i, j such that $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta^2$. This is because if we have found less than k pairs so far, then for every node i, there are still $\frac{\varepsilon}{4}n$ nodes j such that $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta^2$, so a randomly chosen pair $i, j \in W$ satisfies $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta^2$ with constant probability.

   Let these pairs be $(i_1, j_1), \ldots, (i_k, j_k)$. For all these pairs, we set their $y_{ij} = -\frac{8\alpha}{\delta^2 \varepsilon n}$. We set all $x_i = \frac{\alpha}{2n}$. Then
$$\frac{\alpha}{2n}(\mathbf{I} \bullet \mathbf{X}) - \frac{8\alpha}{\delta^2 \varepsilon n} \sum_{t=1}^{k} (\mathbf{E}_{i_t j_t} \bullet \mathbf{X}) \;\leq\; \alpha - \alpha \;=\; 0.$$
   Furthermore, $-\frac{16\alpha}{\delta^2 \varepsilon n}\mathbf{I} \;\preceq\; \frac{\alpha}{2n}\mathbf{I} - \frac{8\alpha}{\delta^2 \varepsilon n}\sum_{t=1}^{k}\mathbf{E}_{i_t j_t} \;\preceq\; \frac{\alpha}{2n}\mathbf{I}$.

3. Now, assume that there is an $i \in W$ such that for some constant $|C(i, \delta)| \geq (1 - \varepsilon)n$. Then any $k \in C(i, \delta)$ satisfies $|C(k, 2\delta)| \geq (1 - \varepsilon)n$. So we can find such an k by simple random sampling. Let $S := C(k, 2\delta)$. Since $|S| \geq (1 - \varepsilon)n$, and we have constraint $\mathbf{K}_S \bullet \mathbf{X} \geq an^2$ corresponding to the set S in the SDP.

   Now we check if $\mathbf{K}_S \bullet \mathbf{X} \leq \frac{an^2}{2}$, and if it is, then we set $z_S = \frac{2\alpha}{an^2}$ all $x_i = -\frac{\alpha}{2n}$. Then
$$\left(-\frac{\alpha}{2n}\mathbf{I} + \frac{2\alpha}{an^2}\mathbf{K}_S\right) \bullet \mathbf{X} \;\leq\; \alpha - \alpha \;=\; 0.$$
   Also, since $\mathbf{0} \preceq \mathbf{K}_S \preceq n\mathbf{I}$, we have $\| -\frac{\alpha}{2n}\mathbf{I} + \frac{2\alpha}{an^2}\mathbf{K}_S\| \leq O(\frac{\alpha}{n})$.

4. Finally, assume that $\mathbf{K}_S \bullet \mathbf{X} \geq \frac{an^2}{2}$, i.e. $\sum_{ij \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$. Furthermore, for any $i, j \in S = C(k, 2\delta)$, we have $\|\mathbf{w}_i - \mathbf{w}_j\| \leq 4\delta$. We can now apply Lemma 6 below to the set S with the vectors $\mathbf{v}_i$, $\mathbf{w}_i$.

   Lemma 6. *Let $S \subseteq V$ be a set of nodes of size $\Omega(n)$. Suppose we are given, for all $i \in S$, vectors $\mathbf{v}_i, \mathbf{w}_i$ of length $O(1)$, such that $\sum_{i,j \in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$, and $\forall ij, \|\mathbf{w}_i - \mathbf{w}_j\| \leq 4\delta$, where $\delta = \Omega(\frac{1}{\log n})$ is a parameter that can be set as desired. Define the directed distance $d(i, j) = \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2$. For any given value α,*

(a) *There is an algorithm, which, using a single max-flow computation, either outputs a valid $O(\frac{\log(n)\alpha}{n})$-regular directed flow $\mathbf{f} = \langle f_p \rangle_p$ such that $\sum_{ij} f_{ij} d(i,j) \geq \alpha$, or a c'-balanced cut of expansion $O(\log(n)\frac{\alpha}{n})$.*

(b) *There is an algorithm, which, using $O(\log n)$ max-flow computations, plus an additional time of $O(\frac{1}{\mu}n^{2+\mu})$ for any constant $\mu > 0$, outputs either:*

   i. *a c'-balanced cut of expansion $O(\sqrt{\log(n)}\frac{\alpha}{n})$, or*

   ii. *a valid $O(\frac{\alpha}{n})$-regular directed flow $\mathbf{f} = \langle f_p \rangle_p$ such that $\sum_{ij} f_{ij} d(i,j) \geq \alpha$, or*

   iii. *$\Omega(\frac{n}{\sqrt{\log n}})$ vertex-disjoint paths such that the path inequality along these paths is violated by $\Omega(1)$.*

If we get a d-regular directed flow such that $\sum_{ij} f_{ij} d(i,j) \geq \alpha$, where $d = O(\frac{\log(n)\alpha}{n})$ or $O(\frac{\alpha}{n})$ depending on the two cases, then we set $\mathbf{F}$ to be the directed Laplacian (of the first kind) of the flow, and $\mathbf{D}$ to be the directed Laplacian (of the first kind) of the demand graph. Then $\mathbf{D} \bullet \mathbf{X} = \sum_{i \in L, j \in R} f_{ij} d(i,j) \geq \alpha$. Again, just as in the undirected case, we can set all $x_i = \frac{\alpha}{2n}$, so that $(\frac{\alpha}{2n}\mathbf{I} - \mathbf{D}) \bullet \mathbf{X} \leq \alpha - \alpha \leq 0$, and $-4d\mathbf{I} \preceq \frac{\alpha}{2n}\mathbf{I} - \mathbf{D} \preceq \frac{\alpha}{2n}\mathbf{I}$.

Finally, if we get $k = \Omega(\frac{n}{\sqrt{\log n}})$ vertex-disjoint paths $p_1, \ldots, p_k$, such that the path inequality along these paths is violated by $\Omega(1)$, $\mathbf{T}_p \bullet \mathbf{X} \leq -s$. Then, we set $f_{p_k} = \frac{\alpha}{sk}$ and all $x_i = \frac{\alpha}{2n}$. So,

$$\left(\frac{a}{2n}\mathbf{I} + \sum_{k=1}^{m} \frac{\alpha}{sk}\mathbf{T}_{p_k} \right) \bullet \mathbf{X} \ \leq \ \alpha - \alpha \ = \ 0.$$

We can bound the width as $\|\frac{a}{2n}\mathbf{I} + \sum_{k=1}^{m} \frac{\alpha}{sk}\mathbf{T}_{p_k}\| \leq O(\frac{\sqrt{\log n}\cdot\alpha}{n})$.

The overall running time is bounded just as in the undirected case. Again, we have $\tilde{O}(1)$ iterations, and in each iteration the most expensive operations are the polylog(n) max-flow computations, which take $\tilde{O}(m^{1.5})$ time on a directed graph, using the algorithm of Goldberg and Rao [47]. For the $O(\sqrt{\log n})$ approximation algorithm, we may incur an additional cost of $\tilde{O}(\frac{1}{\mu}n^{2+\mu})$ per iteration in order to find $\tilde{O}(n)$ paths with violated path inequalities. Thus, we get the stated running time. □

We now prove Lemma 6.

Proof:[Lemma 6]

We may assume, by scaling down if necessary, that all vectors $\mathbf{v}_i, \mathbf{w}_i$ have length at most 1. Let $\|\mathbf{w}_i - \mathbf{w}_j\| \leq \delta$, where δ is a parameter that we can make as small as needed (but always $\Omega(\frac{1}{\log n})$).

We choose a direction represented by a unit vector $\mathbf{u}$ at random. Lemma 14 shows that he fact that $\sum_{ij \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$ implies that with constant probability γ, we can find sets $L_0, R_0 \subseteq S$ each of size at least cn for some constant c, such that for any $i \in L_0$ and $j \in R_0$, $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$.

Next, let r be the median distance from $\mathbf{w}_{i_0}$ to the vectors $\{\mathbf{v}'_i : \ i \in L_0\}$. Let $L_0^+ := \{i \in L_0 : \|\mathbf{v}'_i - \mathbf{w}_{i_0}\| \geq r\}$ and $L_0^- := \{i \in L_0 : \|\mathbf{v}'_i - \mathbf{w}_{i_0}\| \leq r\}$. Define R_0^+ and

R_0^- analogously. If $|R_0^+| \geq |R_0^-|$, then set $L = L_0^-$, and $R = R_0^+$, else set $L = R_0^-$ and $R = L_0^+$. Note that both L and R have at least $\frac{cn}{2}$ nodes each.

For any pair of nodes $i \in L$ and $j \in R$, we have

$$\begin{aligned} d(i,j) &= \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2 \\ &= [\|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_{i_0} - \mathbf{v}_i\|^2 + \|\mathbf{w}_{i_0} - \mathbf{v}_j\|^2] - (\mathbf{w}_{i_0} - \mathbf{w}_i) \cdot (2\mathbf{v}_i - \mathbf{w}_{i_0} - \mathbf{w}_i) \\ &+ (\mathbf{w}_{i_0} - \mathbf{w}_j) \cdot (2\mathbf{v}_j - \mathbf{w}_{i_0} - \mathbf{w}_j) \\ &\geq \|\mathbf{v}_i - \mathbf{v}_j\|^2 - 8\delta \end{aligned}$$

The last inequality follows because $\|\mathbf{w}_{i_0} - \mathbf{w}_j\| \leq \delta$, and $\|2\mathbf{v}_j - \mathbf{w}_{i_0} - \mathbf{w}_j\| \leq 4$ since all vectors are of length at most 1. Now we have two cases, depending on what approximation we need:

Part 1. To get an $O(\log n)$ approximation, we can proceed exactly as in the case of undirected graphs. Namely, we may assume that all pairs $i \in L$ and $j \in R$ satisfy $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{\gamma}{\log n}$ for some constant $\gamma > 0$. We choose $\delta = \frac{\gamma}{16 \log n}$, so that $d(i,j) \geq \frac{\gamma}{2 \log n}$. Next, we connect all nodes in L to a single source with edges of capacity $\frac{\beta \log(n)\alpha}{n}$, and all nodes in R to a single sink with edges of capacity $\frac{\beta \log(n)\alpha}{n}$. We now compute the (directed) max-flow in this network. Just as in the undirected case, if we choose β large enough, either we get a flow such that $\sum_{i \in L, j \in R} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 2\alpha$ or we get a cut of expansion at most $O(\log(n) \cdot \frac{\alpha}{n})$. In the former case, by our choice of δ, we conclude that $\sum_{i \in L, j \in R} f_{ij} d(i,j) \geq \alpha$.

Part 2. To get an $O(\sqrt{\log n})$ approximation, we connect all nodes in L to a single source with edges of capacity $\frac{\beta\sqrt{\log(n)}\alpha}{n}$, and all nodes in R to a single sink with edges of capacity $\frac{\beta\sqrt{\log(n)}\alpha}{n}$. We now compute the (directed) max-flow in this network. Now there are three cases:

1. If the total flow obtained is less than $\frac{c\beta}{4}\sqrt{\log(n)} \cdot \alpha$, then by the max-flow-min-cut theorem, the cut obtained is also at most this size. Note that this cut will be $c/4$-balanced, since at most $\frac{c\beta}{4}\sqrt{\log(n)} \cdot \alpha / d = cn/4$ source (and sink) edges can be cut. Thus, the expansion of the cut is $O(\sqrt{\log(n)} \cdot \frac{\alpha}{n})$.

2. Now assume that the total flow obtained is at least $\frac{c\beta}{4}\sqrt{\log(n)} \cdot \alpha$. Then we check if $\sum_{i \in L, j \in R} f_{ij} d(i,j) \geq \alpha$. If it is, then we are done.

3. Otherwise, if the total flow is at least $\frac{c\beta}{4}\sqrt{\log(n)} \cdot \alpha$, but $\sum_{i \in L, j \in R} f_{ij} d(i,j) < \alpha$, then we repeat this process for $O(\log(n))$ different random directions. If we always end up in this case, then we try to find many paths for which the path inequality is violated to a large extent.

   For each direction $\mathbf{u}$, at least half the flow is between demand pairs (i,j) such that $d(i,j) \leq \frac{8}{c\beta\sqrt{\log n}}$. Now we choose $\delta = \frac{1}{4c\beta\sqrt{\log n}}$. Thus for all such pairs,

$\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{10}{c\beta\sqrt{\log n}}$. Since at most $\frac{\beta\sqrt{\log(n)}\alpha}{n}$ flow enters or leaves any node in L and R, we can show that in fact there is a matching of size εn, with $\varepsilon = \frac{c}{64}$, of pairs $i \in L$ and $j \in R$ such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\eta}{\sqrt{\log n}}$ where $\eta = \frac{10}{c\beta}$. (This argument is essentially the one given in Lemma 15. Scale down the flow by $\frac{\beta\sqrt{\log(n)}\alpha}{n}$ so that at most a unit flow enters or leaves any node in L and R, and the total amount of flow between pairs i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\eta}{\sqrt{\log n}}$ is at least $\frac{c}{8}n$. Then Lemma 15 shows that we obtain a matching of size $\frac{c}{64}n$ by a random selection process).

Note also that such pairs satisfy $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$. We call i, j a "(η, σ)-stretched pair along $\mathbf{u}$" (or often simply "stretched pair"). Let the fraction of directions $\mathbf{u}$ such that there is a matching of stretched pairs along $\mathbf{u}$ of size at least $\frac{c}{64}n$ be γ'. If $\gamma' < \gamma/2$, then after a constant number of repetitions of this procedure, we will end up in case 1, with high probability.

So assume that in each repetition we find large matchings of stretched pairs. In this case, we conclude that $\gamma' \geq \gamma/2$ with high probability. Now, we apply the algorithm of the following lemma, proved in Section 4.6:

Lemma 7. *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots$ be vectors of length at most 1 such for a γ fraction of directions $\mathbf{u}$, there is a matching of (η, σ)-stretched pairs along $\mathbf{u}$ of size εn. Let $\mu > 0$ be a given constant. There is a randomized algorithm which finds k vertex-disjoint paths p of length at most $\frac{2C}{\mu}\sqrt{\log n}$ such that the triangle inequality along p is violated by at least s, in time $\tilde{O}(n^2 + \frac{1}{\mu}kn^{1+\mu})$. Here, s, C are constants that depend only on $\gamma, \varepsilon, \sigma$, and we assume that $k \leq (\frac{\mu\varepsilon}{4C}) \cdot \frac{n}{\sqrt{\log n}}$ and that $\eta \leq \frac{\mu s}{4C}$.*

By setting $\beta = \frac{40C}{\mu s c}$, we can ensure that $\eta \leq \frac{\mu s}{4C}$. Then the lemma above implies that we can find $\Theta(\frac{n}{\sqrt{\log n}})$ paths p on which the path inequality is violated by at least a constant, and return them.

□

4.5.4 Directed Sparsest Cut

We have the same setup as for directed Balanced Separator. The Sparsest Cut in a directed graph $G = (V, E)$ is the cut $(S, \bar{S})$ with minimum expansion, $\frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}$.

Theorem 22. *1. An $O(\log n)$ pseudo-approximation to the* Sparsest Cut *in directed graphs can be computed in $\tilde{O}(m^{1.5})$ time using polylog(n) single-commodity flow computations.*

2. An $O(\sqrt{\log n})$ pseudo-approximation to the Sparsest Cut *in directed graphs can be computed in $\tilde{O}(m^{1.5} + n^{2+\mu})$ time using polylog(n) single-commodity flow computations, for any specified constant μ. The $\tilde{O}$ notation hides polynomial dependence on $\frac{1}{\mu}$.*

PROOF: We consider the directed SPARSEST CUT SDP (see [2]). As in the minimum c-BALANCED SEPARATOR problem, we have vectors $\mathbf{v}_i, \mathbf{w}_i \equiv \mathbf{v}_{n+i}$ corresponding to every node. In addition, we also have a vector $\mathbf{w}_i$. Thus, the matrix $\mathbf{X} \in \mathbb{R}^{2n \times 2n}$. For a directed edge (i, j), we define its directed length $d(i, j) := \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2$. With the same notation as in the case of directed minimum c-BALANCED SEPARATOR, the SDP, in vector and matrix form, is given below:

$$\begin{array}{rrcl}
 & \min \sum_{e=(i,j)\in E} c_e d(i,j) & & \\
\forall i,j \in [n]: & \|\mathbf{w}_i - \mathbf{w}_j\|^2 & = & 0 \\
\forall p: & \sum_{j=1}^{k-1}\|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_{i_1} - \mathbf{v}_{i_k}\|^2 \\
 & \sum_{i,j\in[n]} \|\mathbf{v}_i - \mathbf{v}_j\|^2 & = & n^2 \\
 & \sum_{i=1}^{2n} \|\mathbf{v}_i\|^2 & \leq & n
\end{array}
\qquad
\begin{array}{rrcl}
 & \min \mathbf{C} \bullet \mathbf{X} & & \\
\forall i,j \in [n]: & \mathbf{E}_{ij} \bullet \mathbf{X} & = & 0 \\
\forall p: & \mathbf{T}_p \bullet \mathbf{X} & \geq & 0 \\
 & \mathbf{K}_V \bullet \mathbf{X} & = & n^2 \\
 & \mathbf{Tr}(\mathbf{X}) & \leq & n \\
 & \mathbf{X} & \succeq & \mathbf{0}
\end{array}$$

Here, $\mathbf{C}$ is the directed Laplacian of the first kind (c.f. Section 4.3.2) of the graph. Also, $\mathbf{K}_V$ has the Laplacian of the complete graph on the vertices in $[n]$, in the top left $n \times n$ block, and zero elsewhere.

The optimum of this SDP divided by $4n$ is a lower bound on the expansion of the SPARSEST CUT. This can be seen as follows. Given a cut $(S, \bar{S})$, select an arbitrary unit vector $\mathbf{v}_0$, and set $\mathbf{w}_i = \frac{n}{2\sqrt{|S||\bar{S}|}}\mathbf{v}_0$ for all $i \in [n]$, and set $\mathbf{v}_i = \frac{n}{2\sqrt{|S||\bar{S}|}}\mathbf{v}_0$ for all $i \in S$, and $\mathbf{v}_i = -\frac{n}{2\sqrt{|S||\bar{S}|}}\mathbf{v}_0$ for all $i \in -S$. Then $d(i,j) = \frac{2n^2}{|S||\bar{S}|}$ if $i \in S$ and $j \in -S$, and 0 otherwise. Thus, the objective function is

$$\sum_{e=\{i,j\}\in E} c_e \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;=\; \frac{2n^2 E(S,\bar{S})}{|S||\bar{S}|} \;\leq 4n \frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}.$$

The dual to the SDP is as follows:

$$\begin{array}{rcl}
 & \max & nx + n^2 z \\
x\mathbf{I} + \sum_{i,j\in[n]} y_{ij}\mathbf{E}_{ij} + \sum_p f_p \mathbf{T}_p + z\mathbf{K}_V & \preceq & \mathbf{C} \\
\forall p : f_p, x & \geq & 0
\end{array}$$

Given a candidate solution $\mathbf{X}$, the ORACLE works as follows:

1. If $\mathbf{K}_V \bullet \mathbf{X} \geq \frac{6}{5}n^2$, then we set $z = -\frac{5\alpha}{n^2}$, and $x = \frac{6\alpha}{n}$. Then we have

$$x(\mathbf{I} \bullet \mathbf{X}) + z(\mathbf{K}_V \bullet \mathbf{X}) \;\leq\; 6\alpha - \frac{5\alpha}{n^2} \cdot \frac{6}{5}n^2 \;=\; 0.$$

   Furthermore, $-\frac{5\alpha}{n}\mathbf{I} \preceq x\mathbf{I} - z\mathbf{K}_V \preceq \frac{6\alpha}{n}\mathbf{I}$.

   Similarly, if $\mathbf{K}_V \bullet \mathbf{X} \leq \frac{4}{5}n^2$, then we set $z = \frac{5\alpha}{n^2}$, and $x = -\frac{4\alpha}{n}$ and get similar bounds.

2. This step is identical to step 2 in the implementation of the ORACLE for directed BALANCED SEPARATOR. For a node $i \in [n]$, and radius r, let $C(i,r) = \{j \in [n] : \|\mathbf{w}_i - \mathbf{w}_j\| \leq r\}$. Suppose for all $i \in [n]$ we have $|C(i,\delta)| \leq n/6$ for some $\delta = \Omega(\frac{1}{\log n})$ to be fixed later. Equivalently, for every $i \in [n]$, there are at least $n/6$ nodes j such that $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta^2$. Then as before, by randomly sampling, in expected $O(n)$ time, we can greedily remove $k := \frac{n}{24}$ disjoint pairs i,j such that $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \geq \delta^2$. Let these pairs be $(i_1,j_1),\ldots,(i_k,j_k)$. For all these pairs, we set their $y_{ij} = -\frac{24\alpha}{\delta^2 n}$. We set $x = \frac{\alpha}{n}$. Then

$$\frac{\alpha}{n}(\mathbf{I} \bullet \mathbf{X}) - \frac{24\alpha}{\delta^2 n}\sum_{t=1}^{k}(\mathbf{E}_{i_t j_t} \bullet \mathbf{X}) \;\leq\; \alpha - \alpha \;=\; 0.$$

Furthermore, $-\frac{48\alpha}{\delta^2 n}\mathbf{I} \;\preceq\; \frac{\alpha}{n}\mathbf{I} - \frac{24\alpha}{\delta^2 n}\sum_{t=1}^{k}\mathbf{E}_{i_t j_t} \;\preceq\; \frac{\alpha}{n}\mathbf{I}$.

3. Now assume that the vectors satisfy $\frac{6}{5}n^2 \geq \sum_{i,j\in[n]}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{4}{5}n^2$. Furthermore, assume that there is a node i such that $|C(i,\delta)| \geq \frac{5}{6}n$. By random sampling, we can find a node $k \in C(i,\delta)$, and thus there are $\frac{5}{6}n$ nodes $j \in C(k,2\delta)$. Let $W = V \setminus C(i_2, 2\delta)$. Note that $|W| \leq n/6$. Note that for all $i,j \in C(k,2\delta)$, we have $\|\mathbf{w}_i - \mathbf{w}_j\| \leq 4\delta$. Furthermore, we claim that $\|\mathbf{w}_k\| \leq 2$. Otherwise, for any node $j \in C(i_2,2\delta)$, we have $\|\mathbf{w}_j\| \geq \|\mathbf{w}_k\| - 2\delta \geq 3/2$ if we choose $\delta \leq 1/4$. But then $\mathbf{Tr}(\mathbf{X}) \geq \sum_{j\in C(k,2\delta)}\|\mathbf{w}_j\|^2 \geq \frac{9}{4}\cdot\frac{5}{6}n > n$, a contradiction. Thus, for all $j \in C(i_2,2\delta)$, we have $\|\mathbf{w}_j\| \leq \|\mathbf{w}_i\| + 2\delta \leq \frac{5}{2}$.

   Now, we can apply the algorithm of the Lemma 8 below to the set of vectors $\mathbf{v}_i, \mathbf{w}_i$:

   Lemma 8. *Suppose we are given, for all $i \in V$, vectors $\mathbf{v}_i, \mathbf{w}_i$, such that $\frac{6}{5}n^2 \geq \sum_{i,j\in[n]}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{4}{5}n^2$ and there is a set of nodes W of size $n/6$ such that for any nodes $i,j \in V \setminus W$, we have $\|\mathbf{w}_i\| \leq O(1)$ and $\|\mathbf{w}_i - \mathbf{w}_j\| \leq 4\delta$. Here, $\delta = \Omega(\frac{1}{\log n})$ is a parameter that can be set as desired. Define the directed distance $d(i,j) = \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_j - \mathbf{v}_j\|^2$. Let α be any given value. Then there is an algorithm, which using a single max-flow computation, outputs either*

   (a) a valid $O(\frac{\alpha}{n})$-regular directed flow $\mathbf{f} = \langle f_p\rangle_p$ such that $\sum_{ij} f_{ij} d(i,j) \geq \alpha$, or

   (b) a cut of expansion $O(\frac{\alpha}{n})$, or

   (c) a set S of nodes of size $\Omega(n)$ and a node $i_0 \in S$ such that $\|\mathbf{v}_{i_0} - \mathbf{v}_i\|^2 \leq O(1)$. Furthermore, $\sum_{ij\in S}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$ and for any pair $i,j \in S$, we have $\|\mathbf{w}_i\| \leq O(1)$ and $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \leq 4\delta$.

   If we get a cut of expansion $O(\frac{\alpha}{n})$, we output it.

   If we get a directed flow $\mathbf{f} = \langle f_p\rangle_p$ such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, then just as in step 4 of the ORACLE for directed BALANCED SEPARATOR, we can set $\mathbf{F}$ and $\mathbf{D}$ to be the directed flow and demand graph Laplacians (of the first kind) respectively, and make progress, since

$$\alpha + \sum_p f_p(\mathbf{T}_p \bullet \mathbf{X}) - (\mathbf{F} \bullet \mathbf{X}) \;=\; \alpha - \mathbf{D} \bullet \mathbf{X} \;=\; \alpha - \textstyle\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\leq\; 0.$$

Finally, in the last case, we get a set S of nodes of size $\Omega(n)$ and a node $i_0 \in S$ such that $\|\mathbf{v}_{i_0} - \mathbf{v}_i\|^2 \leq O(1)$. Also, $\sum_{ij \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \Omega(n^2)$ and for any pair $i, j \in S$, we have $\|\mathbf{w}_i\| \leq O(1)$ and $\|\mathbf{w}_i - \mathbf{w}_j\|^2 \leq 4\delta$. Then we can apply Lemma 6 to S with the vector $\mathbf{v}_i - \mathbf{v}_{i_0}$ and $\mathbf{w}_i$ associated to node $i \in S$, with the current α.

This will again yield either a cut of small expansion, in which case we stop, or a directed $\tilde{O}(\frac{\alpha}{n})$-regular flow such that $\sum_{ij} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, in which case we again make progress by setting $\mathbf{F}$ and $\mathbf{D}$ to be the flow and demand graph Laplacians (of the first kind) respectively of this new flow.

The running time is bounded in the same way as in the case of directed Balanced Separator, noting that the most expensive additional step required here is a max-flow computation, which can be done in $\tilde{O}(m^{1.5})$ time using the algorithm of Goldberg and Rao [47]. □

Now, we prove Lemma 8.

Proof:[Lemma 8] We run the following steps:

1. For a node i, and radius r, let $B(i, r) = \{j : \|\mathbf{v}_i - \mathbf{v}_j\| \leq r\}$. If there is a node i such that $|B(i, \frac{1}{2\sqrt{10}})| \geq n/4$, then any $i_0 \in B(i, \frac{1}{2\sqrt{10}})$ satisfies $|B(i_0, \frac{1}{\sqrt{10}})| \geq n/4$. So we can find such an i_0 by simple random sampling. Let $L = B(i_0, \frac{1}{\sqrt{10}})$ and $R = V \setminus L$. Now just as in step 1 of the proof of Lemma 5, if for any node j, $\Delta(j, L) := \min_{i \in L} \|\mathbf{v}_i - \mathbf{v}_j\|^2$, then $\sum_{j \in R} \Delta(j, L) \geq \frac{n}{10}$. Let $k = \frac{|R|}{|L|}$. Note that $k \leq 4$.

   Now consider the two quantities $\sum_{i \in L} k\|\mathbf{w}_i - \mathbf{v}_i\|^2$ and $\sum_{j \in R} \|\mathbf{w}_j - \mathbf{v}_i\|^2$. One of the two is smaller. Assume that it is the former; the other case is analogous, we just reverse the direction of the flow computation. We connect all nodes in L to a single source with edges of capacity $\frac{10k\alpha}{n}$ and all nodes in R to a single sink with edges of capacity $\frac{10\alpha}{n}$, and compute the max-flow $\mathbf{f}$ in the graph. Again, we associate the flow to paths between node pairs $i \in L$, $j \in R$ in the natural way. For such a node pair i, j, let f_{ij} be the total flow from i to j. If the flow saturates all source and sink nodes, then we have

$$\sum_{i \in L, j \in R} f_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\geq\; \sum_{j \in R} \frac{10\alpha}{n} \cdot \Delta(j, L) \;\geq\; \alpha.$$

   Now, we have that for any $i \in L$, $j \in R$, $d(i, j) + d(j, i) = 2\|\mathbf{v}_i - \mathbf{v}_j\|^2$. So $\sum_{i \in L, j \in R} f_{ij}[d(i, j) + d(j, i)] \geq 2\alpha$. Consider

$$\begin{aligned}\sum_{i \in L, j \in R} f_{ij} d(i, j) - \sum_{i \in L, j \in R} f_{ij} d(j, i) \;&=\; \sum_{i \in L, j \in R} f_{ij}\|\mathbf{w}_j - \mathbf{v}_j\|^2 - \sum_{i \in L, j \in R} f_{ij}\|\mathbf{w}_i - \mathbf{v}_i\|^2 \\ &=\; \sum_{j \in R} \frac{10\alpha}{n}\|\mathbf{w}_j - \mathbf{v}_j\|^2 - \sum_{i \in L} \frac{10k\alpha}{n}\|\mathbf{w}_i - \mathbf{v}_i\|^2 \\ &\geq\; 0,\end{aligned}$$

by assumption. The third equality follows because each node $i \in L$ is a source for exactly $\frac{10k\alpha}{n}$ flow, and each node $j \in R$ is a sink for exactly $\frac{10\alpha}{n}$ flow. Thus, we conclude that $\sum_{i \in L, j \in R} f_{ij} d(i,j) \geq \alpha$.

2. If the flow doesn't saturate all source and sink edges, then in the resulting cut, let the number of nodes in L connected to the source be n_s and the number of nodes in R connected to the sink be n_t. Then the capacity of the graph edges cut is at most $\frac{10\alpha}{n}(|R| - kn_s - n_t)$, and the smaller side of the cut has at least $\min\{|L| - n_s, |R| - n_t\}$ nodes. Thus, the expansion of the cut obtained is at most $\frac{10k\alpha}{n} = O(\frac{\alpha}{n})$.

3. Now assume that for all nodes i we have $|B(i, \frac{1}{2\sqrt{10}})| < n/4$. Then we claim that there is a node i such that $|B(i,3)| \geq n/2$. Otherwise, for all nodes i, there are more than $n/2$ nodes j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 3^2 = 9$. This is a contradiction since this would imply that

$$\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ > \ n \cdot \frac{1}{2} n \cdot 9 \cdot \frac{1}{2} \ > \ \frac{6}{5} n^2.$$

Again, by random sampling, we can find an i_0 such that $|B(i,6)| \geq n/2$. Let $S = B(i,6) \setminus W$. Note that $|S| \geq n/3$ since $|W| \leq n/6$. Since for every $i \in S$, $|B(i, \frac{1}{2\sqrt{10}})| < n/4$, we conclude that there are at least $n/3 - n/4 = n/12$ nodes $j \in S$ such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{1}{40}$. Thus, we have

$$\sum_{i,j \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ \geq \ \frac{1}{3} n \cdot \frac{1}{12} n \cdot \frac{1}{40} \cdot \frac{1}{2} \ = \ \Omega(n^2).$$

Furthermore, for any $i, j \in S$, we have $\|\mathbf{w}_i\| \leq O(1)$ and $\|\mathbf{w}_i - \mathbf{w}_j\| \leq 4\delta$. We return this set S.

□

4.5.5 Min UnCut

The Min UnCut problem has various equivalent forms (see [2]). The one we will use is the following. We are given an edge capacitated graph $G = (V, E)$ on the set of nodes $V = \{-n, \ldots, -2, -1, 1, 2, \ldots, n\}$ such that $\{i, j\} \in E$ if and only if $\{-j, -i\} \in E$. Also, we assume that the capacities satisfy $c_{ij} = c_{-j,-i}$. A cut $(S, \bar{S})$ is called symmetric if $\bar{S} = -S$ where $-S = \{-i : \ i \in S\}$. The Min UnCut problem is to find the symmetric cut $(S, -S)$ on minimum capacity, denoted $E(S, -S)$.

Theorem 23. *An $O(\sqrt{\log n})$ approximation to* Min UnCut *can be computed in $\tilde{O}(n^3)$ time.*

Proof: As usual, we start by writing the SDP for the problem. The SDP has vectors $\mathbf{v}_i$ for all $i \in V$, and the condition that $\mathbf{v}_i = -\mathbf{v}_{-i}$. We do not explicitly maintain the vectors for negatively indexed nodes; rather, wherever $\mathbf{v}_{-i}$ appears in the SDP, we replace it by $-\mathbf{v}_i$. Assuming we have made this transformation, the following SDP results:

$$\begin{array}{rrcl}
& \min \displaystyle\sum_{e=\{i,j\}\in E} c_e \|\mathbf{v}_i - \mathbf{v}_j\|^2 & & \\
& \forall i: \quad \|\mathbf{v}_i\|^2 & = & 1 \\
\forall p: & \sum_{j=1}^{k-1} \|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_k - \mathbf{v}_{i_k}\|^2
\end{array}
\qquad\qquad
\begin{array}{rrcl}
& \min \mathbf{C} \bullet \mathbf{X} & & \\
\forall i: & X_{ii} & = & 1 \\
\forall p: & \mathbf{T}_p \bullet \mathbf{X} & \geq & 0 \\
& \mathbf{X} & \succeq & \mathbf{0}
\end{array}$$

The optimum of this SDP divided by 4 is a lower bound on the value of the minimum symmetric cut. This can be seen as follows. For a symmetric cut $(S, -S)$, select an arbitrary unit vector $\mathbf{v}_0$, and set $\mathbf{v}_i = \mathbf{v}_0$ for all $i \in S$, and $\mathbf{v}_i = -\mathbf{v}_0$ for all $i \in -S$. Then $\|\mathbf{v}_i - \mathbf{v}_j\|^2 = 4$ if $i \in S$ and $j \in -S$ (or *vice-versa*), and 0 otherwise. Thus, the objective function is

$$\sum_{e=\{i,j\}\in E} c_e \|\mathbf{v}_i - \mathbf{v}_j\|^2 \; = \; 4E(S, -S).$$

The dual SDP is the following:

$$\begin{array}{rcl}
\max \; \sum_i x_i & & \\
\mathrm{diag}(\mathbf{x}) + \sum_p f_p \mathbf{T}_p & \preceq & \mathbf{C} \\
\forall p: \quad f_p & \geq & 0
\end{array}$$

Given a candidate solution $\mathbf{X}$, the Oracle works as follows:

1. If there is an i, say $i = 1$, such that $X_{ii} \geq 2$. Then we set $x_1 = -\alpha$, and $x_i = \frac{2\alpha}{n-1}$ for all $i \geq 2$. Since $\sum_{i\geq 2} X_{ii} \leq n - 2$, we have

   $$\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} \;\leq\; -\alpha \cdot 2 + \frac{2\alpha}{n-1} \cdot (n-2) \;\leq\; 0.$$

   Also, $-\alpha \mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) \preceq \frac{2\alpha}{n-1}\mathbf{I}$.

   Similarly, if there is an i, say $i = 1$, such that $X_{ii} \leq \frac{1}{2}$. Then we set $x_1 = 2\alpha$, and $x_i = -\frac{\alpha}{n-1}$ for all $i \geq 2$. Since $\sum_{i\geq 2} X_{ii} \geq n - \frac{1}{2}$, we have

   $$\mathrm{diag}(\mathbf{x}) \bullet \mathbf{X} \;\leq\; 2\alpha \cdot \frac{1}{2} - \frac{\alpha}{n-1} \cdot \left(n - \frac{1}{2}\right) \;\leq\; 0.$$

   Also, $-\frac{\alpha}{n-1}\mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) \preceq 2\alpha \mathbf{I}$.

2. Assume for all i, $\frac{1}{2} \leq X_{ii} \leq 2$. We can now apply Lemma 9 below.

   Lemma 9. *Assume we are given vectors $\mathbf{v}_i$ for every node in G such that for all i, $\|\mathbf{v}_i\|^2 = \Theta(1)$ and $\mathbf{v}_i = -\mathbf{v}_{-i}$ and a value α. Then there is an algorithm, which, using a single multicommodity flow computation, can obtain either:*

(a) *a valid $O(\alpha)$-regular flow $\mathbf{f} = \langle f_p \rangle_p$, such that $\sum_{ij} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, or*

(b) *a symmetric cut $(S, -S)$ of value $O(\sqrt{\log n} \cdot \alpha)$.*

If we get a $O(\alpha)$-regular flow such that $\sum_{i,j \in D} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, then we set $\mathbf{F}$ to be flow Laplacian, and $\mathbf{D}$ to be the demand Laplacian of the flow, and we set all $x_i = \frac{\alpha}{n}$. Then $\mathbf{D} \bullet \mathbf{X} = \sum_{i,j \in D} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, and so

$$(\operatorname{diag}(\mathbf{x}) - \mathbf{D}) \bullet \mathbf{X} \ \leq \ \alpha - \alpha \ = \ 0.$$

Note that $-\frac{2\alpha}{s}\mathbf{I} \preceq \frac{\alpha}{n}\mathbf{I} - \mathbf{D} \preceq \frac{\alpha}{n}\mathbf{I}$.

Otherwise, we get a symmetric cut $(S, -S)$ of value at most $O(\sqrt{\log n} \cdot \alpha)$, and we stop.

Now, we bound the running time. First, since we are working with an undirected graph, we may assume that the graph has been sparsified using the algorithm of Benczúr and Karger [22], to leave only $\tilde{O}(n)$ edges. Each iteration may involve one maximum multicommodity flow problem, which takes $\tilde{O}(n^2)$ using Fleischer's algorithm. To bound the number of iterations, we estimate the relevant parameters. The Oracle we described is $(O(\frac{\alpha}{n}, O(\alpha))$-bounded, and the parameter $R = n$. Thus, the number of iterations is $\tilde{O}(n)$ by Theorem 15. Finally, Lemma 24 indicates that we can compute an approximation to the matrix exponential in $\tilde{O}(n^2)$ time. Overall, the algorithm takes $\tilde{O}(n^3)$ time. □

Now, we prove Lemma 9.

Proof:[Lemma 9] We consider a maximum multicommodity flow problem where the set of demand pairs D consists of all pairs i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s to be fixed in Lemma 10. We only impose the edge capacity constraints $f_e \leq c_e$, and no d-regularity constraints, and solve the maximum multicommodity flow problem $\max \sum_{i,j \in D} f_{ij}$, using Fleischer's algorithm, say to a factor of $\frac{1}{2}$. We have the following cases:

1. If $\max \sum_{i,j \in D} f_{ij} \geq \frac{\alpha}{s}$, then assume that the flow is scaled down so that the total flow is exactly $\frac{\alpha}{s}$. So $\sum_{i,j \in D} f_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \alpha$, and we return this flow.

2. If the total flow is less than $\frac{\alpha}{s}$, then the actual max flow is less than $\frac{2\alpha}{s}$, since we have a $\frac{1}{2}$ approximation. Fleischer's algorithm then yields weights w_e on the edges with $\sum_e c_e w_e \leq \frac{2\alpha}{2}$ so that for any demand pair i, j and any path p connecting them, $\sum_{e \in p} w_e \geq 1$.

   Given a subset of nodes $S \subseteq V$, let $E(S)$ be the set of edges in the subgraph induced by S, and define $\operatorname{Vol}(S) := \sum_{e \in E_S} c_e w_e$. Then we have the following lemma:

   Lemma 10. *Let G be a graph with nodes $V = \{-n, \ldots, -1, 1, \ldots, n\}$ and assume we are given vectors $\mathbf{v}_i$ for every node i such that $\|\mathbf{v}_i\|^2 = \Theta(1)$ and for all i, $\mathbf{v}_i = -\mathbf{v}_{-i}$. Also, assume that we are given some weights on edges w_e such that for any node pair i, j with $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$, and for any path p connecting them, $\sum_{e \in p} w_e \geq 1$. Then it is possible to partition the graph into sets $(S, R, -S)$ such that the capacity of edges cut is at most $O(\sqrt{\log n}\, Vol(V))$, and $Vol(R) \leq (1 - c)\, Vol(V)$ for some constant $c < 1$.*

Thus, we can recursively partition the graph to get a symmetric cut of value at most $O(\sqrt{\log n} \cdot \alpha)$ as follows. We first apply the procedure of Lemma 10 to $R_0 := V$. The procedure returns a partitioning of vertices $(S, R, -S)$ such that the capacity of edges cut is at most $O(\sqrt{\log n}\text{Vol}(V))$, and $\text{Vol}(R) \leq (1-c)\text{Vol}(V)$. Then we apply the procedure again to $R_1 := R$, and continue aggregating cuts recursively until we end up with a symmetric cut $(S, -S)$. The capacity of this cut is bounded by

$$O(\sqrt{\log n}[\text{Vol}(R_0)+\text{Vol}(R_1)+\text{Vol}(R_2)+\cdots]) \;=\; O(\sqrt{\log n}\text{Vol}(R_0)) \;=\; O(\sqrt{\log n}\cdot\alpha),$$

because $\text{Vol}(R_0), \text{Vol}(R_1), \ldots$ is a geometrically decreasing sequence. We return the cut $(S, -S)$.

□

Now we prove Lemma 10.

Proof:[Lemma 10] We repeat each node i $m_i := \lfloor \frac{4n\sum_{e\ni i} c_e w_e}{\text{Vol}(V)} \rfloor$ times. Let N be the total number of nodes created this way. Then

$$8n \;\geq\; \sum_{i\in V} \frac{4n\sum_{e\ni i} c_e w_e}{\text{Vol}(V)} \;\geq\; N \;\geq\; \sum_{i\in V} \left[\frac{4n\sum_{e\ni i} c_e w_e}{\text{Vol}(V)} - 1\right] \;\geq\; 2n.$$

Since each node i has a partner node $-i$ with vector $-\mathbf{v}_i$, and all $\|\mathbf{v}_i\|^2 \geq \frac{1}{2}$, we have

$$\begin{aligned}\sum_{ij\in V} \|\mathbf{v}_i - \mathbf{v}_j\|^2 &= \sum_{i,j>0} [\|\mathbf{v}_i - \mathbf{v}_j\|^2 + \|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 + \|\mathbf{v}_{-i} - \mathbf{v}_j\|^2 + \|\mathbf{v}_{-i} - \mathbf{v}_{-j}\|^2] \\ &= \sum_{i,j>0} 4\|\mathbf{v}_i\|^2 + 4\|\mathbf{v}_j\|^2 \\ &\geq 2n^2 \\ &= \Omega(N^2).\end{aligned}$$

We now apply Corollary 5 of Lemma 14 to conclude that for a constant fraction of directions $\mathbf{u}$, there exists a set S' of nodes of size $2cN$ such that for all $i \in S$, we have $\mathbf{v}_i \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$ for some constants $c, \sigma > 0$. We assume that if any copy of a node i is included in S, then so are all the rest. We now apply the algorithm of Theorem 19 to the set S' and $T = -S'$. Now Theorem 19 implies that a max-flow-min-cut computation gives us a cut a c-balanced cut of value C such that there is a pair of nodes i, j and a path p connecting them such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$, for some constant $s > 0$, such that $\sum_{e\in p} w_e \leq O(\sqrt{\log n}\frac{\text{Vol}(G)}{C})$. Now we symmetrize the cut as follows. If the edge $\{k, \ell\}$ is in the cut, then so we cut the edge $\{-\ell, -k\}$ as well. The total capacity of cut edges is at most $2C$. Let S be the set of nodes connected to the source after cutting all these edges. Then by symmetry, $-S$ is the set of all nodes connected to the sink. Note that $|S| \geq cN$. Let $R = V \setminus (S \cup -S)$. Now note that

$$2cn \;\leq\; cN \;\leq\; \sum_{i\in S} m_i \;\leq\; \sum_{i\in S} \frac{2n\sum_{e\ni i} c_e w_e}{\text{Vol}(V)}$$

Thus, we have $\sum_{i \in S} \sum_{e \ni i} c_e w_e \geq c\mathrm{Vol}(V)$. So $\mathrm{Vol}(R) \leq (1-c)\mathrm{Vol}(V)$.

Now since $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$, we conclude that i, j is a demand pair. Then we get that $O(\sqrt{\log n}\frac{\mathrm{Vol}(G)}{C}) \geq \sum_{e \in p} w_e \geq 1$, which implies that $C \leq O(\sqrt{\log n}\mathrm{Vol}(G))$. Thus, $(S, R, -S)$ is the required partitioning of the graph. □

4.5.6 Min 2CNF Deletion

The Min 2CNF Deletion problem has various equivalent forms (see [2]). The one we will use is the following. We are given an edge capacitated directed graph $G = (V, E)$ on the set of nodes $V = \{-n, \ldots, -2, -1, 1, 2, \ldots, n\}$ such that $(i, j) \in E$ if and only if $(-j, -i) \in E$. Also, we assume that the capacities satisfy $c_{ij} = c_{-j,-i}$. A cut $(S, \bar{S})$ is called symmetric if $\bar{S} = -S$ where $-S = \{-i : i \in S\}$. For such a cut, let $E(S, -S)$ denote the total capacity of all arcs from S into $-S$. The Min 2CNF Deletion problem is to find the symmetric cut $(S, -S)$ with minimum $E(S, -S)$.

Theorem 24. *An $O(\sqrt{\log n})$ approximation to the* Min 2CNF Deletion *problem can be computed in $\tilde{O}(nm^{1.5} + n^3)$ time.*

Proof: Consider the Min 2CNF Deletion SDP (see [2]). The SDP has vectors $\mathbf{v}_i$ for all $i \in V$, and the condition that $\mathbf{v}_i = -\mathbf{v}_{-i}$. So we do not explicitly maintain the vectors for negatively indexed nodes; rather, wherever $\mathbf{v}_{-i}$ appears in the SDP, we replace it by $-\mathbf{v}_i$. We have an additional unit vector $\mathbf{v}_0$. For a directed edge (i, j), we define its directed length $d(i, j) := \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{v}_0 - \mathbf{v}_i\|^2 + \|\mathbf{v}_0 - \mathbf{v}_j\|^2$. The SDP, in vector and matrix form, is given below:

$$\begin{array}{rrcl}
 & \min \displaystyle\sum_{e=(i,j) \in E} c_e d(i,j) & & \\
\forall i \in \{0, \ldots, n\}: & \|\mathbf{v}_i\|^2 & = & 1 \\
\forall p: & \sum_{j=1}^{k-1} \|\mathbf{v}_{i_j} - \mathbf{v}_{i_{j+1}}\|^2 & \geq & \|\mathbf{v}_k - \mathbf{v}_{i_k}\|^2
\end{array}
\qquad
\begin{array}{rrcl}
 & \min \mathbf{C} \bullet \mathbf{X} & & \\
\forall i \in \{0, \ldots, n\}: & X_{ii} & = & 1 \\
\forall p: & \mathbf{T}_p \bullet \mathbf{X} & \geq & 0 \\
 & \mathbf{X} & \succeq & \mathbf{0}
\end{array}$$

The optimum of this SDP divided by 8 is a lower bound on the value of the minimum symmetric cut. This can be seen as follows. For a directed symmetric cut $(S, -S)$, set $\mathbf{v}_i = \mathbf{v}_0$ for all $i \in S$, and $\mathbf{v}_i = -\mathbf{v}_0$ for all $i \in -S$. Then $d(i, j) = 8$ if $i \in S$ and $j \in -S$, and 0 otherwise. Thus, the objective function is

$$\sum_{e=(i,j) \in E} c_e d(i,j) \; = \; 8E(S, -S).$$

The dual SDP is the following:

$$\begin{array}{rcl}
 & \max \sum_i x_i & \\
\mathrm{diag}(\mathbf{x}) + \sum_p f_p \mathbf{T}_p & \preceq & \mathbf{C} \\
\forall p: \quad f_p & \geq & 0
\end{array}$$

Now we describe the Oracle. Given a candidate solution $\mathbf{X}$, the Oracle works as follows:

1. Let $\lambda = \Theta(\frac{1}{\sqrt{\log n}})$ be a parameter to be fixed later.

   Now suppose there is an $i \in \{0, \dots, n\}$, say $i = k$, such that $X_{kk} \geq 1 + \lambda$. Then we set $x_k = -\frac{\alpha}{\lambda} + \frac{(1+1/\lambda)\alpha}{n+1}$, and for all $i \neq k$, $x_i = \frac{(1+1/\lambda)\alpha}{n+1}$. Note that $\sum_i x_i = \alpha$, and $\operatorname{diag}(\mathbf{x}) = -\frac{\alpha}{\lambda}\mathbf{E}_k + \frac{(1+1/\lambda)\alpha}{n+1}\mathbf{I}$. Thus,

$$\operatorname{diag}(\mathbf{x}) \bullet \mathbf{X} \;=\; -\frac{\alpha}{\lambda}X_{kk} + \frac{(1+1/\lambda)\alpha}{n+1}(\mathbf{I} \bullet \mathbf{X}) \;\leq\; -\frac{\alpha}{\lambda}(1+\lambda) + (1+1/\lambda)\alpha \;=\; 0.$$

   Also, $-\frac{\alpha}{\lambda}\mathbf{I} \preceq \operatorname{diag}(\mathbf{x}) \preceq \frac{(1+1/\lambda)\alpha}{n+1}\mathbf{I}$.

2. Similarly, if there is an $i \in \{0, \dots, n\}$, say $i = k$, such that $X_{kk} \leq 1 - \lambda$, then we set we set $x_k = \frac{\alpha}{\lambda} + \frac{(1-1/\lambda)\alpha}{n+1}$, and for all $i \neq k$, we set $x_i = \frac{(1-1/\lambda)\alpha}{n+1}$. Note that $\sum_i x_i = \alpha$, and $\operatorname{diag}(\mathbf{x}) = \frac{\alpha}{\lambda}\mathbf{E}_k + \frac{(1-1/\lambda)\alpha}{n+1}\mathbf{I}$. Thus,

$$\operatorname{diag}(\mathbf{x}) \bullet \mathbf{X} \;=\; \frac{\alpha}{\lambda}X_{kk} + \frac{(1-1/\lambda)\alpha}{n+1}(\mathbf{I} \bullet \mathbf{X}) \;\leq\; \frac{\alpha}{\lambda}(1-\lambda) + (1-1/\lambda)\alpha \;=\; 0.$$

   Also, $\frac{(1-1/\lambda)\alpha}{n+1}\mathbf{I} \preceq \operatorname{diag}(\mathbf{x}) \preceq \frac{\alpha}{\lambda}\mathbf{I}$.

3. Now, assume that all $X_{ii} \in [1-\lambda, 1+\lambda]$. Now we apply the procedure of Lemma 11 below to the vectors $\mathbf{v}_i$ obtained from the Cholesky decomposition of $\mathbf{X}$, with the current value of α.

   Lemma 11. *Let G be a graph with the vertex set $V = \{-n, -(n-1), \dots, -1, 1, 2, \dots, n\}$, and for all $i \in V$, let $\mathbf{v}_i$ be a vector of squared length in $[1-\lambda, 1+\lambda]$, for some $\lambda = \Theta(\frac{1}{\sqrt{\log n}})$, such that $\mathbf{v}_{-i} = -\mathbf{v}_i$. Let $\mathbf{v}_0$ be a unit vector, and define $d(i,j) = \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{v}_0 - \mathbf{v}_i\|^2 + \|\mathbf{v}_0 - \mathbf{v}_j\|^2$. Given a value α, there is an algorithm, which, using $O(\log^2 n)$ single-commodity flow computations, and an additional time of $\tilde{O}(n^2)$, can obtain either:*

   (a) a capacity-respecting flow $\mathbf{f} = \langle f_p \rangle_p$, such that the total flow, $\sum_p f_p$, is at most $O(\sqrt{\log n} \cdot \alpha)$, and $\sum_{ij} f_{ij} d(i,j) \geq 2\alpha$, or

   (b) a path p such that the path inequality along p is violated by at least a constant, or

   (c) a symmetric cut $(S, -S)$ of value at most $O(\sqrt{\log n} \cdot \alpha)$.

   If we obtain a symmetric cut $(S, -S)$ of capacity $O(\sqrt{\log n} \cdot \alpha)$, then we stop, because we have the required approximation.

   If we obtain a path p such that the path inequality along p is violated by at least a constant, then we fall through to the next case.

   So assume that we obtain a flow $\mathbf{f} = \langle f_p \rangle_p$ such that the total flow, $F := \sum_p f_p$, is at most $O(\sqrt{\log n} \cdot \alpha)$, and $\sum_{ij} f_{ij} d(i,j) \geq 2\alpha$, where for any node pair (i,j), f_{ij} is the total flow from i to j.

Now we set $\mathbf{F}$ to be the directed Laplacian (of the second kind) of the flow $\mathbf{f}$, and $\mathbf{D}$ to be the directed Laplacian (of the second kind) of the demand graph. Then $\mathbf{D} \bullet \mathbf{X} = \sum_{ij} f_{ij} d(i,j) \geq 2\alpha$.

Note that the matrix corresponding to the directed distance $d(i,j)$ is exactly $\mathbf{T}_{ij0}$. Thus, $\mathbf{D} = \sum_{ij} f_{ij} \mathbf{T}_{ij0}$, where $\mathbf{T}_{ij}$ is the matrix. Using the inequality that for any two vectors $\mathbf{v}$ and $\mathbf{w}$, $\|\mathbf{v} - \mathbf{w}\|^2 \leq 2\|\mathbf{v}\|^2 + 2\|\mathbf{w}\|^2$, we conclude that the matrix $\mathbf{T}_{ij0} + 2\mathbf{E}_i + 2\mathbf{E}_0 \succeq \mathbf{0}$, since for any $\mathbf{X} \succeq \mathbf{0}$, if $\mathbf{v}_0, \ldots, \mathbf{v}_n$ are the vectors obtained from its Cholesky decomposition, we have

$$(\mathbf{T}_{ij} + 2\mathbf{E}_i + 2\mathbf{E}_0) \bullet \mathbf{X} \;=\; \|\mathbf{v}_i - \mathbf{v}_j\|^2 + \|\mathbf{v}_0 - \mathbf{v}_j\|^2 - \|\mathbf{v}_0 - \mathbf{v}_j\|^2 + 2\|\mathbf{v}_0\|^2 + 2\|\mathbf{v}_i\|^2 \;\geq\; 0.$$

Thus,

$$\mathbf{0} \;\preceq\; \mathbf{D} + \sum_{ij} f_{ij}(2\mathbf{E}_i + 2\mathbf{E}_0) \;\preceq\; 4F\mathbf{I}.$$

Now, we set $x_i = -\sum_j 2f_{ij} + \frac{4F+\alpha}{n+1}$, for $1 \leq i \leq n$, and $x_0 = -2F + \frac{4F+\alpha}{n+1}$. Note that

$$\sum_i x_i = -\sum_{1 \leq i \leq n} \sum_j 2f_{ij} - 2F + 4F + \alpha \;=\; -2F - 2F + 4F + \alpha \;=\; \alpha.$$

Also,

$$\mathrm{diag}(\mathbf{x}) - \mathbf{D} \;=\; \frac{4F+\alpha}{n+1}\mathbf{I} - [\mathbf{D} + \sum_{ij} f_{ij}(2\mathbf{E}_i + 2\mathbf{E}_0)],$$

and hence

$$-4F\mathbf{I} \;\preceq\; (\mathrm{diag}(\mathbf{x}) - \mathbf{D}) \;\preceq\; \frac{4F+\alpha}{n+1}\mathbf{I}.$$

Finally,

$$\begin{aligned}
(\mathrm{diag}(\mathbf{x}) - \mathbf{D}) \bullet \mathbf{X} \;&=\; \left(\frac{4F+\alpha}{n+1}\mathbf{I} - [\mathbf{D} + \sum_{ij} f_{ij}(2\mathbf{E}_i + 2\mathbf{E}_0)]\right) \bullet \mathbf{X} \\
&\leq\; 4F + \alpha - 2\alpha - \sum_{ij} 4f_{ij}(1-\lambda) \\
&\leq\; 4F\lambda - \alpha \\
&\leq\; 0,
\end{aligned}$$

if we choose $\lambda \leq \frac{\alpha}{4F} = \Theta(\frac{1}{\sqrt{\log n}})$.

4. If we find a path p connecting node k to node ℓ, such that the path inequality along p is violated by a constant s, i.e. $\mathbf{T}_p \bullet \mathbf{X} \leq -s$, then we set $f_p = \frac{2\alpha}{s}$. As in the previous case, we can infer that the matrix $4\mathbf{I} \succeq \mathbf{T}_p + 2\mathbf{E}_k + 2\mathbf{E}_\ell \succeq \mathbf{0}$.

So we set $x_k = x_\ell = 2f_p + \frac{-4f_p+\alpha}{n+1}$, and $x_i = \frac{-4f_p+\alpha}{n+1}$ for all $i \neq k, \ell$. Then $\sum_i x_i = 4f_p + [-4f_p + \alpha] = \alpha$. Also,

$$\mathrm{diag}(\mathbf{x}) + f_p\mathbf{T}_p = f_p(T_p + 2\mathbf{E}_k + 2\mathbf{E}_\ell) + \frac{-4f_p+\alpha}{n+1}\mathbf{I},$$

so

$$\frac{-4f_p+\alpha}{n+1}\mathbf{I} \preceq \mathrm{diag}(\mathbf{x}) + f_p\mathbf{T}_p \preceq \left(4f_p + \frac{\alpha}{n+1}\right)\mathbf{I}.$$

Also,

$$\begin{aligned}(\mathrm{diag}(\mathbf{x}) + f_p\mathbf{T}_p) \bullet \mathbf{X} &= \left(f_p(T_p + 2\mathbf{E}_k + 2\mathbf{E}_\ell) + \frac{-4f_p+\alpha}{n+1}\mathbf{I}\right) \bullet \mathbf{X} \\ &\leq -2\alpha + 4f_p(1+\lambda) - 4f_p + \alpha \\ &= 4f_p\lambda - \alpha \\ &\leq 0,\end{aligned}$$

if we set $\lambda \leq \frac{\alpha}{4f_p} = \Theta(1)$.

Now we can estimate the running time. The implementation of ORACLE described above is $(O(\sqrt{\log n}\frac{\alpha}{n}), O(\sqrt{\log n}\cdot\alpha))$-bounded, and the parameter $R = n+1$. Thus, Theorem 15 implies that the number of iterations is $\tilde{O}(n)$. In each iteration, we may need to do polylog(n) single commodity flow computations, plus $\tilde{O}(n^2)$ additional time. Each flow computation can be done in $\tilde{O}(m^{1.5})$ time using the algorithm for Goldberg and Rao [47]. The matrix exponential can be approximated in $\tilde{O}(n^2)$ time using Lemma 24. Overall, the running time becomes $\tilde{O}(nm^{1.5} + n^3)$. □

Now, we prove Lemma 11.

PROOF:[Lemma 11] We consider a max-multicommodity flow problem, with the demand pairs being all nodes i, j such that $d(i,j) \geq \Delta$, where $\Delta = \Theta(\frac{1}{\sqrt{\log n}})$ is a fixed quantity. Let D be the set of demand pairs. The multicommodity flow problem can be formulated as the following LP, where p refers to a generic (directed) path in the graph between some demand pair in D:

$$\begin{aligned}\max \sum_p f_p& \\ \forall e: \sum_{p \ni e} f_p &\leq c_e \\ \forall p: f_p &\geq 0\end{aligned}$$

This LP can be (approximately) solved using Fleischer's algorithm in $O(m^2)$ time, but here we describe a more efficient algorithm. We run the Multiplicative Weights algorithm in the gain form (see Section 2.2.1), with the parameter $\varepsilon = 1/4$, on this LP. We treat the LP as a packing program (see Section 2.3.2).

In each round t of the Multiplicative Weights algorithm, we are given a probability distribution on the constraints (alternatively, on the edges) $\mathbf{p}^{(t)} = \langle {p_e}^{(t)} \rangle_e$, and our goal is to find a flow $\mathbf{f}^{(t)} = \langle f_p{}^{(t)} \rangle_p$ which minimizes the expected payoff $\sum_e p_e{}^{(t)} \sum_{p \ni e} \frac{f_p{}^{(t)}}{c_e}$. Let $w_e = \frac{\alpha p_e{}^{(t)}}{c_e}$, so that $\sum_e c_e w_e = \alpha$. Thus, the expected payoff becomes

$$\sum_e p_e{}^{(t)} \sum_{p \ni e} \frac{f_p{}^{(t)}}{c_e} = \frac{\sum_e w_e \sum_{p \ni e} f_p{}^{(t)}}{\alpha} = \frac{\sum_p f_p{}^{(t)} \sum_{e \in p} w_e}{\alpha}.$$

Now, we use the procedure of Lemma 12 recursively as follows. If at any stage in the recursion, this procedure returns a path p such that the triangle inequality along p is violated by at least a constant, then we immediately return this path. So assume that the procedure never finds such a path.

For a subset of nodes S, define $\mathrm{Vol}(S) = \sum_{e \in E(S)} c_e w_e$, where $E(S)$ is the set of edges in the induced subgraph on S. Thus, $\mathrm{Vol}(V) = \alpha$.

We start out with the set of vertices $V^{(0)} = V$. At any stage r, for $r \geq 0$, let $V^{(r)}$ be the set of vertices remaining. We will ensure that $V^{(r)}$ is always a symmetric set (i.e. $V^{(r)} = -V^{(r)}$). We run the procedure of Lemma 12 on the subgraph induced by $V^{(r)}$, with the weight on node i, $w_i = \frac{1}{2} \sum_{e \in E(V^{(r)}):\ e \ni i} c_e w_e$, so that $\mathrm{Vol}(V^{(r)}) = \sum_{i \in V^{(r)}} w_i$.

The procedure gives us a subset of nodes $S^{(r)}$ such that $\sum_{i \in S^{(r)}} w_i \geq c\mathrm{Vol}(V^{(r)})$ for some constant $c > 0$, and for all $i, j \in S^{(r)}$, $\|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 \geq \Omega(\frac{1}{\sqrt{\log |V^{(r)}|}}) \geq \frac{\eta}{\sqrt{\log n}}$ for some small constant η.

Now define $S_+{}^{(r)} = \{i \in S^{(r)} :\ \mathbf{v}_i \cdot \mathbf{v}_0 \geq 0\}$, and $S_-{}^{(r)} = \{i \in S^{(r)} :\ \mathbf{v}_i \cdot \mathbf{v}_0 \leq 0\}$. Then for any $i, j \in S_+{}^{(r)}$, we have

$$\begin{aligned} d(i, -j) &= \|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 - \|\mathbf{v}_0 - \mathbf{v}_i\|^2 + \|\mathbf{v}_0 - \mathbf{v}_{-j}\|^2 \\ &\geq \|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 + 2\mathbf{v}_0 \cdot (\mathbf{v}_i - \mathbf{v}_{-j}) - \|\mathbf{v}_i\|^2 + \|\mathbf{v}_{-j}\|^2 \\ &\geq \|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 - 2\lambda \end{aligned}$$

Now, if we choose $\lambda \leq \frac{\eta}{4\sqrt{\log n}}$, and if we set $\Delta = \frac{\eta}{2\sqrt{\log n}}$, then we conclude that for all $i, j \in S_+{}^{(r)}$, $d(i, -j) \geq \Delta$. Similarly, we get that for all $i, j \in S_-{}^{(r)}$, $d(-j, i) \geq \Delta$. Now, if $\sum_{i \in S_+{}^{(r)}} w_i \geq \sum_{i \in S_-{}^{(r)}} w_i$, we set $L^{(r)} = S_+{}^{(r)}$, else we set $L^{(r)} = -S_-{}^{(r)}$. We have $\sum_{i \in L^{(r)}} \geq \frac{c}{2}\mathrm{Vol}(V^{(r)})$, and for any $i, j \in L^{(r)}$, we have $d(i, -j) \geq \Delta$.

Now, we collapse all nodes in $L^{(r)}$ into a single source, and all nodes in $-L^{(r)}$ into a single sink, and find the min-cut between them by computing a max-flow (where each edge has its original capacity c_e). Let $\mathbf{g}^{(r)}$ be the flow, and let $C^{(r)}$ be the value of the min-cut (which is also the value of the max-flow).

Now, we have

$$\sum_p g_p{}^{(r)} \sum_{e \in p} w_e = \sum_{e \in E(V^{(r)})} w_e \sum_{p \ni e} g_p{}^{(r)} \leq \sum_{e \in E(V^{(r)})} w_e c_e = \mathrm{Vol}(V^{(r)}). \qquad (4.5)$$

Also, by the max-flow-min-cut theorem, $\sum_p g_p{}^{(r)} = C^{(r)}$.

Now, let $W^{(r)}$ be the set of nodes reachable from $L^{(r)}$ after the edges in the min-cut have been removed. Then, we claim that $W^{(r)}$ and $-W^{(r)}$ are disjoint. This is because the existence of such a node implies that there is a directed path from some node in $L^{(r)}$ to some node in $-L^{(r)}$, by the symmetry of the edges. This is impossible since we removed all the edges in the min-cut between them. Thus, there can be no directed edge from a node in $W^{(r)}$ to any node in $-W^{(r)}$ either, because then the endpoint of the arc would also be reachable from $L^{(r)}$ and thus be in $W^{(r)}$. Thus, the min-cut disconnects $W^{(r)}$ from $-W^{(r)}$.

Now, let $V'^{(r)} = V^{(r)} \setminus (W^{(r)} \cup -W^{(r)})$. Since $L^{(r)} \subseteq W^{(r)}$, we conclude that $\text{Vol}(V'^{(r)}) \leq (1-c)\text{Vol}(V^{(r)})$.

Next, we apply the procedure of Lemma 12, to the set $V'^{(r)}$, with the weights on all nodes set to 1. We get a set $S'^{(r)}$ of size at least $c|V'^{(r)}|$ such that for all $i, j \in S'^{(r)}$, $\|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 \geq \Omega(\frac{1}{\sqrt{\log n}})$. As before, we construct sets ${S'_+}^{(r)} = \{i \in T^{(r)} : \mathbf{v}_i \cdot \mathbf{v}_0 \geq 0\}$ and ${S'_-}^{(r)} = \{i \in T^{(r)} : \mathbf{v}_i \cdot \mathbf{v}_0 \leq 0\}$, and we get that for any $i, j \in {S'_+}^{(r)}$, $d(i,j) \geq \Delta$, and for any $i, j \in {S'_-}^{(r)}$, $d(-j, i) \geq \Delta$. If $|{S'_+}^{(r)}| \geq |{S'_-}^{(r)}|$, then we set $L'^{(r)} = {S'_+}^{(r)}$, else $L'^{(r)} = -{S'_-}^{(r)}$. Thus, $|L'^{(r)}| \geq \frac{c}{2}|V'^{(r)}|$, and for any $i, j \in L'^{(r)}$, we have $d(i, -j) \geq \Delta$.

Now, we collapse all nodes in $L'^{(r)}$ into a single source, and all nodes in $-L'^{(r)}$ into a single sink, and find the min-cut between them by computing a max-flow (where each edge has its original capacity c_e). Let $\mathbf{g'}^{(r)}$ be the flow, and let $C'^{(r)}$ be the value of the min-cut (which is also the value of the max-flow). Now, we have

$$\sum_p {g'_p}^{(r)} \sum_{e \in p} w_e \;=\; \sum_{e \in E(V'^{(r)})} w_e \sum_{p \ni e} {g'_p}^{(r)} \;\leq\; \sum_{e \in E(V'^{(r)})} w_e c_e \;=\; \text{Vol}(V'^{(r)}) \;\leq\; \text{Vol}(V^{(r)}). \tag{4.6}$$

Also, by the max-flow-min-cut theorem, $\sum_p {g'_p}^{(r)} = C'^{(r)}$.

Let $W'^{(r)}$ be the set of nodes reachable from $L'^{(r)}$ after the edges in the min-cut have been removed. As before, $W'^{(r)}$ and $-W'^{(r)}$ are disjoint, and the min-cut disconnects $W'^{(r)}$ from $-W'^{(r)}$. Now, let $V^{(r+1)} = V'^{(r)} \setminus (W'^{(r)} \cup -W'^{(r)})$. Since $L'^{(r)} \subseteq W'^{(r)}$, we conclude that $|V^{(r+1)})| \leq (1-c)|V'^{(r)}| \leq (1-c)|V^{(r)}|$. Also, $\text{Vol}(V^{(r+1)} \leq \text{Vol}(V'^{(r)}) \leq (1-c)\text{Vol}(V^{(r)})$. We recur on the set of nodes $V^{(r+1)}$.

Eventually, we end up getting a symmetric cut $(S, -S)$ in the graph. Let $\alpha' = \sum_{r \geq 0} \text{Vol}(V^{(r)})$. Note that since the volumes decrease geometrically by a factor of $(1-c)$ in every round, we get that $\alpha' \leq \frac{1}{c}\text{Vol}(V) = \frac{1}{c}\alpha$. Let $C = \sum_{r \geq 0} C^{(r)} + C'^{(r)}$. C is an upper bound on the total capacity of the cut $(S, -S)$.

If $C \leq \frac{8\alpha}{c\Delta}$, then we can return the cut $(S, -S)$, because the cut has capacity at most $O(\sqrt{\log n} \cdot \alpha)$.

So assume that $C > \frac{8\alpha}{c\Delta}$. The total flow over all levels in the recursion is

$$\sum_{r \geq 0} \sum_p [{g_p}^{(r)} + {g'_p}^{(r)}] \;=\; \sum_{r \geq 0} [C^{(r)} + C'^{(r)}] \;=\; C \;>\frac{8\alpha}{c\Delta}.$$

Let t be the current round in the Multiplicative Weights algorithm. Then we set $\mathbf{f}^{(t)} = \frac{\beta c}{2} \sum_{r \geq 0} [\mathbf{g}^{(r)} + \mathbf{g'}^{(r)}]$, where $\beta \leq 1$ is the scaling factor needed so that the total flow $\sum_p {f_p}^{(t)} = \frac{4\alpha}{\Delta}$, i.e. $\beta = \frac{c\Delta C}{8\alpha}$.

Then we have

$$\begin{aligned}\sum_p f_p{}^{(t)} \sum_{e \in p} w_e &= \frac{\beta c}{2} \sum_{r \geq 0} \sum_p [g_p{}^{(r)} + g_p'{}^{(r)}] \sum_{e \in p} w_e \\ &\leq \frac{\beta c}{2} \sum_{r \geq 0} 2\mathrm{Vol}(V^{(r)}) \\ &\leq \alpha.\end{aligned}$$

The second inequality above follows from (4.5) and (4.6).

Thus, the expected payoff for using the flow $\mathbf{f}^{(t)}$ in round t in the Multiplicative Weights algorithm is

$$\frac{\sum_p f_p{}^{(t)} \sum_{e \in p} w_e}{\alpha} \;\leq\; 1.$$

Furthermore, for any path p between the pair of nodes i, j carrying non-zero flow in $\mathbf{f}^{(t)}$, we have that $d(i,j) \geq \Delta$, so $(i,j) \in D$. Let $f_{ij}{}^{(t)}$ be the total flow from node i to node j in the flow $\mathbf{f}^{(t)}$. Then we have

$$\sum_{ij} f_{ij}{}^{(t)} d(i,j) \;\geq\; \sum_{ij} f_{ij}{}^{(t)} \cdot \Delta \;=\; \frac{4\alpha}{\Delta} \cdot \Delta \;=\; 4\alpha.$$

Now, in each level of the recursion, we have $|V^{(r+1)}| \leq (1-c)|V^{(r)}|$, so there at most $O(\log n)$ levels in all. In each level we use edge capacities at most twice, in two max-flow computations. Overall, any edge is overloaded by at most a factor of $O(\log n)$. Thus, the width ρ in this case is $\max_e \frac{\sum_{p \ni e} f_p{}^{(t)}}{c_e} \leq O(\log n)$.

Now, since we applied the gain form of the Multiplicative Weights algorithm, by Theorem 3, we get the following inequality for any edge e:

$$T \;\geq\; (1-\varepsilon) \sum_{t=1}^{T} \frac{f_e{}^{(t)}}{c_e} - \frac{\rho \ln m}{\varepsilon}.$$

When $T = \lceil \frac{\rho \ln m}{\varepsilon^2} \rceil = O(\log^2 n)$, we get that

$$(1+\varepsilon)T \;\geq\; (1-\varepsilon) \sum_{t=1}^{T} \frac{f_e{}^{(t)}}{c_e}.$$

Let $\mathbf{f} = (1-2\varepsilon)\frac{1}{T}\sum_{t=1}^{T} \mathbf{f}^{(t)}$. We conclude that for all edges e, $\sum_{p \ni e} f_p \leq c_e$. Thus, $\mathbf{f}$ is a capacity respecting flow. The total flow in $\mathbf{f}$ is at most $(1-2\varepsilon)\frac{4\alpha}{\Delta} = O(\sqrt{\log n} \cdot \alpha)$, since in each round t, the total flow in $\mathbf{f}^{(t)}$ is at most $\frac{4\alpha}{\Delta}$.

Finally, since we set $\varepsilon = 1/4$ in the Multiplicative Weights algorithm, we have

$$\sum_{ij} f_{ij} d(i,j) \;=\; \frac{1}{2T} \sum_{t=1}^{T} \sum_{ij} f_{ij}{}^{(t)} d(i,j) \;\geq\; \frac{1}{2T} \cdot T \cdot 4\alpha \;=\; 2\alpha.$$

Thus, we have found the required flow, and we return it. □

Lemma 12. *Let G be a graph with the vertex set $V = \{-n, -(n-1), \dots, -1, 1, 2, \dots, n\}$, and for all $i \in V$, let $\mathbf{v}_i$ be a vector of length $\Theta(1)$ such that $\mathbf{v}_{-i} = -\mathbf{v}_i$. For every node i, let w_i be an associated weight, such that $w_i = w_{-i}$. For $S \subseteq V$, let $w(S) = \sum_{i \in S} w_i$. Then there is an algorithm that in $\tilde{O}(n^2)$ time, finds either*

1. *a set of nodes S of such that $w(S) \geq cw(V)$ for some constant $c > 0$, and for all $i, j \in S$, $\|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 \geq \Omega(\frac{1}{\sqrt{\log n}})$, or,*
2. *a path p such that the triangle inequality along p is violated by a constant.*

Proof: We now repeat each node i, $m_i := \lfloor \frac{4nw_i}{w(V)} \rfloor$ times. Let V' be the new set of vertices, and let $N = |V'|$. Note that

$$4n \;\geq\; \sum_{i \in V} \frac{4nw_i}{w(V)} \;\geq\; N \;\geq\; \sum_{i \in V} \left[\frac{4nw_i}{w(V)} - 1 \right] \;\geq\; 2n.$$

With each copy of a node i, we associate the same vector $\mathbf{v}_i$, so that we obtain a centrally symmetric set of vectors.

Now, we apply Lemma 14, Corollary 5 to the set of vertices V' (with the vectors $\mathbf{v}_i$ scaled down by a constant so that their length is at most 1), to conclude that for a constant fraction γ of directions $\mathbf{u}$, there is a set S_0 of size at least $4cN$ such that for any $i \in S$, we have $\mathbf{v}_i \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{N}}$, for some constants c, σ.

We start removing all pairs i, j such that $i \in S_0$, and $j \in -S_0$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\eta}{\sqrt{\log N}}$ for a suitably small constant η. We do this while preserving symmetry, i.e. if we remove a pair i, j, we also remove the pair $-j, -i$ so that we are left with the sets $S_1, -S_1$ at the end. Now we have two cases:

1. $|S_1| \geq 2cN$: Let S be the subset of nodes in G which corresponds to the new nodes in S_1. Then we have

$$4cn \;\leq\; |S_1| \;=\; \sum_{i \in S} m_i \;\leq\; \sum_{i \in S} \frac{4nw_i}{w(V)},$$

which implies that $w(S) = \sum_{i \in S} w_i \geq cw(V)$. We return the set S. Note that for all $i, j \in S$, we have $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \frac{\eta}{\sqrt{N}} \geq \Omega(\frac{1}{\sqrt{N}})$.

2. $|S'| \leq 2cN$: This means that we removed a matching of size at least $2cN$ of node pairs i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\eta}{\sqrt{\log N}}$ but $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{N}}$. We call such pairs "$(\eta, \sigma)$-stretched pairs along $\mathbf{u}$". Let the fraction of directions $\mathbf{u}$ such that there is a matching of stretched pairs along $\mathbf{u}$ of size at least $2cN$ be γ'. If $\gamma' < \gamma/2$, then after a constant number of repetitions of this procedure, we will end up in case 1, with high probability.

   So assume that in each repetition we find large matchings of stretched pairs. In this case, we conclude that $\gamma' \geq \gamma/2$ with high probability. Now, we can apply the algorithm of Lemma 7 to find a path p such that the triangle inequality along p is violated by at least a constant, and we return this path. This procedure takes time $\tilde{O}(N^2) = \tilde{O}(n^2)$.

□

4.6 Proofs of Theorem 19 and Lemma 7

The proofs of Theorem 19 and Lemma 7 are based on the analysis of Arora, Rao and Vazirani [18] and Lee [73].

4.6.1 Proof of Theorem 19

We restate Theorem 19 here for convenience:

Theorem 19. *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors of length at most 1, such that $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq an^2$. Let w_e be weights on edges and nodes and let $\alpha = \sum_e c_e w_e$. Then there is an algorithm which runs in $\tilde{O}(m^{1.5})$ time and finds a cut of value C which is c-balanced for some constant c, such that there exists a pair of nodes i, j with the property that the graph distance between i and j is at most $O(\sqrt{\log n} \cdot \frac{\alpha}{C})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ where s is a constant which only depends on a. Furthermore, this is true even if any fixed set of τn nodes are prohibited from being i or j, for some small constant τ.*

The Project/Max-Flow Algorithm

0. Let $\sigma = \frac{a}{48}$, and $c = \frac{a}{256}$.
1. Choose a random direction $\mathbf{u}$, and project all $\mathbf{v}_i$ on it. By a linear scan, find sets S and T of size at least $2cn$ each such that for all $i \in S$ and $j \in T$, $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$. If no such sets exist, repeat this step with a new random direction $\mathbf{u}$.
2. Connect all nodes of S to (an artificial) source with edges of capacity 1, and connect all nodes of T to (an artificial) sink with edges of capacity 1. Scale the capacity of the original graph edges by a factor k, and compute the max flow. Let κ be the value of k at which the max flow in the network is cn, and output the min cut found.

Figure 4.2: The Project/Max-Flow algorithm.

The Project/Max-Flow algorithm is given in Figure 4.2. Note that this algorithm may need to do at most $O(\log n)$ max-flow computations (since the source can send out at most n flow), and each max-flow computation can be done in $\tilde{O}(m^{1.5})$ time using the algorithm of Goldberg and Rao [47], thus giving the stated running time.

Let C_{obs} be the capacity of the cut obtained, and let C_{med} be the median value of C_{obs}. We show that C_{med} satisfies the conditions of Theorem 19.

We need the following lemma regarding the Gaussian nature of projections (see [18]):

Lemma 13. *Let $\mathbf{v} \in \mathbb{R}^n$ be a vector. Let $\mathbf{u} \in \mathbb{S}^n$ be a randomly chosen unit vector. Then, for any $t \leq \sqrt{n}/4$, we have*

$$\mathbf{Pr}_{\mathbf{u}}\left[|\mathbf{v} \cdot \mathbf{u}| \;\geq\; \tfrac{t\|\mathbf{v}\|}{\sqrt{n}}\right] \;\leq\; \exp(-t^2/4).$$

Also, for any $t \le 1$, we have

$$\mathbf{Pr_u}\left[|\mathbf{v}\cdot\mathbf{u}| \;\le\; \tfrac{t\|\mathbf{v}\|}{\sqrt{n}}\right] \;\le\; 3t.$$

To begin the analysis of the algorithm, we need the following lemma, which shows that with constant probability, Step 1. of the algorithm succeeds:

Lemma 14. *For at least an $8c$ fraction of directions $\mathbf{u}$, there are efficiently computable subsets of the nodes S and T, each of size at least $2cn$, such that for any $i \in S$ and $j \in T$, $(\mathbf{v}_j - \mathbf{v}_i)\cdot\mathbf{u} \ge \frac{\sigma}{\sqrt{n}}$.*

Proof: Since $\sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge an^2$, and the maximum value of $\|\mathbf{v}_i - \mathbf{v}_j\|$ is 2, we get that $\sum_{ij}\|\mathbf{v}_i - \mathbf{v}_j\| \ge \frac{a}{2}n^2$. For any node i,

$$\frac{a}{2}n^2 \;\le\; \sum_{jk}\|\mathbf{v}_j - \mathbf{v}_k\| \;\le\; \sum_{jk}\|\mathbf{v}_j - \mathbf{v}_i\| + \|\mathbf{v}_i - \mathbf{v}_k\| \;\le\; n\sum_j \|\mathbf{v}_i - \mathbf{v}_j\|,$$

so $\sum_j \|\mathbf{v}_i - \mathbf{v}_j\| \ge \frac{a}{2}n$. Since the maximum value of $\|\mathbf{v}_i - \mathbf{v}_j\|$ is 2, we get that there must be at least $\frac{a}{8}n$ nodes j such that $\|\mathbf{v}_i - \mathbf{v}_j\| \ge \frac{a}{4}$. For any such pair i, j, the Gaussian behavior of projections (Lemma 13) implies that $|\mathbf{v}_i\cdot\mathbf{u} - \mathbf{v}_j\cdot\mathbf{u}| \ge \frac{a}{24\sqrt{n}}$ with probability at least $\frac{1}{2}$. Call such a pair of nodes a *stretched pair.* The expected number of stretched pairs is at least $\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{a}{8}n\cdot n = \frac{a}{32}n^2$. Since there are at most $\frac{1}{2}n^2$ pairs in all, we get that for at least $\frac{a}{32} = 8c$ fraction of directions $\mathbf{u}$, we get $\frac{a}{64}n^2$ stretched pairs.

Let $\mathbf{u}$ be such a direction. Let $\delta = \frac{\sigma}{\sqrt{n}} = \frac{a}{48\sqrt{n}}$. Let m be the median value of $\mathbf{v}_i\cdot\mathbf{u}$. Define the sets $L = \{i : \;\mathbf{v}_i\cdot\mathbf{u} \le m - \delta\}$, $M^- = \{i : \;\mathbf{v}_i\cdot\mathbf{u} \in [m-\delta, m]\}$, $M^+ = \{i : \;\mathbf{v}_i\cdot\mathbf{u} \in [m, m+\delta]\}$and $R = \{i : \mathbf{v}_i\cdot\mathbf{u} \ge m + \delta\}$. Then any stretched pair has at least one node in $L \cup R$. Now we claim that at least one of L and R has size at least $\frac{a}{128}n = 2cn$, because otherwise, the number of stretched pairs is less than $2\cdot\frac{a}{128}n\cdot n = \frac{a}{64}n$, a contradiction. If, say, $|L| \ge \frac{a}{128}n$, we can set $S = L$, and $T = M^+ \cup R$. Note that $|T| \ge \frac{1}{2}n$, since T is the set of all points with projection higher than the median. □

The following corollary to Lemma 14 is needed for the analysis of algorithms for Min UnCut and Min 2CNF Deletion.

Corollary 5. *Let G be a graph with the vertex set $V = \{-n, -(n-1), \ldots, -1, 1, 2, \ldots, n\}$, and for all $i \in V$, let $\mathbf{v}_i$ be a vector with squared length in $[a, 1]$ such that $\mathbf{v}_{-i} = -\mathbf{v}_i$. Then for a $\frac{a}{8}$ fraction of directions $\mathbf{u}$, there is a set S of size at least cn such that for all $i \in S$, $\mathbf{v}_i\cdot\mathbf{u} \ge \frac{\sigma}{\sqrt{n}}$, for $c = \frac{a}{32}, \sigma = \frac{a}{12\sqrt{2}}$.*

Proof: We have

$$\begin{aligned}
\sum_{ij\in V}\|\mathbf{v}_i - \mathbf{v}_j\|^2 &= \sum_{i,j>0}[\|\mathbf{v}_i - \mathbf{v}_j\|^2 + \|\mathbf{v}_i - \mathbf{v}_{-j}\|^2 + \|\mathbf{v}_{-i} - \mathbf{v}_j\|^2 + \|\mathbf{v}_{-i} - \mathbf{v}_{-j}\|^2] \\
&= \sum_{i,j>0} 4\|\mathbf{v}_i\|^2 + 4\|\mathbf{v}_j\|^2 \\
&\ge 4an^2.
\end{aligned}$$

Thus, Lemma 14 implies that for a $\frac{a}{8}$ fraction of directions $\mathbf{u}$, there are sets S and T of size cn such that for any $i \in S$ and $j \in T$, we have $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$, for $c = \frac{a}{32}, \sigma = \frac{a}{12\sqrt{2}}$.

Closer examination of the procedure of Lemma 14 reveals that the sets S and T have projections on $\mathbf{u}$ which are less than and greater than the median projection respectively. The median projection is 0, because we have a centrally symmetric set of vectors. Furthermore, one of S and T is a set of vectors whose projection on $\mathbf{u}$ is at least $\frac{\sigma}{\sqrt{n}}$ away on one side of the median projection. If S is this set, then for all $i \in S$, $\mathbf{v}_i \cdot \mathbf{u} \leq -\frac{\sigma}{\sqrt{n}}$, and thus $-S = \{-i : \ i \in S\}$ is the set we need. Otherwise, T is the set we need. □

Now we show that in a constant fraction of directions, we get a linear-sized matching of "stretched pairs" of nodes:

Lemma 15. *For at least $4c$ fraction of directions $\mathbf{u}$, there is a matching $M_{\mathbf{u}}$ of node pairs (i, j) such that:*

1. $|M_{\mathbf{u}}| \geq \frac{cn}{16}$,
2. $\forall (i, j) \in M_{\mathbf{u}}$ *we have* $\ell_{ij} \leq \frac{2\alpha}{C_{med}}$, *and*
3. $\forall (i, j) \in M_{\mathbf{u}}$, *we have* $|(\mathbf{v}_i - \mathbf{v}_j) \cdot \mathbf{u}| \geq \frac{\sigma}{\sqrt{n}}$.

PROOF: Lemma 14 shows that with probability at least $8c$, both S and T have size at least $2cn$. Conditioned on this happening, with a probability of $\frac{1}{2}$, the expansion of the cut, $C_{\text{obs}} \geq C_{\text{med}}$. So overall, for c fraction of directions, $C_{\text{obs}} \geq C_{\text{med}}$.

Assume this is the case. Since the max flow is cn, the min cut must attach the source to at least cn nodes to S; and the sink to at least cn nodes of T. So, the min cut is c-balanced.

For a path p in the graph from S to T, let $\mathbf{f} = \langle f_p \rangle_p$ denote the flow on it and ℓ_p its length (under w_e). We have:

$$\sum_p f_p \ell_p \ = \ \sum_p f_p \sum_{e \in p} w_e \ = \ \sum_e w_e \sum_{p \ni e} f_p \ \leq \ \sum_e w_e c_e \cdot \kappa \ = \ \kappa\alpha. \tag{4.7}$$

Let S_{obs} be the set of nodes in G which lie on the same side as the source in the output min cut. We assume, without loss of generality, that S_{obs} is the smaller side on the min cut. By the max-flow min-cut theorem, we have

$$C_{\text{obs}} \cdot \kappa + |S - S_{\text{obs}}| + |T \cap S_{\text{obs}}| = cn$$

This implies that

$$\kappa \ \leq \ \frac{cn}{C_{\text{obs}}}. \tag{4.8}$$

Inequalities (4.7) and (4.8) imply that

$$\sum_p f_p \ell_p \ \leq \ \frac{cn\alpha}{C_{\text{obs}}}.$$

Since $\sum_p f_p = cn$, by Markov's inequality we conclude the following:

$$\text{At least } \frac{cn}{2} \text{ flow is on paths } p \text{ with length } \ell_p \ \le\ \frac{2\alpha}{C_{\text{med}}}. \tag{4.9}$$

Let $N_{\mathbf{u}}$ be the set of all pairs (i,j) with $i \in S$ and $j \in T$ such that $\ell_{ij} \le \frac{2\alpha}{C_{\text{med}}}$. We now use the probabilistic method to show the existence of the desired matching. Let $M'_{\mathbf{u}}$ be the set of pairs (i,j) obtained as follows. For a pair (i,j) in $N_{\mathbf{u}}$, let f_{ij} be the total flow from i to j. For every $i \in S$, note that $d_i := \sum_{j:(i,j)\in N_{\mathbf{u}}} f_{ij} \le 1$ because the total flow into i from the source is 1. Interpreting the f_{ij}'s as a probability distribution over j's (here, we discard all j's with probability $1 - d_i$), randomly choose a single pair (i,j) for inclusion in $M'_{\mathbf{u}}$. Next, let $M_{\mathbf{u}}$ be the set obtained from $M'_{\mathbf{u}}$ by removing all pairs (i,j) where j occurs again in a different pair.

We now analyze the expected size of $M_{\mathbf{u}}$. For any node $j \in T$, let X_j be the number of times j occurs as a pair $(i,j) \in M'_{\mathbf{u}}$. Then $\mathbb{E}[|M_{\mathbf{u}}|] = \sum_j \mathbf{Pr}[X_j = 1]$. We have that $X_j = \sum_{(i,j)\in N_{\mathbf{u}}} X_{ij}$ where X_{ij} is an indicator random variable which is set to 1 with probability f_{ij}. Thus, $\mathbb{E}[X_j] = d_j := \sum_{i:(i,j)\in N_{\mathbf{u}}} f_{ij}$. Note that $d_j \le 1$ since the total flow out of j into the sink is at most 1. Now, $\mathbf{Pr}[X_j = 1] = 1 - \mathbf{Pr}[X_j \ge 2] - \mathbf{Pr}[X_j = 0]$. By Markov's inequality, we have that $\mathbf{Pr}[X_j \ge 2] \le \frac{d_j}{2}$. Also,

$$\mathbf{Pr}[X_j = 0] \ =\ \prod_{(i,j)\in N_{\mathbf{u}}} (1 - f_{ij}) \ \le\ \prod_{(i,j)\in N_{\mathbf{u}}} \exp(-f_{ij}) \ =\ \exp(-d_j) \ \le\ 1 - (1 - 1/e)d_j.$$

Here, the last inequality uses the fact that for $x \in [0,1]$, $e^{-x} \le 1 - (1-1/e)x$. Thus,

$$\mathbf{Pr}[X_j = 1] \ \ge\ \left(\frac{1}{2} - \frac{1}{e}\right) d_j \ \ge\ \frac{d_j}{8}.$$

This implies:

$$\mathbb{E}[|M_{\mathbf{u}}|] \ \ge\ \frac{1}{8}\sum_j d_j \ =\ \frac{1}{8}\sum_{(i,j)\in N_{\mathbf{u}}} f_{ij} \ \ge\ \frac{cn}{16}.$$

Here, the last inequality follows from (4.9). Thus, there must exist a matching $M_{\mathbf{u}}$ of size at least $\frac{cn}{16}$. Also, $M_{\mathbf{u}}$ satisfies conditions 2 and 3 by construction. □

Next, we show the existence of a large *core*, which is a set X of linear size such that each node in X participates in a constant fraction of the matchings found in Lemma 15. For this, consider a complete weighted graph on the same nodes, such that for every pair of nodes i, j the weight is $w_{ij} = \Pr_{\mathbf{u}}[\{i,j\} \in \mathbf{M_u}]$.

Lemma 16. *There is a subset of nodes, X, of size at least $\frac{c^2}{4}n$, such that for every $i \in X$, the degree of i on the subgraph induced by X is at least $\frac{c^2}{8}$.*

PROOF: Lemma 15 implies that the total weight of all edges is at least $4c \cdot \frac{c}{16}n = \frac{c^2}{4}n$. Now, we recursively remove all nodes with (remaining) degree less than $\frac{c^2}{8}$, and the edges incident on them. Altogether, we may remove at most $\frac{c^2}{8} \cdot n$ from the total weight. Thus, at least $\frac{c}{8}n$ weight remains. Since the degree of any node is at most 1, this implies that at least $\frac{c^2}{4}n$ nodes remain, each with degree at least $\frac{c^2}{8}$. □

A node i is said to be (ε, δ) centrally covered by X if for a δ fraction of directions $\mathbf{u}$, there exists a node $j \in X$ such that $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \varepsilon$. Similarly, a set of nodes Y is said to be (ε, δ) centrally covered by X if every node in Y is. Lemma 16 implies that X is (ε, δ) centrally covered by itself, where $\varepsilon = \frac{\sigma}{\sqrt{n}}$ and $\delta = \frac{c^2}{8}$. For a vertex $i \in X$, define $\Gamma(i) = \{j \in X : \exists \mathbf{u} \text{ s.t. } (i,j) \in M_{\mathbf{u}}\} \cup \{i\}$. Extend the definition of Γ to subsets $S \subseteq V$ as $\Gamma(S) = \bigcup_{i \in S} \Gamma(i)$. Define $\Gamma^r(S) = \underbrace{\Gamma(\Gamma(\cdots\Gamma}_{r \text{ times}}(S)\ldots))$. Nodes in $\Gamma^r(i)$ are said to be within r *matching hops* of i.

The following three lemmas discuss various useful properties of (ε, δ) covers.

Lemma 17 (Chaining Lemma). *Let X be the subset of nodes from Lemma 16. Let $S \subseteq X$ be a set of nodes that is $(\varepsilon_1, 1 - \frac{\delta}{2})$ centrally covered by X. Let $\rho = |S|/|\Gamma(S)|$. Then there is a set S' of size at least $\frac{\delta}{4}|S|$ which is $(\varepsilon_1 + \varepsilon, \frac{\delta\rho}{4})$ centrally covered by X.*

PROOF: Since $S \subseteq X$, every node $i \in S$ is (ε, δ) centrally covered by $\Gamma(i)$. Since for a uniformly random direction $\mathbf{u}$, the direction $-\mathbf{u}$ is also a uniformly random direction, so we conclude that for a δ fraction of directions $\mathbf{u}$, there is a node $j \in \Gamma(i)$ such that $(\mathbf{v}_j - \mathbf{v}_i) \cdot (-\mathbf{u}) \geq \varepsilon$.

Also, since i is $(\varepsilon_1, 1 - \frac{\delta}{2})$ centrally covered by X, for a $1 - \frac{\delta}{2}$ fraction of directions $\mathbf{u}$, there is a node $k \in \Gamma(i)$ such that $(\mathbf{v}_k - \mathbf{v}_i) \cdot \mathbf{u} \geq \varepsilon_1$. Thus, for a $\frac{\delta}{2}$ fraction of directions $\mathbf{u}$, both these events happen simultaneously, and thus we find a pair of nodes j, k such that $(\mathbf{v}_k - \mathbf{v}_j) \cdot \mathbf{u} \geq \varepsilon_1 + \varepsilon$. We say that i contributes the vector $\mathbf{v}_k - \mathbf{v}_j$ to j in the direction $\mathbf{u}$ towards building a central cover with projection $\varepsilon_1 + \varepsilon$. We we may assume that j is the matched node to i in $M_{-\mathbf{u}}$, thus ensuring that for any given direction, each node in S contributes such a vector to at most one node in $\Gamma(S)$.

Now, for every node $j \in \Gamma(S)$, let

$$d_j \;=\; \mathbf{Pr}_{\mathbf{u}}[\exists k \in X : \;(\mathbf{v}_k - \mathbf{v}_j) \cdot \mathbf{u} \geq \varepsilon_1 + \varepsilon].$$

We have $\sum_{j \in \Gamma(S)} d_j \geq \sum_{i \in S} \frac{\delta}{2} = \frac{\delta}{2}|S|$, since we assumed that for any given direction, each node in S contributes a vector to at most one node in $\Gamma(S)$. Since $|\Gamma(S)| = \frac{1}{\rho}|S|$, the average d_j over nodes $j \in \Gamma(S)$ is at least $\frac{\delta\rho}{2}$. Since all $d_j \leq 1$, by Markov's inequality, at most $1 - \frac{\delta\rho}{4}$ fraction of the d_j's can be less than $\frac{\delta\rho}{4}$. Thus, at least $\frac{\delta\rho}{4}|\Gamma(S)| = \frac{\delta}{4}|S|$ nodes j have d_j at least $\frac{\delta\rho}{4}$, i.e. these many nodes are $(\varepsilon_1 + \varepsilon, \frac{\delta\rho}{4})$ centrally covered by X. □

Lemma 18 (Close-by Covering Lemma). *Let X be the subset of nodes from Lemma 16. Let i be a node which is (ε, δ) centrally covered by X. Let j be another node and let $s := \|\mathbf{v}_j - \mathbf{v}_i\|$. Then for any $t \geq 0$, j is $(\varepsilon - \frac{ts}{\sqrt{n}}, \delta - \exp(-\frac{t^2}{4}))$ centrally covered by X.*

PROOF: Since $\|\mathbf{v}_j - \mathbf{v}_i\| = s$, by the Gaussian nature of projections (Lemma 13), we have $\mathbf{Pr}_{\mathbf{u}}[|(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u}| \geq \frac{ts}{\sqrt{n}}] \leq \exp(-\frac{t^2}{4})$. Thus, for a $\delta - \exp(-\frac{t^2}{4}))$ fraction of directions, we have a node $k \in X$ such that $(\mathbf{v}_k - \mathbf{v}_i) \cdot \mathbf{u} \geq \varepsilon$, and $|(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u}| \leq \frac{ts}{\sqrt{n}}$. Thus, for such a direction $\mathbf{u}$ we have $(\mathbf{v}_k - \mathbf{v}_j) \cdot \mathbf{u} \geq \varepsilon - \frac{ts}{\sqrt{n}}$, and hence j is $(\varepsilon - \frac{ts}{\sqrt{n}}, \delta - \exp(-\frac{t^2}{4}))$ centrally covered by X. □

Lemma 19 (Concentration of Measure Lemma)**.** *Let i be a node which is (ε, δ) centrally covered by a set of nodes S such that for any $j \in S$, $\|\mathbf{v}_j - \mathbf{v}_i\| \leq s$. Then for any $t \geq 0$, j is $(\varepsilon - \frac{t's}{\sqrt{n}}, 1 - \exp(-\frac{t^2}{2}))$ centrally covered by S, where $t' = \sqrt{2\log(\frac{2}{\delta})} + t$.*

Proof: The proof use P. Levy's isoperimetric inequality for measurable subsets of unit sphere $\mathbb{S}^{n-1}$ (see [18]), which says that of all measurable sets $A \subseteq \mathbb{S}^{n-1}$ of a given measure δ, the one that minimizes the measure of the set A_γ of all points within a distance of γ of A is the spherical cap of measure δ. Standard then calculations show that this minimum measure is $1 - \exp(-\frac{t^2}{2})$ if $\gamma > \frac{\sqrt{2\log(\frac{2}{\delta})}+t}{\sqrt{n}}$.

Now, consider the set A of unit vectors $\mathbf{u}$ such that there is a node $j \in S$ such that $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \varepsilon$. This is a measurable set and has measure at least δ, since S is an (ε, δ) central cover for i. Now, for any $t \geq 0$, set $\gamma = \frac{t'}{\sqrt{n}}$, where $t' = \sqrt{2\log(\frac{2}{\delta})} + t$. Then the set A_γ has measure at least $1 - \exp(-\frac{t^2}{2})$. Let $\mathbf{v} \in A_\gamma$. Then there is a vector $\mathbf{u} \in A$ such that $\|\mathbf{v} - \mathbf{u}\| \leq \gamma$, and a node $j \in S$ such that $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \varepsilon$, we have

$$(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{v} \;\geq\; (\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} - |(\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{v} - \mathbf{u})| \;\geq\; \varepsilon - s\gamma,$$

since $|(\mathbf{v}_j - \mathbf{v}_i) \cdot (\mathbf{v} - \mathbf{u})| \leq \|\mathbf{v}_j - \mathbf{v}_i\| \|\mathbf{v} - \mathbf{u}\| \leq s\gamma$ by the Cauchy-Schwarz inequality. Thus, S forms a $(\varepsilon - \frac{t's}{\sqrt{n}}, 1 - \exp(-\frac{t^2}{2}))$ central cover for i. □

Now, we can prove the following lemma which forms the key inductive step in the proof of Theorem 19:

Lemma 20. *Let X be the subset of nodes from Lemma 16. For every $r \geq 0$, one of the following two cases holds:*

1. *There is a non-empty set $S_r \subseteq X$ such that:*
   (a) *$|S_r| \geq (\frac{\delta}{4})^r |X|$,*
   (b) *$|S_r| \geq \delta |\Gamma(S_r)|$, and*
   (c) *S_r is $(\frac{r\varepsilon}{2}, 1 - \frac{\delta}{2})$ centrally covered by X.*
2. *There is a pair of points i, j such that $j \in \Gamma^{3r}(i)$ and $\|\mathbf{v}_i - \mathbf{v}_j\| \geq \beta$, where $\beta = \frac{\sigma}{16\sqrt{\log(4/\delta)}}$.*

Proof: We prove this by induction on r. For $r = 0$, we can choose $S_0 = X$. Every point i in X is covered by itself, so X itself is a $(0, 1 - \delta/2)$ cover for i. Further, $\Gamma(X) = X$, so all conditions required for case 1 hold.

For $r > 0$, assume that case 2 doesn't hold, but case 1 holds for $r - 1$. Then by the Chaining Lemma 17, and the observation that $|S_{r-1}|/|\Gamma(S_{r-1})| \geq \delta$, to get a set S'_r such that $|S_r| \geq (\frac{\delta}{4})|S_{r-1}| \geq (\frac{\delta}{4})^r |X|$, and S'_r is $(\frac{(r-1)\varepsilon}{2} + \varepsilon, \frac{\delta^2}{4})$ centrally covered by X. Now we expand the set S'_r using the Γ operation as follows. Let k be the first value for which $|\Gamma^k(S'_r)| \geq \delta|\Gamma^{k+1}(S'_r)|$. Let $S_r = \Gamma^k(S'_r)$. Since $|S'_r| \geq (\frac{\delta}{4})^r|X|$, it follows that $k \leq \log_{1/\delta}(\frac{4}{\delta})^r \leq 2r$. Let i be any point in S_r. Then there is a point $i' \in S'_r$ such that $i \in \Gamma^{3r}(i')$. Since we assumed that case 2 doesn't hold, we must have that $\|\mathbf{v}_i - \mathbf{v}_{i'}\| \leq \beta$. By the Close-by Covering Lemma 18, we conclude that the central cover X of point i is also a $(\frac{(r-1)\varepsilon}{2} + \varepsilon - \frac{t\beta}{\sqrt{n}}, \frac{\delta^2}{4} - \exp(-\frac{t^2}{4}))$ central cover for i. Choose $t = 2\sqrt{\log(\frac{8}{\delta^2})} \leq 4\sqrt{\log(\frac{4}{\delta})}$. Thus, $\frac{t\beta}{\sqrt{n}} \leq \frac{\sigma}{4\sqrt{n}} = \frac{\varepsilon}{4}$, and so X becomes a $(\frac{(r-1)\varepsilon}{2} + \frac{3\varepsilon}{4}, \frac{\delta^2}{8})$ central cover for i.

Next, if (i, j) is an edge in the central cover for i, then we claim that j is within $3r$ matching hops of i. This is because there is a path from i to j of matching hops which is composed of alternate expansion steps and chaining steps. There are at most r chaining steps, which may reduce the size of the current set by a factor of $\frac{\delta}{4}$, and expansion steps, which increase the size of the current set by a factor of $\frac{1}{\delta}$. Thus, the total number of expansion steps so far is at most $\log_{1/\delta}(\frac{4}{\delta})^r \leq 2r$. In addition to the r chaining steps, we get a total of at most $3r$ matching hops. Again, since we assumed that case 2 doesn't hold, we must have that $\|\mathbf{v}_i - \mathbf{v}_j\| \leq \beta$. Now we can use the Concentration of Measure Lemma 19 to conclude that X forms a $(\frac{(r-1)\varepsilon}{2} + \frac{3\varepsilon}{4} - \frac{t''\beta}{\sqrt{n}}, 1 - \exp(-\frac{t'^2}{2}))$ central cover for i, where $t'' = \sqrt{2\log(\frac{16}{\delta^2})} + t'$. If we set $t' = \sqrt{2\log(\frac{2}{\delta})}$, we get that $t'' \leq 4\sqrt{\log(\frac{4}{\delta})}$. Thus, $\frac{t\beta}{\sqrt{n}} \leq \frac{\sigma}{4\sqrt{n}} = \frac{\varepsilon}{4}$, and so X becomes a $(\frac{r\varepsilon}{2}, 1 - \frac{\delta}{2})$ cover for i.

To complete the induction, note that if case 2 holds for $r - 1$, then it trivially holds for r also. □

Finally, we can prove Theorem 19.

Proof:[Theorem 19] If a vector $\mathbf{v}_i$ is $(\frac{r\varepsilon}{2}, 1 - \frac{\delta}{2})$ covered by vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, and all vectors are of length at most 1, then $n \geq (1 - \frac{\delta}{2}) \cdot \exp(\frac{r^2\varepsilon^2 n}{64})$, since $\|\mathbf{v}_j - \mathbf{v}_i\| \leq 2$ and thus by Lemma 13, we have $\mathbf{Pr}_{\mathbf{u}}[|(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u}| \geq \frac{r\varepsilon}{2}] \leq \exp(-\frac{r^2\varepsilon^2 n}{64})$.

Thus, since $\delta < 1$ and $\varepsilon = \frac{\sigma}{\sqrt{n}}$, case 1 of Lemma 20 cannot hold for $r \geq \frac{\sqrt{64\log(2n)}}{\sigma} = \Theta(\sqrt{\log n})$. So for some $r \leq O(\sqrt{\log n})$, case 2 must hold. Case 2 implies the existence of a pair of points i, j which are within a graph distance of $3r \times \frac{2\alpha}{C_{\text{med}}} = O(\sqrt{\log n}\frac{2\alpha}{C_{\text{med}}})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \beta^2 = \Omega(1)$. We set $s = \beta^2$.

The robustness property follows because we can set aside some τn nodes from participating in the matchings with a small degradation in the constants. For instance, we can set $\tau = \frac{c}{32}$, and then we can ensure that there is a matching of stretched pairs of size at least $\frac{c}{32}n$ in at least $4c$ fraction of directions, which do not involve any of the τn forbidden nodes. The rest of the analysis follows just as before. □

4.6.2 Proof of Lemma 7

We now turn to the proof of Lemma 7. In this context, we call a pair of nodes i, j is a (η, σ)-*stretched pair along* $\mathbf{u}$ if $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\eta}{\sqrt{\log n}}$ but $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$ for constants $\eta, \sigma > 0$. The first step is to show that there are many pairs of nodes with a path connecting them with the desired parameters:

Lemma 21. *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots$ be vectors of length at most 1 such for at least a γ fraction of directions $\mathbf{u}$, there is a matching of (η, σ)-stretched pairs i, j of size εn along $\mathbf{u}$. Then there is a pair i, j of nodes such that there is a path p of stretched pairs of length $C\sqrt{\log n}$ such that each edge in the path is a stretched pair, and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s, where $C = C(\gamma, \varepsilon, \sigma), s = s(\gamma, \varepsilon, \sigma)$ are constants (independent of n and δ). Furthermore, there are $\frac{\varepsilon}{4}n$ pairs i, j of nodes such that there is a path p of stretched pairs connecting them of length $C'\sqrt{\log n}$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s'$, where $C' = C(\gamma, \varepsilon/2, \sigma)$ and $s' = s(\gamma, \varepsilon/2, \sigma)$.*

PROOF: The proof uses Lemmas 16 and 20. Note that neither of these lemmas need any upper bound on the distance between a pair of matched nodes i, j, and so their proofs work as long as for a constant fraction of directions $\mathbf{u}$, we have $\Omega(n)$ sized matchings of node pairs i, j such that $(\mathbf{v}_j - \mathbf{v}_i) \cdot \mathbf{u} \geq \frac{\sigma}{\sqrt{n}}$. Thus, for some $r \leq O(\sqrt{\log n})$, case 2 of Lemma 20 must hold, at which point we get a pair of nodes i, j within $3r$ matching hops such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ for some constant s. Thus, i, j are connected with a path of stretched pairs of length at most $3r = O(\sqrt{\log n})$.

We can show the existence of many such pairs using a greedy algorithm. We keep finding disjoint pairs i, j which satisfy the properties until the number of pairs found is $\Omega(n)$. When we find a pair i, j, we discard the nodes i, j from further consideration. If we have found only $o(n)$ pairs i, j so far, then we still have matchings of stretched pairs of size $\Omega(n)$ among the nodes still under consideration, and so more such pairs remain to be found.

More precisely, suppose we have found less than $\frac{\varepsilon}{4}n$ pairs i, j so far. Then excluding all such nodes from consideration in the matchings of stretched pairs, for a γ fraction of directions $\mathbf{u}$, we have a matching of size $\frac{\varepsilon}{2}n$ of stretched pairs along $\mathbf{u}$. Thus, there is a pair of nodes i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s'$, for $s' = s(\gamma, \varepsilon/2, \sigma)$, and a path p of stretched pairs connecting them of length at most $C'\sqrt{\log n}$ for $C' = C(\gamma, \varepsilon/2, \sigma)$. We can continue finding pairs this way until we have at least $\frac{\varepsilon}{4}n$ of them. $\square$

We can now prove Lemma 7, restated here for convenience.

Lemma 7. *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots$ be vectors of length at most 1 such for a γ fraction of directions $\mathbf{u}$, there is a matching of (η, σ)-stretched pairs of size εn along $\mathbf{u}$. Let $\alpha > 0$ be a given constant. There is a randomized algorithm which finds k vertex-disjoint paths p of length at most $\frac{2C}{\alpha}\sqrt{\log n}$ such that the triangle inequality along p is violated by at least s, in time $\tilde{O}(n^2 + \frac{1}{\alpha}kn^{1+\alpha})$. Here, s, C are constants that depend only on $\gamma, \varepsilon, \sigma$, and we assume that $k \leq (\frac{\alpha\varepsilon}{4C}) \cdot \frac{n}{\sqrt{\log n}}$ and that $\eta \leq \frac{\alpha s}{2C}$.*

PROOF: The idea is the same as before, i.e. to keep finding the required paths greedily and discarding all nodes on them from future consideration. Since we need to find only $O(\frac{n}{\sqrt{\log n}})$ paths, and each path has length at most $O(\sqrt{\log n})$, we end up discarding at most $O(n)$ nodes, where the constant in the O-notation can be made as small as we please. This allows us to conclude that we still have large matchings in many directions, and thus more paths remain to be found.

Now we describe this process in more detail. We construct a new graph, M with the same set of nodes, where there is an edge between nodes i and j if $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \leq \frac{\eta}{\sqrt{\log n}}$. We now use the dynamic decremental spanners algorithm of Baswana [21], which, for any given constant α, maintain a spanner of size $O(\frac{1}{\alpha}n^{1+\alpha})$, such that all distances are stretched by at most a factor of $\frac{2}{\alpha}$, which support edge deletion in polylog(n) time. The algorithm requires $\tilde{O}(m)$ preprocessing time, where m is the number of edges in M (which is $O(n^2)$ in the worst case).

Suppose we have found less than k of the required paths. We discard the nodes on all the paths we find. The number of nodes thus discarded is less than $\frac{2C}{\alpha}\sqrt{\log n} \times k \leq \frac{\varepsilon}{2}n$. Thus, for a γ fraction of directions $\mathbf{u}$, there are matchings of stretched pairs of size at least $\frac{\varepsilon}{2}n$ along $\mathbf{u}$. Thus, by Lemma 21, there are $\frac{\varepsilon}{8}n$ pairs of nodes i, j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 2s$ and there is a path p of stretched pairs length at most $C\sqrt{\log n}$ connecting them, where $s = \frac{1}{2}s(\gamma, \varepsilon/4, \sigma)$ and $C = C(\gamma, \varepsilon/4, \sigma)$.

Since the graph spanner has stretch at most $\frac{2}{\alpha}$, the nodes i, j are connected by a path of length at most $\frac{2C}{\alpha}\sqrt{\log n}$ in the spanner. By randomly sampling nodes i and doing a breadth-first search i of depth at most $\frac{2C}{\alpha}\sqrt{\log n}$, we find a node j such that $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 2s$ with probability at least $\frac{\varepsilon}{8}$. Thus, after constant number of random samples, we get the required node pair i, j with high probability. Thus, the time taken for finding such a pair is $O(\frac{1}{\alpha}n^{1+\alpha})$ since the graph spanner has only so many edges.

Let $p = \langle i = i_1, i_2, \ldots, i_k = j\rangle$ be the path of length at most $\frac{2C}{\alpha}\sqrt{\log n}$ connecting i, j. Since for all edges (i_t, i_{t+1}) along the path p, we have $\|\mathbf{v}_{i_t} - \mathbf{v}_{i_{t+1}}\|^2 \leq \frac{\eta}{\sqrt{\log n}}$, we conclude that

$$\sum_{t=1}^{k-1} \|\mathbf{v}_{i_t} - \mathbf{v}_{i_{t+1}}\|^2 - \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\leq\; \frac{2\eta C}{\alpha} - 2s \;\leq\; -s,$$

i.e. the path inequality along p is violated by at least s.

We now remove all nodes in p from the spanner. This procedure amounts to removing all edges incident on nodes in p in the graph M. So we remove at most $O(\sqrt{\log n} \cdot n)$ edges, and that takes time $\tilde{O}(n)$. Overall, the time taken to find k paths becomes $\tilde{O}(n^2 + \frac{1}{\alpha}kn^{1+\alpha})$. □

4.7 Computing the Matrix Exponential

In our general framework of Section 4.4 for maximization SDPs, the candidate solution $\mathbf{X}^{(t)}$ at each step is $\exp(\mathbf{M})$ for some $\mathbf{M}$. We show how to do this exponentiation faster than the trivial $O(n^3)$ time for the SDPs considered here, based on the idea that approximate computation suffices.

4.7.1 Johnson-Lindenstrauss Dimension reduction

Let α be our current estimate of the optimum and $\delta > 0$ be such that we desire a dual solution of cost at most $(1+\delta)\alpha$. The ORACLE's task, when given a candidate solution $\mathbf{X}^{(t)}$, is to find appropriate dual variables $y_1, \ldots, y_m$ such that $\sum_{i=1}^{j}(\mathbf{A}_j \bullet \mathbf{X}^{(t)})y_j - (\mathbf{C} \bullet \mathbf{X}^{(t)}) \geq 0$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be vectors obtained from the Cholesky decomposition of $\mathbf{X}^{(t)}$ such that $X_{ij}{}^{(t)} = \mathbf{v}_i \cdot \mathbf{v}_j = \frac{1}{2}[\|\mathbf{v}_i\|^2 + \|\mathbf{v}_j\|^2 - \|\mathbf{v}_i - \mathbf{v}_j\|^2]$. Thus ORACLE's task is to find appropriate variables s_i and t_{ij} for $i, j \in [n]$ such that $\sum_i s_i\|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij}\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq 0$. (Note that the s_i and t_{ij} variables cannot be set independently, since they are a linear transformation of $y_1, \ldots, y_m$.)

The vectors $\mathbf{v}_i$ obtained from the Cholesky decomposition of $\mathbf{X}^{(t)} = \exp(\mathbf{M})$ are just the row vectors of $\exp(\frac{1}{2}\mathbf{M})$. Since we are only interested in the square lengths of the row vectors and their differences, we can try Johnson-Lindenstrauss dimension reduction. If we project the vectors $\mathbf{v}_i$ on a random $d = O(\frac{\log n}{\varepsilon^2})$ dimensional subspace, and scale the projections up by $\sqrt{n/d}$ to get vectors $\mathbf{v}'_i$, then by the Johnson-Lindenstrauss lemma, with high probability, the squared lengths $\|\mathbf{v}'_i\|^2$ and $\|\mathbf{v}'_i - \mathbf{v}'_j\|^2$ are within $(1 \pm \varepsilon)$ of $\|\mathbf{v}_i\|^2$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2$ respectively, for all $i, j \in [n]$. Thus, we could run the ORACLE with the $\mathbf{X}'$ which is the Gram matrix of the vectors $\mathbf{v}'_i$, and hope that the ORACLE's "feedback" for $\mathbf{X}'$ would be also valid for $\mathbf{X}^{(t)}$. (Note that the exponential is only used for the ORACLE's computation at this step, and never used again in the algorithm.)

Now we mention why Johnson-Lindenstrauss dimension reduction does not suffice in general, and then state conditions under which it does. (All our ORACLEs satisfy these conditions.) The problem is that the s_i and t_{ij} variables could take both positive and negative values, so $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2$ may no longer be non-negative, even though the corresponding sum for the $\mathbf{v}'_i$'s is. But the difference between the two is at most $\varepsilon \sum_i |s_i| \|\mathbf{v}'_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}'_i - \mathbf{v}'_j\|^2$. So if the ORACLE can also ensure that $\sum_i |s_i| \|\mathbf{v}'_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}'_i - \mathbf{v}'_j\|^2 \le C\alpha$, where α is the current estimate of the optimum, and $C = O(\mathrm{polylog}(n))$, then we can set $\varepsilon = \delta/3C$, so that $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge -\frac{\delta}{3}\alpha$. Thus, we have a $\frac{\delta}{3}$-approximate ORACLE (see Section 4.4.2), and so Theorem 16 applies, and we conclude that the Primal-Dual SDP algorithm takes the same number of iterations, up to constant factors. The following lemma formalizes this.

Lemma 22. *In the above setting, if the* ORACLE *always finds values for the s_i's and t_{ij}'s such that $\sum_i s_i \|\mathbf{v}_i\|^2 + \sum_{ij} t_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \ge 0$ and also $\sum_i |s_i| \|\mathbf{v}_i\|^2 + \sum_{ij} |t_{ij}| \|\mathbf{v}_i - \mathbf{v}_j\|^2 \le \tilde{O}(\alpha)$, then the algorithm works even if instead of $\mathbf{v}_i$, the* ORACLE *uses vectors $\mathbf{v}'_i$ obtained by Johnson-Lindenstrauss dimension reduction as above.*

The lemma also works for minimization SDPs analogously.

4.7.2 Matrix exponential-Vector Products

Our task is now reduced to finding the projection of the vectors $\mathbf{v}_i$ on a given unit vector $\mathbf{u}$. Note that $\exp(\mathbf{A}) = \exp(\frac{1}{2}\mathbf{A}) \exp(\frac{1}{2}\mathbf{A})$, so it easy to compute the Cholesky decomposition of a matrix exponential. The Johnson-Lindenstrauss lemma requires computing the projection of the vectors $\mathbf{v}_i$ onto $O(\frac{\log n}{\varepsilon^2})$ random vectors $\mathbf{u}$, which is simply $\exp(\frac{1}{2}\mathbf{A})\mathbf{u}$. This has fast implementations in solvers for linear differential equations, and one can give good complexity bounds.

It suffices to do this approximately: all we have to do is find a vector $\mathbf{v}$ such that $\|\exp(\frac{1}{2}\mathbf{A})\mathbf{u} - \mathbf{v}\| \le \|\exp(\frac{1}{2}\mathbf{A})\|\tau$ for some inverse polynomial τ (we will obtain precise bounds on τ momentarily). We call such a vector $\mathbf{v}$ an "approximation within error parameter τ".

One obvious way to do this is to use the Taylor series expansion for $\exp(\frac{1}{2}\mathbf{A})$, which exploits the sparsity of $\mathbf{A}$. A matrix vector product $\mathbf{Au}$ can be computed in time which is proportional to the number of non-zero entries of $\mathbf{A}$.

Lemma 23. *Let $t_{\mathbf{A}}$ be the time needed to compute the matrix-vector product $\mathbf{Au}$. Then the vector $\mathbf{v} = \sum_{i=0}^{k} \frac{\mathbf{A}^i}{i!}\mathbf{u}$ can be computed in $O(kt_{\mathbf{A}})$ time and if $k \geq \max\{e^2\lambda, \ln(\frac{1}{\tau})\}$ then $\|\exp(\mathbf{A})\mathbf{u} - \mathbf{v}\| \leq \|\exp(\mathbf{A})\|\tau$.*

Proof: It is easy to see that $\mathbf{v}$ can be computed in $O(kt_{\mathbf{A}})$ time, since each successive term $\frac{\mathbf{A}^i}{i!}\mathbf{u}$ can be computed from the previous one with one matrix-vector product. It remains to bound $\|\exp(\mathbf{A})\mathbf{u} - \mathbf{v}\|$. This can be bounded as follows. Let $\lambda = \|\mathbf{A}\|$. We have

$$\|\exp(\mathbf{A})\mathbf{u} - \mathbf{v}\| \;=\; \left\|\exp(\mathbf{A})\mathbf{u} - \textstyle\sum_{i=0}^{k} \dfrac{\mathbf{A}^i}{i!}\mathbf{u}\right\| \;\leq\; \left\|\exp(\mathbf{A}) - \textstyle\sum_{i=0}^{k} \dfrac{\mathbf{A}^i}{i!}\right\| \;\leq\; \textstyle\sum_{i=k+1}^{\infty} \dfrac{\lambda^i}{i!}$$

The last inequality follows because the eigenvalues of $\exp(\mathbf{A}) - \sum_{i=0}^{k} \frac{\mathbf{A}^i}{i!}$ are $\sum_{i=k+1}^{\infty} \frac{\mu^i}{i!}$ where μ is an eigenvalue of $\mathbf{A}$. Now, using standard approximations for the factorial function, it is easy to see that if $k \geq \max\{e^2\lambda, \ln(\frac{1}{\tau})\}$, then $\sum_{i=k+1}^{\infty} \frac{\lambda^i}{i!} \leq e^{\lambda}\tau = \|\exp(\mathbf{A})\|\tau$. $\square$

Since the matrices exponentiated in the Primal-Dual SDP algorithm are $-\varepsilon\sum_{\tau=1}^{t}\mathbf{M}^{(t)}$, for $t = 1, 2, \ldots, T$, and since for all t, $\mathbf{0} \preceq \mathbf{M}^{(t)} \preceq \mathbf{I}$, we conclude that $\|-\varepsilon\sum_{\tau=1}^{t}\mathbf{M}^{(t)}\| \leq \varepsilon T \leq O(\frac{\rho R \ln(n)}{\delta\alpha})$. So when this is small (such as $O(\log(n))$ as in the case of MaxCut, Balanced Separator and Sparsest Cut (both directed and undirected), this simple method is good enough.

However, in some situations, this quantity may be large (like in the case of Min UnCut and Min 2CNF Deletion where it is $\tilde{O}(n)$), it may be prohibitively expensive to use this method. We now present a more sophisticated method, the *Shift-and-Invert Lanczos* (SI-Lanczos) method of van den Eshof and Hochbruck [94], adapted by Iyengar, Phillips, and Stein [57] for the very setting we have. We sketch out this method now.

Let $\mathbf{S} := \sum_{\tau=1}^{t}\mathbf{M}^{(\tau)}$. In the Primal-Dual SDP algorithm, we need to compute $\exp(-\frac{\varepsilon}{2}\mathbf{S})\mathbf{u}$. Now, the idea is that this product is mostly determined by the smallest eigenvalues of $\mathbf{S}$. Thus, one can apply the Lanczos iteration to $(\mathbf{I}+\gamma\mathbf{S})^{-1}$ for some $\gamma > 0$. This emphasizes the eigenvalues of interest.

In each iteration t of the Lanczos method, the most expensive operation is the computing the vector $(\mathbf{I}+\gamma\mathbf{S})^{-1}\mathbf{v}_t$ where $\mathbf{v}_t$ is a unit vector. If we assume that $\mathbf{S}$ is *well-conditioned* (for e.g. if $\mathbf{S} \succ \mathbf{0}$ and the condition number of $\mathbf{I} + \gamma\mathbf{S}$, $\kappa(\mathbf{S}) := \frac{1+\gamma\lambda_1(\mathbf{S})}{1+\gamma\lambda_n(\mathbf{S})}$ is bounded by a constant) then we can use the conjugate gradient method to compute this vector to any relative accuracy τ in $O(\log(\frac{1}{\tau}))$ iterations. Each iteration of the conjugate gradient method requires one matrix-vector product $(\mathbf{I} + \gamma\mathbf{S})\mathbf{u}$ for some vector $\mathbf{u}$.

Now, the key idea is that eventually, we normalize by dividing by the trace. So for any μ,

$$\frac{\exp(-\varepsilon\mathbf{S})}{\mathbf{Tr}(\exp(-\varepsilon\mathbf{S}))} \;=\; \frac{\exp(-\varepsilon(\mu\mathbf{I}+\mathbf{S}))}{\mathbf{Tr}(\exp(-\varepsilon(\mu\mathbf{I}+\mathbf{S})))}.$$

Thus, instead of using $\mathbf{S}$ in the SI-Lanczos algorithm, we can use $\mu\mathbf{I}+\mathbf{S}$. We can now choose μ judiciously, so that $\mathbf{S}$ is well-conditioned. For example, since for all $\tau = 1, 2, \ldots, t$, we have $-\mathbf{I} \preceq \mathbf{M}^{(\tau)} \preceq \mathbf{I}$, we conclude that $-t\mathbf{I} \preceq \mathbf{S} \preceq t\mathbf{I}$. So if we set $\mu = 2t$, then we have $2t\mathbf{I}+\mathbf{S} \succeq t\mathbf{I} \succ \mathbf{0}$ and

$$\kappa(\mathbf{I}+\gamma(2t\mathbf{I}+\mathbf{S})) \;=\; \frac{1+2t\gamma+\gamma\lambda_1(\mathbf{S})}{1+2t\gamma+\gamma\lambda_n(\mathbf{S})} \;\leq\; \frac{1+3t\gamma}{1+t\gamma} \;\leq\; 3.$$

Lemma 24 ([94])**.** *Let $\mathbf{A} \succeq \mathbf{0}$. The SI-Lanczos algorithm with $\gamma = 1/2$ requires $O(\log^2(\frac{1}{\tau}))$ iterations to find a vector $\mathbf{v}$ such that*

$$\|\exp(-\mathbf{A})\mathbf{u}-\mathbf{v}\| \;\leq\; \|\exp(-\mathbf{A})\|\tau,$$

for any given unit vector $\mathbf{u}$. Each iteration of SI-Lanczos takes time $O(t_{\mathbf{A}})$, where $t_{\mathbf{A}}$ is the time needed to compute the matrix-vector product $\mathbf{Au}$. If $\mathbf{A}$ has m non-zero entries, then $t_{\mathbf{A}} = O(m)$, and the product $\exp(\mathbf{A})\mathbf{u}$ can be approximated within error parameter τ in $\tilde{O}(m\log^3(\frac{1}{\tau}))$ time.

We now turn to the question of how small τ needs to be so that the Primal-Dual algorithm can be made to work with the approximate matrix exponential.

Lemma 25. *The Primal-Dual algorithm converges in $O(\frac{\ell\rho R^2\log(n)}{\delta^2\alpha^2})$ iterations if the density matrix is computed using matrix exponential-vector products approximated within the error parameter $\tau = \frac{\delta\alpha}{48n^{5/2}R(\ell+\rho)}$.*

Proof: Let $\mathbf{A} = \varepsilon(2t\mathbf{I}+\mathbf{S})$. In the Primal-Dual SDP algorithm, we construct a candidate solution by taking the density matrix $\mathbf{P} = \frac{\exp(-\mathbf{A})}{\mathbf{Tr}(\exp(-\mathbf{A}))}$ and scaling it by R, to get $\mathbf{X} = R\mathbf{P}$. In response, the Oracle finds a matrix $-\mathbf{I} \preceq \mathbf{M} \preceq \mathbf{I}$ such that $\mathbf{M} \bullet \mathbf{P} \geq \frac{\pm\ell}{\ell+\rho}$, where the sign depends on the current iteration (i.e. the sign is that of $\ell^{(t)}$ in the description of the algorithm).

The Cholesky factorization of $\exp(-\mathbf{A})$ is $\exp(-\frac{1}{2}\mathbf{A})\exp(-\frac{1}{2}\mathbf{A})$. To compute the projection of the rows of $\exp(-\frac{1}{2}\mathbf{A})$ on a $d = O(\frac{\log n}{\varepsilon^2})$ dimensional subspace, scaled up by $\sqrt{n/d}$ we consider a matrix $\mathbf{U} \in \mathbb{R}^{n\times d}$ whose d columns are of length $\sqrt{n/d}$ and form an orthogonal basis for the subspace. Then the projected vectors are the rows of $\mathbf{W} := \exp(-\frac{1}{2}\mathbf{A})\mathbf{U}$. We showed in the previous section that the Primal-Dual SDP algorithm can be made to work using the density matrix $\mathbf{P}' = \frac{\mathbf{W}\mathbf{W}^\top}{\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top)}$ instead of $\mathbf{P}$.

Now, by Lemma 24 above, we can compute a matrix $\mathbf{V} \in \mathbb{R}^{n\times d}$ such that each column vector of $\mathbf{V}$ is within an a radius of $\sqrt{n/d}\|\exp(-\frac{1}{2}\mathbf{A})\|\tau$ of the corresponding column vector in $\mathbf{W}$.

We use the density matrix $\mathbf{P}'' = \frac{\mathbf{V}\mathbf{V}^\top}{\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)}$ as an approximation to $\mathbf{P}'$. We now estimate the error in this approximation. Let $\mathbf{E} = \mathbf{W}-\mathbf{V}$. Then we have $\|\mathbf{E}\|_F \leq \sqrt{n}\|\exp(-\frac{1}{2}\mathbf{A})\|\tau$ by the bound from Lemma 24. Here, $\|\cdot\|_F$ is the Frobenius norm. Now, consider

$$\begin{aligned}\|\mathbf{W}\mathbf{W}^\top - \mathbf{V}\mathbf{V}^\top\| &= \|-\mathbf{E}\mathbf{U}^\top\exp(-\tfrac{1}{2}\mathbf{A})^\top - \exp(-\tfrac{1}{2}\mathbf{A})\mathbf{U}\mathbf{E}^\top + \mathbf{E}\mathbf{E}^\top\| \\ &\leq 2\|\mathbf{E}\|\|\exp(-\tfrac{1}{2}\mathbf{A})\|\|\mathbf{U}\| + \|\mathbf{E}\|^2 \\ &\leq 3n\|\exp(-\mathbf{A})\|\tau. \qquad (4.10)\end{aligned}$$

Here, we used the facts that $\|\exp(-\frac{1}{2}\mathbf{A})\|^2 = \|\exp(-\mathbf{A})\|$ and $\|\mathbf{U}\| = \sqrt{n/d}$. Also, we have

$$\begin{aligned}
|\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top) - \mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)| &= |-2\mathbf{Tr}(\mathbf{E}\mathbf{U}^\top \exp(-\tfrac{1}{2}\mathbf{A})^\top) + \mathbf{Tr}(\mathbf{E}\mathbf{E}^\top)| \\
&\leq 2|\mathbf{Tr}(\mathbf{E}\mathbf{U}^\top \exp(-\tfrac{1}{2}\mathbf{A})^\top)| + |\mathbf{Tr}(\mathbf{E}\mathbf{E}^\top)| \\
&\leq 2\sqrt{n}\|\mathbf{E}\mathbf{U}^\top \exp(-\tfrac{1}{2}\mathbf{A})^\top\|_F + \|\mathbf{E}\|_F^2 \\
&\leq 2\sqrt{n}\|\mathbf{E}\|_F\|\mathbf{U}\|_F\|\exp(-\tfrac{1}{2}\mathbf{A})\|_F + \|\mathbf{E}\|_F^2 \\
&\leq 3n^{3/2}\|\exp(-\mathbf{A})\|\tau. \qquad (4.11)
\end{aligned}$$

By the Johnson-Lindenstrauss Lemma, with high probability the rows of $\mathbf{W}$ have lengths within $1 \pm \varepsilon$ of the lengths of the corresponding rows in $\exp(-\frac{1}{2}\mathbf{A})$. Since $\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top)$ is the sum of the squared lengths of rows in $\mathbf{W}$, and $\mathbf{Tr}(\exp(-\mathbf{A}))$ is the sum of the squared lengths of rows in $\exp(-\frac{1}{2}\mathbf{A})$, we have

$$|\mathbf{Tr}(\mathbf{W}\mathbf{W})^\top - \mathbf{Tr}(\exp(-\mathbf{A}))| \leq \varepsilon^2\mathbf{Tr}(\exp(-\mathbf{A})). \qquad (4.12)$$

Now, putting everything together, we have

$$\begin{aligned}
\|\mathbf{P}' - \mathbf{P}''\| &= \left\|\frac{\mathbf{W}\mathbf{W}^\top}{\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top)} - \frac{\mathbf{V}\mathbf{V}^\top}{\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)}\right\| \\
&\leq \left\|\frac{\mathbf{W}\mathbf{W}^\top}{\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top)} - \frac{\mathbf{W}\mathbf{W}^\top}{\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)}\right\| + \left\|\frac{\mathbf{W}\mathbf{W}^\top}{\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)} - \frac{\mathbf{V}\mathbf{V}^\top}{\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)}\right\| \\
&\leq \frac{\|\mathbf{W}\mathbf{W}^\top\||\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top) - \mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)|}{\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top)\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)} + \frac{\|\mathbf{W}\mathbf{W}^\top - \mathbf{V}\mathbf{V}^\top\|}{\mathbf{Tr}(\mathbf{V}\mathbf{V}^\top)} \\
&\leq \frac{(3n^{3/2} + 3n)\|\exp(-\mathbf{A})\|\tau}{\mathbf{Tr}(\mathbf{W}\mathbf{W}^\top) - 3n^{3/2}\|\exp(-\mathbf{A})\|\tau} \qquad \text{from (4.10) and (4.11)} \\
&\leq \frac{(3n^{3/2} + 3n)\|\exp(-\mathbf{A})\|\tau}{(1 - \varepsilon^2 - 3n^{3/2}\tau)\mathbf{Tr}(\exp(-\mathbf{A}))} \qquad \text{from (4.12)} \\
&\leq \frac{(3n^{3/2} + 3n)\tau}{1 - \varepsilon^2 - 3n^{3/2}\tau} \\
&\leq 8n^{3/2}\tau,
\end{aligned}$$

if we set $\tau \leq 1/12n^{3/2}$ and $\varepsilon \leq 1/2$. Here, we also used the facts that for any symmetric matrix $\mathbf{B}$, we have $\|\mathbf{B}\| \leq \mathbf{Tr}(\mathbf{B})$.

Thus, for any matrix $-\mathbf{I} \preceq \mathbf{M} \preceq \mathbf{I}$ found by the Oracle, we have

$$\begin{aligned}
|\mathbf{P}' \bullet \mathbf{M} - \mathbf{P}'' \bullet \mathbf{M}| &= |(\mathbf{P}' - \mathbf{P}'') \bullet \mathbf{M}| \\
&\leq |8n^{3/2}\tau\mathbf{I} \bullet \mathbf{M}| \qquad \because -8n^{3/2}\tau\mathbf{I} \preceq \mathbf{P}' - \mathbf{P}'' \preceq 8n^{3/2}\tau\mathbf{I} \\
&\leq 8n^{5/2}\tau \qquad \because -\mathbf{I} \preceq \mathbf{M} \preceq \mathbf{I}
\end{aligned}$$

Finally, if we set $\tau = \frac{\delta\alpha}{48n^{5/2}R(\ell+\rho)}$, then we have $\mathbf{M} \bullet \mathbf{P}' \geq \mathbf{M} \bullet \mathbf{P}'' - \frac{\delta\alpha}{6R(\ell+\rho)}$. Assuming the Oracle finds $\mathbf{M}$ such that $\mathbf{M} \bullet \mathbf{P}'' \geq \frac{\pm\ell}{\ell+\rho}$, then we have $\mathbf{M} \bullet \mathbf{P}' \geq \frac{\frac{-\delta\alpha}{6R}\pm\ell}{\ell+\rho}$.

Now, we set the dimension of the subspace d appropriately so that the Johnson-Lindenstrauss lemma implies that we have a $\mathbf{M} \bullet \mathbf{P} \geq \mathbf{M} \bullet \mathbf{P}' - \frac{\delta\alpha}{6R(\ell+\rho)}$. Thus, we get that $\mathbf{M} \bullet \mathbf{P} \geq \frac{-\frac{\delta\alpha}{3R}\pm\ell}{\ell+\rho}$, and so the Primal-Dual SDP algorithm converges in $O(\frac{\ell\rho R^2\log(n)}{\delta^2\alpha^2})$ rounds, just as in the proof of Theorem 16. $\square$

Chapter 5

Derandomization and Quantum Algorithms

In this chapter, we discuss a few additional applications of the Matrix Multiplicative Weights algorithm of Chapter 3 in derandomization and quantum computing.

The basic idea in two of these applications is the same: there is a certain matrix whose eigenvalues need to be bounded, and this is achieved by the Matrix Multiplicative Weights algorithm. In the case of derandomizing the Alon-Roichman construction of expander graphs (Section 5.1), the second largest eigenvalues of the normalized adjacency matrix of the graph needs to be bounded away from 1. In the quantum hypergraph covering problem (Section 5.2), a minimal multi-set of matrices needs to be picked from a given collection such that the smallest eigenvalue of their sum is at least 1.

Finally, in the problem of learning quantum states (Section 5.3), the Matrix Multiplicative Weights algorithm implicitly gives an efficient algorithm to produce a quantum state that is consistent with a given set of measurements, and thus gives an upper bound on a complexity measure (related to the VC-dimension) of the hypothesis space of quantum states.

5.1 Derandomization of the Alon-Roichman Theorem

Expander graphs are extremely useful tools in computer science, with applications ranging from derandomization, to hardness of approximation results, to error-correcting codes (see [54] for a comprehensive survey). Explicit constructions of expanders are important for various deterministic constructions. The Alon-Roichman theorem [9] gives a generic way to construct an expander graph from any algebraic group by random sampling. We derandomize the proof of the Alon-Roichman theorem given by [72] (see also [79]) to give a deterministic and efficient construction of the expanding generating set. A similar result was also obtained by Wigderson and Xiao [101].

We start by describing expander graphs. Given a connected undirected d-regular graph $G = (V, E)$ on n vertices, let $\mathbf{A}$ be the normalized adjacency matrix, i.e. $A_{ij} = \frac{w_{ij}}{d}$ where w_{ij} is the number of edges between vertices i and j, including self-loops and multiple edges.

Note that $\mathbf{A}$ is a real, symmetric matrix. We have $\lambda_1(\mathbf{A}) = 1$, and the corresponding eigenvector is $\mathbf{e}$, which is the all 1's vector scaled down by $\sqrt{n}$. The graph G is called an expander if $\lambda_2(\mathbf{A}) \leq \beta$ for some constant $\beta < 1$.

Now, let H be an arbitrary algebraic group of order n. A multi-set of elements of H, S is called symmetric if for all $i \in H$, the multiplicities i and i^{-1} in S are the same. The Cayley graph $\mathrm{Cay}(H; S)$ is the $|S|$-regular graph whose vertex set is the elements of H, and for $i, j \in H$, we have $w_{ij} = |\{k \in S : \ i \circ k = j\}|$, where $\circ$ is the group operation. Alon and Roichman [9] prove the following theorem:

Theorem 25 ([9])**.** *Fix $\beta < 1$. For an arbitrary group H of size n, by picking a random multi-set of size $O(\frac{1}{\beta^2}\log n)$ and taking its symmetric closure $S = T \sqcup T^{-1}$ we have*

$$\Pr[\lambda_2(\mathrm{Cay}(H; S)) \leq \beta] \ > \ 0.$$

Here, the symmetric closure of a multi-set T of elements of H, denoted by $T \sqcup T^{-1}$, is the set S such that for each element $i \in T$, we include i and i^{-1} in S. We have the following derandomization of this theorem, using the Matrix Multiplicative Weights algorithm:

Theorem 26. *There is a deterministic algorithm which, given β and H, runs in $\tilde{O}(n^3/\beta^2)$ time and produces a set T of size $O(\frac{1}{\beta^2}\log n)$ so that if $S = T \sqcup T^{-1}$, then we have*

$$\lambda_2(\mathrm{Cay}(H; S)) \ \leq \ \beta.$$

Proof: For every element $i \in H$, define $\mathbf{N}(i) = \frac{1}{2}(\mathbf{R}(i) + \mathbf{R}(i^{-1}) - \frac{2}{n}\mathbf{J})$, where $\mathbf{R}(i)$ is the permutation matrix associated with i (i.e. $\mathbf{R}(i)_{jk} = 1$ if $k = j \circ i$, and 0 otherwise), and $\mathbf{J}$ is the all ones matrix. Now, it is easy to check that for any multi-set $T \subseteq H$ and $S = T \sqcup T^{-1}$, we have the normalized adjacency matrix of $\mathrm{Cay}(H; S)$ is given by $\mathbf{A} := \frac{1}{|T|}\sum_{t=1}^{T} \frac{1}{2}(\mathbf{R}(i) + \mathbf{R}(i^{-1}))$. Since $\lambda_1(\mathbf{A}) = 1$ and the corresponding eigenvector is $\mathbf{e}$, we conclude that

$$\begin{aligned}
\lambda_2(\mathbf{A}) \ &= \ \lambda_1\Big(\tfrac{1}{|T|}\textstyle\sum_{i\in T}\frac{1}{2}(\mathbf{R}(i) + \mathbf{R}(i^{-1})) - \mathbf{e}\mathbf{e}^\top\Big) \\
&= \ \lambda_1\Big(\tfrac{1}{|T|}\textstyle\sum_{i\in T}\frac{1}{2}(\mathbf{R}(i) + \mathbf{R}(i^{-1}) - \frac{2}{n}\mathbf{J})\Big) \\
&= \ \lambda_1\left(\tfrac{1}{|T|}\textstyle\sum_{i\in T}\mathbf{N}(i)\right).
\end{aligned}$$

The algorithm consists of an online matrix game in the gain form (see Section 3.2.1). In every round, we choose an element $i \in H$ and the corresponding gain matrix is $\mathbf{M}(i) = \frac{1}{2}(\mathbf{N}(i) + \mathbf{I})$. We now estimate $\|\mathbf{N}(i)\|$. Note that $\mathbf{N}(i)\mathbf{e} = 0$, so the largest eigenvalue of $\mathbf{N}(i)$ in absolute value corresponds to a unit vector $\mathbf{v}$ that is perpendicular to $\mathbf{e}$. For such a vector $\mathbf{v}$, we have

$$\|\mathbf{N}(i)\mathbf{v}\| \ = \ \|\tfrac{1}{2}(\mathbf{R}(i) + \mathbf{R}(i^{-1}))\mathbf{v}\| \ \leq \ \tfrac{1}{2}[\|\mathbf{R}(i)\| + \|\mathbf{R}(i^{-1})\|] \ = \ 1,$$

since $\|\mathbf{R}(i)\| = \|\mathbf{R}(i^{-1})\| = 1$. Thus, $\|\mathbf{N}(i)\| \leq 1$, and so the gain matrices satisfy $\mathbf{0} \preceq \mathbf{M}(i) \preceq \mathbf{I}$. In every round i [1], given a density matrix $\mathbf{P}^{(i)}$, the gain matrix chosen is

[1]For convenience of notation, we index rounds by i rather than t.

$\mathbf{M}(i)$ for any i which satisfies

$$\mathbf{M}(i) \bullet \mathbf{P}^{(i)} \;\leq\; \frac{1}{2}. \tag{5.1}$$

We show later that such an i always exists and so it can be found by exhaustive search.

Suppose the game is played for T rounds. With some abuse of notation, we also let T be the multi-set of elements i played in the T rounds. Let $S = T \sqcup T^{-1}$. By Corollary 4 to Theorem 11, we have

$$(1+\varepsilon)\frac{T}{2} \;\geq\; (1+\varepsilon)\textstyle\sum_{i\in T}\mathbf{M}(i)\bullet\mathbf{P}^{(i)} \;\geq\; \lambda_1\big(\textstyle\sum_{i\in T}\mathbf{M}(i)\big) - \dfrac{\ln n}{\varepsilon}.$$

Divide by T, and note that

$$\tfrac{1}{T}\lambda_1\left(\textstyle\sum_{i\in T}\mathbf{M}(i)\right) \;=\; \tfrac{1}{2}\left(\lambda_1\big(\tfrac{1}{T}\textstyle\sum_{i\in T}\mathbf{N}(i)\big)+1\right).$$

Rearranging, and setting $\varepsilon = \frac{\beta}{2}$ and $T = \frac{8\ln n}{\beta^2}$, we get that

$$\lambda_1\big(\tfrac{1}{T}\textstyle\sum_{i\in T}\mathbf{N}(i)\big) \;\leq\; \beta,$$

as required.

We only need to show the existence of the element $i \in H$ satisfying (5.1) as claimed. For any density matrix $\mathbf{P}$, we have

$$\frac{1}{n}\sum_{i\in H}\mathbf{M}(i)\bullet\mathbf{P} \;=\; \frac{1}{n}\sum_{i\in H}\frac{1}{2}(\mathbf{N}(i)+\mathbf{I})\bullet\mathbf{P} \;=\; \frac{1}{2n}\sum_{i\in H}\mathbf{N}(i)\bullet\mathbf{P} + \frac{1}{2}\mathbf{I}\bullet\mathbf{P}.$$

Now, we have $\sum_{i\in H}\mathbf{N}(i) = \sum_{i\in H}\mathbf{R}(i) - \mathbf{J} = \mathbf{0}$, since for any $j,k \in H$, there is a unique i such that $\mathbf{R}(i)_{jk} = 1$, viz. $i = j^{-1}\circ k$. Thus,

$$\frac{1}{n}\sum_{i\in H}\mathbf{M}(i)\bullet\mathbf{P} \;=\; \frac{1}{2}\mathbf{I}\bullet\mathbf{P} \;=\; \frac{1}{2}$$

and so there is an element $i \in H$ such that $\mathbf{M}(i)\bullet\mathbf{P} \leq \frac{1}{2}$. $\square$

5.2 Covering Quantum Hypergraphs

In the Set Cover problem, we are given a universe V of size n, and a collection of subsets of V, E, whose union is V. The Set Cover problem is to find the smallest collection of subsets of V from E whose union is V (alternatively, we can imagine a hypergraph with V being the set of vertices and E being the set of hyperedges, and the goal is to find the smallest set of hyperedges which cover all vertices). The Set Cover problem can be cast in a matrix framework as follows. For every subset of nodes $S \in E$, we associate a diagonal matrix $\mathbf{M}(S)$ which has the indicator vector for S on the diagonal. Note that such matrices satisfy $\mathbf{0} \preceq \mathbf{M}(S) \preceq \mathbf{I}$. A collection of subsets $E' \subseteq E$ covers all elements in V if and only if $\sum_{S\in E'}\mathbf{M}(S) \succeq \mathbf{I}$. The greedy algorithm of repeatedly picking subsets that cover the maximum number of uncovered elements obtains an $O(\log n)$ approximation to the Set Cover problem.

The Quantum Hypergraph Cover problem is a quantum generalization of the Set Cover problem, defined by Ahlswede and Winter [4]. A quantum hypergraph is $\Gamma = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a n-dimensional Hilbert space and $\mathcal{E}$ is a finite set of m hyperedges e which are identified with a Hermitian operator $\mathbf{M}(e)$ such that $\mathbf{0} \preceq \mathbf{M}(e) \preceq \mathbf{I}$.

A cover of the quantum hypergraph Γ is a multi-set $\mathcal{C}$ of hyperedges e such that $\sum_{e \in \mathcal{C}} \mathbf{M}(e) \succeq \mathbf{I}$. The cover number $c(\Gamma)$ is the size of the smallest cover of Γ. The problem is to approximate $c(\Gamma)$. This problem has important applications in quantum information theory, see [4] for more details.

If $c(\Gamma) > m$, then we can approximate it to within a factor of 2, by solving an associated covering SDP and rounding up the fractional solution, see Lemma 26. We therefore assume that $c(\Gamma) \leq m$.

Ahlswede and Winter [4] showed that by solving the associated covering SDP and running a randomized rounding algorithm on the fractional solution, we obtain a cover of size $O(\log n) \cdot c(\Gamma)$. We derandomize their algorithm using the Matrix Multiplicative Weights algorithm. A similar algorithm was independently discovered by Wigderson and Xiao [101], by derandomizing the randomized rounding of the associated SDP using the method of pessimistic estimators.

Theorem 27. *There is a deterministic algorithm that obtains an $O(\log n)$ approximation to the Quantum Hypergraph Cover problem in $\tilde{O}(n^3 \cdot c(\Gamma))$ time. This is a polynomial time algorithm if we assume that $c(\Gamma) \leq m$.*

Proof: The algorithm consists of an online matrix game. For every hyperedge $e \in \mathcal{E}$, we associate the loss matrix $\mathbf{M}(e)$. We run the Matrix Multiplicative Weights algorithm with this setup, with the parameter $\varepsilon = \frac{1}{2}$. At each step t, we choose the hyperedge $e^{(t)} = \arg\max_e \mathbf{M}(e) \bullet \mathbf{P}^{(t)}$. We claim that $e^{(t)}$ satisfies

$$\mathbf{M}(e^{(t)}) \bullet \mathbf{P}^{(t)} \;\geq\; \frac{1}{c(\Gamma)}. \tag{5.2}$$

This is because for the optimal cover $\mathcal{C}^*$, we have

$$\sum_{e \in \mathcal{C}^*} \mathbf{M}(e) \bullet \mathbf{P}^{(t)} \;\geq\; \mathbf{I} \bullet \mathbf{P}^{(t)} \;=\; 1,$$

so some $e \in \mathcal{C}^*$ must satisfy (5.2). By Corollary 3 to Theorem 10, we get

$$\frac{T}{2c(\Gamma)} \;\leq\; \lambda_n \left(\textstyle\sum_{t=1}^{T} \mathbf{M}(e^{(t)}) \right) + 2\ln(n).$$

By setting $T = (4\ln(n) + 2) \cdot c(\Gamma)$, and rearranging the inequality we get

$$\lambda_n \left(\textstyle\sum_{t=1}^{T} \mathbf{M}(e^{(t)}) \right) \;\geq\; 1 \quad \Longrightarrow \quad \sum_{t=1}^{T} \mathbf{M}(e^{(t)}) \succeq \mathbf{I}.$$

So we have a cover $\mathcal{C} = \{e^{(1)}, e^{(2)}, \ldots, e^{(T)}\}$ of size $O(\log n) \cdot c(\Gamma)$ as desired. □

Lemma 26. *If $c(\Gamma) > m$, then there is a deterministic algorithm that obtains a quantum hypergraph cover of size at most $2c(\Gamma)$.*

Proof: Consider the following covering SDP:

$$\begin{array}{rcl} \min \ \sum_e y_e & & \\ \sum_e y_e \mathbf{M}(e) & \succeq & \mathbf{I} \\ \forall e: \quad y_e & \geq & 0 \end{array}$$

If we restrict y_e to non-negative integers, then the SDP becomes equivalent to the Quantum Hypergraph Cover problem, so the optimum of the SDP is at most $c(\Gamma)$. We solve the SDP to obtain a fractional solution y_e^* for every hyperedge e, such that $\sum_e y_e^* \mathbf{M}(e) \succeq \mathbf{I}$ and $\sum_e y_e^* \leq c(\Gamma)$. Then the integer solution which takes $\tilde{y}_e := \lceil y_e^* \rceil$ copies of hyperedge e also satisfies $\sum_e \tilde{y}_e \mathbf{M}(e) \succeq \mathbf{I}$ and $\sum_e \tilde{y}_e \leq \sum_e (y_e^* + 1) \leq c(\Gamma) + m \leq 2c(\Gamma)$, and so we have our desired 2-approximation. $\Box$

5.3 Fat-shattering dimension of quantum states

Aaronson [1] considered the problem of learning a quantum state. Suppose we have a physical process that produces a quantum state. By applying the process repeatedly, we can prepare as many copies of the state as we want, and can then measure each copy in a basis of our choice. The goal is to learn an approximate description of the state by combining the various measurement outcomes.

An n-qubit quantum state can be represented by a density matrix $\mathbf{P}$ in the Hilbert space of dimension 2^n. A measurement of $\mathbf{P}$ is two-outcome positive operator valued measure (POVM), which is represented by a matrix $\mathbf{F}$ such that $\mathbf{0} \preceq \mathbf{F} \preceq \mathbf{I}$. If we measure $\mathbf{P}$ using $\mathbf{F}$, the outcome is 1 with probability $\mathbf{F} \bullet \mathbf{P}$, and 0 with probability $1 - \mathbf{F} \bullet \mathbf{P}$. Let $\mathcal{S}$ be the set of all two-outcome POVMs. The function $f_{\mathbf{P}} : \mathcal{S} \to [0, 1]$ defined as $f_{\mathbf{P}}(\mathbf{F}) = \mathbf{F} \bullet \mathbf{P}$ is a probabilistic concept (p-concept) as defined by Kearns and Schapire [61]. In general, a p-concept is a mapping from the set of learning examples to $[0, 1]$ which assigns to each example the probability that its label is 1.

We now discuss the learning problem in detail. There is an unknown quantum state, $\mathbf{P}^*$, that we can prepare as many times as needed. We now sample measurements from an unknown distribution $\mathcal{D}$ on $\mathcal{S}$, and measure $\mathbf{P}^*$ with the sampled distributions. For the purpose of this section, we assume that the measurement actually yields the value of $f_{\mathbf{P}^*}(\mathbf{F}) = \mathbf{F} \bullet \mathbf{P}^*$ for any measurement $\mathbf{F}$, though Aaronson [1] shows that this assumption is not necessary [2]. The goal is to learn a quantum state $\tilde{\mathbf{P}}$ using as few measurements as possible that approximates $\mathbf{P}^*$ in the following sense:

$$\mathbf{Pr}_{\mathbf{F} \in \mathcal{D}}[|f_{\tilde{\mathbf{P}}}(\mathbf{F}) - f_{\mathbf{P}^*}(\mathbf{F})| \leq \gamma] \ \geq \ 1 - \varepsilon,$$

where $\gamma, \varepsilon > 0$ are error parameters.

[2] Thus, strictly speaking, we no longer have a p-concept, rather a real valued function that we're trying to learn.

This Probably Approximately Correct (PAC)-style learning framework also has an analogous concept to the Vapnik-Chervonenkis (VC) dimension in standard PAC learning, viz. the fat-shattering dimension. Just as the VC dimension enables us to bound sample complexity in standard PAC learning (see Blumer *et al* [25]), the fat-shattering dimension enables us to bound the sample complexity of learning p-concepts, by the results of Kearns and Schapire [61], Anthony and Bartlett [11], and Bartlett and Long [20].

We define the concept of fat-shattering dimension only in the current context. Let $\mathcal{C}_n = \{f_{\mathbf{P}}\}_{\mathbf{P}}$ be the set of the p-concepts induced by all density matrices.

Definition 3. *Let $\mathcal{T} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_k\} \subseteq \mathcal{S}$ be a set of measurements of size k. Then $\mathcal{T}$ is said to be γ-fat shattered by $\mathcal{C}_n$ if there are real numbers $\alpha_1, \alpha_2, \ldots, \alpha_k \in [0,1]$ such that for all k-bit vectors $\mathbf{y} = \langle y_1, y_2, \ldots, y_k \rangle \in \{0,1\}^k$, there is a density matrix $\mathbf{P}_y$ such that for all i,*

$$\begin{aligned} \text{if } \ y_i = 0 \ \text{ then } \ f_{\mathbf{P}_y}(\mathbf{F}_i) &\geq \alpha_i + \gamma \\ \text{if } \ y_i = 1 \ \text{ then } \ f_{\mathbf{P}_y}(\mathbf{F}_i) &\leq \alpha_i - \gamma \end{aligned}$$

or equivalently, for all i,

$$(-1)^{y_i}(\mathbf{F}_i - \alpha_i \mathbf{I}) \bullet \mathbf{P}_y \geq \gamma.$$

The γ-fat shattering dimension of $\mathcal{C}_n$, $\mathrm{fat}(\gamma)$, is defined to be the largest k such that there is a set of k measurements which is γ-fat shattered by $\mathcal{C}_n$.

Aaronson shows that the results of Anthony and Bartlett [11] and Bartlett and Long [20] imply the following:

Theorem 28. *Let $\varepsilon, \gamma, \delta > 0$ be given error parameters. Let X be a set of m sample measurements drawn i.i.d. from $\mathcal{D}$. Suppose that there is an algorithm that produces a quantum state $\tilde{\mathbf{P}}$ such that for any measurement $\mathbf{F} \in X$, we have $|f_{\tilde{\mathbf{P}}}(\mathbf{F}) - f_{\mathbf{P}^*}(\mathbf{F})| \leq \frac{\gamma\varepsilon}{7}$. Then, for some constant c, with probability at least $1-\delta$ over the choice of the training set X, we have*

$$\mathbf{Pr}_{\mathbf{F}\in\mathcal{D}}[|f_{\tilde{\mathbf{P}}}(\mathbf{F}) - f_{\mathbf{P}^*}(\mathbf{F})| \leq \gamma] > 1 - \varepsilon,$$

provided

$$m \ \geq \ \tfrac{c}{\gamma^2\varepsilon^2}\left(\mathrm{fat}(\tfrac{\gamma\varepsilon}{35}) \cdot \log^2\left(\tfrac{1}{\gamma\varepsilon}\right) + \log\left(\tfrac{1}{\delta}\right)\right).$$

The main technical part is therefore to bound the γ-fat shattering dimension of $\mathcal{C}_n$. Aaronson proves the following theorem using a lower bound due to Ambainis *et al* [10] for quantum codes. We give an alternative proof using the Matrix Multiplicative Weights algorithm.

The approach taken in this section is analogous to the $O(\frac{1}{\rho^2})$ bound one obtains on the fat-shattering dimension of hyperplane classifiers with margin ρ using the perceptron algorithm: the idea is that the fat-shattering dimension gives a lower bound on the number of mistakes an algorithm for the learning problem must make in the worst case; conversely, an upper bound on the number of mistakes made by an algorithm gives an upper bound on the fat-shattering dimension. In our setting, the Matrix Multiplicative Weights algorithm is used implicitly as a learning algorithm. Tsuda, Rätsch and Warmuth [92] also consider a similar learning algorithm.

Theorem 29. *The γ-fat shattering dimension of $\mathcal{C}_n$ is $O(\frac{n}{\gamma^2})$.*

Proof: Let $\mathcal{T} = \{\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_k\} \subseteq \mathcal{S}$ be a set of measurements of size k which is γ-fat shattered by $\mathcal{C}_n$. Let $\alpha_1, \alpha_2, \ldots, \alpha_k \in [0,1]$ be the corresponding threshold parameters. Given a bit vector $\mathbf{y} = \langle y_1, y_2, \ldots, y_k \rangle$, we will show, using the Matrix Multiplicative Weights algorithm, a procedure to obtain a density matrix $\mathbf{P}$ such that for all i,

$$(-1)^{y_i}(\mathbf{F}_i - \alpha_i \mathbf{I}) \bullet \mathbf{P} \ \geq \ \frac{\gamma}{2} \tag{5.3}$$

by inspecting only $O(\frac{n}{\gamma^2})$ of the measurements in $\mathcal{T}$. This implies that $k = O(\frac{n}{\gamma^2})$, because if k were bigger, then there is measurement, say $\mathbf{F}_i$, which is not inspected by the algorithm. Thus, we could set its bit y_i to be such that $(-1)^{y_i}(\mathbf{F}_i - \alpha_i \mathbf{I}) \bullet \mathbf{P} \leq 0$, which contradicts (5.3).

We now describe the algorithm. We run the Matrix Multiplicative Weights algorithm in the gain form (see Section 3.2.1) with the gain matrices $\mathbf{M}_i := \frac{1}{2}[(-1)^{y_i}(\mathbf{F}_i - \alpha_i \mathbf{I}) + \mathbf{I}]$. Note that $\mathbf{0} \preceq \mathbf{M}_i \preceq \mathbf{I}$, since $-\mathbf{I} \preceq (-1)^{y_i}(\mathbf{F}_i - \alpha_i \mathbf{I}) \preceq \mathbf{I}$.

In each round, given the density matrix $\mathbf{P}$ generated by the Matrix Multiplicative Weights algorithm, we choose the loss matrix $\mathbf{M}_i$ which maximizes $\mathbf{M}_i \bullet \mathbf{P}$. If it is the case that for all i, $\mathbf{M}_i \bullet \mathbf{P} \geq \frac{1}{2}(1 + \frac{\gamma}{2})$, then we stop, because $\mathbf{P}$ now satisfies (5.3). Otherwise, we continue. After T iterations, by Corollary 4 to Theorem 11, for any density matrix $\mathbf{P}$, we have

$$(1+\varepsilon)\sum_{t=1}^{T} \mathbf{M}_{i^t} \bullet \mathbf{P}^t \ \geq \ \sum_{t=1}^{T} \mathbf{M}_{i^t} \bullet \mathbf{P} - \frac{\ln(2^n)}{\varepsilon}.$$

The left hand side is bounded from above by $\frac{T}{2}(1+\varepsilon)(1+\frac{\gamma}{2})$ by assumption. Now, since $\mathcal{T}$ is γ-fat shattered by $\mathcal{C}_n$, there is density matrix $\mathbf{P}_\mathbf{y}$ such that for all i, $(-1)^{y_i}(\mathbf{F}_i - \alpha_i \mathbf{I}) \bullet \mathbf{P}_\mathbf{y} \geq \gamma$. By substituting $\mathbf{P} = \mathbf{P}_\mathbf{y}$, we have $\mathbf{M}_i \bullet \mathbf{P}_\mathbf{y} \geq 1 + \frac{\gamma}{2}$ for all i, so we can lower bound the first term on the right hand side by $\frac{T}{2}(1+\gamma)$. Thus, we get

$$\frac{T}{2}(1+\varepsilon)(1+\tfrac{\gamma}{2}) \ \geq \ \frac{T}{2}(1+\gamma) - \frac{\ln(2)}{\varepsilon}n.$$

If we set $\varepsilon = \gamma/5$, then assuming $\gamma \leq 1/2$, we have $(1+\varepsilon)(1+\frac{\gamma}{2}) \leq 1 + \frac{3\gamma}{4}$, and so this inequality implies that $T \leq \frac{40\ln(2)n}{\gamma^2}$. Thus, within $\frac{40\ln(2)}{\gamma^2}n$ iterations, we will obtain the desired density matrix. □

Remark: Aaronson obtains a better constant factor than the $40\ln(2)$ factor we obtain in Theorem 29. It is possible to improve our constant somewhat by relaxing condition (5.3) to require only non-negativity of the LHS.

Chapter 6

Fast Algorithms for Approximate Semidefinite Programming

In this chapter, we return to the Semidefinite Programming (SDP) problem of Chapter 4. Many recent combinatorial optimization algorithms utilize SDP at their core, and as discussed earlier, even though interior point methods solve such SDPs quite efficiently, specialized algorithms which obtain faster running time using the primal-dual schema or Lagrangian relaxation are still quite desirable, and quite rare in the SDP world. The results of Chapter 4 gave primal-dual algorithms for SDP; in this chapter, we describe how fast algorithms for SDP can be designed using Lagrangian relaxation techniques. These algorithms make use of the basic Multiplicative Weights algorithm in the manner of Section 2.3.2, and they directly work on the constraints of the primal SDP. In this sense, they may be called *primal-only* methods, in contrast to the primal-dual algorithms of Chapter 4. We apply this method to the problems of MaxQP, a quadratic programming problem which has MaxCut as a subcase, HaploFreq, a probability estimation problem in computational biology, and Embedding, the problem of embedding a finite metric space in ℓ_2 with the least distortion, and obtain fast (approximation) algorithms for each of them.

The results of this chapter originally appeared in a joint paper with Sanjeev Arora and Elad Hazan [14].

6.1 Semidefinite Programming: a recapitulation

We recall the notation of Chapter 4, and we consider minimization SDPs of the following form:

$$
\begin{aligned}
&\min\ \mathbf{C} \bullet X \\
\forall j \in [m] :\ \ &\mathbf{A}_j \bullet \mathbf{X} \ \geq\ b_j \\
&\mathbf{Tr}(\mathbf{X}) \ \leq\ R \\
&\mathbf{X} \ \succeq\ 0
\end{aligned}
\tag{6.1}
$$

Here, $\mathbf{X} \in \mathbb{R}^{n\times n}$ is a matrix of variables and $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m \in \mathbb{R}^{n\times n}$. Also, $[m]$ is notation for the set $\{1, 2, \ldots, m\}$. Throughout this chapter, without loss of generality, we assume that all matrices are real and symmetric. For matrices $\mathbf{A}, \mathbf{B}$, we use the notation $\mathbf{A} \bullet \mathbf{B} = \mathbf{Tr}(\mathbf{AB}) = \sum_{ij} A_{ij}B_{ij}$. This SDP is quite general. The trace bound is the only non-standard constraint.

As discussed in the introductory chapter, the first polynomial-time algorithm for semidefinite programming (strictly speaking, an approximation algorithm that computes the solution up to any desired accuracy ε) used the ellipsoid method [50] but faster interior-point methods were later given by Alizadeh [5], and Nesterov and Nemirovskii [83, 84]. The running time of these interior point algorithms is $\tilde{O}(\sqrt{m}(m + n^3)L)$ where L is an input size parameter. In this chapter, the $\tilde{O}$ notation is used to suppress polylog$(\frac{mn}{\varepsilon})$ factors.

In addition to well-known approximation algorithms based on SDP such as MaxCut, Sparsest Cut, Coloring, etc., SDP has also proved useful in a host of other settings. In this chapter, we consider a few of these applications. For instance, Linial, London, and Rabinovich [75] observe that given an n-point metric space, finding its minimum-distortion embedding into ℓ_2 can be formulated as an SDP with $m = O(n^2)$ constraints, which takes $\tilde{O}(n^4)$ time to solve. Recent approximation algorithms for the *cut norm* of a matrix [8] and for certain subcases of correlation clustering [31] 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 [8] is $O(1)$ uses the famous *Grothendieck inequality* from analysis.) Halperin and Hazan [51] showed that a biological probability estimation problem, which estimates the frequencies of haplotypes from a noisy sample, can be solved using SDP with $m = O(n^2)$.

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.

The results of this chapter give such algorithms for various specific SDPs. These algorithms are of type (B) considered in Chapter 1. We use the technique of Lagrangian relaxation with the basic Multiplicative Weights algorithm to approximately solve the SDPs.

This generalizes the work of Klein and Lu [67] who use the algorithm of Section 2.3.2 based on the Multiplicative Weights algorithm to approximately solve SDPs that arose in the algorithms of Goemans-Williamson for Max Cut and Karger, Motwani, and Sudan for Coloring. 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*.

6.2 Description of results

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 (see Definition 1), 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 Multiplicative Weights algorithm is proportional to ρ^2. 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.

In this chapter, we describe how to modify the Multiplicative Weights algorithm to handle some of these high-width SDPs. Our technique is a hybrid of the Multiplicative Weights 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 Weights update idea, that combines old constraints into new, fewer constraints. Intuitively, this works because the Multiplicative Weights algorithm excels at handling many low-width constraints and exterior point methods excel at handling a few, high-width constraints. The idea is related to the observation in [85] that their packing-covering problems are solvable in polynomial time using the dual ellipsoid method.

Next, we 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 [71] to our needs (see Lemma 31). The Lanczos algorithm is quite efficient because it can exploit the sparsity of the matrix. We also provide a sparsification procedure for matrices (see Section 6.7) based on random sampling to further speed up the process.

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.

$$\begin{aligned} \forall j \in [m]: \ \mathbf{A}_j \bullet \mathbf{X} &\geq b_j \\ \mathbf{Tr}(\mathbf{X}) &\leq R \\ \mathbf{X} &\succeq 0 \end{aligned} \tag{6.2}$$

The upper bound on the trace, $\mathbf{Tr}(\mathbf{X}) = \sum_i X_{ii}$, is usually absent in the textbooks, but is natural for relaxation SDPs. For instance, in combinatorial optimization, usually we have some unit vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ associated with, say, the nodes in a graph, and $X_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. Then $\mathbf{Tr}(\mathbf{X}) = n$. In any case, $\mathbf{Tr}(\mathbf{X})$ for the optimum $\mathbf{X}$ 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. $\mathbf{A}_j \bullet \mathbf{X} \geq b_j - \varepsilon$ for $j = 1, 2, \ldots, m$, or conclude correctly that the SDP is infeasible.

We use the algorithm of Section 2.3.2, which approximately solves the feasibility problem

$$\exists?\ \mathbf{X} \in \mathcal{P} : \quad \forall j \in [m] : \ \mathbf{A}_j \bullet \mathbf{X} \ \geq \ b_j \tag{6.3}$$

where $\mathcal{P} = \{\mathbf{X} \in \mathbb{R}^{n \times n} : \ \mathbf{X} \succeq \mathbf{0}, \mathbf{Tr}(\mathbf{X}) \leq R\}$. In each round t of this algorithm, we get a probability distribution $\mathbf{p}^{(t)} = \langle {p_1}^{(t)}, {p_2}^{(t)}, \ldots, {p_m}^{(t)} \rangle$ on the constraints, and then Oracle's task is the following feasibility problem

$$\exists?\ \mathbf{X} \in \mathcal{P} : \quad \sum_{j=1}^{m} {p_j}^{(t)} (\mathbf{A}_j \bullet \mathbf{X} - b_j) \ \geq \ 0 \tag{6.4}$$

This is actually an eigenvalue problem in disguise, since, as we will momentarily show, assuming the system is feasible, then there is a feasible solution $\mathbf{X}$ that has rank 1.

Table 6.1 summarizes the improvements we obtain for various problems using this approach. The main point to stress is that in practice our algorithm may run even faster than the worst-case estimates in Table 6.1. Throughout this chapter we carefully list times in terms of number of eigenvalue/eigevector computations required, and these tend to run much faster than our worst-case estimate (which are formally given in Lemma 31 below). By contrast, each iteration of Alizadeh's SDP solver requires Cholesky decomposition, which is inherently a $\tilde{\Theta}(n^3)$ operation.

Problem	Previous best	Our algorithm	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	$\tilde{O}(n^4)$	$\tilde{O}\left(\frac{n^{2.5}}{\varepsilon^{2.5}}\right)$	$\Omega(n^{1.5})$
Embedding	$\tilde{O}(n^4)$	$\tilde{O}\left(\frac{n^3}{d_{\min}^{2.5}\varepsilon^{3.5}}\right)$	For $d_{\min} = \omega(n^{-0.4})$

Table 6.1: Running time obtained for several applications. The running times shown are for a multiplicative $1 - \varepsilon$ approximation to the MaxQP and Embedding problems, and an additive ε approximation to the HaploFreq problem.

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, our algorithm for MaxQP provides speedups when either of the following two conditions are true (a) the number of nonzero entries in the matrix $\mathbf{A}$ in the objective function is $N = o(n^2)$ or (b) the optimum of the SDP, α^*, is at least $\frac{1}{\sqrt{n}} \sum_{i,j} |A_{ij}|$. This covers a wide range of matrices, including the ones arising in the MaxCut SDP.

Likewise, in the Embedding problem, where one is seeking the minimum distortion embedding into ℓ_2, the ε is benign, say $\Omega(1)$. 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 speedup when $d_{\min}$ is at least $n^{-0.4}$. (This is still an interesting set of metrics.)

In the original paper on the topic of this chapter [14], we also considered applications to the following SDPs: undirected Sparsest Cut and Balanced Separator, Min UnCut, and Min 2CNF Deletion. We omit these applications in this chapter because the improvements obtained were for a narrow range of parameters, and the results of Chapter 4 subsume these applications anyway.

6.3 Description of the method

In this section, we give more details of the method, 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 [31] for a discussion.

$$\begin{aligned} \max\ \mathbf{A} \bullet \mathbf{X} & \\ \forall i \in [n] :\ X_{ii} &\leq 1 \\ \mathbf{X} &\succeq 0 \end{aligned} \qquad \text{(MaxQP)}$$

We assume here that $\mathrm{diag}(\mathbf{A}) \geq 0$, i.e. all diagonal entries of $\mathbf{A}$ are nonnegative. Let $N \geq n$ be the number of non-zero entries of $\mathbf{A}$. We wish to get a multiplicative $1 - O(\varepsilon)$ approximation to the optimum value of the SDP. Alizadeh's interior point method solves the SDP in $\tilde{O}(n^{3.5})$ time.

Step I: Bounding the optimum and trace. We consider the general SDP (6.1). The first step is to compute bounds on the optimum $\alpha*$ of the SDP, i.e. we find $L \leq U$ such that $\alpha^* \in [L, U]$. We implicitly assume that all our SDPs have positive optimum value. We conduct a binary search for the optimum in the range $[L, U]$. We also compute a bound on the trace of the optimum, $\mathbf{Tr}(\mathbf{X}) \leq R$, if it is not explicitly specified in the SDP.

Example: SDP (MaxQP). We assume that $\sum_{ij} |A_{ij}| = 1$, this amounts to scaling the optimum by a fixed quantity, and the multiplicative approximation guarantees translate to the unscaled problem directly.

Let $\mathbf{X}^*$ be the optimum solution. Since $\mathbf{X}^*$ is positive semidefinite, there are vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathbb{R}^n$ such that $X^*_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. Thus, the Cauchy-Schwarz inequality implies that for any i, j, $(X^*_{ij})^2 \leq X^*_{ii} X^*_{jj} \leq 1$, so $|X^*_{ij}| \leq 1$. Hence, $\alpha^* = \mathbf{A} \bullet \mathbf{X}^* = \sum_{ij} A_{ij} X^*_{ij} \leq \sum_{ij} |A_{ij}| = 1$. Conversely, the matrix $\mathbf{X}$ specified by $X_{ij} = \frac{\mathrm{sgn}(A_{ij})}{n}$ and $X_{ii} = 1$ is diagonally dominant and hence positive semidefinite, and achieves an objective value of $\frac{1}{n}$. This gives a lower bound on α. We can bound the trace of the optimum as $\mathbf{Tr}(X) \leq n$.

Step II: Reduction to feasibility problem. We "guess" the value of α^* using binary search in the range $[L, U]$ computed in Step I. Let α be our current guess. Define a convex set $\mathcal{P} = \{\mathbf{X} \in \mathbb{R}^{n \times n} :\ \mathbf{X} \succeq 0,\ \mathbf{Tr}(\mathbf{X}) \leq R\}$. We rewrite the SDP as a feasibility problem for the binary search as follows:

$$\begin{aligned} -\frac{1}{\alpha}\mathbf{C} \bullet \mathbf{X} \ &\geq \ -1 \\ \forall j \in [m]: \ \mathbf{A}_j \bullet \mathbf{X} \ &\geq \ b_j \\ \mathbf{X} \ &\in \mathcal{P} \end{aligned} \tag{6.5}$$

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 $1 + \varepsilon$.

Example: SDP (MaxQP). We set $\mathcal{P} = \{\mathbf{X} \in \mathbb{R}^{n\times n} : \ \mathbf{X} \succeq 0, \ \mathbf{Tr}(\mathbf{X}) \leq n\}$, and consider the following feasibility problem, for $\alpha \in [\frac{1}{n}, 1]$:

$$\begin{aligned} \frac{1}{\alpha}\mathbf{A} \bullet \mathbf{X} \ &\geq \ 1 \\ \forall i \in [n]: \ -X_{ii} \ &\geq \ -1 \\ \mathbf{X} \ &\in \mathcal{P} \end{aligned} \tag{6.6}$$

Step III: The Multiplicative Weights Update algorithm. Now, we turn to the general SDP feasibility problem (6.3), since we have reduced the MaxQP problem to a feasibility problem of this kind in Step II. Let T_{oracle} be the time needed to implement the Oracle. We can run the algorithm of Theorem 7, which we restate here in the context of solving SDPs:

Theorem 30. *Let $\varepsilon > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded $\frac{\varepsilon}{3}$-approximate* Oracle *for the feasibility problem (6.4). Assume that $\ell \geq \frac{\varepsilon}{3}$. Then there is an algorithm which either solves the problem (6.3) up to an additive error of ε, or correctly concludes that the system is infeasible, in time*

$$O\left(\frac{\ell\rho\log(m)}{\delta^2} \cdot T_{oracle} + m\right).$$

Example: SDP (MaxQP). We estimate width bounds on the Oracle. In (6.6), we have $\frac{1}{\alpha}\mathbf{A} \bullet \mathbf{X} - 1 \in [-\frac{n}{\alpha} - 1, \frac{n}{\alpha}]$ for all $\mathbf{X} \in \mathcal{P}$, since

$$|\mathbf{A} \bullet \mathbf{X}| \ \leq \ |\mathbf{A} \bullet n\mathbf{I}| \ = \ n|\mathbf{Tr}(\mathbf{A})| \ \leq \ n.$$

Similarly, $1 - X_{ii} \in [-n, 1]$ for all $\mathbf{X} \in \mathcal{P}$. Thus, any implementation of the Oracle which solves the feasibility problem (6.4) is $(\frac{n}{\alpha}, \frac{n}{\alpha} + 1)$-bounded. We have $m = n + 1$, $R = n$. Thus, the running time from Theorem 30 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 will how to improve the running time momentarily.

Step IV: Oracle **from eigenvector computations.** Note that the Oracle can be implemented by maximizing $\sum_{j=1}^{m} p_j(\mathbf{A}_j \bullet \mathbf{X} - b_j)$ over the set $\mathcal{P} = \{\mathbf{X} \in \mathbb{R}^{n\times n} : \ \mathbf{X} \succeq$

$\mathbf{0}, \mathbf{Tr}(\mathbf{X}) \leq R\}$, where $\mathbf{p} = \langle p_1, p_2, \ldots, p_m \rangle$ is a given probability distribution on the constraints.

We show in Lemma 30 that this amounts to approximately computing the largest eigenvector of the matrix $\mathbf{C} = \sum_{j=1}^{m} p_j(\mathbf{A}_j - \frac{b_j}{R}\mathbf{I})$ up to tolerance $\delta = \frac{\varepsilon}{3R}$. Define $T_{\text{ev}}(\mathbf{C}, \delta)$ to be the time needed for this. Thus, $T_{\text{oracle}} = O(T_{\text{ev}}(\mathbf{C}, \delta))$.

Step V: Inner and Outer SDPs. Now we indicate our width reduction technique. 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. We prove the following theorem in Section 6.4:

Theorem 31. *With the setup as in Theorem 30, there is an algorithm which produces an ε approximate solution to the general SDP (6.3) 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. On the other hand, the Multiplicative Weights algorithm has good dependence on the number of constraints but poor dependence on the width. We therefore seek to combine the two algorithms so as to be able to exploit the advantages of both. For this, we define a more constraint specific notion of width:

Definition 4 (Constraint width)**.** *A constraint of the kind $\mathbf{A} \bullet \mathbf{X} \geq b$ where $\mathbf{X} \in \mathcal{P}$, is said to be (ℓ, ρ)-bounded over $\mathcal{P}$ for some parameters $0 \leq \ell \leq \rho$, if one of the following two conditions holds for all $\mathbf{X} \in \mathcal{P}$: either*

$$\begin{aligned} \mathbf{A} \bullet \mathbf{X} - b &\in [-\ell, \rho], \text{ or} \\ \mathbf{A} \bullet \mathbf{X} - b &\in [-\rho, \ell] \end{aligned}$$

Then value ρ is called the width of the constraint.

Observe that in the SDP (MaxQP), there is a single constraint, $\frac{1}{\alpha}\mathbf{A} \bullet \mathbf{X} - 1 \geq 0$, which has high width: $O(\frac{n}{\alpha})$. The other constraints have width bounded by n. This phenomenon happens in all our applications: we 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.

To describe this idea, let $J \subseteq [m]$ be a subset of constraints, such that for all $j \in J$, the constraint $\mathbf{A}_j \geq b_j$ is (ℓ_H, ρ_H)-bounded, and for all $j \in [m] \setminus J$ the constraint $\mathbf{A}_j \geq b_j$ is (ℓ_L, ρ_L)-bounded, for some parameters $\ell_H, \rho_H, \ell_L, \rho_L$. Here, we assume that $\rho_H \gg \rho_L$, so that J is the index set of the high-width constraints. Let $m_H = |J|$, and let $m_L = m - m_H$.

Now we push the high width constraints,into the convex domain for $\mathbf{X}$, thus creating a new convex set $\mathcal{Q} = \{\mathbf{X} \in \mathcal{P},\ \forall j \in J :\ \mathbf{A}_j \bullet \mathbf{X} \geq b_j\}$, and run the Multiplicative Weights Update algorithm of Theorem 30 on the other constraints with $\mathbf{X} \in \mathcal{Q}$. We call this the *outer SDP*.

The (outer) ORACLE now needs to solve the following feasibility problem, given a probability distribution $\mathbf{p}$ the set of constraints $[m] \setminus J$:

$$\exists?\ \mathbf{X} \in \mathcal{Q} : \quad \sum_{j \in [m] \setminus j} p_j(\mathbf{A}_j \bullet \mathbf{X} - b_j) \ \geq \ 0 \tag{6.7}$$

As usual, it suffices to find an $\mathbf{X} \in \mathcal{Q}$ which satisfies the feasibility problem (6.7) up to an additive error of $-\frac{\varepsilon}{3}$. This can be achieved by approximately solving the following SDP with $m_H + 1$ constraints:

$$\begin{aligned} \sum_{j \in [m] \setminus j} p_j(\mathbf{A}_j \bullet \mathbf{X} - b_j) \ &\geq \ 0 \\ \forall j \in J : \ \mathbf{A}_j \bullet \mathbf{X} \ &\geq b_j \\ \mathbf{X} \ &\in \mathcal{P} \end{aligned}$$

We call this the *inner SDP*. The (inner) ORACLE for this SDP needs to optimize a weighted combination of *all* m constraints over $\mathcal{P}$, which is just the one we needed in Step IV.

We solve the inner SDP using the algorithm of Theorem 31. The number of constraints in the inner SDP is $m_H + 1$. Thus, the algorithm of Theorem 31 implements the outer ORACLE in

$$\tilde{O}(m_H \log(\rho_H) \cdot T_{\text{oracle}} + m_H \log(\rho_H) \mathcal{M}(m_H \log(\rho_H)))$$

time. Here, T_{oracle} is the time for implementing the inner ORACLE, which, as proved in Lemma 30, is $O(T_{\text{ev}}(\mathbf{C}, \frac{\varepsilon}{3R})$, where $\mathbf{C}$ is an arbitrary convex combination of all the constraints.

Putting Theorem 30 and Theorem 31 together in this hybrid fashion, we obtain the following theorem:

Theorem 32. *With the given setup, the hybrid algorithm which composes an outer and inner SDP produces an ε approximate solution to the general SDP (6.3) or declares correctly its infeasibility in time*

$$\tilde{O}\left(\frac{\ell_L \rho_L}{\varepsilon^2}\left[m_H \log(\rho_H) \cdot T_{oracle} + m_H \log(\rho_H) \mathcal{M}(m_H \log(\rho_H))\right] + m_L\right).$$

The time bound given in Theorem 32 is horrendous; we can simplify it considerably by making the following assumptions on m_H and ρ_H, which are satisfied in all our applications:

Corollary 6. *If $m_H = \tilde{O}(1)$, and $\rho_H = poly(mn)$, then the running time of the algorithm of Theorem 32 reduces to*

$$\tilde{O}\left(\frac{\ell_L \rho_L}{\varepsilon^2} \cdot T_{oracle} + m\right).$$

Here, $T_{oracle} = O(T_{ev}(\mathbf{C}, \frac{\varepsilon}{3R})$, where $\mathbf{C}$ is an arbitrary convex combination of all the constraints.

Example: SDP (MaxQP**).** We have only one high width constraint, $\frac{1}{\alpha}\mathbf{A} \bullet \mathbf{X} - 1 \geq 0$, and so the set J consists of only this single constraint. Thus, $m_H = 1$, $m_L = n$, $\rho_H = O(\frac{n}{\alpha}) = O(n^2)$, $\ell_L = 1$, $\rho_L = n$. Thus, by Corollary 6, the SDP can be solved in time $\tilde{O}(\frac{n}{\varepsilon^2} \cdot T_{\text{oracle}} + n)$. Factoring in the $T_{\text{oracle}} = O(T_{\text{ev}}(\mathbf{C}, \frac{\varepsilon}{3n}))$, we get the following theorem, which will be proved in section 6.6.1:

Theorem 33. *A multiplicative* $1-O(\varepsilon)$ *approximation to SDP* (MaxQP) *can be obtained in time*

$$\tilde{O}\left(\frac{n^{1.5}}{\varepsilon^{2.5}} \cdot \min\left\{N,\ \frac{n^{1.5}}{\varepsilon\alpha^*}\right\}\right).$$

Here, N is the number of non-zero entries in the matrix $\mathbf{A}$ in the objective function.

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 $\mathbf{A}$ 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 $\mathbf{A}$ is the combinatorial Laplacian of the input graph, divided by $4m$ (to make the $\sum_{ij} |A_{ij}| = 1$), where m is the total weight of all edges in the graph. Since a random cut in the graph has expected value at least $\frac{m}{2}$, we get that $\alpha^* \geq \frac{m/2}{4m} = \frac{1}{8}$. 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 [67], 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 [67] does not extend to this general problem.

6.4 Proofs of the Main Theorems

In this section we prove Theorem 31. For convenience of notation, we define the linear functions $f_j(\mathbf{X}) = \mathbf{A}_j \bullet \mathbf{X} - b_j$ for $j \in [m]$. The problem is to check the feasibility of the system of inequalities

$$\exists ?\ \mathbf{X} \in \mathcal{P} : \quad \forall j \in [m] :\ f_j(\mathbf{X})\ \geq\ 0 \tag{6.8}$$

We assume that there is an Oracle which does the following task: given a distribution on the constraints $\mathbf{p} = \langle p_1, p_2, \ldots, p_m \rangle$, solves the feasibility problem

$$\exists ?\ \mathbf{X} \in \mathcal{P} : \quad \sum_{j=1}^{m} p_j f_j(\mathbf{X})\ \geq\ 0 \tag{6.9}$$

approximately, i.e. it either finds an $\mathbf{X} \in \mathcal{P}$ which makes the weighted combination $\sum_j p_j f_j(\mathbf{X}) \geq -\frac{\varepsilon}{3}$ or declares correctly that (6.9) is infeasible.

In the algorithm we describe, we do not use any special properties of the convex domain $\mathcal{P}$. So this algorithm applies to any general linear feasibility problem with an arbitrary convex domain $\mathcal{P}$, in the same setting as in Section 2.3.2, as long as we have access to an Oracle of the kind described above. Let T_{oracle} be the time needed for the Oracle. We restate Theorem 31 in this setting:

Theorem 31. *Let $\varepsilon > 0$ be a given error parameter. Suppose there exists an (ℓ, ρ)-bounded, $\frac{\varepsilon}{3}$-approximate* Oracle *for the feasibility problem (6.9). Assume that $\ell \geq \frac{\varepsilon}{3}$. Then there is an algorithm which either solves the feasibility problem (6.8) up to an additive error of ε, or correctly concludes that the system is infeasible, in time*

$$\tilde{O}(m \log(\rho) \cdot T_{oracle} + m \log(\rho)\mathcal{M}(m \log(\rho))).$$

Proof: To derive the algorithm, we will need the following version of Farkas' lemma:

Lemma 27 (Farkas)**.** *Given a matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$ and a vector $\mathbf{c} \in \mathbb{R}^m$, one and only one of the following systems has a solution:*

1. $\mathbf{A}\mathbf{p} \leq 0$, $\mathbf{p} \geq 0$ *and* $\mathbf{c} \cdot \mathbf{p} > 0$ *for some* $\mathbf{p} \in \mathbb{R}^m$*;*
2. $\mathbf{A}^\top \mathbf{q} \geq \mathbf{c}$ *and* $\mathbf{q} \geq 0$ *for some* $\mathbf{q} \in \mathbb{R}^n$*.*

Farkas' Lemma implies the following lemma:

Lemma 28. *Consider $\mathbf{X}_1, ..., \mathbf{X}_n \in \mathcal{P}$. Then exactly one of the following holds:*

1. *There exists a distribution $\mathbf{p} = \langle p_1, p_2, \ldots, p_m \rangle^\top$ such that for all $i \in [n]$, we have $\sum_j p_j f_j(\mathbf{X}_i) \leq -\varepsilon$.*
2. *There exist a distribution $\mathbf{q} = \langle q_1, q_2, ..., q_n \rangle^\top$ such that for $\mathbf{Y} = \sum_i q_i \mathbf{X}_i$ we have that for all $j \in [m]$, $\mathbf{Y}$ satisfies $f_j(\mathbf{Y}) \geq -\varepsilon$.*

Proof: In Lemma 27, choose the matrix $\mathbf{A}$ to be $A_{ij} = f_j(\mathbf{X}_i) + \varepsilon$, and the vector $\mathbf{c}$ as $c_j = 1$ for all $j \in [m]$. Since $\mathbf{c} > 0$, we conclude that the vectors $\mathbf{p}$ and $\mathbf{q}$ in Lemma 27 are non-zero. By scaling we may assume that $\mathbf{p}$ is a distribution as required in this lemma. Then the first case exactly corresponds to the first case of Lemma 27. The second case of Lemma 27 translates to the following: there is a vector $\tilde{\mathbf{q}} = \langle \tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_n \rangle^\top$ such that $\tilde{\mathbf{q}} \geq 0$ and $\forall j \in [m]$, we have $\sum_i \tilde{q}_i (f_j(\mathbf{X}_i) + \varepsilon) \geq 1$. Set $q_i = \tilde{q}_i / \sum_k \tilde{q}_k$ so that $\mathbf{q} = \langle q_1, q_2, \ldots, q_n \rangle^\top$ is a distribution. Then for all $j \in [m]$, we have $\sum_i q_i f_j(\mathbf{X}_i) \geq \frac{1}{\sum_i \tilde{q}_i} - \varepsilon \geq -\varepsilon$ which implies that $f_j(\sum_i q_i \mathbf{X}_i) \geq -\varepsilon$, since f_j is a linear function. Setting $\mathbf{Y} = \sum_i q_i \mathbf{X}_i$ concludes the proof. □

Lemma 29. *Let $\mathbf{X}_1, \ldots, \mathbf{X}_n \in \mathcal{P}$. Suppose the first case of Lemma 28 holds, i.e. there is a distribution $\mathbf{p}^* = \langle p_1^*, p_2^*, \ldots, p_m^* \rangle^\top$ such that for all $i \in [n]$, we have $\sum_j p_j^* f_j(\mathbf{X}_i) \leq -\varepsilon$. Then the polytope $\mathcal{Q}$ of vectors $\langle p_1, p_2, \ldots, p_m \rangle^\top \in \mathbb{R}^m$ defined by the linear inequalities*

$$\begin{aligned} \forall i \in [n]: \quad \sum_j f_j(\mathbf{X}_i) p_j &\leq -\frac{3\varepsilon}{4} \\ \sum_j p_j &\leq 1 + \frac{\varepsilon}{4\rho} \\ \sum_j p_j &\geq 1 - \frac{\varepsilon}{4\rho} \\ \forall j \in [m]: \quad p_j &\geq -\frac{\varepsilon}{2m\rho} \end{aligned} \tag{6.10}$$

has volume at least $(\frac{\varepsilon}{2m\rho})^m$. Also, it is contained in an ℓ_∞ box of volume 2^m.

Proof: We show that the ℓ_∞ box around $\mathbf{p}^*$ defined by $\mathcal{B} = \{\mathbf{p} \in \mathbb{R}^m : \|\mathbf{p}-\mathbf{p}^*\|_\infty \leq \frac{\varepsilon}{4m\rho}\}$ is contained in $\mathcal{Q}$. This box has volume $(\frac{\varepsilon}{2m\rho})^m$. This is true because for any $\mathbf{p} \in \mathcal{B}$, we have:

$$\begin{aligned}
\forall i \in [n]: \sum_j f_j(\mathbf{X}_i)p_j \ &\leq\ \sum_j \left[f_j(\mathbf{X}_i)p_j^* + |f_j(\mathbf{X}_i)| \cdot \frac{\varepsilon}{4m\rho}\right] \leq -\varepsilon + m\rho \cdot \frac{\varepsilon}{4m\rho} = -\frac{3\varepsilon}{4} \\
\sum_j p_j \ &\leq\ \sum_j p_j^* + m \cdot \frac{\varepsilon}{4m\rho} \ =\ 1 + \frac{\varepsilon}{4\rho} \\
\sum_j p_j \ &\geq\ \sum_j p_j^* - m \cdot \frac{\varepsilon}{4m\rho} \ =\ 1 - \frac{\varepsilon}{4\rho} \\
\forall j \in [m]: \ p_j \ &\geq\ p_j^* - \frac{\varepsilon}{4m\rho} \ \geq\ -\frac{\varepsilon}{4m\rho}
\end{aligned}$$

Here, the first inequality uses the fact that $|f_j(\mathbf{X}_i)| \leq \rho$.

Next, note that for all $j \in [m]$, we have $-\frac{\varepsilon}{4m\rho} \leq p_j \leq 1 + \frac{\varepsilon}{4\rho}$. So $\mathcal{Q}$ is contained in the ℓ_∞ box

$$\left\{\mathbf{p} \in \mathbb{R}^m : \ \forall j \in [m]: \ -\frac{\varepsilon}{4m\rho} \leq p_j \leq 1 + \frac{\varepsilon}{4\rho}\right\},$$

which has volume $((1 + \frac{\varepsilon}{4\rho}) + \frac{\varepsilon}{4m\rho})^m \leq 2^m$. □

At this point, we are ready to present the algorithm. We run Vaidya's algorithm [93] 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 of Vaidya's algorithm, a point $\mathbf{X} \in \mathcal{P}$ may be generated by running Oracle on a distribution in $\mathbb{R}^{n\times n}$. Let $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ be all the points generated in this way over all iterations. Then the polytope is $\mathcal{Q}$ defined by the linear inequalities (6.10) in Lemma 29.

The separation oracle works as follows: given a point $\mathbf{p} = \langle p_1, \ldots, p_m\rangle^\top$, it first checks whether $\mathbf{p}$ satisfies the last three constraints, and returns one of them as a separating hyperplane if it is violated. So assume that the last three constraints are satisfied by $\mathbf{p}$. Now, it calls Oracle on the distribution $\tilde{\mathbf{p}}$ which is obtained as follows. Let $\mathbf{q} = \langle q_1, \ldots, q_m\rangle^\top$ be defined by $q_j = p_j + \frac{\varepsilon}{4m\rho}$. Note that $q_j \geq 0$ and

$$\sum_j q_j \ =\ \sum_j p_j + \frac{\varepsilon}{4m\rho} \ \leq\ 1 + \frac{\varepsilon}{4\rho} + \frac{\varepsilon}{4m\rho} \cdot m \ =\ 1 + \frac{\varepsilon}{2\rho} \ <\ \frac{3}{2}, \tag{6.11}$$

if we assume that $\varepsilon < \rho$. Also,

$$\sum_j q_j \ \geq\ \sum_j p_j \ \geq\ 1 - \frac{\varepsilon}{4\rho} \ >\ 0.$$

Now, we define the distribution $\tilde{\mathbf{p}} = \langle \tilde{p}_1, \ldots, \tilde{p}_m\rangle^\top$ as $\tilde{p}_j = q_j/Z$ where $Z = \sum_{j'=1}^m q_{j'}$. If Oracle declares that there for all $\mathbf{X} \in \mathcal{P}$, $\sum_j \tilde{p}_j f_j(\mathbf{X}) < 0$, then the algorithm immediately aborts and declares infeasibility of the system. Otherwise, Oracle returns a point

$\mathbf{X} \in \mathcal{P}$ such that $\sum_j \tilde{p}_j f_j(\mathbf{X}) \geq -\frac{\varepsilon}{3}$. Thus, we have

$$\sum_j \left(p_j + \frac{\varepsilon}{4m\rho}\right) \cdot f_j(\mathbf{X}) \;\geq\; -\frac{\varepsilon}{3} \cdot Z \;>\; -\frac{\varepsilon}{2},$$

since $Z < \frac{3}{2}$ by (6.11). Also,

$$\sum_j p_j f_j(\mathbf{X}) + \frac{\varepsilon}{4} \;\geq\; \sum_j \left(p_j + \frac{\varepsilon}{4m\rho}\right) \cdot f_j(\mathbf{X}),$$

since $|f_j(\mathbf{X})| \leq \rho$. From the above two inequalities, we have $\sum_j p_j f_j(\mathbf{X}) > -\frac{3\varepsilon}{4}$. Then the constraint $\sum_j p_j f_j(\mathbf{X}) \leq -\frac{3\varepsilon}{4}$ serves as a separating hyperplane for Vaidya's algorithm and $\mathbf{X}$ becomes one of the $\mathbf{X}_i$'s mentioned above.

Vaidya's algorithm needs $n = O\left(\log\left(\frac{2^m}{(\frac{\varepsilon}{2m\rho})^m}\right)\right) = \tilde{O}(m\log(\rho))$ iterations to decide emptiness of the polytope $\mathcal{Q}$. In this case, the first case of Lemma 28 doesn't hold, so the second must. Since the existence of the distribution $\mathbf{q} = \langle q_1, q_2, \ldots, q_n\rangle^\top$ has been established, it can be found by solving the linear program:

$$\begin{aligned} \forall j \in [m]: \quad \sum_i f_j(\mathbf{X}_i) q_i \;&\geq\; -\varepsilon \\ \sum_i q_i \;&=\; 1 \\ \forall i \in [n]: \quad q_i \;&\geq\; 0 \end{aligned} \tag{6.12}$$

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 [102]. 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 $\tilde{O}(m\log(\rho) \cdot T_{\text{oracle}} + m\log(\rho)\mathcal{M}(m\log(\rho))))$. $\Box$

6.5 Implementing Oracle using Approximate Eigenvector Computations

In this section, we present lemmas which describe how to efficiently implement the Oracle. Recall that the Oracle needs to solve the feasibility problem (6.4), which we restate here: given a probability distribution $\mathbf{p} = \langle p_1, p_2, \ldots, p_m\rangle$ on the constraints, consider the following feasibility problem:

$$\exists?\ \mathbf{X} \in \mathcal{P}: \quad \sum_{j=1}^{m} p_j(\mathbf{A}_j \bullet \mathbf{X} - b_j) \;\geq\; 0$$

The Oracle needs to find an $\mathbf{X} \in \mathcal{P}$ which satisfies this up to an additive error of $\frac{\varepsilon}{3}$, or declare correctly that no $\mathbf{X}$ can satisfy this feasibility problem. We now show how to implement the Oracle using an approximate eigenvector finding procedure:

Lemma 30. *Suppose we have a procedure, that given a matrix* $\mathbf{C} \in \mathbb{R}^{n\times n}$ *and a tolerance* $\delta > 0$*, computes a unit vector* $\mathbf{x}$ *which satisfies* $\mathbf{x}^\top\mathbf{C}\mathbf{x} \geq -\delta$*, in time* $T_{ev}(\mathbf{C}, \delta)$*, or declares correctly that* $\mathbf{C}$ *is negative definite. Then, given a distribution* $\mathbf{p} = \langle p_1, p_2, \ldots, p_m\rangle^\top$*, we can implement* ORACLE *by applying this procedure once with* $\mathbf{C} = \sum_{j=1}^m p_j(\mathbf{A}_j - \frac{b_j}{R}\mathbf{I})$ *and* $\delta = \frac{\varepsilon}{3R}$.

PROOF: Suppose $\sum_{j=1}^m -p_j \cdot b_j \geq 0$. In this case, the ORACLE can simply return $\mathbf{X} = 0$. So assume that $\sum_{j=1}^m -p_j \cdot b_j < 0$, or equivalently, $\sum_{j=1}^m p_j \cdot b_j > 0$.

Then the ORACLE constructs the matrix $\mathbf{C} = \sum_{j=1}^m p_j(\mathbf{A}_j - \frac{b_j}{R}\mathbf{I})$ and applies the procedure of the lemma on $\mathbf{C}$ with $\delta = \frac{\varepsilon}{3R}$. Suppose the procedure yields a unit vector $\mathbf{x}$ such that $\mathbf{x}^\top\mathbf{C}\mathbf{x} \geq -\delta$, then the ORACLE returns the matrix $\tilde{\mathbf{X}} = R\mathbf{x}\mathbf{x}^\top$. Note that $\mathbf{Tr}(\tilde{\mathbf{X}}) = R$. For this matrix, we have

$$\begin{aligned}\sum_{j=1}^m p_j(\mathbf{A}_j \bullet \tilde{\mathbf{X}} - b_j) \;&=\; \sum_{j=1}^m p_j(\mathbf{A}_j \bullet \tilde{\mathbf{X}} - \frac{b_j}{R}\mathbf{I} \bullet \tilde{\mathbf{X}}) \\ =\; \mathbf{C} \bullet \tilde{\mathbf{X}} \;=\; \mathbf{C} \bullet R\mathbf{x}\mathbf{x}^\top \;&=\; R\mathbf{x}^\top\mathbf{C}\mathbf{x} \;\geq\; R \cdot -\frac{\varepsilon}{3R} \;=\; -\frac{\varepsilon}{3}\end{aligned}$$

as required.

Otherwise, the procedure declares correctly that $\mathbf{C}$ is negative definite. In this case, the ORACLE declares that no $\mathbf{X} \in \mathcal{P}$ satisfies the feasibility problem (6.4). We show now that this is correct. Suppose that in fact there were an $\mathbf{X} \in \mathcal{P}$ which satisfies $\sum_{j=1}^m p_j(\mathbf{A}_j \bullet \mathbf{X} - b_j) \geq 0$. Because $\mathbf{X} \in \mathcal{P}$, $\mathbf{Tr}(\mathbf{X}) \leq R$ and $\mathbf{X} \succeq 0$. Since $\mathbf{C} \prec 0$, we have $\mathbf{C} \bullet \mathbf{X} < 0$ (note that $\mathbf{X} \neq \mathbf{0}$, since $\sum_{j=1}^m p_j \cdot b_j > 0$). Then we have

$$0 \;>\; \mathbf{C} \bullet \mathbf{X} \;=\; \sum_{j=1}^m p_j \left(\mathbf{A}_j \bullet \mathbf{X} - \frac{b_j}{R}\mathbf{I} \bullet \mathbf{X}\right) \;\geq\; \sum_{j=1}^m p_j(\mathbf{A}_j \bullet \mathbf{X} - b_j) \;\geq\; 0.$$

The second inequality follows because $\mathbf{I} \bullet \mathbf{X} = \mathbf{Tr}(\mathbf{X}) \leq R$. Thus, we have a contradiction. □

By Lemma 30, the ORACLE needed for our algorithms can be implemented by computing the eigenvector belonging to 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 31. *Let* $\mathbf{C} \in \mathbb{R}^{n\times n}$ *be a matrix with* $N \geq n$ *non-zero entries. Let* $\lambda_1 := \lambda_1(\mathbf{C})$ *and* $\lambda_n := \lambda_n(\mathbf{C})$*. Let* $\delta > 0$ *be a given error parameter, and let* Λ *be an upper bound on* $|\lambda_n|$*. Let* $\gamma = \frac{\lambda_1+\delta}{\lambda_1+\Lambda}$*. If* $\lambda_1 \geq -\frac{\delta}{2}$*, then with high probability, the Lanczos algorithm with a random start applied to the matrix* $\mathbf{C} + \Lambda\mathbf{I}$ *yields a unit vector* $\mathbf{x}$ *which satisfies* $\mathbf{x}^\top\mathbf{C}\mathbf{x} \geq -\delta$ *in time* $\tilde{O}(\frac{N}{\sqrt{\gamma}})$*. Thus,* $T_{ev}(\mathbf{C}, \delta) = \tilde{O}(\frac{N}{\sqrt{\gamma}})$.

PROOF: We need Theorem 3.2(a) of Kuczyński and Woźniakowski [71]:

Theorem 34 (KW). *Let $\mathbf{M} \in \mathbb{R}^{n\times n}$ be a positive semidefinite matrix. Then with high probability, the Lanczos algorithm produces in $O(\frac{\log(n)}{\sqrt{\gamma}})$ iterations a unit vector $\mathbf{x}$ such that $\frac{\mathbf{x}^\top \mathbf{M}\mathbf{x}}{\lambda_1(\mathbf{M})} \geq 1-\gamma$.*

Now let $\mathbf{M} = \mathbf{C} + \Lambda\mathbf{I}$. Notice that $\mathbf{M}$ is positive semidefinite, and $\lambda_1(\mathbf{M}) = \lambda_1 + \Lambda$. We set $\gamma = \frac{\lambda_1+\delta}{\lambda_1+\Lambda}$. Note that if $\lambda_1 > \delta$, then $\gamma > 0$. We apply Theorem 34 with the given value of γ to conclude that with high probability, in $\tilde{O}(\frac{\log(n)}{\sqrt{\gamma}})$ iterations a unit vector $\mathbf{x}$ such that:

$$\frac{\mathbf{x}^\top \mathbf{M}\mathbf{x}}{\lambda(\mathbf{M})} = \frac{\mathbf{x}^\top(\mathbf{C}+\Lambda\mathbf{I})\mathbf{x}}{\lambda_1+\Lambda} = \frac{\mathbf{x}^\top\mathbf{C}\mathbf{x}+\Lambda}{\lambda_1+\Lambda} \geq 1-\gamma$$

Simplifying, we get $\mathbf{x}^\top\mathbf{C}\mathbf{x} \geq (1-\gamma)\lambda_1 - \gamma\Lambda = \delta$, so we return $\mathbf{x}$.

The most expensive operation in each iteration of the Lanczos algorithm is a product of some vector with the matrix $\mathbf{M}$, and this can implemented in time $O(N)$. Thus, the overall running time becomes $\tilde{O}(\frac{N}{\sqrt{\gamma}})$. $\square$

Remark: The parameters γ in Lemma 31 is not known *a priori*, but in applications we will derive suitable bounds on them. One convenient lower bound on γ can be obtained as follows.

Lemma 32. *Assume $\lambda_1 \geq -\frac{\delta}{2}$, and $\Lambda \geq \delta$. Then $\gamma \geq \frac{\delta}{2\Lambda}$.*

Proof: If $\lambda_1 > 0$, then

$$\frac{\lambda_1+\delta}{\lambda_1+\Lambda} \geq \frac{\delta}{\Lambda} \geq \frac{\delta}{2\Lambda},$$

because $\delta \leq \Lambda$. If $0 \geq \lambda_1 \geq -\frac{\delta}{2}$, then

$$\frac{\lambda_1+\delta}{\lambda_1+\Lambda} \geq \frac{-\frac{\delta}{2}+\delta}{\Lambda} = \frac{\delta}{2\Lambda}.$$

$\square$

Lemma 31 shows that the running time of the Oracle depends on the sparsity of $\mathbf{C}$, i.e. the number on non-zero entries in it. In Section 6.7 we provide a randomized sparsification procedure:

Lemma 33. *Let $\mathbf{C} \in \mathbb{R}^{n\times n}$ 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 $\mathbf{C}'$ such that $\mathbf{C}'$ has $O(\frac{\sqrt{n}S}{\delta})$ non-zero entries and $\|\mathbf{C}-\mathbf{C}'\|_2 \leq \delta$.*

Suppose we obtain $\mathbf{C}'$ by sparsifying $\mathbf{C}$ using Lemma 33 with error parameter $\frac{\delta}{4}$, and then we use the Lanczos algorithm of Lemma 31 with error parameter $\frac{\delta}{2}$. Then for any unit vector $\mathbf{x}$, we have $|\mathbf{x}^\top\mathbf{C}'\mathbf{x} - \mathbf{x}^\top\mathbf{C}\mathbf{x}| \leq \|\mathbf{C}'-\mathbf{C}\| \leq \delta$. So if the Lanczos procedure yields a unit vector $\mathbf{x}$ such that $\mathbf{x}^\top\mathbf{C}'\mathbf{x} \geq -\frac{\delta}{2}$, then $\mathbf{x}^\top\mathbf{C}\mathbf{x} \geq -\frac{3\delta}{4}$. Furthermore, if $\lambda_1(\mathbf{C}') < -\frac{\delta}{4}$, then $\lambda_1(\mathbf{C}) < 0$, and hence $\mathbf{C} \prec \mathbf{0}$. The upshot is that we can use $\mathbf{C}'$ in place of $\mathbf{C}$ 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 $\frac{\sqrt{n}S}{\delta}$.

6.6 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.

6.6.1 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 33.

Theorem 33. *A multiplicative* $1-O(\varepsilon)$ *approximation to SDP* (MaxQP) *can be obtained in time*

$$\tilde{O}\left(\frac{n^{1.5}}{\varepsilon^{2.5}} \cdot \min\left\{N,\ \frac{n^{1.5}}{\varepsilon\alpha^*}\right\}\right).$$

Here, N is the number of non-zero entries in the matrix $\mathbf{A}$ in the objective function.

Proof: We apply Corollary 6. 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 $\mathbf{X} \in \mathcal{Q}$. 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 $p_0, p_1, \ldots, p_n$ which sum to 1, the matrix $\mathbf{C}$ from Lemma 30 in this case is $p_0(\frac{1}{\alpha}\mathbf{A} - \frac{1}{n}\mathbf{I}) + \sum_{i=1}^n p_i(\frac{1}{n}\mathbf{I} - \mathbf{e}_i\mathbf{e}_i^\top)$, where $\mathbf{e}_i$ is the i^{th} standard basis vector, and $\delta = \frac{\varepsilon}{6n}$ or $\frac{\varepsilon}{3n}$, depending on whether we use the sparsification procedure or not.

To apply Lemma 31, we need to bound the most negative eigenvalue, λ_n, of $\mathbf{C}$. Observe that $\mathbf{Tr}(\mathbf{C}) = p_0(\frac{1}{\alpha}\mathbf{Tr}(\mathbf{A}) - 1) \geq -1$. Since the trace equals the sum of the eigenvalues, we conclude that $(n-1)\lambda_1 + \lambda_n \geq -1$. If $\lambda_1 \geq 0$ then we conclude that $|\lambda_n| \leq (n-1)\lambda_1 + 1$, which implies that

$$\gamma \;=\; \frac{\lambda_1 + \delta}{\lambda_1 + |\lambda_n|} \;\geq\; \frac{\lambda_1 + \delta}{n\lambda_1 + 1} \;\geq\; \delta,$$

since $\delta < \frac{1}{n}$. If $\lambda_1 < 0$, then $|\lambda_n| \leq 1$, and hence

$$\gamma \;=\; \frac{\lambda_1 + \delta}{\lambda_1 + |\lambda_n|} \;\geq\; \frac{\lambda_1 + \delta}{\lambda_1 + 1} \;\geq\; \delta.$$

Thus, in either case $\gamma \geq \delta \geq \Omega(\frac{\varepsilon}{n})$, and by Lemma 31, the eigenvector procedure takes $\tilde{O}(N\frac{\sqrt{n}}{\sqrt{\varepsilon}})$ time.

If we apply the sparsification procedure of Lemma 33, 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

$$\tilde{O}\left(\frac{n^{1.5}}{\varepsilon^{2.5}} \cdot \min\left\{N,\ \frac{n^{1.5}}{\varepsilon\alpha^*}\right\}\right).$$

In each step of the binary search when $\alpha \le \alpha^*$, SDP (6.6) is feasible, we get a solution $\mathbf{X} \succeq \mathbf{0}$ which satisfies the SDP up to an additive error of ε, i.e. for all $i \in [n]$, $X_{ii} \le 1+\varepsilon$, and $\mathbf{A} \bullet \mathbf{X} \ge (1-\varepsilon)\alpha$. Now consider the solution $\tilde{\mathbf{X}} = \frac{1}{1+\varepsilon}\mathbf{X}$. Then for all $i \in [n]$, $\tilde{X}_{ii} \le 1$, and $\mathbf{A} \bullet \tilde{\mathbf{X}} \ge \frac{1-\varepsilon}{1+\varepsilon}\alpha \ge (1-2\varepsilon)\alpha$. So $\tilde{\mathbf{X}}$ is a feasible solution with objective value at least $(1-2\varepsilon)\alpha$. When $\alpha > \frac{1}{1-2\varepsilon}\alpha^*$, then no feasible solution with objective value at least $(1-2\varepsilon)\alpha$ exists, so the Oracle declares the system infeasible in some iteration, and we stop the binary search. In the binary search, we increase α by a factor of $1+\varepsilon$ every time. Thus, we get a $1-3\varepsilon$ approximation to the α^* when we stop. □

6.6.2 SDP relaxations of biological probability estimation problems

The following SDP arises in the context of the biologically-motivated problem of estimating haplotype frequencies. See [51] for a more detailed description of the problem.

$$\begin{aligned} \max\ \mathbf{A} \bullet \mathbf{X} & \\ \sum_{ij} X_{ij} &= 1 \\ \forall i,j \in [n]:\ X_{ij} &\ge 0 \\ \mathbf{X} &\succeq 0 \end{aligned} \qquad (\textsc{HaploFreq})$$

where $\mathbf{A}$ 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 $\sum_{ij} \mathbf{A}_{ij} p_i p_j$. 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 $\max_{ij}\{A_{ij}\} \cdot [\frac{1}{2}, 1]$. The upper bound is trivial since the objective is a convex combination (with the weights X_{ij}) of the A_{ij} values. Let $A_{k\ell}$ be the maximal A_{ij}. Then the lower bound is obtained by taking the vector $\mathbf{u} = \frac{1}{2}(\mathbf{e}_\ell + \mathbf{e}_k)$, where $\mathbf{e}_i$ is the i^{th} standard basis vector, and letting $\mathbf{X}$ be the positive semidefinite matrix $\mathbf{u}\mathbf{u}^\top$. We have $\mathbf{A} \bullet \mathbf{X} = \frac{1}{4}[A_{kk} + 2A_{k\ell} + A_{\ell\ell}]$. Since all A_{ij} are non-negative, this solution has value at least $\frac{1}{2}A_{kl}$.

The trace of $\mathbf{X}$ is trivially bounded by 1 from the first constraint. Note also that without loss of generality we can relax the first constraint to be $\sum_{ij} X_{ij} \le 1$. This is because in the optimum solution $\mathbf{X}$ the sum has to equal 1: all the quantities are non-negative, and so scaling up the matrix $\mathbf{X}$ to make $\sum_{ij} X_{ij} = 1$ doesn't decrease the objective value.

Theorem 35. *SDP* (HaploFreq) *can be approximated up to an additive error of $O(\varepsilon)$ in $\tilde{O}(\frac{n^{2.5}}{\varepsilon^{2.5}})$ time.*

Proof: According to step II, we "guess" α using binary search and reduce the SDP to the following problem. Here, $\mathcal{P}$ is the convex set $\{\mathbf{X} \in \mathbb{R}^{n\times n} : \ \mathbf{X} \succeq \mathbf{0}, \ \mathbf{Tr}(\mathbf{X}) \leq 1\}$.

$$\begin{aligned} \frac{1}{\alpha}\mathbf{A} \bullet \mathbf{X} - 1 &\geq 0 \\ 1 - \sum_{ij} X_{ij} &\geq 0 \\ \forall i,j \in [n] : \ X_{ij} &\geq 0 \\ \mathbf{X} &\in \mathcal{P} \end{aligned}$$

We now estimate the width of each constraint. We have $\max_{\mathbf{X}\in\mathcal{P}}(\mathbf{A} \bullet \mathbf{X}) = \|\mathbf{A}\| \leq n\max_{ij}\{A_{ij}\}$. Thus, the width of the first constraint is $\frac{1}{\alpha}\|\mathbf{A}\| + 1 = n + 1$. Similarly, the width of the second constraint is $\|\mathbf{J}\| + 1 = n + 1$, where $\mathbf{J}$ is the all 1's matrix. These two constraints have high width and we will put into the inner SDP.

For any i, j, the width of the constraint $X_{ij} \geq 0$ is 1: this is because $\mathbf{X} \succeq \mathbf{0}$ implies (by the Cauchy-Schwarz inequality) that $|X_{ij}| \leq \sqrt{X_{ii}X_{jj}} \leq 1$ since $\mathbf{Tr}(\mathbf{X}) \leq 1$. We will put these constraints in the outer SDP. Thus, $\ell_L = \rho_L = 1$, and $\delta = \frac{\varepsilon}{3}$ (sparsification is not needed here). Let $\mathbf{C}$ represent the weighted combination of the constraints for the Oracle. According to Corollary 6, the SDP can be ε approximately in $\tilde{O}(\frac{1}{\varepsilon^2} \cdot [T_{ev}(\mathbf{C}, \frac{\varepsilon}{3}) + n^2])$ time.

It remains to estimate $T_{ev}(\mathbf{C}, \frac{\varepsilon}{2})$. The matrix $\mathbf{C}$ is of the form $p_0(\frac{1}{\alpha}\mathbf{A} - \frac{1}{n}\mathbf{I}) + p_1(\frac{1}{n}\mathbf{I} - \mathbf{J}) + \sum_{ij} p_{ij}\mathbf{E}_{ij}$, where $\mathbf{J}$ is the all 1's matrix, and p_0, p_1, p_{ij} for $1 \leq i, j \leq n$ are non-negative weights summing to 1.

To bound the most negative eigenvalue, λ_n, of $\mathbf{C}$, we use the Gershgorin circle theorem. This implies that $|\lambda_n| \leq \max_i\{\sum_j |\mathbf{C}_{ij}|\}$. For the matrix $\mathbf{C}$, the dominant contributors to this maximum are the matrices $\frac{1}{\alpha}\mathbf{A}$ and $\mathbf{J}$, the other matrices put together contribute at most $O(1)$ since $p_0 + p_1 + \sum_{ij} p_{ij} = 1$. For any i, we have $\sum_j \frac{1}{\alpha}|A_{ij}| \leq 2n$ since $\alpha \geq \frac{1}{2}\max_{ij} A_{ij}$. Also, for any i, $\sum_j |J_{ij}| = n$. Thus, the bound on $|\lambda_n|$ is $O(n)$.

Thus, by Lemma 32, $\gamma \geq \Omega(\frac{\varepsilon}{n})$, and hence by Lemma 33, $T_{\text{ev}}(\mathbf{C}, \frac{\varepsilon}{2}) = \tilde{O}(\frac{n^{2.5}}{\sqrt{\varepsilon}})$ because $\mathbf{C}$ is a dense matrix. Since $\sum_{ij} |C_{ij}|$ can be as large as $\Omega(n^2)$, sparsification does not help here.

Overall, the running time from Corollary 6 is $\tilde{O}(\frac{n^{2.5}}{\varepsilon^{2.5}})$. $\Box$

For comparison, the best known interior point algorithm solves this SDP in $\tilde{O}(n^4)$ time.

6.6.3 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 $d_{ij} = D_{ij}^2$.

$$\begin{aligned} &\min \alpha \\ \forall i,j \in [n], i < j : \quad d_{ij} \ \leq \ &X_{ii} - 2X_{ij} + X_{jj} \leq \ \alpha \cdot d_{ij} \\ &\mathbf{X} \ \succeq \ \mathbf{0} \end{aligned} \qquad \text{(Embedding)}$$

Here, since $\mathbf{X} \succeq \mathbf{0}$, there are vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathbb{R}^n$ such that for all $i, j \in [n]$, we have $X_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$, and thus $X_{ii} - 2X_{ij} + X_{jj} = \|\mathbf{v}_i - \mathbf{v}_j\|^2$. Thus, if α^* is the optimum of this program, then the minimum distortion for embedding the metric into ℓ_2 is $\sqrt{\alpha^*}$. By Bourgain's theorem [26], this 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 $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ which satisfies $\sum_i \|\mathbf{v}_i\|^2 \le \alpha^* n$: we may assume that the optimal solution satisfies $\sum_i \mathbf{v}_i = 0$, otherwise we can shift the origin to the sum of the vectors; this does not change the pairwise distances $\|\mathbf{v}_i - \mathbf{v}_j\|^2$. Thus we have $\alpha^* \sum_{ij} d_{ij} \ge \sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 = n \sum_i \|\mathbf{v}_i\|^2$, which implies the claim. Since $\alpha^* = O(\log^2 n)$, we may use the trace bound $\mathbf{Tr}(\mathbf{X}) \le O(\log^2(n) \cdot n)$ in the SDP. Let the minimum squared internode distance, $\min_{ij}\{d_{ij}\}$, be denoted $d_{\min}$.

Theorem 36. *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: We guess α using binary search in the range $[1, O(\log^2 n)]$, and reduce the problem to the following feasibility SDP. Here, $\mathcal{P}$ is the convex set $\mathcal{P} = \{\mathbf{X} \in \mathbb{R}^{n\times n} : \mathbf{X} \succeq \mathbf{0}, \mathbf{Tr}(\mathbf{X}) \le \alpha n\}$.

$$\begin{aligned}
\forall i, j \in [n], i < j: \quad \frac{1}{d_{ij}}(X_{ii} - 2X_{ij} + X_{jj}) - 1 \;&\ge\; 0 \\
\forall i, j \in [n], i < j: \quad 1 - \frac{1}{\alpha d_{ij}}(X_{ii} - 2X_{ij} + X_{jj}) \;&\ge\; 0 \\
\mathbf{X} \;&\in\; \mathcal{P} \qquad (6.13)
\end{aligned}$$

We now estimate the width of each constraint. First, we note that since $\mathbf{X} \succeq 0$, let $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^n$ be vectors such that for all $i, j \in [n]$, we have $X_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. Thus, $X_{ii} - 2X_{ij} + X_{jj} = \|\mathbf{v}_i - \mathbf{v}_j\|^2$. So we have

$$0 \;\le\; X_{ii} - 2X_{ij} + X_{jj} \;=\; \|\mathbf{v}_i - \mathbf{v}_j\|^2 \;\le\; 2\|\mathbf{v}_i\|^2 + 2\|\mathbf{v}_j\|^2 \;\le\; 2\mathbf{Tr}(\mathbf{X}) \;\le\; 2\alpha n.$$

Thus, the first set of constraints of SDP (6.13) is $(1, \frac{2\alpha n}{d_{\min}})$ bounded. The second set of constraints $(1, \frac{2n}{d_{\min}})$ bounded.

Thus, by Theorem 30, an ε approximate solution to the SDP can be found in time $\tilde{O}(\frac{n}{d_{\min}\varepsilon^2} \cdot (T_{\text{oracle}} + n^2))$. Note that we do not apply the hybrid algorithm of Corollary 6 here. It remains to estimate $T_{\text{oracle}} = O(T_{\text{ev}}(\mathbf{C}, \delta)$, where $\mathbf{C}$ is the matrix representing the weighted combination of all the constraints, and $\delta = \frac{\varepsilon}{6\alpha n}$ or $\frac{\varepsilon}{3\alpha n}$, depending on whether we use the sparsification procedure or not.

The matrix $\mathbf{C}$ is of the form $\sum_{ij} \frac{p_{ij}}{d_{ij}}(\mathbf{T}_{ij} - \frac{1}{2\alpha n}\mathbf{I}) + \sum_{ij} \frac{q_{ij}}{\alpha d_{ij}}(\frac{1}{2\alpha n}\mathbf{I} - \mathbf{T}_{ij})$, where for all $i, j \in [n]$, $p_{ij} \ge 0$ and $q_{ij} \ge 0$ and $\sum_{ij} p_{ij} + q_{ij} = 1$, and $\mathbf{T}_{ij}$ is the matrix such that $\mathbf{T}_{ij} \bullet \mathbf{X} = X_{ii} - 2X_{ij} + X_{jj}$. Note that $\|\mathbf{T}_{ij}\| = 2$, so we conclude that $\|\mathbf{C}\| \le O(\frac{1}{d_{\min}})$, since $\mathbf{C}$ is a convex combination of the matrices $\frac{1}{d_{ij}}(\mathbf{T}_{ij} - \frac{1}{2\alpha n}\mathbf{I})$ and $\frac{1}{\alpha d_{ij}}(\frac{1}{2\alpha n}\mathbf{I} - \mathbf{T}_{ij})$, all of which have norms bounded by $O(\frac{1}{d_{\min}})$. Thus, the most negative eigenvalue of $\mathbf{C}$,

```
Procedure SPARSIFY($\mathbf{C}, \varepsilon$)
for each $i \le j \in [n]$ do
if $|C_{ij}| > \frac{\varepsilon}{\sqrt{n}}$ then
    $C'_{ji} = C'_{ij} = C_{ij}$
else
    $C'_{ji} = C'_{ij} = \begin{cases} \mathrm{sgn}(C_{ij}) \cdot \frac{\varepsilon}{\sqrt{n}} & \text{with probability } p_{ij} = \frac{\sqrt{n}|C_{ij}|}{\varepsilon} \\ 0 & \text{with probability } 1 - p_{ij} \end{cases}$
return $\mathbf{C}'$
```

Figure 6.1: The SPARSIFY procedure.

λ_n, can be bounded in absolute value by $O(\frac{1}{d_{\min}})$. Thus, by Lemma 31, $T_{\text{ev}}(\mathbf{C}, \delta)$ can be bounded by $\tilde{O}(\frac{N\sqrt{n}}{\sqrt{\varepsilon} d_{\min}})$, where N is the number of non-zero entries in $\mathbf{C}$, since $\delta \ge \Omega(\frac{\varepsilon}{\alpha n})$.

Sparsification could potentially reduce the number of matrix entries. Note that we have $\sum_{ij} |C_{ij}| = \tilde{O}(\frac{1}{d_{\min}})$, again since $\mathbf{C}$ is a convex combination of matrices such that the sum of the absolute values of their entries is $O(\frac{1}{d_{\min}})$. So by Lemma 32, we have $\gamma \ge \Omega(\frac{\varepsilon d_{\min}}{\alpha n})$, and so by Lemma 33 the number of entries could be reduced to $\tilde{O}(\frac{n^{1.5}}{\varepsilon d_{\min}})$.

Thus, the overall running time comes to

$$\tilde{O}\left(\min\left\{\frac{n^3}{d_{\min}^{2.5}\varepsilon^{3.5}}, \frac{n^{3.5}}{d_{\min}^{1.5}\varepsilon^{2.5}}\right\}\right).$$

The first expression is better when $d_{\min} = \omega(n^{-0.5})$. Interior point methods can solve this SDP in time $\tilde{O}(n^4)$. Thus, the first expression is better than interior point methods when $d_{\min} = \omega(n^{-4})$, and the second expression is better when $d_{\min} = \omega(n^{-0.333})$. Thus, in the range of parameters when this algorithm beats the running time of interior point methods, the first expression is always better. □

6.7 Matrix sparsification

The results of this section originally appeared in a joint paper with Sanjeev Arora and Elad Hazan [16]. In this section, we prove Lemma 33:

Lemma 33. *Let $\mathbf{C} \in \mathbb{R}^{n \times n}$ 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 $\mathbf{C}'$ such that $\mathbf{C}'$ has $O(\frac{\sqrt{n}S}{\delta})$ non-zero entries and $\|\mathbf{C} - \mathbf{C}'\|_2 \le \delta$.*

PROOF: The required procedure, SPARSIFY, is given in Figure 6.1. We set the parameter $\varepsilon = \frac{\delta}{16}$. We now prove that it produces the desired result with high probability.

First, we prove that the number of non-zero entries in $\mathbf{C}$ is $O(\frac{\sqrt{n}S}{\varepsilon})$ with high probability.

Lemma 34. *With probability at least* $1 - \exp(-\Omega(\frac{\sqrt{n}S}{\varepsilon}))$*, the matrix* $\mathbf{C}'$ *contains at most* $O(\frac{\sqrt{n}S}{\varepsilon})$ *non-zero entries.*

PROOF: Since $\sum_{ij} |C_{ij}| = S$, 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 [82] asserts that if $X_1, X_2, \ldots, X_n$ are indicator random variables and $X = \sum_i X_i$ with $\mathbb{E}[X] = \mu$, then

$$\mathbf{Pr}[X > (1+\varepsilon)\mu] < \left[\frac{e^{\varepsilon}}{(1+\varepsilon)^{1+\varepsilon}}\right]^{\mu}$$

In our case, we set up indicator random variables X_{ij} for $i \leq j$ which are 0 or 1 depending on whether $C'_{ij} = 0$ or not. Let $X = \sum_{i \leq j} X_{ij}$. Then $2X$ is an upper bound on the number of non-zero entries of $\mathbf{C}'$. We have

$$\mathbb{E}[X] \;=\; \sum_{i \leq j} p_{ij} \;=\; \sum_{i \leq j} \frac{\sqrt{n}|C_{ij}|}{\varepsilon} \;\leq\; \frac{\sqrt{n}S}{\varepsilon}.$$

The claim follows by using the Chernoff bound with $\varepsilon = e - 1$. □

Next, define $\mathbf{E} = \mathbf{C} - \mathbf{C}'$. We will show that with high probability, for all unit vectors $\mathbf{x}$, we have $|\mathbf{x}^\top \mathbf{E} \mathbf{x}| \leq O(\varepsilon)$, which implies $\|\mathbf{C} - \mathbf{C}'\|_2 \leq O(\varepsilon)$.

Notice that for all coordinates i, j such that $|C_{ij}| \geq \frac{\varepsilon}{\sqrt{n}}$, we have $E_{ij} = 0$. For the rest of the coordinates, since $\mathbb{E}[C'_{ij}] = \mathrm{sgn}(C_{ij}) \cdot \frac{\varepsilon}{\sqrt{n}} \times \frac{\sqrt{n}|C_{ij}|}{\varepsilon} = C_{ij}$, we conclude that $\mathbb{E}[E_{ij}] = 0$. We will now consider a $\frac{\varepsilon_0}{\sqrt{n}}$-grid on the unit sphere (ε_0 is set to some constant, say $\frac{1}{2}$),

$$T = \left\{ \mathbf{x} : \; \mathbf{x} \in \frac{\varepsilon_0}{\sqrt{n}} \mathbb{Z}^n, \; \|\mathbf{x}\|_2 \leq 1 \right\}.$$

Feige and Ofek [39] 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 35. *The size of* $|T|$ *is at most* $\exp(cn)$ *for* $c = (\frac{1}{\varepsilon_0} + 2)$*. If for every* $\mathbf{x}, \mathbf{y} \in T$ *we have* $|\mathbf{x}^\top \mathbf{E} \mathbf{y}| \leq \varepsilon$*, then for every unit vector* $\mathbf{x}$*, we have* $|\mathbf{x}^\top \mathbf{E} \mathbf{x}| \leq \frac{\varepsilon}{(1-\varepsilon_0)^2}$.

PROOF: Map every point in $\mathbf{x} \in T$ in a one-to-one correspondence with a n-dimensional hypercube of side length $\frac{\varepsilon_0}{\sqrt{n}}$ on the grid:

$$\mathbf{x} \mapsto \mathcal{C}_{\mathbf{x}} = \left\{ \mathbf{x} + \mathbf{u} : \; \mathbf{u} \geq \mathbf{0}, \; \|\mathbf{u}\|_\infty \leq \frac{\varepsilon_0}{\sqrt{n}} \right\}.$$

The maximum length of any vector in $\mathcal{C}_{\mathbf{x}}$ is bounded by $\|\mathbf{x}\| + \varepsilon_0 \leq 1 + \varepsilon_0$, and thus the union of these cubes is contained in the n-dimensional ball B of radius $(1 + \varepsilon_0)$. We conclude:

$$|T| \times \left(\frac{\varepsilon_0}{\sqrt{n}}\right)^n \;=\; \sum_{\mathbf{x} \in T} \mathrm{Vol}(\mathcal{C}_{\mathbf{x}}) \;\leq\; \mathrm{Vol}(B) \;=\; \frac{\pi^{n/2}}{\Gamma(n/2+1)}(1+\varepsilon_0)^n.$$

And so:

$$|T| \;\le\; \frac{\pi^{n/2}}{\Gamma(n/2+1)}\left(\frac{(1+\varepsilon_0)\sqrt{n}}{\varepsilon_0}\right)^n \;\le\; \exp\left(\left(\frac{1}{\varepsilon_0}+2\right)n\right).$$

Next, given any unit vector, $\mathbf{x}$, let $\mathbf{y} = (1-\varepsilon_0)\mathbf{x}$. By "rounding down" the coordinates of $\mathbf{y}$ to the nearest multiple of $\frac{\varepsilon_0}{\sqrt{n}}$, we get a grid point $\mathbf{z}$ such that $\mathbf{y} \in \mathcal{C}_\mathbf{z}$. Thus, the maximum length of any vertex of $\mathcal{C}_\mathbf{z}$ is bounded by $\|\mathbf{y}\| + \varepsilon_0 = 1$, so all vertices of $\mathcal{C}_\mathbf{z}$ are grid points in T. Express $\mathbf{y}$ as a convex combination of the vertices $\mathbf{v}_i$ of $\mathcal{C}_z$; viz. $\mathbf{y} = \sum_i \alpha_i \mathbf{v}_i$ with $\alpha_i \ge 0$ and $\sum_i \alpha_i = 1$. Then we have

$$|\mathbf{y}^\top \mathbf{E}\mathbf{y}| \;=\; |(\sum_i \alpha_i \mathbf{v}_i)^\top \mathbf{E}(\sum_i \alpha_i \mathbf{v}_i)| \;\le\; \sum_{i,j} \alpha_i\alpha_j |\mathbf{v}_i^\top \mathbf{E}\mathbf{v}_j| \;\le \sum_{i,j}\alpha_i\alpha_j\varepsilon \;=\; \varepsilon.$$

The second inequality above follows because we assumed that for all $\mathbf{x}', \mathbf{y}' \in T$, $|\mathbf{x}'^\top \mathbf{E}\mathbf{y}'| \le \varepsilon$. Finally, since $\mathbf{y} = (1-\varepsilon_0)\mathbf{x}$, we have

$$|\mathbf{x}^\top \mathbf{E}\mathbf{x}| \;=\; \frac{|\mathbf{y}^\top \mathbf{E}\mathbf{y}|}{(1-\varepsilon_0)^2} \;\le\; \frac{\varepsilon}{(1-\varepsilon_0)^2}.$$

□

Let $\mathbf{x}, \mathbf{y} \in T$. Since $\mathbb{E}[E_{ij}] = 0$, we conclude that $\mathbb{E}[\mathbf{x}^\top \mathbf{E}\mathbf{y}] = 0$. We now a prove strong concentration bound:

Lemma 36. *With probability at least* $1 - \exp(-\Omega(n))$, *for every* $\mathbf{x}, \mathbf{y} \in T$ *it holds that* $|\mathbf{x}^\top \mathbf{E}\mathbf{y}| \le c\varepsilon$.

PROOF: We use the following bound from Hoeffding's original paper [53]: 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 $\mathbb{E}[X] = \mu$. Then for any $t > 0$

$$\Pr[|X-\mu| \ge t] \;\le\; 2\exp\left(-\frac{2t^2}{\sum_i (b_i - a_i)^2}\right).$$

Consider the random variables $Z_{ij} = E_{ij}x_iy_j$, then $\mathbf{x}^\top \mathbf{E}\mathbf{y} = \sum_{ij} E_{ij}x_iy_j = \sum_{ij} Z_{ij}$. Since C'_{ij} is either $\mathrm{sgn}(C_{ij}) \cdot \frac{\varepsilon}{\sqrt{n}}$ or 0, the squared range of E_{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 $\mathbb{E}[Z_{ij}] = 0$, by the Hoeffding bound we have:

$$\Pr\left[|\sum_{i\le j} Z_{ij}| \;\ge\; c\varepsilon\right] \;\le\; 2\exp\left(-\frac{2c^2\varepsilon^2}{\frac{\varepsilon^2}{n}}\right) \;=\; 2\exp(-2c^2 n).$$

A similar bound holds for $\Pr[|\sum_{i>j} Z_{ij}| \;\ge\; c\varepsilon]$. Since $|\mathbf{x}^\top \mathbf{E}\mathbf{y}| = |\sum_{i\le j} Z_{ij} + \sum_{i>j} Z_{ij}|$, by the union bound we have

$$\Pr[|\mathbf{x}^\top \mathbf{E}\mathbf{y}| \;\ge\; 2c\varepsilon] \;\le\; 4\exp(-2c^2 n).$$

Since there are $\exp(2cn)$ pairs of vectors $\mathbf{x}, y \in T$, the union bound implies that with probability at least $1 - \exp(-\Omega(n))$, for all vectors $\mathbf{x}, \mathbf{y} \in T$, we have $|\mathbf{x}^\top \mathbf{E} \mathbf{y}| \leq c\varepsilon$. □

Now, we finish the proof of Lemma 33. We set $\varepsilon_0 = \frac{1}{2}$, so that $c = 4$. We set $\varepsilon = \frac{\delta}{16}$ in the Sparsify procedure. Thus, Lemma 36 shows that for all $\mathbf{x}, \mathbf{y} \in T$, we have $|\mathbf{x}^\top \mathbf{E} \mathbf{y}| \leq \frac{\delta}{4}$ with high probability. Then Lemma 35 shows that for all unit vectors $\mathbf{x}$, we have $|\mathbf{x}^\top \mathbf{E} \mathbf{x}| \leq \delta$, as required. □

6.8 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 require 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 $\frac{1}{\varepsilon}$.

Chapter 7

Graph Partitioning using Expander Flows

In this chapter, we focus on some graph partitioning problems considered in Chapter 4, viz. the Sparsest Cut and Balanced Separator problems for undirected graphs. These problems ask for a partition of the input graph into two large pieces while minimizing the size of the "interface" between them, as measured by the number of edges crossing the partition. As mentioned previously, such graph partitions are of fundamental importance, appearing in the theory of Markov chains, geometric embeddings, divide-and-conquer algorithms, clustering algorithms etc. In Chapter 4, we showed how to obtain an $O(\sqrt{\log n})$ factor approximation to the Sparsest Cut and Balanced Separator problems in $\tilde{O}(n^2)$ time by approximately solving the SDP using a primal-dual approach.

In this chapter, we consider alternative approximation algorithms for these problems using a notion (which appeared in the original Arora, Rao Vazirani (ARV) paper [18]) called *expander flows*. These are multicommodity flows in the graph whose *demand graph* is an expander. The existence of an expander flow provides a lower bound on the expansion of a graph, and ARV showed that an expander flow which bounds the expansion of the graph to within a $O(\sqrt{\log n})$ factor can be found, though their algorithm made use of the ellipsoid algorithm and hence is quite inefficient.

We cast the problem of routing an expander as a linear feasibility program, and use the basic Multiplicative Weights to solve it in time $\tilde{O}(n^2)$. Thus, we compute an $O(\sqrt{\log n})$ factor approximation to the Sparsest Cut and Balanced Separator problems in $\tilde{O}(n^2)$ time. This matches the running time of the algorithms of Chapter 4, but the technique is quite different, involving flow computations which also produce a certificate of expansion.

The results of this chapter first appeared in a joint paper with Sanjeev Arora and Elad Hazan [12].

7.1 Graph partitioning: a recapitulation

We now recall some notation for the problems considered in this chapter. We are given a graph $G = (V, E)$ with specified capacities c_e for every edge e. Let $n := |V|, m := |E|$. For any cut $(S, \bar{S})$ where $\bar{S} = V \setminus S$ and $|S| \leq |V|/2$, the *edge expansion* of the cut is $E(S, \bar{S})/|S|$, where $E(S, \bar{S})$ is the total capacity of the edges crossing the cut. In the SPARSEST CUT problem we wish to determine the cut with the smallest edge expansion:

$$\alpha(G) = \min_{S \subseteq V, |S| \leq |V|/2} \frac{E(S, \bar{S})}{|S|}. \tag{7.1}$$

A cut $(S, \bar{S})$ is *c-balanced*, for some parameter $c \leq 1/2$, if both $S, \bar{S}$ have at least $c|V|$ vertices. In the minimum c-BALANCED SEPARATOR problem we wish to determine $\alpha_c(G)$, the minimum expansion of c-balanced cuts. The *conductance* of a cut $(S, \bar{S})$ is the quantity $E(S, \bar{S})/E(S)$, where $E(S)$ denotes the sum of degrees (in terms of edge capacities) of nodes in S, and here we assume that $E(S) \leq E(V)/2$. In the GRAPH CONDUCTANCE problem we wish to determine the cut with the smallest conductance:

$$\Phi(G) = \min_{S \subseteq V, E(S) \leq E(V)/2} \frac{E(S, \bar{S})}{E(S)}. \tag{7.2}$$

Efforts to design good approximation algorithms for these NP-hard problems have spurred the development of many subfields of theoretical computer science. The earliest algorithms relied on spectral methods introduced — in the context of Riemannian manifolds — by Cheeger [35] and improved by Alon and Milman [7] and Alon [6]. Though this connection between eigenvalues and conductance only yields a weak approximation (the worst-case approximation ratio is n), it has had enormous influence in a variety of areas, including random walks, pseudorandomness, error-correcting codes, and routing.

Leighton and Rao [74] designed the first true approximation by giving $O(\log n)$-approximations for SPARSEST CUT and GRAPH CONDUCTANCE and $O(\log n)$-pseudo-approximations for the minimum c-BALANCED SEPARATOR. They used a linear programming relaxation of the problem based on multicommodity flows proposed in [88]. Leighton, Rao and others used similar ideas to design approximation algorithms for numerous **NP**-hard problems, see the surveys [89, 95]. Furthermore, efforts to improve these ideas led to progress in other areas, such as fast computations of multicommodity flows and packing-covering linear programs [103, 85, 44], and efficient geometric embeddings of metric spaces [75]; see also [19].

Arora, Rao and Vazirani (ARV) [18] designed an $O(\sqrt{\log n})$-approximation algorithm. They use semidefinite programming (SDP), a technique introduced in approximation algorithms by Goemans and Williamson [45]. The running time of the ARV algorithm is dominated by the solution of this SDP, which takes $\tilde{O}(n^{4.5})$ time using interior point methods [5]. (Here and in the rest of this chapter, $\tilde{O}(\cdot)$ notation is used to suppress polylogarithmic factors.) New techniques in high-dimensional geometry introduced by their analysis found immediate application in algorithms for other problems; e.g. the graph partitioning algorithms considered in Chapter 4 (which are in turn inspired by the work of [2]), and embedding algorithms of negative type metric spaces into ℓ_2 [32, 17]. In Chapter 4, we showed how to obtain an $O(\sqrt{\log n})$ factor approximation to the Sparsest Cut and Balanced Separator problems in $\tilde{O}(n^2)$ time by approximately solving the SDP using a primal-dual approach.

In this chapter, we consider alternative approximation algorithms for these problems using a notion called *expander flows*. These are multicommodity flows in the graph whose *demand graph* (i.e., the weighted graph $\langle d_{ij} \rangle_{ij}$ where d_{ij} is the flow shipped between nodes i, j) is an expander. Of course, Leighton and Rao had shown how to embed even the complete graph (which in particular is an expander) in the host graph, so the important issue here is the *edge congestion* (maximum amount of flow using an edge). The flows exhibited by Arora, Rao, Vazirani are efficient enough to work with a $\sqrt{\log n}$ factor lower congestion than Leighton-Rao flows. Thus these flows can be used to certify that the expansion is $\Omega(\alpha(G)/\sqrt{\log n})$. (Note that determining graph expansion is **coNP**-complete [24], so we cannot expect to have succinct certificates that prove that the expansion is exactly $\alpha(G)$.)

In addition to being an interesting graph theoretic fact — analogous to, say, the approximate max-flow min-cut theorem that underlies Leighton-Rao's result — the existence of such expander flows seems to hint at a faster version of the ARV approximation algorithm for Sparsest Cut. After all, computation of multicommodity flows is a highly developed area today. Thus an approximation algorithm for Sparsest Cut could try to route a multicommodity flow in the graph, and modify it using eigenvalue computations that check if the current demands form an expander. If an expander flow exists (given a certain upper bound on congestion) then the final multicommodity flow would converge to it. If the expander flow doesn't exist, the algorithm would presumably find a very sparse cut that "proves" this fact. ARV [18] suggested this approach for designing faster algorithms, though the best algorithm they could come up with used the Ellipsoid method, and hence was less efficient even than the SDP-based one.

This chapter presents an $\tilde{O}(n^2)$ time randomized algorithm that uses expander flows to compute a $O(\sqrt{\log n})$-approximation to Sparsest Cut. This essentially matches the running time of the best implementations of Leighton-Rao's $O(\log n)$-approximation (Benczúr and Karger [22], the last paper in a long line of work), and also matches the running time of the algorithms described in Chapter 4. The algorithm computes an expander flow in a sparse weighted graph that is obtained by Benczúr and Karger using a (nontrivial) random sampling on the original graph. This expander flow suffices to certify the expansion of the original graph. Furthermore, the algorithm produces, in addition to expander flows, a distribution on $O(\log n)$ balanced cuts, which can be viewed as an ℓ_1 metric, and thus can be embedded in ℓ_2^2. Then we use the ideas of Arora-Rao-Vazirani to obtain the $O(\sqrt{\log n})$-approximate sparsest cut from this metric.

The algorithm can also yield expander flows in any weighted graph on m edges in $\tilde{O}(m^2)$ time.

Overview of methodology

As mentioned, we first use the sparsification technique of Benczúr-Karger (see Theorem 17) to transform our graph to a sparse weighted graph in which the number of edges

$m = \tilde{O}(n)$. The value of the sparsest cut is essentially unchanged, so we find expander flows in the sparse graph.

Many papers (such as [85, 103]) have described efficient algorithms for packing and covering problems based on the Multiplicative Weights algorithm of Section 2.3.2. The linear program for finding expander flows also has packing and covering constraints, and so we can apply the algorithm of Section 2.3.2 to solve it (actually, the dual to the linear program). The issue is that the linear program has exponentially many variables, corresponding to every path in the graph, and so the dual has exponentially many constraints. However, by only considering shortest paths, we can restrict the number of constraints to $O(n^2)$, at the expense of getting concave constraints (which can be handled using the algorithm of Section 2.3.2).

The weights generated in the algorithm can now be interpreted as *demands* for a multi-commodity flow. The Oracle in the Multiplicative Weights algorithm now needs to check whether the demands do correspond to an actual expander flow. In the interest of keeping the width low, the Oracle only checks if the demands correspond to an approximate expander flow, viz. a "pseudo-expander flow". Another reason for the need to consider pseudo-expander flows arises from a lack of precision because the Oracle uses approximate flow algorithms and eigenvalue computations. The final pseudo-expander flow obtained is thus fairly coarse, but an expander flow is such a robust object that anything even remotely resembling it can be easily turned into a true expander flow.

We design the Oracle carefully so that it runs in $\tilde{O}(n^2)$ time and yet manages to keep the width $O(1)$, thus ensuring that the Multiplicative Weights algorithm converges in $O(\log n)$ rounds. The Oracle uses a combination of random sampling (above and beyond the use of Benczúr-Karger at the very start), approximate min-cost concurrent multicommodity computations, and approximate eigenvalue computations to generate its response.

The outline appears in Section 7.2 and subsequent sections fill in the details. We note that when the algorithm fails to result in an expander flow, then one can produce an approximately optimum sparsest cut; this part relies on the analysis in [18].

7.2 Expander flows and algorithm overview

We now define expander flows and outline the main ideas in our algorithm.

All weighted graphs in this paper are symmetric, that is $d_{ij} = d_{ji}$ for all node pairs i, j. We call $d_i = \sum_j d_{ij}$ the *degree* of node i. We emphasize that degrees can be fractions (i.e., less than 1).

A *multicommodity flow* $\mathbf{f}$ in an capacitated graph $G = (V, E)$ is an assignment of *demand* $d_{ij} \geq 0$ to each node pair $\{i, j\}$ and a flow f_p to every path p in the graph which meets the demands; i.e. we can route d_{ij} units of flow between i and j, and can do this simultaneously for all pairs without violating any edge capacities. We refer to the complete weighted graph with edge $\{i, j\}$ having weight d_{ij} as the *demand graph* of the flow. Given a subset $S \subseteq V$ the *demand crossing the cut* $(S, \bar{S})$ is the capacity of the cut $(S, \bar{S})$ in the demand graph, i.e. $d(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} d_{ij}$.

Definition 5. *The graph with edge weights $\langle d_{ij} \rangle_{ij}$ is a D-regular β-expander if it has maximum degree at most D and for any subset $S \subseteq V$ such that $|S| \leq n/2$ the total weight crossing the cut $(S, \bar{S})$, viz. $d(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} d_{ij}$, satisfies*

$$d(S, \bar{S}) \; \geq \; \beta D |S|.$$

A multi-commodity flow $\mathbf{f}$ *is called a D-regular β-pseudo expander flow (or just expander flow for short) if its demand graph is a D-regular β-expander.*

Note that we have relaxed D-regularity and only require maximum degree D; this is without loss of generality since one could add self-loops to raise all degrees to D.

Lemma 37. *If a graph G admits a multicommodity flow whose demand graph is a D-regular β-expander, then its expansion is at least βD.*

Proof: Let $\langle d_{ij} \rangle_{ij}$ be the demands in the D-regular β-expander flow. Then for any $S \subseteq V$ with $|S| \leq n/2$, the capacity of the cut $(S, \bar{S})$ must be at least the demand crossing it. Thus,

$$E(S, \bar{S}) \; \geq \; d(S, \bar{S}) \; \geq \; \beta D |S|,$$

which implies that $\frac{E(S,\bar{S})}{|S|} \geq \beta D$. Thus, the expansion of G is at least βD. □

Another notion we will need is that of a pseudo-expander flow:

Definition 6. *The graph with edge weights $\langle d_{ij} \rangle_{ij}$ is a D-regular (c, β)-pseudo expander if it has maximum degree at most D and for any subset $S \subseteq V$ such that $cn \leq |S| \leq n/2$ the total weight crossing the cut $(S, \bar{S})$, viz. $d(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} d_{ij}$, satisfies*

$$d(S, \bar{S}) \; \geq \; \beta D |S|.$$

A multi-commodity flow $\mathbf{f}$ *is called a D-regular (c, β)-pseudo expander flow (or just pseudo-expander flow for short) if its demand graph is a D-regular (c, β)-pseudo expander.*

Notice that a D-regular β-expander flow is in particular a D-regular (γ, β)-pseudo expander flow for each γ. Just like expander flows, pseudo-expander flows can be used to obtain lower bounds on the expansion of balanced cuts:

Lemma 38. *If a graph G admits a multicommodity flow whose demand graph is a D-regular (c, β)-pseudo expander, then the expansion of all c-balanced cuts is at least βD.*

This lemma is proved just as before. The following theorem of Arora, Rao and Vazirani [18] shows that the notions of expander flows and pseudo-expander flows allow us to obtain $O(\sqrt{\log n})$ approximations to the expansions of the Sparsest Cut and minimum c-Balanced Separator respectively:

Theorem 37 ([18]). *There is a constant $\beta_0 > 0$ such that every graph $G = (V, E)$ admits a D-regular β_0-expander flow, where $D = \Omega(\alpha(G)/\sqrt{\log n})$. Further, every graph G admits a D-regular (c, β_0)-pseudo expander flow, where $D = \Omega(\alpha_{c'}(G)/\sqrt{\log n})$ for some $c' \leq c$.*

The essence of our work is to show how to efficiently compute such expander flows. To understand this algorithm it helps to first look at an LP whose feasibility is implied by the above Theorem. (This LP is mentioned in the introduction of [18] and motivated the results of that paper.) Let D be the degree of interest; think of it as a "constant" in the LP, not as a variable.

For all simple paths p in the graph, we have a non-negative variable f_p. Let $\mathcal{P}_{ij}$ be the set of paths connecting node pair $\{i, j\}$, $\mathcal{P}_{i*}$ be the set of paths originating from node i, and $\mathcal{P}_{S,\bar{S}}$ be the set of paths having end points on either side of the cut $(S, \bar{S})$. In the following LP, we use the notation "$\forall S$" to only refer to subsets of vertices of size at most $n/2$. The Primal LP is

$$\begin{aligned} \forall i : \sum_{p \in \mathcal{P}_{i*}} f_p &\leq D \\ \forall e : \sum_{p:\ e \in p} f_p &\leq c_e \\ \forall S : \sum_{p \in \mathcal{P}_{S,\bar{S}}} f_p &\geq \beta_0 D |S| \end{aligned} \tag{7.3}$$

The LP for D-regular (c, β_0)-pseudo expander flows is the same as the one above except that the third set of constraints is only over subsets of vertices S such that $cn \leq |S| \leq n/2$. We give a unified treatment of both LPs, and in the following sections, whenever a subset of vertices S occurs, it is implicitly assumed that $|S| \leq n/2$, and, in the case of the c-Balanced Separator problem, $|S| \geq cn$. We will need to use the constant c even in the context of Sparsest Cut, and in this case we set $c = \frac{1}{2}$.

As outlined in [18], the Primal LP can be solved to near-optimality in polynomial time by an Ellipsoid-like method, using an eigenvalue computation as a separation oracle. To design a better algorithm we consider the dual to the LP and solve it using the Multiplicative Weights algorithm of Section 2.3.2. Since the algorithm in that framework maintains a distribution on all pure row strategies, it is important to work with games where the number of pure row strategies is polynomial. In particular, we need a polynomial-size representation of the flow. The standard representation uses variables f_{ije} for each demand pair $(i, j) \in V \times V$ and edge $e \in E$. We do not know how to formulate our algorithm using this representation, and even if we did, the number of variables (i.e., number of pure strategies for the row player) would be $\Omega(n^2 m)$, which would be a lower bound on the running time[1]. The idea instead is to not use any representation of the flows at all, and to maintain only the *demands* d_{ij}. Now the number of variables is $\binom{n}{2}$ and so we at least have a prayer of achieving $\tilde{O}(n^2)$ running time.

We now consider the Dual LP to (7.3). In this Dual LP we have non-negative

[1]Note that it is possible to use random sampling to reduce the number of nonzero demands to $O(n \log n)$; in fact this is done during every iteration of our algorithm while computing the best response for the column player. However, the Multiplicative Weights update rule seems to require updating the distribution on all row strategies, and indeed we update all $O(n^2)$ demands at every iteration.

variables s_i, w_e, z_S corresponding to vertex i, edge e, and subset $S \subseteq V$ respectively.

$$\begin{aligned} \min \quad & D\sum_i s_i + \sum_e c_e w_e - \beta_0 D \sum_S |S| z_S \\ \forall ij, \forall p \in \mathcal{P}_{ij}: \quad & s_i + s_j + \sum_{e \in p} w_e - \sum_{S:\ i \in S, j \in \bar{S}} z_S \geq 0 \end{aligned} \tag{7.4}$$

Lemma 39. *If the* Primal *is feasible, then the optimum of the* Dual *LP is non-negative.*

Proof: This is a simple consequence of weak duality. Let f_p be an assignment of flow to paths p which satisfies (7.3). Then, if $\mathbf{x} = \langle \mathbf{s}, \mathbf{w}, \mathbf{z} \rangle$ is any feasible dual solution, then

$$\begin{aligned} 0 &\leq \sum_{ij} \sum_{p \in \mathcal{P}_{ij}} f_p \left[s_i + s_j + \sum_{e \in p} w_e - \sum_{S:\ i \in S, j \in \bar{S}} z_S \right] \\ &= \sum_i \sum_{p \in \mathcal{P}_{i*}} f_p s_i + \sum_e \sum_{p \ni e} f_p w_e - \sum_S \sum_{p \in \mathcal{P}_{S,\bar{S}}} f_p z_S \\ &\leq \sum_i D s_i + \sum_e c_e w_e - \beta_0 D \sum_S |S| z_S, \end{aligned}$$

because the f_p's satisfy (7.3). Thus, the Dual objective value is non-negative, as required. □

As it stands, the Dual LP has exponentially many constraints corresponding to all possible paths p in the graph. The Dual can be equivalently written with only $O(n^2)$ constraints with the following observation. For any pair i, j, and for any dual solution $\mathbf{x} = \langle \mathbf{s}, \mathbf{w}, \mathbf{z} \rangle$, let p be the shortest path between i and j under the edge lengths w_e, i.e. $p = \arg\min_{p \in \mathcal{P}_{ij}} \sum_{e \in p} w_e$. Then if the dual constraint in (7.4) corresponding to p is satisfied, then so are the constraints to all the other paths in $\mathcal{P}_{ij}$. Let $l_{ij}(\mathbf{x})$ be the length of the shortest path between i and j under the edge lengths w_e. Then we have the following equivalent Dual program which has only $\binom{n}{2}$ constraints:

$$\begin{aligned} \min \quad & D\sum_i s_i + \sum_e c_e w_e - \beta_0 D \sum_S |S| z_S \\ \forall ij: \quad & s_i + s_j + l_{ij}(\mathbf{x}) - \sum_{S:\ i \in S, j \in \bar{S}} z_S \geq 0 \end{aligned} \tag{7.5}$$

In this case, the primal variables corresponding to the constraints for a pair of nodes $\{i, j\}$ is interpreted as aggregate *demand* d_{ij} between them.

Lemma 40. *All constraints in the* Dual *program (7.5) are concave functions of the dual variables* $\mathbf{x}$.

Proof: Since the only non-linear part of the constraints of the Dual program (7.5) are the functions l_{ij}, it suffices to show that these functions are concave. For this purpose, let $\mathbf{x} = \langle \mathbf{s}, \mathbf{w}, \mathbf{z} \rangle$ and $\mathbf{x}' = \langle \mathbf{s}', \mathbf{w}', \mathbf{z}' \rangle$ be two vectors of dual variables, and let $\lambda \in [0, 1]$. We need to show that $l_{ij}(\lambda\mathbf{x}+(1-\lambda)\mathbf{x}') \geq \lambda l_{ij}(\mathbf{x})+(1-\lambda)l_{ij}(\mathbf{x}')$ to show that l_{ij} is concave. Now, let p be the shortest path from i to j under edge lengths $\lambda\mathbf{w}+(1-\lambda)\mathbf{w}'$. Then

$$\begin{aligned} l_{ij}(\lambda\mathbf{x}+(1-\lambda)\mathbf{x}') &= \sum_{e\in p}\lambda w_e+(1-\lambda)w'_e \\ &= \lambda\sum_{e\in\tilde{p}} w_e+(1-\lambda)\sum_{e\in\tilde{p}} w'_e \\ &\geq \lambda l_{ij}(\mathbf{x})+(1-\lambda)l_{ij}(\mathbf{x}'), \end{aligned}$$

because the path p has length at least $l_{ij}(\mathbf{x})$ under $\mathbf{w}$, and at least $l_{ij}(\mathbf{x}')$ under $\mathbf{w}'$, by the definition of l_{ij}. $\Box$

We now consider the convex domain of dual variables

$$\mathcal{Q}' = \{\mathbf{x}: \; D\sum_i s_i+\sum_e c_e w_e \leq \beta nD; \; \sum_S |S| z_S = n\}.$$

where $\beta \ll \beta_0$ is a small positive constant to be chosen later. This set is chosen so that all dual vectors $\mathbf{x}\in\mathcal{Q}'$ make the Dual objective negative: for all $\mathbf{x}\in\mathcal{Q}'$, we have

$$D\sum_i s_i+\sum_e c_e w_e-\beta_0 D\sum_S |S| z_S \;\leq\; -(\beta-\beta_0)nD \;<\; 0.$$

Now, consider the following feasibility problem:

$$\exists?\ \mathbf{x}\in\mathcal{Q}': \quad \forall ij: \quad s_i+s_j+\ell_{ij}(\mathbf{x})-\sum_{S:\ i\in S, j\in\bar{S}} z_S \geq 0 \tag{7.6}$$

The significance of this feasibility problem is given by the following lemma, proved in Section 7.4:

Lemma 41. *There is a constant $\delta>0$ such that if $\mathbf{x}\in\mathcal{Q}'$ satisfies the feasibility problem (7.6) up to additive error δ, then there is an algorithm that uses $\mathbf{x}$ to find a c'-balanced cut of expansion $O(\sqrt{\log n}\cdot D)$, for some $c'\leq c$.*

We can now run the Multiplicative Weights algorithm of Section 2.3.2 to solve the feasibility problem (7.6). If the algorithm manages to satisfy (7.6) over $\mathcal{Q}'$, then it implies that the current value of D is too high: there is no expander flow which meets the degree bound D.

The Multiplicative Weights algorithm generates probability distributions $\mathbf{p}=\langle p_{ij}\rangle_{ij}$ on the constraints in every round. By scaling up the distribution by $\frac{1}{2}nD$, we obtain a set of demands $\langle d_{ij}\rangle_{ij}$ such that $\sum_{i<j} d_{ij}=\frac{1}{2}nD$. These d_{ij}'s correspond to demands for a multicommodity flow problem; we emphasize that the demands need not correspond to an expander flow: there may not be a flow $\langle f_p\rangle_p$ satisfying these demands which also satisfies the Primal constraints.

We can implement the required Oracle for the Multiplicative Weights algorithm using a combination of mincost concurrent flow, eigenvalue computations, and random sampling in $\tilde{O}(n^2)$ time. The width of the implementation of the Oracle turns out to be

$O(n)$, and thus by Theorem 6, the Multiplicative Weights algorithm converges in $\tilde{O}(n^2)$ rounds. Thus the overall running time would be $\tilde{O}(n^2 \times n^2) = \tilde{O}(n^4)$. This is only slightly better than solving SDPs or LPs.

Now we describe a related setting in which the losses are truncated and the convex domain is restricted, so that the width is $O(1)$. The implementation of the ORACLE is considerably more complicated now, but it still runs in $\tilde{O}(n^2)$ time. Then Theorem 6 implies that convergence occurs in $O(\log n)$ rounds, and the overall running time is $\tilde{O}(n^2)$.

First, we restrict the convex domain for the problem as follows:

$$\mathcal{Q} := \left\{\mathbf{x} \in \mathcal{Q}' : \quad \forall i : \ s_i \leq 1/\varepsilon \text{ and } \forall S \text{ s.t. } |S| < cn : \ z_S = 0\right\}. \tag{7.7}$$

Here ε is a small constant defined in Section 7.3. Next, we truncate the l_{ij}'s: define

$$\ell_{ij}(\mathbf{x}) \ = \ \min\left\{l_{ij}(\mathbf{x}),\ 1/\varepsilon\right\}. \tag{7.8}$$

Since ℓ_{ij} is defined as the minimum of two concave functions (l_{ij} and the constant function which takes the value $1/\varepsilon$), it is also a concave function. Now, we consider the following feasibility problem (here, we set $c = \frac{1}{2}$ for the SPARSEST CUT problem):

$$\exists ? \mathbf{x} \in \mathcal{Q} : \quad \forall ij : \quad s_i + s_j + \ell_{ij}(\mathbf{x}) - \sum_{S:\ i \in S, j \in \bar{S}} z_S \geq 0 \tag{7.9}$$

Since we truncated the l_{ij}'s and restricted the domain $\mathcal{Q}'$, if (7.9) is feasible, then so is (7.6). Thus, it suffices to solve (7.9) up to the desired additive error δ.

Now we can see how these changes restrict the width of the ORACLE to $O(1)$: since $z_S > 0$ only for $|S| \geq cn$, and $\sum_S |S|\, z_S = n$, we have $\sum_S z_S \leq 1/c$. Thus, for any i, j and any $\mathbf{x} \in \mathcal{Q}$, we have

$$-\frac{1}{c} \ \leq \ s_i + s_j + \ell_{ij}(\mathbf{x}) - \sum_{S:\ i \in S, j \in \bar{S}} z_S \ \leq \ \frac{3}{\varepsilon}.$$

Thus, any implementation of an ORACLE for this problem is $(O(1), O(1))$-bounded, and thus the width is $O(1)$.

Now, we consider the ORACLE for this problem. Given a set of demands $\mathbf{d} = \langle d_{ij} \rangle_{ij}$, the ORACLE needs to solve following feasibility problem:

$$\exists ?\ \mathbf{x} \in \mathcal{Q}' : \quad \sum_{ij} d_{ij} \left[s_i + s_j + \ell_{ij}(\mathbf{x}) - \textstyle\sum_{S:\ i \in S, j \in \bar{S}} z_S\right] \ \geq \ 0 \tag{7.10}$$

Recall that $d_i = \sum_j d_{ij}$ and $d(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} d_{ij}$. For notational convenience, we define the payoff function for the ORACLE to be the LHS in (7.10):

$$\begin{aligned} v(\mathbf{d}, \mathbf{x}) \ &:= \ \sum_{ij} d_{ij} \left[s_i + s_j + \ell_{ij}(\mathbf{x}) - \textstyle\sum_{S:\ i \in S, j \in \bar{S}} z_S\right] \\ &= \ \sum_i d_i s_i + \sum_{ij} d_{ij} \ell_{ij}(\mathbf{x}) + \sum_S d(S, \bar{S}) z_S. \end{aligned} \tag{7.11}$$

The ORACLE's objective now is to enforce non-negative payoff (as measured by $v(\mathbf{d}, \mathbf{x})$) when given a set of demands $\mathbf{d}$. The next theorem (proved in Section 7.3), describes an implementation for the ORACLE which runs in $\tilde{O}(n^2)$ time, with the property that if the current set of demands $\mathbf{d}$ is such that the ORACLE cannot enforce non-negative payoff (which must happen close to convergence), then $\mathbf{d}$ "almost" represents an expander flow; in fact we obtain a pseudo-expander flow. A pseudo-expander flow can be used to find either an expander flow, or a cut of expansion $O(D)$. We prove the following lemma in Section 7.3:

Lemma 42. *There is a randomized procedure,* ORACLE, *which given a set of demands* $\mathbf{d}$, *runs in* $\tilde{O}(n^2)$ *time and*

1. *either produces an* $\mathbf{x} \in \mathcal{Q}$ *with* $v(\mathbf{d}, \mathbf{x}) \geq 0$ *in which exactly one set* $S \subseteq V$ *with at least* cn *vertices has a non-zero* z_S,
2. *or else,* ORACLE *fails. In this case, a* D*-regular* (c, β_2)*-pseudo-expander flow can be constructed from* $\mathbf{d}$, *for some constant* β_2 *which depends only on* β.

This lemma is sufficient for the BALANCED SEPARATOR problem. For the SPARSEST CUT problem, whenever the ORACLE finds, and we obtains a D-regular (c, β_2)-pseudo expander flow, it runs the algorithm given by the following lemma (the stipulation on the number of non-zero demands will be met by random sampling), which shows that we can pass from a pseudo-expander-flow to an expander flow (and when we fail to so, we can obtain a reason for the failure in form of a sparse cut). This lemma is proved in Section 7.6. Note that the number of edges, m, is $\tilde{O}(n)$, since we sparsify the graph first using the techniques of Benczúr and Karger [22].

Lemma 43. *Let* $\mathbf{f} = \langle f_p \rangle_p$ *be a* D*-regular* (c, β)*-pseudo-expander flow on a graph* G. *Assume that the flow has non-zero demand on only* $O(n \log n)$ *pairs of vertices. Then, there is a procedure that in time* $\tilde{O}(m^{1.5})$, *finds either*

1. *a* D*-regular* $\frac{\beta^2}{130}$ *expander flow,*
2. *or, a cut of expansion at most* $\frac{1}{c}D$.

Finally, if the ORACLE never fails, then the Multiplicative Weights algorithm finds an feasible solution for the feasibility problem (7.9) up to any specified additive error δ. In this case, by Lemma 41, we can find a balanced cut of value $O(\sqrt{\log n} \cdot D)$ of the optimal. Thus, putting Lemmas 41, 42 and 43 together (noting that we set $c = \frac{1}{2}$), we immediately have the following main theorem:

Theorem 38. *For some universal constant* β, *there is a procedure, that given a graph* G *and a value* D, *finds in* $\tilde{O}(n^2)$ *time either*

1. *a* D*-regular* β*-expander flow, or*
2. *a cut of expansion at most* $O(\sqrt{\log n} \cdot D)$.

If $\mathbf{f}$ is a D-regular β-expander flow, then it is easy to check that for any $D' \leq D$, $\frac{D'}{D}\mathbf{f}$ is a D'-regular β-expander flow. Thus, by starting with a low value of D and doubling it every time, we can find a value D^* such the algorithm finds a D^*-regular β-expander flow, and a cut of expansion at most $O(\sqrt{\log n} \cdot D^*)$. Thus, by Lemma 37 we have $D^* \leq \alpha(G) \leq O(\sqrt{\log n} \cdot D^*)$, and so we have the desired $O(\sqrt{\log n})$ approximation to the Sparsest Cut.

For the minimum c-Balanced Separator, by applying Lemmas 41 and 42 as before, we immediately have the following theorem:

Theorem 39. *For some universal constant β, there is a procedure, that given a graph G and a value D, finds in $\tilde{O}(n^2)$ time either*

1. *a D-regular (c, β)-pseudo expander flow, or*
2. *a c' balanced cut of expansion at most $O(\sqrt{\log n} \cdot D)$, for some $c' \leq c$.*

7.2.1 A note on running time:

We make a few remarks on the $\tilde{O}(n^2)$ running time, which occurs many times in this chapter and in particular in the implementation of Oracle. First, one can reduce the number of nonzero demands to $\tilde{O}(n)$ by random sampling. This is a known technique from existing Sparsest Cut implementations (see eg [68], [22]) though we occasionally need to add a few simple ideas.

In many places we need to find cuts $(S, \bar{S})$ where the demand graph fails to expand (i.e. $d(S, \bar{S}) = o(nD)$) and the cut is large, namely $|S| = \Omega(n)$. Using the well-known results of Cheeger and Alon we can do this using approximate eigenvalue computations on the Laplacian of this sparse graph, which takes $\tilde{O}(n)$ time by repeated matrix-vector products. (This idea has been repeatedly rediscovered, but one reference is [71]). Using the eigenvector and Theorem 40 we can find cuts (if any exist) where the demand graph does not expand. Repeating the eigenvector method $O(n)$ times we try to aggregate these small cuts to have size $\Omega(n)$. If this aggregation fails to produce any large cuts that do not expand, then we can throw away $o(n)$ of the graph such that in the remaining graph all cuts expand well. (In other words, we have a pseudo-expander flow already.) Thus the total time is $\tilde{O}(n^2)$.

The Oracle procedure performs a min-cost concurrent multicommodity flow computation using the algorithm of Fleischer [40], which also takes time $\tilde{O}(n^2)$ since the number of demands has been reduced to $\tilde{O}(n)$ by random sampling.

Finally, we repeat the algorithm of Theorem 39 for successively doubling values of D. Thus overall, the algorithm for approximating Sparsest Cut takes $\tilde{O}(n^2 \cdot \log(\frac{U}{L}))$ time, where $[L, U]$ is a range of values for $\alpha(G)$.

We can bound $\frac{U}{L}$ by $O(n)$ as follows. Let the global min-cut value in the graph G be C (this value can be approximated to a constant factor in $O(m+n)$ time using Matula's algorithm [80]). Then, for any cut $(S, \bar{S})$ in the graph, $\frac{E(S,\bar{S})}{|S|} \geq \frac{C}{n}$, so $\alpha(G) \geq \frac{C}{n}$. On the other hand, the expansion of the min-cut is at most C, so $\alpha(G) \leq C$. Thus, we can take the range of $\alpha(G)$ to be $[\frac{C}{n}, C]$, so that the $O(\sqrt{\log n})$ approximation algorithm for Sparsest Cut takes $\tilde{O}(n^2)$ time overall.

Similarly, for c-Balanced Separator, by removing minimum cuts recursively as long as the total size of the removed subgraph is at most $c'n$, we can obtain a factor $O(n)$ approximation to $\alpha_{c'}(G)$, as shown by Leighton and Rao [74]. Since we may have to aggregate $O(n)$ minimum cuts, the total amount of time needed to obtain the $O(n)$ approximation is $O(mn) = \tilde{O}(n^2)$. Thus, the $O(\sqrt{\log n})$ pseudo-approximation algorithm for the c-Balanced Separator takes $\tilde{O}(n^2)$ time as well.

7.3 Implementing Oracle

In this section we prove Lemma 42, restated here for convenience:

Lemma 42. *There is a randomized procedure,* Oracle*, which given a set of demands* $\mathbf{d}$*, runs in* $\tilde{O}(n^2)$ *time and*

1. *either produces an* $\mathbf{x} \in \mathcal{Q}$ *with* $v(\mathbf{d}, \mathbf{x}) \geq 0$ *in which exactly one set* $S \subseteq V$ *with at least* cn *vertices has a non-zero* z_S*,*
2. *or else,* Oracle *fails. In this case, a* D*-regular* (c, β_2)*-pseudo-expander flow can be constructed from* $\mathbf{d}$*, for some constant* β_2 *which depends only on* β*.*

In each of the following cases, Oracle attempts to exploit certain characteristics of the demands to find a suitable $\mathbf{x}$, failing which, execution falls through to the next case. To facilitate the search, Oracle may temporarily neglect part of the demands, however, the final $\mathbf{x}$ it finds gives non-negative payoff even with the original demands. Recall that for all $i \in V$, $d_i = \sum_j d_{ij}$ and for all $S \subseteq V$, $d(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} d_{ij}$. Also, let $\varepsilon_1, \varepsilon_2$ be constants to be chosen later, and set $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\}$ in (7.7) and (7.8).

Case 1: Many large degrees

Sort the vertices in decreasing order by d_i.
Case 1a: If the largest $\varepsilon_1 \beta n$ degrees account for more than $\varepsilon_1 nD$ demand, then we can find an $\mathbf{x} \in \mathcal{Q}$ with non-negative payoff by setting $s_i = 1/\varepsilon_1$ for all these vertices. Set $z_S = 2$ for any S with $n/2$ vertices. All other variables are 0. Since $d(S, \bar{S}) \leq \frac{1}{2}nd$, we have

$$v(\mathbf{d}, \mathbf{x}) \;\geq\; \varepsilon_1 nD \cdot (1/\varepsilon_1) - d(S, \bar{S}) \cdot 2 \;\geq\; nD - nD \;\geq\; 0.$$

Case 1b: Otherwise, Oracle modifies the demand graph. Vertices with the top $\varepsilon_1 \beta n$ degrees have their demands set to 0. The remaining degrees must be at most $\frac{D}{\beta}$: otherwise, the removed demands summed up to at least $\varepsilon_1 \beta n \cdot \frac{D}{\beta} = \varepsilon_1 nD$, which we assumed is not the case. The total demand discarded is at most $\varepsilon_1 nD$. Execution falls through to the next case.

Case 2: A large non-expanding cut

First, Oracle applies the Benczúr-Karger reduction to the (modified) demand graph to reduce it to a set of $O(n \log n)$ non-zero demands such that *all* cuts (in particular, degrees too) are approximately preserved. Let G_d be the demand graph obtained this way. Oracle runs the procedure FindLargeCut($G_d, \frac{D}{\beta}, \frac{c}{2}, \frac{\beta^2}{2}$) (see Lemma 48 in Section 7.5) This runs in $\tilde{O}(n^2)$ time since there are only $O(n \log n)$ non-zero demands in G_d.
Case 2a: Suppose it gives $\frac{c}{2}$-balanced cut S with expansion at most $\frac{\beta^2}{2} \cdot \frac{D}{\beta} = \frac{\beta}{2}D$. The demand discarded in Case 1 is at most $\varepsilon_1 nD \leq \frac{2\varepsilon_1}{c}|S|D = \frac{\beta}{2}D|S|$, if we set $\varepsilon_1 = \frac{\beta c}{4}$. Even including this discarded demand we have $d(S, \bar{S}) \leq \beta D|S|$. Oracle constructs an $\mathbf{x} \in \mathcal{Q}$ with non-negative payoff by setting $z_S = n/|S|$, all $s_i = \beta$, all other variables are 0. The payoff is

$$v(\mathbf{d}, \mathbf{x}) \;\geq\; nD \cdot \beta - d(S, \bar{S}) \cdot (n/|S|) \;\geq\; \beta nD - \beta nD \;\geq\; 0.$$

Case 2b: Otherwise, FindLargeCut returns a graph on at least $(1 - \frac{c}{2})n$ nodes such that the induced demand graph has expansion least $\frac{\beta^4}{32} \cdot \frac{D}{\beta} = \frac{\beta^3}{32}D$. The demand on the nodes left out is discarded. On the entire graph, all c-balanced cuts still expand by at least $\beta_1 D$ for $\beta_1 = \frac{\beta^3}{64}$. Execution falls through to the next case.

Case 3: Unroutable demands

First, Oracle performs random sampling on the demands so that the number of nonzero demands is $\tilde{O}(n)$. In Section 7.7, we prove the following lemma via simple applications of the Chernoff-Hoeffding bounds:

Lemma 44. *We can randomly sample the demands to produce new demands, $\tilde{d}_{ij}$, of which at most $O(n \log^2 n)$ are non-zero, so that for any $\delta > 0$, with probability at least $1 - n^{-\Omega(\log n)}$, we have:*

$$\begin{aligned} \forall i : \quad & \tilde{d}_i \;\leq\; d_i + D \\ \forall S,\ n/2 \geq |S| \geq cn : \quad & \tilde{d}(S, \bar{S}) \;\geq\; (1 - \delta) d(S, \bar{S}) \\ \forall \mathbf{x} \in \mathcal{Q} : \quad & \sum_{ij} d_{ij}\ell_{ij}(\mathbf{x}) \;<\; nD \implies \sum_{ij} \tilde{d}_{ij}\ell_{ij}(\mathbf{x}) \;<\; 7nD \end{aligned}$$

Now, Oracle sets all $s_i = 0$, and since $\ell_{ij}(\mathbf{x})$'s are truncated, the optimum choice of w_e's corresponds to solving the following LP (here, we abuse notation a bit and use ℓ_{ij} to refer to the variables with the value $\ell_{ij}(\mathbf{x})$):

$$\begin{aligned} \text{Maximize} \quad & \sum_{ij} \tilde{d}_{ij}\ell_{ij} \quad \text{subject to} \\ \forall ij, \forall p \in \mathcal{P}_{ij} : \quad & \ell_{ij} \;\leq\; \sum_{e \in p} w_e \\ \forall ij : \quad & \ell_{ij} \;\leq\; 1/\varepsilon_2 \\ & \sum_e c_e w_e \;\leq\; \beta nD \qquad (7.12) \end{aligned}$$

We show how to approximately solve the above LP by considering the dual. This can be thought of as a min-cost max-concurrent flow problem, which can be solved in sparse graphs in $\tilde{O}(n^2)$ time using the algorithm of Fleischer [40]. Consider the following instance of a min-cost max-concurrent flow problem: for every pair $\{i, j\}$ we associate demand $\tilde{d}_{ij}$. We also associate a pseudo-edge between every pair $\{i, j\}$ with infinite capacity and cost $b = 1/(\beta\varepsilon_2 nD)$. Any real edge e has cost 0 and its original capacity c_e. We impose the budget constraint 1 on the total cost of the flow. We get the following LP and its dual:

$$\begin{array}{rlrl} & \max t & & \min \sum_e c_e w'_e + \phi' \\ \forall ij : & \sum_{p \in \mathcal{P}_{ij}} y'_p + t'_{ij} \geq \tilde{d}_{ij} t & \forall ij : & l'_{ij} \leq b\phi' \\ \forall e : & \sum_{p \ni e} y'_p \leq c_e & \forall ij, \forall p \in \mathcal{P}_{ij} : & l'_{ij} \leq \sum_{e \in p} w'_e \\ & b \sum_{ij} t'_{ij} \leq 1 & & \sum_{ij} \tilde{d}_{ij} l'_{ij} \geq 1 \end{array} \tag{7.13}$$

ORACLE solves this LP using Fleischer's algorithm to within a constant multiplicative factor, say 2. The algorithm runs in $\tilde{O}(n^2)$ time since there are only $O(n \log^2 n)$ non-zero demands.

The algorithm also yields the weights w_e such that $2t \geq \sum_e c_e w'_e + \phi'$. We also get a flow $\mathbf{g} = \langle g_p \rangle_p$ with congestion C by setting $C = 1/t$ and scaling all g'_p and t'_{ij} by C to get g_p and t_{ij}. This routes all but $\sum_{ij} t_{ij}$ of the demands with congestion C.

Next, we get a feasible solution w_e and ℓ_{ij} for LP (7.12): let $k = \beta nD / (\sum_e c_e w'_e + \phi') \geq \beta nD \cdot C/2$, and scale up the w'_e, l'_{ij}, ϕ' by k to get w_e, ℓ_{ij}, ϕ. Since $\sum_e c_e w_e + \phi = \beta nD$, $\sum_e c_e w_e \leq \beta nD$ and $\phi \leq \beta nD$; so $b\phi \leq 1/\varepsilon_2$ as needed. Also, $\sum_{ij} \tilde{d}_{ij} \ell_{ij} = \sum_{ij} \tilde{d}_{ij} \ell'_{ij} \cdot k \geq \beta nDC/2$.

Case 3a: If $C > 14/\beta$ then $\sum_{ij} \tilde{d}_{ij} \ell_{ij} > 7nD$, so $\sum_{ij} d_{ij} \ell_{ij} \geq nD$. Then ORACLE constructs an $\mathbf{x} \in \mathcal{Q}$ by using the given settings for w_e and ℓ_{ij} and assigning $z_S = 2$ for some S with $n/2$ vertices. Other variables are all 0. Then

$$v(\mathbf{d}, \mathbf{x}) \;\geq\; nD - d(S, \bar{S}) \cdot 2 \;\geq\; nD - nD \;\geq\; 0$$

Case 3b: Otherwise $C \leq 14/\beta$, then the ORACLE fails. We then get a pseudo-expander flow as explained below.

7.3.1 Finding a Pseudo-Expander flow

The flow $\mathbf{g}$ obtained by ORACLE just before it failed routes all but $\sum_{ij} t_{ij}$ of the total demand with congestion at most $14/\beta$. We discard the t_{ij} demands, which amount to at most $\sum_{ij} t'_{ij}/t \leq (1/b) \cdot C \leq 14\varepsilon_2 nD$. The remaining demands are $\frac{D}{\beta} + D$ regular. If we choose $\delta = \frac{1}{2}$ in Lemma 44, then after random sampling, all c-balanced cuts have expansion at least $\frac{\beta_1}{2} D$. By setting $\varepsilon_2 = \frac{\beta_1 c}{56}$, the total demand discarded is at most $\frac{\beta_1 c}{4} nD$. Thus, all c-balanced cuts still have expansion at least $\frac{\beta_1}{4} D$. We then scale the flow by $\frac{\beta}{14}$ so that the congestion becomes 1, all degrees are at most D, and all c-balanced cuts have expansion at least $\beta_2 D$ for $\beta_2 = \frac{\beta_1 \beta}{56}$. Thus, we end up with a D-regular (c, β_2)-pseudo-expander flow.

7.4 Finding a cut of expansion within $O(\sqrt{\log n})$ of optimal

Since the width of the ORACLE is $O(1)$, Theorem 6 shows that if the ORACLE does not fail for $T = O(\frac{\log n}{\delta^2})$ rounds, the Multiplicative Weights algorithm finds an $\mathbf{x} \in \mathcal{Q} \subseteq \mathcal{Q}'$ which satisfies (7.9) (and thus, (7.6)) up to an additive error of δ, where δ is a small constant to be chosen later. In this section, we prove Lemma 41, which shows that in this case we can find a cut of expansion $O(\sqrt{\log n} \cdot D)$. We restate Lemma 41 here for convenience:

Lemma 41. *There is a constant $\delta > 0$ such that if $\mathbf{x} \in \mathcal{Q}'$ satisfies the feasibility problem (7.6) up to additive error δ, then there is an algorithm that uses $\mathbf{x}$ to find a c'-balanced cut of expansion $O(\sqrt{\log n} \cdot D)$, for some $c' \leq c$.*

Let $\mathbf{x}^* = \langle \mathbf{s}^*, \mathbf{w}^*, \mathbf{z}^* \rangle \in \mathcal{Q}'$ satisfy the feasibility problem (7.6) up to additive error δ. Since the Multiplicative Weights algorithm needs only $O(\log n)$ rounds to find such an $\mathbf{x}^*$, and only one z_S is non-zero in any round, we conclude that $\mathbf{x}^*$ has only $O(\log n)$ non-zero z_S^*'s. We now construct an ℓ_2^2 embedding of the vertices $i \in V$ using $\mathbf{x}^*$:

Lemma 45. *We can construct a unit vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ such that $\frac{1}{4}\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 = c(1-c)n^2$, and for all pairs i, j, $\frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 = \sum_{S\colon\, i \in S, j \in \bar{S}} \frac{|S|}{n} z_S^*$*

PROOF: First we note that $\sum_S \frac{|S|}{n} z_S = 1$ for any $\mathbf{x} \in \mathcal{Q}$. There are $N = O(\log n)$ sets S with non-zero z_S^*. We construct vectors in $\mathbb{R}^N$, with a coordinate for each such set S. For any vertex i, construct vector $\mathbf{v}_i$ by setting $\mathbf{v}_i(S) = \pm\sqrt{\frac{|S|}{n} z_S^*}$ depending on whether $i \in S$ or $i \in \bar{S}$. Note that $\|\mathbf{v}_i\|^2 = \sum_S \frac{|S|}{n} z_S = 1$. Also, for any pair i, j, the vector $\mathbf{v}_i - \mathbf{v}_j$ has non-zero coordinates only for S such that $i \in S, j \in \bar{S}$. Thus, $\frac{1}{4}\|\mathbf{v}_i - \mathbf{v}_j\|^2 = \sum_{S\colon\, i \in S, j \in \bar{S}} \frac{|S|}{n} z_S^*$. So,

$$\frac{1}{4}\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 = \sum_{ij} \sum_{S\colon\, i \in S, j \in \bar{S}} \frac{|S|}{n} z_S^* = \sum_S \frac{|S|}{n} z_S^* \cdot \left[\sum_{ij\colon\, i \in S, j \in \bar{S}} 1 \right] = \sum_S \frac{|S|}{n} z_S^* \cdot |S||\bar{S}|.$$

Since $z_S^* \neq 0$ only if the cut $(S, \bar{S})$ is c-balanced, we have $|S||\bar{S}| \geq c(1-c)n^2$ for all such S, and hence

$$\frac{1}{4}\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq \sum_S \frac{|S|}{n} z_S^* \cdot c(1-c)n^2 = c(1-c)n^2.$$

□

Now, we appeal to Theorem 19, restated here for convenience:

Theorem 19. *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ be vectors of length at most 1, such that $\sum_{ij} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq an^2$. Let w_e be weights on edges and nodes and let $\alpha = \sum_e c_e w_e$. Then there is an algorithm which runs in $\tilde{O}(m^{1.5})$ time and finds a cut of value C which is c-balanced for some constant c, such that there exists a pair of nodes i, j with the property that the graph distance between i and j is at most $O(\sqrt{\log n} \cdot \frac{\alpha}{C})$ and $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$ where s is a constant which only depends on a. Furthermore, this is true even if any fixed set of τn nodes are prohibited from being i or j, for some small constant τ.*

Now we can complete the proof of Lemma 41.
Proof:[Lemma 41]
Since $\mathbf{x}^* \in \mathcal{Q}'$ satisfy the feasibility problem (7.6) up to additive error δ, we have

$$s_i^* + s_j^* + \ell_{ij}(\mathbf{x}^*) - \sum_{S:\ i \in S, j \in \bar{S}} z_S^* \ \geq\ -\delta$$
$$\therefore\ s_i^* + s_j^* + l_{ij}(\mathbf{x}^*) \ \geq\ \sum_{S:\ i \in S, j \in \bar{S}} \frac{|S|}{n} z_S^* - \delta,$$

since $\ell_{ij}(\mathbf{x}^*) \leq l_{ij}(\mathbf{x}^*) = \min_{p \in \mathcal{P}_{ij}} \left\{ \sum_{e \in p} w_e^* \right\}$.

Construct the unit vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ of Lemma 45. Then for all pairs $\{i, j\}$, we have

$$s_i^* + s_j^* + l_{ij}(\mathbf{x}^*) \ \geq\ \frac{1}{4} \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \delta.$$

Let c', s, τ be the constants given by Theorem 19 for $a = 4c(1-c)$. Note that $\alpha = \sum_e c_e w_e^* \leq \beta n D$.

Since $\sum_i s_i^* \leq \beta n$, at most τn nodes have $s_i^* > \beta/\tau$. Let all such vertices form the set A. We apply Theorem 19 to G with A being the set of τn forbidden vertices. We thus get a cut of value C such that there is a pair of vertices i, j with the following properties:

1. $s_i^*, s_j^* \leq \beta/\tau$,
2. the graph distance of i, j in G is at most $O(\sqrt{\log n} \cdot \alpha/C) = O(\sqrt{\log n} \cdot nD/C)$, and
3. $\|\mathbf{v}_i - \mathbf{v}_j\|^2 \geq s$.

Further, we choose $\beta = \frac{s\tau}{32}$ and $\delta = \frac{s}{16}$ so that

$$l_{ij}(\mathbf{x}^*) \ \geq\ \frac{1}{4} \|\mathbf{v}_i - \mathbf{v}_j\|^2 - s_i^* - s_j^* - \delta \ \geq\ \frac{s}{4} - \frac{2\beta}{\tau} - \delta \ =\ \frac{s}{8}.$$

Hence, we have $O(\sqrt{\log n} \cdot nD/C) \geq \frac{s}{8}$. Thus $\frac{C}{c'n} \leq O(\sqrt{\log n} \cdot D)$. This implies that the expansion of the cut found is at most $O(\sqrt{\log n} \cdot D)$, as required. Since we have only $O(n \log n)$ edges in the graph, this procedure runs in $\tilde{O}(n^{1.5})$ time. □

7.5 Sparse cuts via eigenvalue computations

For a weighted graph G where the weight for pair $\{i,j\}$ (j could possibly the same as i, to allow for self-loops) is d_{ij}, the Laplacian $\mathcal{L}$ of G is the $n \times n$ symmetric matrix with rows and columns indexed by nodes in G, such that $\mathcal{L} = D^{-1/2}(D-A)D^{-1/2}$, where D is the diagonal matrix of (weighted) node degrees, and A is the (weighted) adjacency matrix of the graph G [2] Note that this definition of the Laplacian is different from the combinatorial Laplacian defined in Chapter 4. However, the two are closely related in regular graphs. The smallest eigenvalue of $\mathcal{L}$ is 0 corresponding to the eigenvector $\langle \sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_n} \rangle^\top$, where d_i is the (weighted) degree of node i. The following well-known theorem that arises from the work of Alon and Cheeger (for a proof see [36]) shows that the second smallest eigenvalue of $\mathcal{L}$ gives us useful information about the conductance of G:

Theorem 40 (Alon-Cheeger)**.** *Let the conductance of a weighted graph G with n vertices and m edges be $c(G)$. Let the Laplacian of the graph be $\mathcal{L}$, and denote its second smallest eigenvalue by $\lambda_{\mathcal{L}}$. Then:*

$$2c(G) \geq \lambda_{\mathcal{L}} \geq \frac{c(G)^2}{2}$$

Furthermore, suppose we are given a vector $\mathbf{x}$ orthogonal to the $\langle \sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_n} \rangle^\top$ eigenvector, i.e. $\sum_i \sqrt{d_i} x_i = 0$. Let $\lambda := \frac{\mathbf{x}^\top \mathcal{L} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}$. Then there is a procedure SweepCut$(G, \mathbf{x})$*, that in time $\tilde{O}(m+n)$ finds a cut with conductance at most $\sqrt{2\lambda}$.*

The procedure SweepCut$(G, \mathbf{x})$ operates as follows. Assume that the coordinates of $\mathbf{x}$ are ordered in increasing value, $x_1 \leq x_2 \leq \ldots x_n$. For $1 \leq k \leq n-1$, let $S_k = \{1, 2, \ldots, k\}$. Then the theorem shows that one of the cuts $(S_k, \bar{S}_k)$ has conductance at most $\sqrt{2\lambda}$, and thus can be found in time $\tilde{O}(m+n)$.

The optimal $\mathbf{x}$ for this procedure is an eigenvector belonging to the second smallest eigenvalue of $\mathcal{L}$. Computing this eigenvector may be computationally expensive if $\lambda_{\mathcal{L}}$ is very close to 0. However, for our application, we only need to find a cut of conductance less than some pre-specified constant $\beta > 0$. In this case, it suffices to find a vector $\mathbf{x}$ such that its λ value is at most $\frac{\beta^2}{2}$. For this, we can use the power method, as explained in the following lemma.

Lemma 46. *Let $\lambda > \lambda_{\mathcal{L}}$ be a given parameter. Then we can find a vector $\mathbf{x}$ such that $\sum_i \sqrt{d_i} x_i = 0$ and $\frac{\mathbf{x}^\top \mathcal{L} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}} \leq \lambda$ using $O(\frac{\log n}{\lambda - \lambda_L})$ matrix vector products with the matrix $\mathcal{L}$.*

Proof: An eigenvector of $\mathcal{L}$ belonging to the 0 eigenvalue is $\mathbf{y} = \langle \sqrt{d_1}, \sqrt{d_2}, \ldots, \sqrt{d_n} \rangle^\top$. Furthermore, the largest eigenvalue of $\mathcal{L}$ is at most 2. Thus, the matrix $2I - \mathcal{L} - \frac{1}{\mathbf{y}^\top \mathbf{y}} \mathbf{y}\mathbf{y}^\top$ is positive semidefinite with largest eigenvalue $2 - \lambda_{\mathcal{L}}$. To get a vector $\mathbf{x}$ with Rayleigh quotient at least $2 - \lambda$, we can use the power method with a random start. The analysis of [71] indicates that the method succeeds with constant probability in $O(\frac{\log n}{\lambda - \lambda_L})$ iterations. $\square$

[2] We assume without loss of generality that no node has zero degree.

Lemma 47. *There is a randomized procedure,* FindCut(G, λ)*, which finds a cut of conductance at most* $\sqrt{2\lambda}$ *in* G*, or with constant probability, concludes correctly that* G *has conductance at least* $\frac{\lambda}{4}$*. The procedure requires* $O(\frac{\log n}{\lambda})$ *matrix vector products with the Laplacian of* G*.*

Proof: Let $\lambda_{\mathcal{L}}$ be the second smallest eigenvalue of the Laplacian of G. If $\lambda_{\mathcal{L}} \leq \frac{1}{2}\lambda$, then with constant probability, $O(\frac{\log n}{\lambda})$ iterations of the power method, as described in Lemma 46, suffice to find the desired $\mathbf{x}$. Once $\mathbf{x}$ is found, we can run SweepCut$(G, \mathbf{x})$ to find a cut of conductance at most $\sqrt{2\lambda}$. Otherwise if $\lambda_{\mathcal{L}} \geq \frac{1}{2}\lambda$, then by Theorem 40, G has conductance at least $\frac{1}{2}\lambda$. □

One can iterate the FindCut procedure to find c-balanced separators:

Lemma 48. *There is a procedure that, given a weighted graph* G *with degrees bounded by* D*, a fraction* $c > 0$*, and a expansion bound* $\beta > 0$*, uses* FindCut $O(n)$ *times and produces either a cut of size* cn *with edge expansion less than* βD*, or a graph on at least* $(1-c)n$ *vertices on which every cut has expansion at least* $\frac{\beta^2}{8}D$*.*

Proof: For a graph G, let G_D be the graph G with each node augmented with (weighted) self-loops to make the weighted degree exactly D. This ensures that a cut of conductance ϕ in G_D has expansion at least ϕD. We repeatedly use FindCut to get cuts of expansion less than βD and aggregate them. The resulting set also has expansion less than βD. When FindCut can no longer find cuts of expansion less than βD, then with constant probability, the second smallest eigenvalue of the Laplacian of the remaining (augmented) graph is at least $\frac{\beta^2}{4}$. Thus, by Theorem 40, every cut in the remaining graph has expansion $\frac{\beta^2}{8}D$. This procedure, FindLargeCut, is given in Figure 7.1. The success probability can be boosted up using standard repetition techniques. Here, $G \setminus S$ is the subgraph induced on the vertex set $V \setminus S$.

```
Procedure FindLargeCut($G, D, c, \beta$)
// Returns a cut of expansion of expansion at most $\beta D$ or
// a subgraph of G of size at least $(1-c)n$ with expansion at least $\frac{\beta^2}{8}D$
Initialization: $S \leftarrow \phi$
while FindCut($(G \setminus S)_D$, $\frac{\beta^2}{2}$) finds a cut $(T, \bar{T})$ of conductance at most $\beta$
    $S \leftarrow S \cup T$
end while
if $|S| \geq cn$ then return the set $S$
else return the graph $(G \setminus S)$
```

Figure 7.1: The FindLargeCut procedure.

□

7.6 Pseudo-Expander Flows

In this section we show how, given a pseudo-expander flow, we can either convert it into an expander flow or find a sparse cut. We recall the definition of pseudo-expander flows.

Definition 6. *The graph with edge weights $\langle d_{ij}\rangle_{ij}$ is a D-regular (c, β)-pseudo expander if it has maximum degree at most D and for any subset $S \subseteq V$ such that $cn \leq |S| \leq n/2$ the total weight crossing the cut $(S, \bar{S})$, viz. $d(S, \bar{S}) = \sum_{i\in S, j\in\bar{S}} d_{ij}$, satisfies*

$$d(S, \bar{S}) \ \geq\ \beta D|S|.$$

A multi-commodity flow $\mathbf{f}$ is called a D-regular (c, β)-pseudo expander flow (or just pseudo-expander flow for short) if its demand graph is a D-regular (c, β)-pseudo expander.

We now turn to the proof of Lemma 43, which we restate here for convenience:

Lemma 43. *Let $\mathbf{f} = \langle f_p\rangle_p$ be a D-regular (c, β)-pseudo-expander flow on a graph G. Assume that the flow has non-zero demand on only $O(n \log n)$ pairs of vertices. Then, there is a procedure that in time $\tilde{O}(m^{1.5})$, finds either*

1. *a D-regular $\frac{\beta^2}{130}$ expander flow,*
2. *or, a cut of expansion at most $\frac{1}{c}D$.*

Proof: Let G_f be the demand graph for the given pseudo-expander flow. Let $\mathcal{D}$ be its Laplacian, and let $\lambda_{\mathcal{D}}$ be the second smallest eigenvalue of $\mathcal{D}$.

First, we run FindLargeCut$(G_f, D, c, \frac{\beta}{2})$. Since G_f is the demand graph of a (c, β)-pseudo-expander flow, the procedure cannot return a cut $(S, \bar{S})$ in G_f that is c balanced with expansion less than $\frac{\beta}{2}D$. Thus, it returns a subset of vertices S of size at most cn such that the induced subgraph $G_f \setminus S$ has expansion at least $\frac{\beta^2}{32}$.

Now, let L be the set S and $cn - |S|$ arbitrarily removed nodes of $\bar{S}$, and let $R = V \setminus L$. Note that $|L| = cn$ and $R = (1 - c)n$. Let $k = \frac{|R|}{|L|} \leq \frac{1}{c}$. We form a flow network by connecting all nodes in L to a single (artificial) source with edges of capacity kD and all nodes in R to a single (artificial) sink with edges of capacity D, and compute the max flow in the network (with all original graph edges in G retaining their capacities). The flow computation runs in $\tilde{O}(m^{1.5})$ time using the algorithm of Goldberg and Rao [47].

Suppose the flow does not saturate all source edges. Then its value is less than $kD|L|$. Let $(T, \bar{T})$ be the min-cut found, with T being the side of the cut containing the source. Let $n_\ell = |T \cap L|$ and let $n_r = |\bar{T} \cap R|$. Then the capacity of the original graph edges cut is at most $kD|L| - kD(|L| - n_\ell) - D(|R| - n_r) = D[kn_\ell + n_r - |R|]$. The smaller side of the cut contains at least $\min\{n_\ell, n_r\}$ nodes, and since $kn_\ell \leq k|L| = |R|$ and $n_r \leq |R|$, the expansion of the cut found is at most $kD \leq \frac{1}{c}D$.

Otherwise, suppose that the flow does saturate all source edges. Let $\mathbf{g}$ be this flow. Consider the flow $\mathbf{h} = \frac{1}{2}\mathbf{f} + \frac{1}{2k}\mathbf{g}$. Then $\mathbf{h}$ is a D-regular flow. We claim that it is an $\Omega(c^2)$ expander flow.

Let $(T, \bar{T})$ be a cut in the graph, with $|T| \leq |\bar{T}|$. Let $x = |T \cap L|$ and $y = |T \cap R|$. Now we have two cases:

1. $x \geq \frac{\beta^2}{64}y$:
   Since **g** pumps kD flow into every node in $T \cap L$, which eventually goes into the sink, at least kDx flow must cross the cut $(T, \bar{T})$. Thus, in the flow **h**, at least $\frac{1}{2}Dx$ flow crosses the cut $(T, \bar{T})$. Thus, the expansion of the cut $(T, \bar{T})$ is at least
   $$\frac{\frac{1}{2}Dx}{x+y} \;\geq\; \frac{\frac{1}{2}Dx}{x+\frac{64}{\beta^2}x} \;\geq\; \frac{\beta^2}{130}D$$
   since $\beta \leq 1$.

2. $x \leq \frac{\beta^2}{64}y$:
   Since $T \cap R \subseteq \bar{S}$, and since the subgraph of G_f induced by $\bar{S}$ has expansion at least $\frac{\beta^2}{32}D$, at least $\frac{\beta^2}{32}Dy$ flow in **f** must leave the set $T \cap R$. Since **f** is D-regular, at most Dx of this flow can terminate in nodes in $T \cap L$. Thus, since $x \leq \frac{\beta^2}{64}y$, at least $\frac{\beta^2}{64}Dy$ flow crosses the cut $(T, \bar{T})$. So in **h**, at least $\frac{\beta^2}{128}Dy$ flow crosses the cut $(T, \bar{T})$. Thus, the expansion of the cut $(T, \bar{T})$ is at least
   $$\frac{\frac{\beta^2}{128}Dy}{x+y} \;\geq\; \frac{\frac{\beta^2}{128}Dy}{y+\frac{\beta^2}{64}y} \;\geq\; \frac{\beta^2}{130}D$$
   since $\beta \leq 1$.

□

7.7 Random sampling on the demands

We now describe how to randomly sample the demands to reduce the number of non-zero demands to $O(n \log^2 n)$ while still preserving degrees, expansion of large cuts, and the values of $\ell_{ij}(x)$'s.

Recall that we only perform the random sampling if the steps corresponding to choosing the s_i's and the z_S's do not result in an $\mathbf{x}$ that has positive payoff. Therefore:

$$\begin{aligned} \forall i: \quad d_i \;&\leq \frac{D}{\beta} \\ \forall S,\ n/2 \geq |S| \geq cn: \quad d(S,\bar{S}) \;&\geq\; \beta_1 D|S| \;\geq\; \beta_1 cnD \end{aligned}$$

For random sampling, we choose the probability distribution $\mathcal{D}$ over pairs of nodes, where the probability of $\{i,j\}$ is $p_{ij} = d_{ij}/Z$ where $Z = \sum_{k\ell} d_{k\ell} \leq \frac{1}{2}nD$. Now we form the multiset S by choosing m independent samples $\{i,j\}$ from $\mathcal{D}$. Thus, in each of the m rounds, we choose only 1 pair, for a total of m pairs. Set indicator random variable $X_{ij}^s = 1$ or 0 depending on whether we choose $\{i,j\}$ on the s^{th} trial, $1 \leq s \leq m$. Finally, set the new sampled demands to be

$$\tilde{d}_{ij} \;=\; \frac{Z\sum_s X_{ij}^s}{|S|}$$

We use the following version of the Chernoff-Hoeffding bounds from [82]:

Lemma 49. *Let $X = \sum X_s$ be the sum of m independent identically distributed random variables in $[0,1]$ such that $\mathbb{E}[X_s] = \mu$. Let $\delta > 0$ be any small error parameter , and $b(\delta) = (1+\delta)\ln(1+\delta) - \delta$. Then*

$$\mathbf{Pr}[X/m \ \geq \ (1+\delta)\mu] \ < \ e^{-mb(\delta)\mu}.$$

If $\delta > 2e-1$, the upper bound above can be replaced by $2^{-m(1+\delta)\mu}$. For $\delta < 1$, we have

$$\mathbf{Pr}[X/m \ \leq \ (1-\delta)\mu] \ < \ e^{-m\delta^2\mu/2}$$

Now we show the sampled demands approximate the original ones well with high probability, if the number of samples is $\Omega(n\log^2 n)$. We restate Lemma 44 here for convenience:

Lemma 44. *We can randomly sample the demands to produce new demands, $\tilde{d}_{ij}$, of which at most $O(n\log^2 n)$ are non-zero, so that for any $\delta > 0$, with probability at least $1 - n^{-\Omega(\log n)}$, we have:*

$$\begin{aligned} \forall i: \quad & \tilde{d}_i \ \leq \ d_i + D \\ \forall S,\ n/2 \geq |S| \geq cn: \quad & \tilde{d}(S,\bar{S}) \ \geq \ (1-\delta)d(S,\bar{S}) \\ \forall \mathbf{x} \in \mathcal{Q}: \quad & \sum_{ij} d_{ij}\ell_{ij}(\mathbf{x}) \ < \ nD \implies \sum_{ij} \tilde{d}_{ij}\ell_{ij}(\mathbf{x}) \ < \ 7nD \end{aligned}$$

Proof: Let $m = \Omega(n\log^2 n)$ be the number of samples.

1. Fix any i. Define, for all $1 \leq s \leq m$, $X_s = \sum_j X^s_{ij}$. All $X_s \in \{0,1\}$ and have expectation d_i/Z. Define $X = \sum_s X_s$. Then $X/m = \tilde{d}_i/Z$. Set $\delta = \frac{D}{d_i}$. Now we have two cases:

   (a) $\delta \leq 2e-1$:
   Then $d_i \geq \frac{D}{2e-1} \geq D/6$, and hence $d_i/Z \geq 1/3n$. By Lemma 49, we conclude that

   $$\mathbf{Pr}[\tilde{d}_i \geq d_i + D] \ = \ \mathbf{Pr}[X/m \ \geq \ (1+\delta)(d_i/Z)] \ < \ e^{-mb(\delta)d_i/Z} \ \leq \ e^{-mb(\beta)/3n}$$

   since $\delta = \frac{D}{d_i} \geq \beta$, and so $b(\delta) \geq b(\beta)$.

   (b) $\delta \geq 2e-1$:
   Then $(1+\delta)d_i/Z > D/Z \geq 2/n$. By Lemma 49, we conclude that

   $$\mathbf{Pr}[\tilde{d}_i \geq d_i + D] \ = \ \mathbf{Pr}[X/m \ \geq \ (1+\delta)(d_i/Z)] \ < \ 2^{-m(1+\delta)d_i/Z} \ < \ 2^{-2m/n}.$$

   If $m = \Omega(n\log^2 n)$, then any such event happens with probability $< n^{-\Omega(\log n)}$. By the union bound over all n nodes,

   $$\mathbf{Pr}[\exists i: \ \tilde{d}_i \ \geq \ d_i + D] \ < \ n^{-\Omega(\log n)}.$$

2. Fix any S. Define, for all $1 \le s \le m$, $X_s = \sum_{i \in S, j \in \bar{S}} X^s_{ij}$. All $X_s \in \{0,1\}$ and have expectation $d(S,\bar{S})/Z$. Define $X = \sum_s X_s$. Then $X/m = \tilde{d}(S,\bar{S})/Z \ge 2\beta_1 c$. By Lemma 49 above, we conclude that

$$\mathbf{Pr}[\tilde{d}(S,\bar{S}) \ge (1-\delta)d(S,\bar{S})] \;<\; e^{-m\delta^2 d(S,\bar{S})/2Z} \;\le\; e^{-\delta^2\beta_1 cm}.$$

If $m = \Omega(n \log^2 n)$, then any such event happens with probability $< e^{-\Omega(n\log^2 n)}$. By the union bound over at most e^n choices of S,

$$\mathbf{Pr}[\exists S,\ |S| > n/5 :\ \tilde{d}(S,\bar{S}) \;\ge\; (1-\delta)d(S,\bar{S})] \;<\; e^{-\Omega(n\log^2 n)}$$

3. Fix an $\mathbf{x} \in \mathcal{Q}$. Let w_e be the weight function on edges specified by $\mathbf{x}$. Note that since we truncate all path lengths to be at most $1/\varepsilon_2$, we may assume that all $w_e \le 1/\varepsilon_2$. We discretize the space of all possible weight functions on edges as follows. Let the number of edges be $N = O(n \log n)$. We round the w_e values downwards to the closest multiple of $1/N$, to obtain the point $\tilde{\mathbf{x}} \in \mathcal{Q}$. The number of possible such discretized weight functions is bounded by $(N/\varepsilon_2)^N = e^{O(n\log^2 n)}$.

   With some abuse of notation, let $\ell_{ij} = \ell_{ij}(\mathbf{x})$ and $\tilde{\ell}_{ij} = \ell_{ij}(\tilde{\mathbf{x}})$. The discretization ensures that $|\ell_{ij} - \tilde{\ell}_{ij}| \le 1$. Since case 1. holds with high probability, we may assume that all $\tilde{d}_i \le d_i + D$. Thus, $|\sum_{ij} \tilde{d}_{ij}\ell_{ij} - \sum_{ij}\tilde{d}_{ij}\tilde{\ell}_{ij}| \le \sum_{ij}\tilde{d}_{ij} \le nD$.

   Define, for all $1 \le s \le m$, $X_s = \varepsilon_2 \sum_{ij} X^s_{ij}\tilde{\ell}_{ij}$. All $X_s \in [0,1]$ (as $\tilde{\ell}_{ij} \le \varepsilon_2^{-1}$) and have expectation $\mu = \varepsilon_2 \sum_{ij} d_{ij}\tilde{\ell}_{ij}/Z$. Define $X = \sum_s X_s$. Then $X/m = \varepsilon_2 \sum_{ij}\tilde{d}_{ij}\tilde{\ell}_{ij}/Z$. Now, if $\sum_{ij} d_{ij}\ell_{ij} < nD$, then $\sum_{ij} d_{ij}\tilde{\ell}_{ij} < nD$, and so $\mu < \varepsilon_2 nD/Z$. Let $1 + \delta = \frac{6\varepsilon_2 nD/Z}{\mu}$. Note that $\delta > 2e - 1$, and $(1+\delta)\mu \ge 12\varepsilon_2$. By Lemma 49 above, we conclude that,

$$\begin{aligned}\mathbf{Pr}[\sum_{ij}\tilde{d}_{ij}\ell_{ij} > 7nD] &\le \mathbf{Pr}[\sum_{ij}\tilde{d}_{ij}\tilde{\ell}_{ij} > 6nD] \\ &\le \mathbf{Pr}[X/m > (1+\delta)\mu] \\ &< 2^{-m(1+\delta)\mu} \\ &< 2^{-12\varepsilon_2 m}.\end{aligned}$$

   Applying the union bound to all the $e^{O(n\log^2 n)}$ possible discretized metrics, we obtain that if $m = \Omega(n\log^2 n)$ then

$$\begin{aligned}\mathbf{Pr}[\exists \mathbf{x} \in \mathcal{Q} :\ \sum_{ij} d_{ij}\ell_{ij} < nD \text{ but } \sum_{ij}\tilde{d}_{ij}\ell_{ij} > 7nD] &< 2^{O(n\log^2 n)} e^{-\Omega(n\log^2 n)} \\ &= e^{-\Omega(n\log^2 n)}.\end{aligned}$$

Finally, the union bound over all three cases implies that the stipulation of the lemma holds with probability at least $1 - n^{-\Omega(\log n)}$. □

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this thesis, we explored the power of the Multiplicative Weights Update rule in various forms: the basic one where real valued weights are updated by multiplying them by appropriate factors, and a matrix version where the "weights" are encapsulated by a density matrix and the update rule uses the matrix exponential.

Both of these algorithms have numerous applications. The basic version, especially, has been known for many years, and consequently has applications in very diverse areas (see the survey [15]). In this thesis, we showed how to apply this algorithm to solve various special semidefinite programs efficiently by reducing the problem to computing approximate eigenvectors. We also showed how to obtain an $O(\sqrt{\log n})$ approximation to the Sparsest Cut and Balanced Separator problems in undirected weighted graphs by embedding expander flows in the input graph using the Multiplicative Weights algorithm.

The bulk of the thesis concerned the matrix generalization, viz. the Matrix Multiplicative Weights algorithm. Using the Matrix Multiplicative Weights algorithm, we were able to provide the first truly general method for designing fast, combinatorial, primal-dual algorithms for semidefinite programming. Thus, we get the fastest known algorithms for approximating the Sparsest Cut and Balanced Separator problems in directed and undirected graphs to factors of $O(\log n)$ and $O(\sqrt{\log n})$. We also obtained the fastest known algorithms for approximating the constraint satisfaction problems Min UnCut and Min 2CNF Deletion to a factor of $O(\sqrt{\log n})$.

We also showed how to apply the Matrix Multiplicative Weights algorithm to derandomize Alon and Roichman's [9] construction of expander graphs, to derandomize the $O(\log n)$ approximation algorithm of Ahlswede and Winter [4], and to obtain an alternative proof of Aaronson's [1] result on the learnability of quantum states by bounding the fat-shattering dimension of the probabilistic concept class of quantum states.

In summary, the Multiplicative Weights framework is a powerful design primitive, and the algorithms it yields usually have particularly simple analyses. It is simple and useful enough that it should be viewed as a basic tool in the hands of an algorithm designer along with divide-and-conquer, dynamic programming, random sampling, and the like.

8.2 Future Work

- **Solving more SDPs efficiently.**
  One direction for future research is approximating other problems based on SDP such as Min Linear Arrangement, for which the currently known algorithms are quite inefficient. We plan to apply the methods developed to SDPs which arise in practice, such as for Air Traffic Flow scheduling and Sparse PCA.

- **Linear time algorithm for approximating** Sparsest Cut.
  While an $\tilde{O}(n^2)$ time algorithm for the Sparsest Cut problem presented in this thesis might seem hard to beat, and is certainly optimal for dense graphs, it is still conceivable that there might be a linear time algorithm, more so if we are willing to settle for a polylogarithmic approximation. Our fastest algorithm takes $\tilde{O}(m+n^{1.5})$ time algorithm (m is the number of edges) for an $O(\log n)$ approximation, and it would be very interesting to obtain an $\tilde{O}(m+n)$ time algorithm. This would involve looking inside the various results such as Benczúr-Karger and Fleischer (or Garg-Könemann) that are used in black-box fashion in the current algorithms.

- **Quantum Computing.**
  The Matrix Multiplicative Weights algorithm generates density matrices, a central notion in quantum mechanics used to model a quantum state. In this thesis we described some quantum computing applications of the algorithm, and certainly there are more applications waiting to be found.

- **Learning algorithms.**
  The Matrix Multiplicative Weights algorithm has been independently discovered by Learning theorists, who applied to online variance minimization [98], online Principal Component Analysis [99, 100], etc. It would be very interesting to obtain more learning algorithms using the Matrix Multiplicative Weights algorithm.

- **"Combinatorializing" Interior Point methods.**
  While the Multiplicative Weights method is certainly quite powerful, its major drawback is that the algorithms it yields have running time dependent on a problem specific parameter, the width, and that its dependence on the error ε is inverse polynomial. Interior point methods do not have this issue. A natural question is whether it is possible to use fast combinatorial subroutines within an interior point algorithm to get a faster algorithm overall. In other words, can interior point algorithms make effective use of the combinatorial structure of a given problem? Such an algorithm will likely be truly practical.

Bibliography

[1] S. Aaronson. The learnability of quantum states. *quant-ph/0608142*, 2006.

[2] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF deletion, and directed cut problems. In *STOC*, pages 573–581, 2005.

[3] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized steiner problem on networks. In *STOC*, pages 134–144, 1991.

[4] R. Ahlswede and A. Winter. Strong converse for identification via quantum channels. *IEEE Transactions on Information Theory*, 48(3):569–579, 2002.

[5] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1):13–51, 1995.

[6] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[7] N. Alon and V. Milman. lambda$_1$, isoperimetric inequalities for graphs, and superconcentrators. *J. Comb. Theory, Ser. B*, 38(1):73–88, 1985.

[8] N. Alon and A. Naor. Approximating the cut-norm via grothendieck's inequality. In *36th STOC*, pages 72–80, 2004.

[9] N. Alon and Y. Roichman. Random cayley graphs and expanders. *Random Struct. Algorithms*, 5(2), 1994.

[10] A. Ambainis, A. Nayak, A. Ta-Shma, and U. V. Vazirani. Dense quantum coding and quantum finite automata. *J. ACM*, 49(4):496–511, 2002.

[11] M. Anthony and P. L. Bartlett. Function learning from interpolation. *Combinatorics, Probability & Computing*, 9(3), 2000.

[12] S. Arora, E. Hazan, and S. Kale. $O(\sqrt{\log n})$ approximation to SPARSEST CUT in $\tilde{O}(n^2)$ time. In *FOCS*, pages 238–247, 2004.

[13] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *FOCS*, pages 339–348, 2005.

[14] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th FOCS*, pages 339–348, 2005.

[15] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. *Manuscript*, 2005. Preliminary draft of paper available online at `http://www.cs.princeton.edu/~satyen/ publications.php`.

[16] S. Arora, E. Hazan, and S. Kale. A fast random sampling algorithm for sparsifying matrices. In *APPROX-RANDOM*, pages 272–279, 2006.

[17] S. Arora, J. R. Lee, and A. Naor. Euclidean distortion and the sparsest cut. In *STOC*, pages 553–562, 2005.

[18] S. Arora, S. Rao, and U. V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC*, pages 222–231, 2004.

[19] Y. Aumann and Y. Rabani. An (log) approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.

[20] P. L. Bartlett and P. M. Long. Prediction, learning, uniform convergence, and scale-sensitive dimensions. *J. Comput. Syst. Sci.*, 56(2):174–190, 1998.

[21] S. Baswana. Dynamic algorithms for graph spanners. In *ESA*, 2006.

[22] A. A. Benczúr and D. R. Karger. Approximating $s-t$ minimum cuts in $\tilde{O}(n^2)$ time. In *STOC*, pages 47–55, 1996.

[23] 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.

[24] M. Blum, R. M. Karp, O. Vornberger, C. H. Papadimitriou, and M. Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Inf. Process. Lett.*, 13(4/5):164–167, 1981.

[25] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.

[26] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel J. Math.*, 52(1–2):46–52, 1985.

[27] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, 2004.

[28] G. W. Brown. *Analysis of Production and Allocation, T. C. Koopmans, ed.*, chapter Iterative solution of games by fictitious play, pages 374–376. Wiley, 1951.

[29] G.W. Brown and J. von Neumann. Solutions of games by differential equations. *Annals of Mathematical Studies*, 24:73–79, 1950.

[30] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *STOC*, pages 1–10, 1999.

[31] M. Charikar and A. Wirth. Maximizing quadratic programs: Extending grothendieck's inequality. In *45th FOCS*, pages 54–60, 2004.

[32] S. Chawla, A. Gupta, and H. Räcke. Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In *SODA*, pages 102–111, 2005.

[33] B. Chazelle. *The Discrepancy Method — Randomness and Complexity*. Cambridge University Press, Cambridge, 2000.

[34] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete & Computational Geometry*, 4, 1989.

[35] J. Cheeger. A lower bound for the smallest eigenvalue of the laplacian. *Problems in Analysis*, pages 195–199, 1970.

[36] F. Chung. *Spectral graph theory*. Number 92 in CBMS Regional Conference Series in Mathematics. AMS, 1997.

[37] T. M. Cover. Universal data compression and portfolio selection. In *37th Annual Symposium on Foundations of Computer Science*, pages 534–538, Burlington, Vermont, 1996. IEEE.

[38] G. B. Dantzig. Maximization of linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pages 339–347, 1951.

[39] U. Feige and E. Ofek. Spectral techniques applied to sparse random graphs. *Random Structures and Algorithms*, 27(2):251–275, September 2005.

[40] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[41] D.P. Foster and R. Vohra. Regret in the on-line decision problem. *Games and Economic Behaviour*, 29:7–35, 1999.

[42] Y. 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.

[43] Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[44] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS*, pages 300–309, 1998.

[45] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[46] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation Algorithms*, pages 144–191, 1997.

[47] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[48] S. Golden. Lower Bounds for the Helmholtz Function. *Physical Review*, 137:1127–1128, February 1965.

[49] M. Grigoriadis and L. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. In *Operations Research Letters*, volume 18, pages 53–58, 1995.

[50] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, 1988.

[51] E. Halperin and E. Hazan. Haplofreq - estimating haplotype frequencies efficiently. In *RECOMB*, pages 553–568, 2005.

[52] 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.

[53] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[54] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Manuscript*, 2006.

[55] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, Cambridge, 1990.

[56] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, Cambridge, 1994.

[57] G. Iyengar, D. J. Phillips, and C. Stein. Approximation algorithms for semidefinite packing problems with applications to maxcut and graph coloring. In *IPCO*, pages 152–166, 2005.

[58] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[59] A. Kalai and S. Vempala. Efficient algorithms for online decision problems. In *16th Annual Conference on Computational Learning Theory*, pages 26–40, 2003.

[60] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[61] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. *J. Comput. Syst. Sci.*, 48(3):464–497, 1994.

[62] J. A. Kelner and D. A. Spielman. A randomized polynomial-time simplex algorithm for linear programming. In *STOC*, pages 51–60, 2006.

[63] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademia Nauk SSSR*, pages 1093–1096, 1979.

[64] R. 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`.

[65] R. Khandekar, S. Rao, and U. V. Vazirani. Graph partitioning using single commodity flows. In *STOC*, pages 385–390, 2006.

[66] S. Khanna, M. Sudan, L. Trevisan, and D. P. Williamson. The approximability of constraint satisfaction problems. *SIAM J. Comput.*, 30(6):1863–1920, 2000.

[67] P. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *STOC*, pages 338–347, 1996.

[68] P. N. Klein, S. A. Plotkin, C. Stein, and É. 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.

[69] P. N. Klein and N. E. Young. On the number of iterations for dantzig-wolfe optimization and packing-covering approximation algorithms. In *IPCO*, pages 320–327, 1999.

[70] A. R. Klivans and R. A. Servedio. Boosting and hard-core set construction. *Machine Learning*, 51(3):217–238, 2003.

[71] J. Kuczyński and H. Woźniakowski. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM J. Matrix Anal. Appl.*, 13(4):1094–1122, 1992.

[72] Z. Landau and A. Russell. Random cayley graphs are expanders: a simple proof of the alon-roichman theorem. *Electr. J. Comb.*, 11(1), 2004.

[73] J. R. Lee. On distance scales, embeddings, and efficient relaxations of the cut cone. In *SODA*, pages 92–101, 2005.

[74] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[75] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[76] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

[77] N. Littlestone. *Mistake bounds and logarithmic linear-threshold learning algorithms.* PhD thesis, University of California at Santa Cruz, Santa Cruz, CA, USA, 1989.

[78] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1 February 1994.

[79] P.-S. Loh and L. J. Schulman. Improved expansion of random cayley graphs. *Discrete Mathematics and Theoretical Computer Science*, 6(2):523 – 528, 2004.

[80] D. W. Matula. A linear time $2+\varepsilon$ approximation algorithm for edge connectivity. In *SODA*, pages 500–504, 1993.

[81] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *J. SIAM Rev.*, 20(4):801–836, October 1978.

[82] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[83] Yu. Nesterov and A. S. Nemirovskii. Conic formulation of a convex programming problem and duality. *Optimization Methods and Software*, 1(2):95–115, 1992.

[84] Yu. Nesterov and A. S. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming: Theory and Applications.* SIAM, Philadelphia, 1994.

[85] S. A. Plotkin, D. B. Shmoys, and E. Tardos. Fast approximation algorithm for fractional packing and covering problems. In *FOCS*, pages 495–504, 1991.

[86] J. Robinson. An iterative method of solving a game. *Ann. Math.*, 54:296–301, 1951.

[87] S.Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007.

[88] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.

[89] D. S. Shmoys. Cut problems and their application to divide and conquer. *Approximation Algorithms for NP-hard problems*, 1995.

[90] D. A. Spielman and S-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3):385–463, 2004.

[91] C. J. Thompson. Inequality with applications in statistical mechanics. *Journal of Mathematical Physics*, 6(11):1812–1823, 1965.

[92] K. Tsuda, G. Rätsch, and M. K. Warmuth. Matrix exponentiated gradient updates for on-line learning and bregman projection. *Journal of Machine Learning Research*, 6:995–1018, 2005.

[93] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Math. Program.*, 73(3):291–341, 1996.

[94] J. van den Eshof and M. Hochbruck. Preconditioning lanczos approximations to the matrix exponential. *SIAM J. Sci. Comput.*, 27(4):1438–1457, 2006.

[95] V. Vazirani. *Approximation algorithms.* Springer Verlag, 2002.

[96] M. K. Warmuth. Winnowing subspaces. In *ICML*, 2007.

[97] M. K. Warmuth and D. Kuzmin. A bayesian probability calculus for density matrices. In *UAI*, 2006.

[98] M. K. Warmuth and D. Kuzmin. Online variance minimization. In *COLT*, pages 514–528, 2006.

[99] M. K. Warmuth and D. Kuzmin. Randomized PCA algorithms with regret bounds that are logarithmic in the dimension. In *NIPS*, 2006.

[100] M. K. Warmuth and D. Kuzmin. Online kernel PCA with entropic matrix updates. In *ICML*, 2007.

[101] A. Wigderson and D. Xiao. Derandomizing the Ahlswede-Winter matrix-valued Chernoff bound using pessimistic estimators and applications. *ECCC TR06-105*, 2006.

[102] Y. Ye. An $O(n^3L)$ potential reduction algorithm for linear programming. *Math. Program.*, 50:239–258, 1991.

[103] N. E. Young. Randomized rounding without solving the linear program. In *SODA*, pages 170–178, 1995.

Appendix A

Algorithms for Linear and Semidefinite Programming

In this chapter, we give a high level overview of the well-known algorithms for LP and SDP. These algorithms are quite general, and the ellipsoid and interior point methods in particular solve LPs and SDPs (approximately) in polynomial time.

A.1 The Simplex Algorithm

The famous Simplex algorithm of Dantzig [38], while not polynomial time in the worst case, gave the first practical way to solve LPs. The Simplex algorithm exploits the geometry of the LP: viz. the fact that the domain for the variables specified by the linear constraints forms a convex polytope, and the linear objective function is optimized at one of the vertices of the polytope. Thus, one can perform a walk on the vertices of the polytope, each time moving to a neighboring vertex that improves the objective, until we reach a vertex that is the local optimum among its neighboring vertices. This vertex is then a global optimum since the objective function is linear.

Even now, the Simplex algorithm is one of the fastest ways to solve LPs in practice. A theoretical justification of why the Simplex algorithm performs so well in practice was given by Spielman and Teng [90] who proved that the *smoothed complexity* of the shadow vertex Simplex algorithm is polynomial, i.e. if the input is perturbed slightly, then with high probability the shadow vertex Simplex algorithm converges in polynomial time. Kelner and Spielman [62] gave the first randomized Simplex type algorithm that runs in polynomial time.

There is no simplex analogue for SDP, however, since the domain is not a convex polytope because of the positive semidefiniteness constraint.

A.2 The Ellipsoid Algorithm

In 1979, Khachiyan [63] showed the breakthrough result that LPs can be solved in polynomial time using the Ellipsoid method developed by Shor, Yudin and Nemirovskii in the 1970s. The Ellipsoid method can be viewed as performing binary search in high dimensional space. A rigorous exposition can be found in [50].

Using a standard binary search on the value of the objective function, a convex optimization problem can be reduced to the following feasibility problem: there is a convex body K in $\mathbb{R}^n$, and the objective is to find a point in K, or declare correctly that it is empty. We are also given a value R such that K is contained in the ball of radius R centered at the origin, and a value r such that if K is non-empty, then it contains a ball of radius r. We are also given a separation oracle for K, i.e. an algorithm that decides whether a given input point is in K or not, and if not, then it returns a hyperplane which separates the input point from K.

Then the Ellipsoid algorithm repeatedly produces ellipsoids of ever-shrinking volume which contain K, starting with the ball of radius R centered at the origin. At each stage, the separation oracle is invoked with the center of the ellipsoid as input; either the point lies in K, in which case we stop, or else it isn't, and we get a hyperplane separating the center of the ellipsoid from K. In the latter case, the algorithm generates an ellipsoid that contains the intersection of the current ellipsoid with the half-space of the hyperplane that contains K, with the property that the new ellipsoid has volume that is a factor $\exp(\frac{-1}{O(n)})$ smaller than the volume of the current ellipsoid. Since the starting ellipsoid, viz. the ball of radius R centered at the origin, has volume $O(R^n)$, and assuming K is non-empty, its volume is at least $\Omega(r^n)$, within $O(n \log(\frac{R^n}{r^n})) = O(n^2 \log(R/r))$ iterations we must obtain a point in K; otherwise, we conclude that K is empty.

In the case of linear programs, we first reduce the problem to a feasibility instance by doing binary search on the objective. Thus, we need to decide non-emptiness of a convex polytope. A separation oracle for the convex polytope simply checks if any of the linear constraints is violated by the input point; if so, the constraint gives the necessary separating hyperplane.

It is possible to show that all vertices of the polytope have rational coordinates with the numerator and denominator bounded by $\exp(L)$ where L is an input bit size parameter. Assuming the polytope is non-degenerate, then it has $n+1$ affinely independent vertices, and hence the simplex formed by them can be shown to have volume at least $\exp(-O(nL))$ (note: all the ellipsoid algorithm needs is a lower bound on the volume of K. The radius r of a contained ball is not necessarily required). Also, K can be shown to be contained in a ball of radius $\exp(O(L))$. Thus, the Ellipsoid algorithm decides feasibility in $O(n^2L)$ iterations. Each iteration involves one call to the separation oracle; since this involves checking each of the constraints at the current point, the separation oracle can be implemented in $O(mn)$ time. This dominates the running time for the other operations in each iteration (including computing the new ellipsoid), so the overall running time becomes $O(mn^3L)$.

Thus, even though the Ellipsoid algorithm runs in polynomial time, it is vastly impractical. Even so, the power of the Ellipsoid method comes from its ability to use a very weak specification of the convex body in question, viz. in terms of a separation oracle. This fact was exploited by Grötschel, Lovász and Schrijver [50] who applied it to SDPs and gave applications of SDPs to combinatorial optimization.

To apply the Ellipsoid method to SDP, all one needs to do is provide a separation oracle. Again, this is simply done by checking if any of the linear constraints is violated by the input point. If all of them are, then one needs to check the positive semidefiniteness constraint, i.e. one needs to check whether the input point $\mathbf{X}$ has only non-negative eigenvalues. This is easily done by computing its smallest eigenvalue λ. If it is negative, then there is an associated eigenvector $\mathbf{v} \in \mathbb{R}^n$ such that the $\mathbf{v}^\top \mathbf{X} \mathbf{v} = \lambda < 0$. This is the same as saying that $\mathbf{X} \bullet \mathbf{v}\mathbf{v}^\top < 0$. On the other hand, any positive semidefinite $\mathbf{X}$ satisfies $\mathbf{X} \bullet \mathbf{v}\mathbf{v}^\top \geq 0$; thus, this linear constraint gives a separating hyperplane.

The Ellipsoid method applied to SDP is even more impractical, needing $O(n^4 L)$ iterations to converge by an analysis similar to the one for LP, since the dimension of the space of variables is now $\Theta(n^2)$.

A more efficient algorithm with the same characteristics as the Ellipsoid algorithm was developed by Vaidya [93]. This algorithm maintains convex polytopes, rather than ellipsoids, which enclose K, such that their volume reduces by a constant factor in every step. This algorithm needs $O(nL)$ iterations to converge in the case of LPs, and $O(n^2 L)$ in the case of SDPs.

A.3 Interior Point Methods

The first truly practical polynomial time algorithm for LP was developed by Karmarkar [60] using Interior Point methods. This method was further developed by Ye [102] who gave an $O(n^3 L)$ time algorithm for LP (here, L is a bit size complexity parameter).

Interior point methods rely on the theory of LP duality and complementary slackness. These algorithms are much more involved than the Ellipsoid method, and we will not consider them in any detail; we only describe a few key ideas.

Interior Point methods maintain a feasible primal and dual solution. The LP is optimized precisely when the duality gap, which is the difference between the values of the primal and dual solution, is reduced to zero. We therefore make use of a carefully constructed potential function of the current primal and dual solutions with two key properties: (a) at optimality, i.e. when the duality gap goes to zero, the potential function goes to negative infinity, and (b) it consists of a logarithmic barrier function which tends to keep the primal solution away from the boundary of the polytope. In fact, it suffices to drive the potential down to $-O(\sqrt{n}L)$; at this point, the primal and dual solutions are already close enough in value that they can be rounded to optimality straight away.

The algorithm starts with a feasible primal and dual solution with potential $O(\sqrt{n}L)$. Then in each iteration, an affine scaling is applied to the primal and dual solution which preserves the potential but takes the primal solution far away from the boundary of the polytope. Then, since we want to decrease the potential function, a gradient descent step is applied to either the primal or the dual variables (only). In either case, the gradient needs to be projected into the feasible region so as to maintain the feasibility of the new solutions. It is then shown that either the primal or dual gradient step reduces the potential by at least a constant. Thus, only $O(\sqrt{n}L)$ steps are needed to reduce the potential to $-O(\sqrt{n}L)$, at which point the algorithm stops. In each iteration, the most expensive operation is the computation of the projected gradient. By using approximate solutions to linear equations, this can be accomplished in $O(n^{2.5})$ time, thus yielding an overall running time complexity of $O(n^3L)$.

This method was extended by Alizadeh [5] to the case of SDPs, and an interior point algorithm for SDPs was presented with running time $O(\sqrt{m}(n^3+m)L)$. A similar result was obtained by Nesterov and Nemirovskii [83, 84].