Efficient Approximation Algorithms for Some Semidefinite Programs

Hsueh-I Lu
Ph.D. Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-96-33
December 1996

Efficient Approximation Algorithms for Some Semidefinite Programs

by
Hsueh-I Lu
B. S., National Taiwan University, 1986
Sc. M., National Taiwan University, 1990
Sc. M., Brown University, 1992

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University

May 1997

This dissertation by Hsueh-I Lu
is accepted in its present form
by the Department of Computer Science
as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ______________ ______________________________________

Philip N. Klein

Recommended to the Graduate Council

Date ______________ ______________________________________

Franco P. Preparata

Date ______________ ______________________________________

Pascal Van Hentenryck

Approved by the Graduate Council

Date ______________ ______________________________________

Vita

I was born in I-Lan, Taiwan on January 12, 1965. Because of moving, I went to three primary schools in Taiwan: Gwan-Fu, Jong-Shan, and Ming-Tsu, two in I-Lan and one in Hualien. I graduated from May-Lwun Junior High in 1978, and Hualien Senior High in 1982, both in Hualien. In 1986 I received my B. S. degree in Computer Science and Information Engineering from National Taiwan University, Taipei, from where I received my Sc. M. degree in 1990. Before joining Brown University in August 1990, I never left the small island Taiwan. In 1992 I received my second Sc. M. degree in Computer Science from Brown University, Providence, RI, USA.

Preface

Linear programming has been fruitful in designing approximation algorithms for many combinatorial optimization problems. Nonlinear programming did not receive as much attention in this respect until the recent work by Goemans and Williamson [62]. They use *semidefinite programs*, which are nonlinear programs, to obtain approximation solutions with much better approximation factors. For example, the best previously known approximation algorithm for MAXCUT, which was invented twenty years ago, has approximation factor 0.5 [137]. The algorithm of Goemans and Williamson dramatically improves the approximation factor to 0.878. Inspired by the work on MAXCUT, Karger, Motwani, and Sudan [86] adapt the same technique and obtain the currently best approximation algorithm for coloring a k-colorable graph with the fewest possible number of colors. The approximation ratio is improved by a factor of $\Omega(n^{2/k})$ over the best previously known result [29]. Later Karger and Blum give the best known approximation algorithm for coloring a 3-colorable graph by combining the results in [86] and [29] and give the best known approximation algorithm for coloring a 3-colorable graph. Besides MAXCUT and COLORING, the technique of semidefinite programming has been successful in designing approximation algorithms for many other optimization problems, such as MAX DICUT [63, 51], MAX SAT [63, 51], MAX BISECTION [54], MAX k-CUT [54], MAX STABLE SET [6], and BETWEENNESS [43].

Each of these improved algorithms is based on obtaining a near-optimal solution to a semidefinite program, which is referred as the *semidefinite relaxation* of the underlining optimization problem. Therefore how to approximately solve these semidefinite relaxations efficiently, in both time and space, is practically important. All the previous work resorts to various general *interior-point methods* [5, 113, 150] for this step. In the thesis we adapt the technique of *Lagrangean relaxation* [96, 105, 121, 87, 158] to cope with this crucial step. Specifically we use the framework of Plotkin, Shmoys, and Tardos [121] to obtain near-optimal solutions to the semidefinite relaxations of MAXCUT and COLORING. Our results significantly reduce the work space required by the best known approximation algorithms for MAXCUT and COLORING if the given graph is sparse. As a result, our algorithms enable the best known approximation algorithms for MAXCUT and COLORING to work on much larger sparse graphs than they did with interior-point methods.

In the first part of the thesis we give the basic approximation algorithms for the semidefinite programs of MAXCUT and COLORING [94]. Starting with finding an initial feasible solution, our algorithms repetitively solve a sequence of optimization subproblems. The solutions to these optimization subproblems are used to "guide" the current feasible solution towards the optimal solution. We show that the subproblem corresponding to the semidefinite relaxation of MAXCUT is in fact an eigenvector problem. More interestingly, we show that the subproblem corresponding to COLORING is equivalent to the semidefinite relaxation for MAXCUT. If the approximation precision is a constant, the time complexity of our algorithms is better than than interior-point methods and have the ability to exploit the sparsity of the given graph [94].

In the second part of the thesis we show the extensions of our basic algorithms. In the first chapter of the second part we show how our basic algorithms can be applied to reduce the required work space of the best known approximation algorithms for MAXCUT and COLORING. Moreover, if the precision of the approximation is a constant and the chromatic number of the input graph is a constant, our algorithms reduces the time complexity of the best known approximation algorithms for MAXCUT and COLORING.

The application of semidefinite programming to combinatorial optimization problems was pioneered by Lovász's work on the Shannon capacity of a graph [106]. Lovász introduced the *theta function*, which is a semidefinite program, to approximate the Shannon capacity of a graph. The polynomial-time solvability of the theta function leads to the only known polynomial-time algorithms for some optimization problems on perfect graphs, such as MAX-CLIQUE (and therefore MAX STABLE SET) [66], and COLORING (and therefore MIN CLIQUE COVER) [67]. In the second chapter of the second part we show how our basic algorithm for the semidefinite relaxation of COLORING can be used to reduce the work space required by the only known polynomial-time algorithms for MAXCLIQUE and MAX STABLE SET. Moreover, if the chromatic number of the input perfect is a constant, then the time complexity of both algorithms is significantly reduced.

The output of interior-point methods or our basic algorithms tends to be a full-rank matrix. It follows from Pataki's results [117, 118] that there exist low-rank optimal solutions to the semidefinite programs for MAXCUT and COLORING. In the last chapter of the second part of the thesis we show how to adapt Pataki's results and give compact approximation algorithms for solving the semidefinite programs for MAXCUT and COLORING [95]. The rank of the solution output by our compact algorithm for MAXCUT's semidefinite relaxation matches Pataki's bound. Interestingly, the rank of the solution output by our compact algorithm for COLORING's semidefinite relaxation is asymptotically lower than Pataki's bound. Moreover, if the input graph is sparse, then the space complexity of our compact algorithms is better than that of interior-point methods. The running-time analysis of our compact algorithms, however, is based on exact arithmetic. Our compact algorithms are only of theoretical interest so far.

We have implemented our basic approximation algorithms for the semidefinite relaxation of MAXCUT. The results of some numerical experiments are shown in the last part of the thesis.

Acknowledgments

I would like to thank Prof. Philip Klein, who has been a great advisor. He teaches me a lot about doing research. One important thing I have learned from him is “never easily give up”. I really appreciate his patience, encouragement, and ideas. Without his guidance, I would not have learned half as much at Brown.

I would like to thank Prof. Franco Preparata and Prof. Pascal Van Hentenryck for being on my thesis committee. Their suggestions and comments help to improve the thesis significantly.

I would like to thank Dr. David Williamson for supplying useful data, program, and information, which help a lot on the work shown in Chapter 7. I also thank him for the suggestion that inspired me to work out the results shown in Chapter 5.

I would like to thank Prof. Michael Jünger for supplying useful information and data on Spin Glasses.

I would like to thank Prof. Rob Netzer and Prof. R. Ravi for their collaboration on the research that is not within the scope of the thesis. Working with them has been very educational to me. I also thank them for helping me to get a teaching job.

I would like to thank Mary Andrade, who is one of the nicest people I have ever met, for her kind help on the thesis. I also would like to thank Jennet Kirschenbaum for her generous help on the defense.

During my graduate study at Brown, there have been some big challenges in my life. I thank God for granting me faith and strength through His Words so that I can confront the difficulties. Also I would like to thank my Christian friends, Dr. and Mrs. Michael Chang, Dr. and Mrs. Nai-Che Tsai, James and Jean Sung, Daniel and Shirley Wu, Edward and Salina Kuo, Yo-Wei and Guo-Luey Sun, Wen-Chun and Wen-Jen Ni, Shih-Hsiu and Jia-Ling Huang, Tim and Nancy Lu, Daphne Kot, Caspar Liou, Chong-Ming and Balinda Wong, Sean and Grace Ho, and many others. Their friendship and prayers have been very strong support to me.

My deepest gratitude is to my parents. Their support and encouragement has always been the treasure of my life. My greatest appreciation is due my wife, Chih-Hui. Without her sweet company and being thoughtful, supportive, understanding, and patient, my graduate study at Brown cannot possibly be successfully finished. I dedicate this thesis to all three of them.

Contents

List of Tables

List of Figures

Part I

Basics

Chapter 1

Introduction

Linear Programming has been a very useful tool for designing approximation algorithms. A combinatorial optimization problem can usually be written as an *integer linear program*, which is NP-complete to be solved [58]. A widely used technique is relaxing the underlining integer linear program to a linear program by allowing the solution to be fractional. The resulting linear program, called the *fractional relaxation*, is often polynomial-time solvable, e.g. if the original integer linear program has a polynomial number of constraints [92]. The optimal fractional solution can be regarded as some kind of approximate solution to the original integer linear program. Sometimes randomly rounding the fractional solution to an integer solution gives a good approximation to the original integer linear program with high probability [128]. Moreover, the randomized-rounding procedure can usually be derandomized to a deterministic algorithm [127]. It is surprising that the technique of fractional relaxation plus randomized rounding gives the best performance guarantees for many NP-hard problems [25]. Linear programming, however, is not always this powerful. Sometimes the performance of linear relaxation is limited. Therefore whether *nonlinear relaxation* can work better has been an open question.

Researchers in the area of combinatorial optimization usually study a mathematical programming problem by examining the structure of the convex hull spanned by the feasible solutions. The convex hull reveals important information about the problem. For example, if the convex hull can be described by polynomial number of constraints, which is the case for linear programs, then any linear objective function can be optimized within the convex hull in polynomial time. The approach of the fractional relaxation described in the previous paragraph is closely related to how to obtain the integral convex hull from the fractional convex hull of the underlining linear program. Gomory [64] proposes a way to obtain the integral convex hull by applying a sequence of *linear cuts* on the fractional convex hull. Chvátal [45] shows that the Gomory-cut procedure always terminates in a finite number of steps and obtains the integral convex hull. The number of steps, however, could be very large. Lovaśz and Schrijver [108] show that *nonlinear cuts* do perform better in this aspect. They introduce a nonlinear cut, whose nonlinearity comes from semidefinite constraints, by "lifting" the fractional convex hull to a higher-dimensional space, applying some linear cuts in the higher dimensional space, and then projecting the resulting convex hull back to the original space. They show that after applying a linear number of such nonlinear cuts to the fractional convex hull, the resulting convex body is the integral convex hull itself. Their result suggests that nonlinear relaxation might perform better in designing approximation algorithms.

Recently Goemans and Williamson give the first approximation algorithm which shows that nonlinear relaxation does perform better than linear relaxation [62]. Specifically they write the NP-complete MAXCUT problem as a nonlinear integer program and then relax it as a *semidefinite program*, which is a nonlinear program. After obtaining a near-optimal solution to this *semidefinite relaxation* of MAXCUT in polynomial time [5], their algorithm "rounds" the solution, which is a positive-semidefinite matrix, to an integral solution for the original nonlinear integer program. The result is exciting: Goemans and Williamson show that the performance ratio is 0.878, which is much better than the performance ratio 0.5 of the best previously known approximation algorithms for twenty years. In the same paper, Goemans and Williamson also give a 0.79607-approximation algorithm for MAX DICUT and a 0.758-approximation algorithm for MAX SAT, where the best previously known approximation algorithms have performance ratios 0.25 and 0.75 respectively. Later the performance ratios for MAX DICUT and MAX 2SAT are improved to 0.859 and 0.991 respectively by Feige and Goemans [51]. Using the same technique of semidefinite relaxation, Frieze and Jerrum [55] improve the approximation algorithms for MAX BISECTION and MAX k-CUT for large k.

Inspired by the work on MAXCUT, Karger, Motwani, and Sudan [86] apply the same idea of semidefinite relaxation to the problem of coloring a k-colorable graph with the fewest possible number of colors. They generalize the notion of color to that of *vector color*. Consider a graph of n nodes. A vector color is an n-dimensional vector assigned to a node. A *vector coloring* of the graph is an assignment of vector colors to the nodes such that the vector colors of every two adjacent nodes have a small inner product. The vector coloring problem can be written as a semidefinite program, whose near-optimal solution can be found in polynomial time [5]. The near-optimal vector coloring is then "rounded" to a regular coloring by vector projection in n-dimensional space. The number of colors required by the algorithm is $\tilde{O}(n^{1-3/(k+1)})$. The number of colors required by the best previously known approximation algorithm is $\tilde{O}(n^{1-1/(k-4/3)})$ [29]. The improvement is by a factor of $\tilde{\Omega}(n^{2/k})$. The approximation ratio for COLORING on a 3-colorable graph was later improved by Blum and Karger [30].

Each of these improved algorithms is based on obtaining a near-optimal solution to a semidefinite program, which is referred as the *semidefinite relaxation* of the underlining optimization problem. Therefore how to approximately solve these semidefinite relaxations efficiently, in both time and space, is practically important.

Semidefinite programming is a generalization of linear programming, and a special case of convex programming. Essentially, what is added to linear programming is the ability to specify constraints of the form "X is a positive-semidefinite matrix", where X is a symmetric matrix whose entries are variables. Such a constraint is written $X \succeq 0$. A positive-semidefinite matrix is a matrix X such that for any column vector u, the value of $u^T X u$ is nonnegative. A symmetric matrix X is positive-semidefinite if and only if every eigenvalue of X is nonnegative.

The first polynomial-time algorithm proposed for solving semidefinite programs was based on the ellipsoid method. Grötschel, Lovász, and Schrijver [66] gave a subroutine that, given a matrix X that is not positive semidefinite, finds a hyperplane separating X from the set of positive-semidefinite matrices. This subroutine, combined with the ellipsoid method, provided a polynomial-time algorithm for semidefinite programming. However, the complexity of this algorithm is quite high.

Nesterov and Nemirovskii [112] showed how to use the interior-point methods to solve semidefinite programs. Alizadeh [5] showed how an interior-point algorithm for linear programming could be directly generalized to handle semidefinite programming. Since the work of Alizadeh, there has been a great deal of research into such algorithms [83, 149, 131, 157, 75, 101, 53]. A simplex-type method was also discovered by Pataki [119]. Among these varieties, the performance of interior-point methods are the best in both theory and practice. Suppose there are ℓ constraints in a semidefinite program, whose variable is an $n \times n$ matrix. A near-optimal solution for the semidefinite program can be obtained in time $\tilde{O}(\sqrt{n}\ell^3)$ using interior-point methods. For example, let n be the number of nodes and m be the number of edges in a given graph. The semidefinite relaxation for MAXCUT has n constraints on an $n \times n$-matrix variable. A near-optimal solution can therefore be obtained in time $\tilde{O}(n^{3.5})$. The semidefinite relaxation of COLORING has m constraints on an $n \times n$-matrix variable. Hence a near-optimal solution can be obtained in time $\tilde{O}(\sqrt{n}m^3)$.

1.1 Memory Space

There are two kinds of storage in most modern computer systems, the local disk and the server disk. The size of the local disk determines the work space of an executable program. The local disk of a computer is usually small, e.g., 2 gigabytes. Many computers, however, can share a fairly large server disk, e.g., 200 gigabytes. In the thesis we emphasize the distinction between these two kinds of storage. We assume that the server disk is significantly larger than the local disk. Therefore we make efforts to reduce the required work space of our algorithms and store the intermediate data on the server disk.

1.2 Our Results

Using interior-point methods to approximate the semidefinite relaxations of MAXCUT and COLORING requires work space quadratic in the number of nodes of the given graph. Specifically, interior-point methods involve operations, such as matrix inversion or Cholesky decomposition, on $n \times n$ matrices, where n is the number of nodes in the input graph. These matrix operations require work space $O(n^2)$. Thus previous sequential implementations of the best known approximation algorithms for MAXCUT and COLORING work only for relatively small graphs, e.g., graphs with no more than 1,000 nodes. Reducing the required work space for approximating the semidefinite relaxations of MAXCUT and COLORING is the main contribution of the thesis. Specifically, we give algorithms for obtaining approximate solutions to both semidefinite relaxations in work space $O(m)$, where m is the number of edges. As a consequence, our algorithms enable the best known approximation algorithms for MAXCUT and COLORING to handle sparse graphs that are much larger, e.g., $n = 100,000$. Although our algorithms perform well only for obtaining rough approximations, as predicted by the analysis and shown in the experiments, they can handle instances that previous algorithms cannot even touch.

The basis of our algorithms is the framework of Plotkin, Shmoys, and Tardos [121], which is related to the technique of *Lagrangean relaxation*. Lagrangean relaxation useful in exploiting the structure of mathematical programs [96, 105, 121, 87, 158]. It basically works as follows: Suppose we are given a set of constraints, over which an objective function is supposed to be optimized. Instead of solving it using general-purpose algorithms such as ellipsoid methods [92, 68] or interior-point methods [89, 147, 50], we could separate the constraints into two sets, say P_{easy} and P_{hard}, where we know how to efficiently optimize a linear objective function over P_{easy}. Starting with an initial feasible solution X that satisfies $P_{\text{easy}} \cup P_{\text{hard}}$, the algorithm proceeds iteratively. In each iteration the algorithm solves an *optimization subproblem*, i.e., optimizing over P_{easy} a carefully chosen objective function based on the current solution X. The solution to the optimization subproblem is then used to guide X towards the optimum solution. If it is much easier to optimize some function over P_{easy} than over $P_{\text{easy}} \cup P_{\text{hard}}$, then the technique of Lagrangean relaxation usually yields great improvement over the algorithms that optimize over $P_{\text{easy}} \cup P_{\text{hard}}$ directly. Based on the technique of Lagrangean relaxation, Plotkin, Shmoys, and Tardos [121] give a framework for designing efficient approximation algorithms for convex programs. All the previous applications of this framework are to linear programs. The results in the thesis are the first applications of the framework to nonlinear programs.

Using the framework of Plotkin, Shmoys, and Tardos, we observe an interesting relation between the semidefinite relaxations of MAXCUT and COLORING: the semidefinite relaxation of MAXCUT can be regarded as an optimization subproblem for solving the semidefinite relaxation of COLORING. Besides our observation, Laurent, Poljak and Rendl [104] show that the semidefinite relaxation of MAX INDEPENDENCE SET can be reduced to the semidefinite relaxation of MAXCUT by introducing a new node adjacent to each node of the original graph. Therefore the semidefinite relaxation of MAXCUT appears to be fundamental.

For each of the semidefinite relaxations of MAXCUT and COLORING we give a *basic approximation algorithm.* Both basic algorithms require work space only $O(m)$. Although the $\Omega(n^2)$ space required by both algorithms is no better than that required by interior-point methods, most of the intermediate data produced during the execution of our basic algorithms can be stored on the server disk. At the completion of execution, the output can be easily obtained from those data stored on the server disk. Moreover, the time complexity of our basic algorithms is better than that of interior-point methods if the approximation precision is a constant. Therefore our basic algorithms are quite good at obtaining rough approximations to the semidefinite relaxations of MAXCUT and COLORING for sparse graphs.

When applied to the best known approximation algorithms for MAXCUT and COLORING, our basic algorithms perform even better. The approximation algorithm of Goemans and Williamson and the algorithm of Karger, Motwani, and Sudan both require that we compute a matrix H such that $H^T H$ equals the near-optimal solution X to the corresponding semidefinite relaxation. Given an $n \times n$ matrix X, it would take time $O(n^3)$ to obtain such a matrix H. The form in which a near-optimal solution is given by our basic algorithms, however, makes it trivial to find such a matrix H. Moreover, when our basic algorithms are applied to the best known approximations for MAXCUT and COLORING, storing those intermediate data on the server disk becomes unnecessary. As as result, the space complexity of the algorithm for MAXCUT is reduced to $O(m)$, and the space complexity of the algorithm for COLORING is reduced to $O(m + n\Delta)$, where Δ is the maximum degree of the input graph. The work space of both algorithms is still $O(m)$.

Our basic algorithm for COLORING's semidefinite relaxation can be applied to the only known polynomial-time algorithms for some optimization problems on perfect graphs. The required space is reduced to $O(m)$. The time complexity is significantly reduced if the chromatic number of the input perfect graph is a constant.

We also show how to modify our basic algorithms so that low-rank solutions to the semidefinite relaxations of MAXCUT and COLORING can be obtained in low space. We refer to these two modified algorithms as the *compact approximation algorithms* for the semidefinite relaxations of MAXCUT and COLORING. The space required by the our compact algorithm for the semidefinite relaxation of MAXCUT is only $O(m+n^{1.5})$. The space required by the compact algorithm for the semidefinite relaxation of COLORING is $O(m+\epsilon^{-2}n^{1.5}\log n)$, where ϵ is the required approximation precision. The work space required by both compact algorithms is $O(m+n^{1.5})$. The running-time analysis, however, is based on exact arithmetic. Therefore our compact algorithms are only of theoretical interest so far.

1.2.1 Basic algorithms

Our basic algorithm for the semidefinite relaxation of MAXCUT starts with finding an initial feasible solution X by a greedy method. The algorithm then proceeds iteratively. In each iteration an optimization subproblem, defined by the current feasible solution X, is approximately solved. Every optimization subproblem is in fact an eigenvector problem that can be efficiently solved using the *power method*. The algorithm then uses the near-optimal solution $\tilde{X}$ to the optimization subproblem to guide the current feasible solution X towards the optimal solution. The current solution X is guaranteed to be ϵ-optimal after $\tilde{O}(\epsilon^{-2}n)$ iterations. It is known that the power method takes $\tilde{O}(\epsilon^{-1})$ matrix-vector multiplications to yield the required solution to each optimization subproblem [102]. Therefore it takes time $\tilde{O}(\epsilon^{-3}mn)$ for our basic algorithm to produce an ϵ-optimal solution to the semidefinite relaxation of MAXCUT, where m is the number of edges. If the approximation precision ϵ is a constant, then our algorithm performs better than the general interior-point methods, which take time $\tilde{O}(n^{3.5})$ [62]. The improvement is particularly striking when the input graph is very sparse, e.g., $m = O(n)$. We have performed some numerical experiments on sparse large graphs. For a graph with $10,000$ nodes and $20,000$ edges, our algorithm can find a .5-optimal solution to the semidefinite relaxation of MAXCUT in 10 hours. By a rough estimate, an interior-point algorithm would take at least one year to output any near-optimal solution to the same semidefinite relaxation, even if it had sufficient work space.

Our basic algorithm for the semidefinite relaxation of COLORING works in a similar way: the sequence of the near-optimal solutions $\tilde{X}$ to the optimization subproblems guides the current solution X towards the optimal solution. The required number of iterations is $\tilde{O}(\epsilon^{-2})$. Each optimization subproblem can be regarded as the semidefinite relaxation of MAXCUT for a graph, whose edge weights are determined by the current feasible solution X to the semidefinite relaxation of COLORING. Therefore we can use our basic algorithm for MAXCUT's semidefinite relaxation to handle these optimization subproblems. We show that the required precision is $O(\epsilon/k)$ if the given graph is k-colorable. Therefore our algorithm for COLORING runs in time $\tilde{O}(\epsilon^{-5}k^3mn)$. If ϵ and k are both constant, then our basic algorithm performs better than the general interior-point methods, which takes time $\tilde{O}(\sqrt{n}m^3)$.

As mentioned above, if the given graph is sparse, our basic algorithms significantly reduce the required work space for obtaining near-optimal solutions to the semidefinite relaxations of MAXCUT and COLORING. Although the space required by our basic algorithms is $\tilde{O}(n^2\epsilon^{-3})$ for MAXCUT and $\tilde{O}(n^2\epsilon^{-5})$ for COLORING, they both require work space only $O(m)$. Specifically, our basic algorithms can store most of the intermediate data on the server disk, and keep only data of linear size in the work space. At the completion of execution, the output can be easily obtained from the data stored on the server disk. Since the server disk is usually much larger than the work space, our algorithms can work on fairly large sparse instances in practice. Although our basic algorithms perform better for rough approximations, they can handle the instances on which interior-point methods would simply run out of memory.

1.2.2 Applications to the approximation algorithms of MAXCUT and COLORING

The best known approximation algorithms for MAXCUT and COLORING have the following three steps:

1. Obtain a near-optimal solution X to the corresponding semidefinite relaxation.

2. Obtain a matrix H such that H^TH equals X. Assign the i^{th} column $h^{(i)}$ of H to the i^{th} node of the input graph.

3. Obtain a near-optimal solution based on the dot-products $z^{(s)} \cdot h^{(i)}$ for some random vectors $z^{(s)}$.

Although our basic algorithms are only for the first step, we can modify them so that all three steps are done incrementally. In particular, in the j^{th} iteration we obtain the j^{th} component of each vector $h^{(i)}$ and the j^{th} components of those random vectors. Therefore each dot-product can be computed up to the j^{th} term. As soon as our algorithms find a near-optimal solution to the corresponding semidefinite relaxation, all the dot-products are known, from which a near-optimal solution to MAXCUT or COLORING can be easily obtained.

Because all we need at the completion of execution are those dot-products, the algorithms no longer need to store the intermediate data on the server disk. As a consequence, the space complexity of the algorithm for MAXCUT is reduced to $O(m)$, and the space complexity of the algorithm for COLORING is reduced to $O(m + n\Delta)$. The required work space of both algorithms is still $O(m)$.

1.2.3 Applications to some algorithms for perfect graphs

The only known polynomial-time algorithms for some optimization problems, such as MAXCLIQUE, MAX STABLE SET, COLORING, and MIN CLIQUE COVER, on perfect graphs [66, 67], are based on the observation that Lovász's *theta function* [106] can be computed in polynomial time. It follows from the journal version of [86] that the optimum value of COLORING's semidefinite relaxation is closely related to Lovász's theta function. We show that our basic algorithm for the semidefinite relaxation of COLORING can thus be used to reduce the time and space complexity of the only known polynomial-time algorithms for finding a maximum clique or a maximum stable set in a perfect graph, if the chromatic number of the given graph is a constant.

1.2.4 Compact algorithms

Pataki shows that any extreme solution to a semidefinite program with ℓ constraints has rank at most $\sqrt{2\ell}$ [117, 118]. (The same result is implied by the work of Barvinok [18].) The space required by any interior-point method is $\Omega(n^2)$, because it has to maintain a full-rank $n \times n$ matrix during the execution. Our basic algorithm for MAXCUT's semidefinite relaxation is potentially better because it does not have to maintain a full-rank matrix all the time. Specifically, it starts with finding an initial rank-one matrix, and then proceeds iteratively to improve its quality. In each iteration it applies a rank-one update to the current solution matrix. Unfortunately the number of iterations can be as large as $\tilde{O}(\epsilon^{-2}n)$. Therefore the output tends to have full rank, and its space complexity is thus asymptotically the same as that of interior-point methods. Similarly the output of our basic algorithm for COLORING's semidefinite relaxation tends to have full rank.

It follows from Pataki's results that no matter how high the rank of the current solution is, there always exists a solution of rank at most $\sqrt{2(n+1)}$ that has the same quality. Therefore our compact algorithm for MAXCUT's semidefinite relaxation is basically the basic algorithm with a number of *rank-reduction* steps, which were proposed by Pataki [118]. Specifically, whenever the rank of the current solution is more than $\sqrt{2(n+1)}$, we replace the current solution by a rank-$\sqrt{2(n+1)}$ solution that has the same quality. That way we maintain the low rank of the current solution during the execution of our basic algorithm. As a result, we reduce required space for obtaining a near-optimal solution to MAXCUT's semidefinite relaxation to $O(n^{1.5}+m)$, where $O(m)$ comes from the input. The work space required by our algorithm is $O(n^{1.5}+m)$, which is more than that required by our basic algorithm, but still less than that required by an interior-point algorithm for sparse graphs. Based on exact arithmetic, the time required by our compact algorithm is $\tilde{O}(\epsilon^{-2}n^4)$.

As shown in §1.2.1, our approximation algorithm for the COLORING semidefinite program runs in $\tilde{O}(\epsilon^{-2})$ iterations. In each iteration an approximate solution for a MAXCUT semidefinite program is obtained. The resulting approximate solution of the COLORING semidefinite program is a linear combination of the solutions obtained from all iterations. If we resort to our compact algorithm for the semidefinite relaxation of MAXCUT in each iteration of our basic algorithm for COLORING's semidefinite relaxation, then we are guaranteed to find an approximate solution for COLORING semidefinite program whose rank is $\tilde{O}(\epsilon^{-2}\sqrt{n})$. This gives a compact approximation algorithm for the COLORING semidefinite program. Note that Pataki's results only guarantee a rank-$O(\sqrt{m})$ solution for the semidefinite relaxation of COLORING. Our compact approximation algorithm for the COLORING semidefinite program is interesting in that the rank of the output is asymptotically lower than the bound given by Pataki, when ϵ is a constant and the graph is dense. The space complexity of our compact algorithm is $\tilde{O}(m+\epsilon^{-2}n^{1.5})$. The required work space is $O(m+n^{1.5})$. Based on exact arithmetic, the time complexity is $\tilde{O}(\epsilon^{-5}k^3n^4)$.

1.3 Outline

In the following chapter we give the preliminaries that are required to make the thesis self-contained. We then explain our basic algorithms in Chapter 3. We spend three chapters on the extensions of our basic algorithms: the applications to the approximation algorithms for MAXCUT and COLORING in Chapter 4, the applications to some perfect-graph algorithms in Chapter 5, and the compact algorithms in Chapter 6. The implementation of the basic algorithm for the semidefinite relaxation of MAXCUT is shown in Chapter 7. In the last chapter, we summarize our contribution, and propose some future work.

Lemma 12, Lemma 13, Lemma 15, Lemma 22, Lemma 23, and Lemma 25 of the thesis are adapted from the paper by Plotkin, Shmoys, and Tardos [122]. We therefore do not include their proofs in the main content of the thesis. However, to make the thesis self-contained, we put their proofs in Appendix A.

The power method is the core of our algorithms. Its analysis of can be found in [102]. For completeness of the thesis, we give an easier-to-read and shorter analysis in Appendix B.

Chapter 2

Background

2.1 Notation

In this section we define some notation that will be used throughout the thesis.

All vectors and matrices in the thesis are real. All vectors are column vectors. Unless otherwise specified, vectors are denoted by lower-case letters, and matrices are denoted by capital letters. We use x_i to denote the i^{th} element of vector x; and use X_{ij} to denote the entry ij of matrix X.

Suppose x is an n-element vector, and X is an $m \times n$ matrix. Define

$$\begin{aligned} \|x\|_2 &= \sqrt{\sum_{1 \le i \le n} x_i} \\ \|X\|_F &= \sqrt{\sum_{1 \le i \le m} \sum_{1 \le j \le n} X_{ij}}. \end{aligned}$$

Let X and Y be two $m \times n$ matrices. Define

$$X \bullet Y = \sum_{1 \le i \le m} \sum_{1 \le j \le n} X_{ij} Y_{ij}.$$

Let x and y be two n-element vectors. Define

$$x \cdot y = \sum_{1 \le i \le n} x_i y_i.$$

We use $X \succeq 0$ to signify that the matrix X is symmetric and positive semidefinite. Let I be the identify matrix. Let J be the all-one matrix. Let $\vec{1}$ be the all-one vector.

Vectors $v_1, \ldots, v_n$ are *orthonormal* if the following holds for every $1 \le i, j \le n$.

$$v_i^T v_j = \begin{cases} 1 & \text{when } i = j \\ 0 & \text{when } i \ne j. \end{cases}$$

Clearly if the columns of an $n \times k$ matrix U are orthonormal, then $U^T U$ is the $k \times k$ identity matrix.

It is well-known that every $n \times n$ symmetric rank-k matrix X can be written as $X = VDV^T$ for some $k \times k$ diagonal matrix D and $n \times k$ matrix V, where the columns of V are orthonormal (see, e.g., Theorem 2.5.4 of [79]).

Let L be a linear subspace of n-dimensional space. We use $L^\perp$ to denote the orthogonal space of L. Namely $L^\perp$ is composed of all the vectors that are orthogonal to each vector in L. Let A be an $n \times n$ matrix. The *null space* of A is the linear subspace of n-dimensional space in which every vector x satisfies $Ax = 0$. It is well-known that the rank of A plus the dimension of the null space of A is exactly n.

Let P be a mathematical program. An assignment to the variables of P is said to be a *feasible solution* to P if the assignment satisfies P's constraints. The *value* of such an assignment is the value of the objective function at that assignment. Let $f(\cdot)$ be the objective function of P. Let X^* be an optimal solution to P. A solution X feasible to P is *ϵ-optimal* if

$$(1+\epsilon)^{-1} \le \frac{f(X^*)}{f(X)} \le (1+\epsilon).$$

For simplicity of time-bounds, we assume throughout that $\epsilon > 1/n$.

2.2 Two Graph Optimization Problems

In the thesis we focus on the following two graph optimization problems.

MAXCUT — Given a graph with weights on edges, we would like to partition its nodes into two sets such that the sum of the weights of the crossing edges is maximized.

COLORING — Given a graph, we would like to color its nodes with the fewest number of distinct colors such that no adjacent nodes have the same color.

Both problems are NP-complete [58].

2.2.1 MAXCUT

MAXCUT [125] has applications in VLSI layout design and statistical physics [16, 17, 40, 42, 120, 142, 143, 132]. It is among Karp's original NP-complete problems [90]. It remains NP-complete even if the given graph is unweighted [60], Algorithm GreedyCut(G)

Input: an undirected graph G with weights on edges

Output: a cut of G

1. Start with both sides empty, so the initial value of the cut is zero.
2. For every node v in G do
   (a) Put v on the side which increases the current cut value more.

Figure 2.1: Algorithm GreedyCut

cubic [123, 155], or quasi-planar [14]. Since this problem is so hard, researchers have been working along two different directions. The first direction is to look for algorithms for MAXCUT on special graphs. It is known that MAXCUT is polynomial-time solvable on planar graphs [72, 114], weakly bipartite graphs [69], and graphs that are not contractible to K_5 [15].

The other direction is to look for approximation algorithms. It is known that MAXCUT is MAX SNP-complete, even if the degree of the graph is bounded by a constant [116]. It follows from [11] that MAXCUT does not have a *polynomial-time approximation scheme* (PTAS) [85] unless P=NP. Namely there exists a positive constant ϵ such that approximating MAXCUT to within a ratio of $1-\epsilon$ is NP-complete. For example it is NP-complete to approximate MAXCUT to within the ratio of 65/66 [20]. However, MAXCUT can be easily approximated to within a constant factor. The first approximation algorithm for MAXCUT, as shown in Figure 2.1, was given by Sahni and Gonzalez in 1976 [137].

One can easily see that the resulting cut output by the above greedy algorithm has value at least one half of the total edge weight of G. Clearly the value of any cut is at most the sum of positive edge weights of G. If G has no edges with negative weights, then the approximation ratio of Sahni and Gonzalez's algorithm is $\frac{1}{2}$. If G has negative weights, then one can verify that the value of the cut reported by GreedyCut minus the sum of negative edge weights of G is at least one half of the sum of the absolute values of the edge weights of G.

Researchers have slightly improved the approximation ratio for unweighted graph MAXCUT [151, 124, 73]. None of the results, however, makes progress on the constant term $\frac{1}{2}$ in the approximation ratio.

In the thesis we are interested in approximation algorithms for MAXCUT.

For a graph with no negative edge weights, Goemans and Williamson [63] give the currently best approximation algorithm for this problem in 1994. They use the technique of *semidefinite relaxation* to significantly improve the approximation ratio of MAXCUT to 0.878, which is provably tight [88]. For a graph with negative edge weights, Goemans and Williamson's algorithm guarantees to find a cut whose value minus the negative edge weights is no less than 0.878 times the maximum cut value minus the negative edge weights. Their result is a major breakthrough in designing approximation algorithms in that for the first time they successfully adapt a nonlinear-programming technique to design approximation algorithm. Much research, including the results described in this thesis, was prompted by their pioneering work [86, 55, 51, 43, 94, 95, 104, 78, 111, 6, 30, 88, 98].

There are also some results on approximation algorithms for MAXCUT on special graphs. It is known that MAXCUT has a polynomial-time approximation scheme when the given graph is dense [49, 10].

2.2.2 COLORING

COLORING [84] has applications in register allocation, time tabling, scheduling, sequencing, and circuit testing [39, 38, 34, 44, 70, 35, 24, 154, 59]. The most famous problem related to graph coloring is probably the four-colorability of planar graphs, which is due to F. Guthrie. The conjecture has been answered affirmatively [7, 9]. Moreover, a planar graph can actually be four-colored in quadratic time [8, 133]. However, COLORING is much harder than four-coloring a planar graph. Coloring a 3-colorable planar graph with three colors is NP-complete [60].

There are many positive results on solving COLORING on special graphs [61, 66, 67, 57, 145, 80, 12, 13], among which a quite general result is that perfect graphs can be colored with the fewest possible number of colors in polynomial time [66, 67].

Approximating COLORING is difficult. There exists a positive constant ϵ such that approximating COLORING to within a factor of n^ϵ is NP-complete [110]. Moreover, COLORING cannot be approximated in polynomial time to within a factor of $n^{1/10}$ (or $n^{1/13}$) unless NQP $\neq$ co-RQP (or NP = co-RP) [21]. Researchers have been trying hard to improve the approximation ratio for COLORING [153, 29, 74]. The currently best approximation ratio for COLORING is $O(n(\log\log n)^2/\log^3 n)$ due to Halldórsson [74].

In the thesis we are interested in approximating COLORING for a k-colorable graph. The problem is also difficult. Given a 3-colorable graph, it is NP-complete to color the graph with four colors [93]. Karger, Motwani, and Sudan give the best currently known approximation algorithm for COLORING on a k-colorable graph [86]. Their result improves the best previously known ratio [29] by a factor of $\Omega(n^{2/k})$. Their work is based on the technique of semidefinite relaxation invented by Goemans and Williamson for MAXCUT [63]. Blum and Karger combine the results in [29] and [86] and give the best known approximation algorithm for coloring a 3-colorable graph.

2.3 Semidefinite Relaxation

We explain the idea of semidefinite relaxation in this section.

2.3.1 Positive semidefinite matrices

An $n \times n$ matrix X is *positive semidefinite* if $v^T X v \geq 0$ for every n-element vector v. It follows directly from the definition that the sum of two positive semidefinite matrices is still positive semidefinite.

If X is symmetric (which implies that all of its eigenvalues are real, see, e.g., §5.5 of [144]), then the following statements are equivalent (see, e.g., [103]):

- X is positive semidefinite.
- All eigenvalues of X are nonnegative.
- X admits a *Cholesky decomposition*. Namely there exists a matrix H such that $X = H^T H$.

Recall that we use $X \succeq 0$ to signify that X is symmetric and positive semidefinite.

The following property of a symmetric and positive-semidefinite matrix will be used throughout the thesis.

Lemma 1 An $n \times n$ matrix X is symmetric and positive semidefinite if and only if there exists a set U of n-element columns vectors such that

$$X = \sum_{u \in U} u u^T.$$

Proof Clearly $uu^T \succeq 0$ for any vector $u \in U$. Therefore $X \succeq 0$. The if-part of the lemma holds.

Let $H^T H$ be the Cholesky decomposition of X. Let U be the set of the columns of H^T. One can easily verify that $X = \sum_{u \in U} uu^T$. Therefore the only-if-part of the lemma holds. $\square$

The following lemma is a corollary of the above lemma.

Lemma 2 Let X and Y be two $n \times n$ matrices. If $X \succeq 0$ and $Y \succeq 0$, then $X \bullet Y \geq 0$.

Proof It follows from Lemma 1 that X and Y can be written as

$$\begin{aligned} X &= \sum_{u \in U} uu^T, \\ Y &= \sum_{v \in V} vv^T. \end{aligned}$$

Clearly

$$\begin{aligned} X \bullet Y &= \sum_{u \in U} \sum_{v \in V} (uu^T) \bullet (vv^T) \\ &= \sum_{u \in U} \sum_{v \in V} (u \cdot v)^2 \\ &\geq 0. \end{aligned}$$

The lemma is proved. $\square$

2.3.2 Semidefinite programs

A *semidefinite program* is a mathematical program for optimizing a linear function of a matrix X subject to linear constraints and the constraint that $X \succeq 0$. An example of a semidefinite program is as follows.

$$\begin{aligned} \min \quad & C \bullet X \\ s.t. \quad & A^{(1)} \bullet X \leq b_1 \\ & \vdots \\ & A^{(\ell)} \bullet X \leq b_\ell \\ & X \succeq 0, \end{aligned} \tag{2.1}$$

where C, $A^{(1)}, \ldots, A^{(\ell)}$, and X are all $n \times n$ matrices. Note that without the last constraint $X \succeq 0$, the above mathematical program is simply a linear program.

The constraint $X \succeq 0$ is nonlinear but convex. Therefore a semidefinite program is a special case of a convex program. Although a semidefinite program is a nonlinear program, surprisingly it preserves many nice properties of a linear program. For instance most methods for solving integer programs (e.g., simplex [48], ellipsoid [92], and interior-point [89, 156]) can be generalized to semidefinite programs [119, 68, 146, 5, 112, 113]. As a result, a semidefinite program can be approximately solved to within an additive error ϵ in polynomial time unless the optimal solution itself is exponentially large [5]. For example if the semidefinite program (2.1) has a feasible solution, then it takes time $O(\sqrt{n}\ell^3 \log \epsilon^{-1})$ for interior-point methods to find a near-optimal solution to within an additive error ϵ.

2.3.3 Semidefinite relaxation

A discrete optimization problem, such as MAXCUT or COLORING, can usually be written as an integer linear program. Solving an integer linear program is known to be NP-complete [90, 31], but linear program can be efficiently solved in polynomial time. Therefore the technique of *fractional relaxation* works as follows. The linear program without the integral constraints is called the linear relaxation of the original integer linear program. We first find the optimal solution $x_{\text{fractional}}$ to the linear relaxation. We then round $x_{\text{fractional}}$ into an integral solution x_{integral}. The vector x_{integral} is a good approximate solution to the original integer linear program if the rounding procedure is successful. This technique has been used in approximation algorithms for many discrete optimization problems [76, 128, 127, 25].

Goemans and Williamson generalized the above linear-relaxation technique to obtain a nonlinear-relaxation one [63]: they write the MAXCUT problem as an integer semidefinite program, i.e., a semidefinite program with some integral constraints,

$$\begin{array}{lll} \max & \frac{1}{4} C \bullet (J - X) & \\ \text{s.t.} & X_{ii} = 1 & \text{for every } 1 \leq i \leq n \\ & X \succeq 0 & \\ & X_{ij} \in \{\pm 1\} & \text{for every } 1 \leq i, j \leq n, \end{array} \tag{2.2}$$

where C is the edge-weight matrix of the given graph. They use interior-point methods to find a near-optimal solution X_{real} of the semidefinite relaxation, i.e., the semidefinite programs without the integral constraints

$$X_{ij} \in \{\pm 1\} \quad \text{for every } 1 \le i, j \le n,$$

They show that X_{real} can always be rounded to a good approximate solution X_{integral} for the original discrete optimization problem.

2.3.4 MAXCUT — an example

To illustrate the technique, we explain in this subsection why the MAXCUT problem can be represented as (2.2). Let C be the edge-weight matrix of the given graph, i.e., C_{ij} is the weight of edge ij. Suppose we use $y_i = 1$ to signify putting node i on the left, and $y_i = -1$ to signify putting node i on the right. Therefore every cut corresponds to an assignment to y. One can easily verify that the weight of the cut is exactly

$$\frac{1}{4} \sum_{1 \le i,j \le n} C_{ij}(1 - y_i y_j).$$

Therefore the weight of the maximum cut can be written as

$$\begin{aligned} \max \quad & \frac{1}{4} \sum_{1 \le i,j \le n} C_{ij}(1 - y_i y_j) \\ \text{s.t.} \quad & y_i \in \{\pm 1\}, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \max \quad & \frac{1}{4} C \bullet (J - X) \\ \text{s.t.} \quad & X = yy^T \\ & X_{ii} = 1 \qquad\qquad \text{for every } 1 \le i \le n. \end{aligned} \tag{2.3}$$

Clearly a feasible solution to the above mathematical program is also a feasible solution to (2.2). It suffices to show that a feasible solution to (2.2) is also a feasible solution to the above mathematical program.

Let X be a feasible solution to (2.2). Since $X \succeq 0$, there exists a matrix H such that $X = H^T H$. Let $h^{(i)}$ be the i^{th} column of H. Since $h^{(i)} \cdot h^{(i)} = X_{ii} = 1$, each $h^{(i)}$ is a unit vector. Since $h^{(1)} \cdot h^{(i)} = X_{1j} \in \{\pm 1\}$, it follows that $h^{(i)} = \pm h^{(1)}$ for every $1 \le i \le n$. Let y be the first column of X. We know $h^{(i)} = y_i h^{(1)}$. We show $X = yy^T$. For simplicity, let $z = h^{(1)}$, so $h^{(i)} = y_i z$. Therefore

$$h_j^{(i)} = y_i z_j, \tag{2.4}$$

for every j. Let $u^{(j)}$ be the j^{th} column of H^T. Clearly

$$u_i^{(j)} = h_j^{(i)}, \tag{2.5}$$

for every i and j. Combining (2.4) and (2.5), we know $u^{(j)} = z_j y$ for every j. It follows that

$$\begin{aligned} X &= H^T H \\ &= \sum_j u^{(j)} u^{(j)T} \\ &= \sum_j (z_j y)(z_j y)^T \\ &= \sum_j z_j^2 y y^T \\ &= (\sum_j z_j^2) y y^T \\ &= y y^T. \end{aligned}$$

Thus X is a feasible solution to (2.3). It follows that (2.3) is equivalent to (2.2). Therefore the MAXCUT problem can be represented as (2.2).

Chapter 3

Basic Algorithms

3.1 Overview

Obtaining a near-optimal solution to the semidefinite relaxation of MAXCUT, which has $O(n)$ constraints, is the crucial step in Goemans and Williamson's algorithm. The time complexity for this step is $O(n^{3.5})$ using interior-point methods. Obtaining a near-optimal solution to the semidefinite relaxation of COLORING, which has $O(m)$ constraints, is the crucial step in Karger, Motwani, and Sudan's algorithm. The time complexity of this step is $O(\sqrt{n}m^3)$ using interior-point methods.

In this chapter, we propose an alternative approach. We apply the method of Plotkin, Shmoys, and Tardos [121] to find approximate solutions to both semidefinite programs. As the core of these algorithms, we use the power method to solve an eigenvalue problem. The power method can exploit the sparsity of the matrix, which in turn reflects the sparsity of the graph.

The basis of our approach is the method of Plotkin, Shmoys, and Tardos for approximately solving what they call fractional packing and covering problems. Their method is analogous to Lagrangean relaxation, in which constraints are replaced by a "penalty" component of the objective function, such that violations of the constraint are penalized. The difficulty in Lagrangean relaxation is in choosing the weights of the penalties. In the method of Plotkin, Shmoys, and Tardos, the penalties are determined directly from a current candidate solution. These penalties are used to select a direction to move from the current candidate solution to obtain the next candidate solution, and the process is repeated.

Some of the particulars of their approach originated in the work of Shahrokhi and Matula [139] on approximate solution of a multicommodity flow problem. Shahrokhi and Matula proved a polynomial but rather high bound on the running time of their algorithm. A much faster algorithm for this problem was developed by Klein, Plotkin, Stein, and Tardos [96]. One important ingredient in the improvement is a formulation of approximate optimality (relaxed complementary slackness conditions) that works well in this setting. This multicommodity flow algorithm was generalized by Leighton, Makedon, Plotkin, Stein, Tardos, and Tragoudas [105]. Plotkin, Shmoys, and Tardos then took a final step and generalized this multicommodity flow algorithm, obtaining not a single algorithm but a whole framework in which algorithms for a variety of problems could be formulated. Grigoriadis and Khachiyan [65] independently generalized the above results on multicommodity flow problem to other applications.

This framework, like that of the ellipsoid algorithm [68] and the method given by Vaidya [147], casts algorithms in terms of a subroutine, an oracle. For the ellipsoid algorithm, the subroutine is called a separation oracle. For Plotkin, Shmoys, and Tardos, the subroutine must find an optimum (or near-optimum) solution to a simpler optimization problem. Specifically, if the goal is to find a vector that satisfies some linear inequalities and in addition lies in a given convex body P, the subroutine must find a vector in P of minimum cost.

3.1.1 The basic algorithm for the semidefinite relaxation of MAXCUT

To cast the semidefinite relaxation of MAXCUT in this framework, we take P to be the set of positive-semidefinite matrices X satisfying the linear inequality $\sum_{ij} L_{ij}X_{ij} = 1$. To obtain an algorithm, we must supply a subroutine to find such a matrix X minimizing a weighted sum of its diagonal elements. That is, we must find the minimizer of

$$\min\{\sum_i y_i X_{ii} : \sum_{ij} L_{ij}X_{ij} = 1, X \succeq 0\}. \tag{3.1}$$

We show that the minimum is achieved by a rank-one matrix, a matrix X of the form uu^T, where u is an n-element vector. Namely, the a minimizer for the following program is also a minimizer of the above program.

$$\min\{\sum_i y_i u_i^2 : \sum_{ij} L_{ij}u_i u_j = 1, uu^T \succeq 0\}. \tag{3.2}$$

Furthermore, we show that the best such vector u can be obtained from a vector v which is the eigenvector corresponding to the maximum eigenvalue of a matrix L' derived from L and y. Specifically, let

$$L'_{ij} = \frac{L_{ij}}{\sqrt{y_i y_j}}.$$

Let v be the eigenvector corresponding to the maximum eigenvalue of L'. Let u be the vector obtained from v and y by letting

$$u_i = \frac{v_i}{\sqrt{y_i}}.$$

Then u is a minimizer of (3.2), and therefore uu^T is a minimizer of (3.1).

Finding an eigenvector of a matrix would take too much time. We exploit the fact that we need only an approximate solution, and can therefore use a few iterations of a method called the *power method*.

Since L' is positive-semidefinite, all its eigenvalues are nonnegative. Hence its maximum eigenvalue is the eigenvalue of maximum absolute value. The power method uses the fact that the eigenvalues of A^k are the k^{th} powers of the eigenvalues of A. If k is big enough, therefore, small differences in eigenvalues of A result in large differences in eigenvalues of A^k. Instead of computing A^k directly, we can make do with the matrix-vector product $A^k x_{(0)}$, where $x_{(0)}$ is a suitably chosen initial vector. We can compute this product by computing a series of k matrix-vector products $Ax_{(i)}$. Computing each of these products takes time proportional to the number of nonzero elements of A. In this case, the number is $O(m)$. It is known that a good enough solution is obtained by taking $k = O(\epsilon^{-1} \log n)$ [102]. Thus we obtain a fast subroutine to optimize over P.

Some additional work is needed. We need to ensure that the vectors obtained by the power method have reasonably small components. We do this by perturbing the initial problem in a way that does not modify the optimum value too much; an approximately optimal solution to the perturbed problem yields an approximately optimal solution to the original problem. We also need to show how to obtain a good initial solution to the semidefinite program. For this, we use the previously known approximation algorithm for MAXCUT, due to Sahni and Gonzalez [137], which is shown in Figure 2.1. Finally, we show that after only $O(\epsilon^{-2} n \log n)$ iterations of optimizing over P, we obtain an ϵ-optimal solution X to the semidefinite program. The time required is thus $O(\epsilon^{-3} nm \log^2 n)$. Furthermore, in practice we could run the power method for many fewer iterations by using as its initial vector the output obtained from the power method the last time. Thus one factor of ϵ^{-1} in our analysis seems superfluous.

Since in each iteration the matrix output by the subroutine has the form uu^T, we obtain X as the sum of such rank-one matrices:

$$X = u^{(1)}u^{(1)^T} + \cdots + u^{(r)}u^{(r)^T}.$$

In order to keep the required work space small, we do not store all those vectors $u^{(i)}$ on the local disk. Since the server disk is usually much larger than the local disk, we can store those vectors $u^{(i)}$ on the server disk. Although it would require space $\tilde{O}(\epsilon^3 n^2)$ on the server disk, the required work space is only $O(m)$, which is used to keep the matrices L and L', and the diagonal elements of the current solution matrix.

3.1.2 The basic algorithm for the semidefinite relaxation of COLORING

In casting this problem in the framework of Plotkin, Shmoys, and Tardos, we take the convex body P to be the set of positive-definite matrices X whose diagonal elements are 1. Thus in this case P involves n linear constraints instead of just one, as in the case of MAXCUT. Moreover, the function we must optimize over P involves all the components of X rather than just the diagonal elements (as was the case in MAXCUT). The objective function has the form $\sum_{ij} C_{ij} X_{ij}$. Thus optimization over P cannot be done using a rank-one matrix. However, by making a small perturbation in this optimization problem, we transform it into precisely the MAXCUT semidefinite program. Thus we can use our algorithm for that problem to approximately optimize over P. (Once again, we show that the perturbation is such that an approximately optimal solution to the perturbed problem yields an approximately optimal solution to the original problem.)

This time, we need to optimize over P only $O(\epsilon^{-2}\log n)$ times. Thus we obtain an algorithm for the semidefinite relaxation of that takes time $O(\epsilon^{-5}k^3 nm \log^3 n)$, if the given graph is k-colorable. As mentioned in §3.1.1, by being more clever about how we use the power method, we can probably remove one factor of ϵ^{-1} in practice.

3.1.3 Outline

Here is the structure for the rest of the chapter. We first give some preliminaries in §3.2. In §3.3 we give and analyze the basic approximation algorithm for solving MAXCUT's semidefinite relaxation. In §3.4 we give and analyze the basic approximation algorithm for solving COLORING's semidefinite relaxation.

3.2 Preliminaries

All vectors in this chapter are n-element real column vectors. All matrices in this chapter are $n \times n$, real, and symmetric, and therefore have real eigenvalues.

Let G be a simple undirected graph of n nodes with costs on edges. Let C be the cost matrix of G, where C_{ij} is the cost of the edge ij of G. Let $\bar{L}(G)$ be the matrix $\bar{L}$ defined as

$$\bar{L}_{ij} = \begin{cases} -C_{ij} & i \neq j \\ \sum_{1 \leq k \leq n} |C_{ik}| & i = j. \end{cases}$$

Note that if G does not have negative edge cost, then $\bar{L}(G)$ is called the *Laplacian matrix* of G. The following lemma is immediate from Geršgorin's theorem (e.g. see §10.6 of [103]).

Lemma 3 Let $\bar{L} = \bar{L}(G)$. Then $\bar{L} \succeq 0$.

The semidefinite relaxation of MAXCUT is the following semidefinite program as shown in §2.3.4:

$$\begin{array}{lll} \max & \frac{1}{4} C \bullet (J - X) & \\ \text{s.t.} & X_{ii} = 1 & \text{for every } 1 \leq i \leq n \\ & X \succeq 0. & \end{array}$$

In order to handle the negative edge costs, however, Goemans and Williamson work on the following semidefinite program [63, Theorem 2.6].

$$\begin{array}{lll} \max & \frac{1}{4} C \bullet (J - X) - c_{\text{negative}} & \\ \text{s.t.} & X_{ii} = 1 & \text{for every } 1 \leq i \leq n \\ & X \succeq 0, & \end{array} \tag{3.3}$$

where c_{negative} is the sum of negative edge costs of G. Lemma 4 ensures that (3.3) can be rewritten as follows.

$$\begin{array}{lll} \max & \frac{1}{4} \bar{L} \bullet X & \\ \text{s.t.} & X_{ii} = 1 & \text{for every } 1 \leq i \leq n \\ & X \succeq 0. & \end{array} \tag{3.4}$$

Lemma 4 Let C be the cost matrix of graph G. Let $\bar{L} = \bar{L}(G)$. Let c_{negative} be the sum of negative edge costs of G. Let X be a matrix such that every diagonal element of X is one. Then $C \bullet (J - X) - 4c_{\mathsf{negative}} = \bar{L} \bullet X$.

Proof Let $D = \bar{L} + C$. Clearly D is a diagonal matrix. Since every diagonal element of X is one, we know that

$$\begin{aligned} D \bullet X &= \sum_{1 \le i \le n} \bar{L}_{ii} \\ &= \sum_{1 \le i,j \le n} |C_{ij}| \\ &= -4c_{\text{negative}} + \sum_{1 \le i,j \le n} C_{ij} \\ &= -4c_{\text{negative}} + C \bullet J. \end{aligned}$$

Therefore

$$\begin{aligned} \bar{L} \bullet X &= D \bullet X - C \bullet X \\ &= -4c_{\text{negative}} + C \bullet J - C \bullet X \\ &= C \bullet (J - X) - 4c_{\text{negative}}. \end{aligned}$$

□

The semidefinite relaxation of COLORING is the following semidefinite program as shown in [86].

$$\begin{aligned} \min \quad & \lambda \\ \text{s.t.} \quad & X_{ii} = 1 \quad \text{for every } 1 \le i \le n \\ & X_{ij} \le \lambda \quad \text{for every edge } ij \text{ of } G \\ & X \succeq 0. \end{aligned} \tag{3.5}$$

3.3 The Basic Algorithm for the Semidefinite Relaxation of MAXCUT

3.3.1 Problem reduction

For simplicity of the analysis, we assume that the edge costs of G are scaled such that the sum of the absolute values of G's edge costs is exactly one. It follows that $\bar{L} \bullet I = 2$. Define matrix L to be the matrix obtained from $\bar{L}$ as follows:

$$L = \bar{L} + I/n. \tag{3.6}$$

Since $\bar{L} \succeq 0$ by Lemma 3, we know $L \succeq 0$. Consider the following semidefinite program.

$$\begin{array}{lll} \min & \lambda & \\ \text{s.t.} & X_{ii} \leq \lambda & \text{for every } 1 \leq i \leq n \\ & L \bullet X = 1 & \\ & X \succeq 0. & \end{array} \tag{3.7}$$

Suppose λ is the maximum diagonal element of X. Define $Z(X)$ to be the matrix obtained from X by increasing each of its diagonal elements to λ. One can easily verify that if (X, λ) is a feasible solution to (3.7), then $Z(X)/\lambda$ is a feasible solution to (3.4).

The following lemma shows how to obtain a feasible solution to (3.7) from a feasible solution to (3.4).

Lemma 5 Let X be a feasible solution to (3.4). Let $c = \bar{L} \bullet X$. Then $(\frac{X}{c+1}, \frac{1}{c+1})$ is a feasible solution to (3.7). Moreover, if X is an optimal solution to (3.4), then $(\frac{X}{c+1}, \frac{1}{c+1})$ is an optimal solution to (3.7).

Proof The first part of the lemma is immediate from the fact that

$$\begin{aligned} L \bullet X &= \bar{L} \bullet X + 1 \\ &= c + 1. \end{aligned}$$

We prove the second part. Suppose (X^*, λ^*) is an optimal solution to (3.7). We know that $Z(X^*)/\lambda^*$ is a feasible solution to (3.4). Therefore

$$\begin{aligned} c + 1 &= \bar{L} \bullet X + 1 \\ &\geq \bar{L} \bullet Z(X^*)/\lambda^* + 1 \\ &= L \bullet Z(X^*)/\lambda^* \\ &\geq 1/\lambda^*. \end{aligned}$$

However by the optimality of (X^*, λ^*) we know that $\lambda^* \leq 1/(1+c)$. Therefore $\frac{1}{c+1} = \lambda^*$, and the lemma follows. $\square$

The following lemma ensures that the problem of finding a near-optimal solution to (3.4) can be reduced to the problem of finding a near-optimal solution to (3.7).

Lemma 6 Suppose $\epsilon \leq 0.5$. Let (X, λ) be an ϵ-optimal solution to (3.7). Then $Z(X)/\lambda$ is a 2ϵ-optimal solution to (3.4).

Proof Let (X^*, λ^*) be an optimal solution to (3.7). By definition $\lambda \leq (1+\epsilon)\lambda^*$. Let $\bar{X} = Z(X)/\lambda$, which is a feasible solution to (3.4). Clearly $L \bullet \bar{X} \geq 1/\lambda$. Let $\bar{X}^*$ be an optimal solution to (3.4). We show $\bar{L} \bullet \bar{X} \geq (1+2\epsilon)^{-1} \bar{L} \bullet \bar{X}^*$.

Since $\bar{X}^*$ is feasible to (3.4), we know $\frac{1}{\bar{L} \bullet \bar{X}^* + 1} \geq \lambda^*$ by Lemma 5. Thus

$$1/\lambda^* - 1 \geq \bar{L} \bullet \bar{X}^*.$$

Since I is feasible to (3.4), we know

$$\begin{aligned} \bar{L} \bullet \bar{X}^* &\geq \bar{L} \bullet I \\ &= 2. \end{aligned}$$

Hence

$$1/\lambda^* - 1 - \epsilon \geq (1 - 0.5\epsilon) \bar{L} \bullet \bar{X}^*.$$

Therefore we have

$$\begin{aligned} \bar{L} \bullet \bar{X} &= L \bullet \bar{X} - 1 \\ &\geq 1/\lambda - 1 \\ &\geq (1+\epsilon)^{-1}/\lambda^* - 1 \\ &= (1+\epsilon)^{-1}(1/\lambda^* - 1 - \epsilon) \\ &\geq (1+\epsilon)^{-1}(1 - 0.5\epsilon) \bar{L} \bullet \bar{X}^* \\ &\geq (1+2\epsilon)^{-1} \bar{L} \bullet \bar{X}^*, \end{aligned}$$

where the last inequality holds because $\epsilon \leq 0.5$. $\square$

For the rest of the section we focus on finding an ϵ-optimal solution to (3.7).

3.3.2 Bounding the values of feasible solutions

We first give a lower bound on the optimum value based on the assumption that the sum of the absolute values of G's edge costs is one. We then give an upper bound and a lower bound on the feasible values. Clearly the bounds on the feasible values hold for the optimal value.

Lemma 7 Let Y be a diagonal matrix such that $Y - L \succeq 0$. Then the optimal value is at least $1/(I \bullet Y)$.

Proof Let (X^*, λ^*) be an optimal solution to (3.7). Since $X^* \succeq 0$ and $Y - L \succeq 0$, it follows from Lemma 2 that

$$X^* \bullet (Y - L) \geq 0.$$

Therefore

$$\begin{aligned} X^* \bullet Y &\geq L \bullet X^* \\ &= 1. \end{aligned}$$

Since $X^*_{ii} \leq \lambda^*$ and Y is a diagonal matrix, we know $X^* \bullet Y \leq \lambda^* I \bullet Y$. It follows that

$$\lambda^* \geq 1/(I \bullet Y).$$

The lemma is proved. □

The following lemma is a corollary of the above lemma.

Lemma 8 The optimum value is at least 0.2.

Proof Let Y be a diagonal matrix such that $Y_{ii} = 2L_{ii} - \frac{1}{n}$. Namely $Y = 2(\bar{L} + C) + I/n$. It follows that

$$\begin{aligned} Y - L &= 2(\bar{L} + C) + I/n - L \\ &= 2(\bar{L} + C) - \bar{L} \\ &= \bar{L} + 2C. \end{aligned}$$

By definition of $\bar{L}$ we know that the i^{th} diagonal element of $\bar{L} + 2C$ is exactly the sum of the absolute value of the off-diagonal elements on its i^{th} row. It follows from Geršgorin's theorem (see, e.g., §10.6 of [103]) that $\bar{L} + 2C \succeq 0$. Therefore $Y - L \succeq 0$.

Note that

$$\begin{aligned} I \bullet Y &= 2I \bullet L - 1 \\ &= 6 - 1 \\ &= 5. \end{aligned}$$

It follows from Lemma 7 that the optimal value is at least 0.2. □

Lemma 9 A feasible value is at most n.

Proof By the feasibility of (X, λ) we know $L \bullet X = 1$. Since $L = \bar{L} + I/n$, it follows that

$$I \bullet X = n(1 - \bar{L} \bullet X).$$

Note that $\bar{L} \succeq 0$ by Lemma 3, and $X \succeq 0$ by the feasibility of (X, λ). Therefore $\bar{L} \bullet X \geq 0$ by Lemma 2, and thus

$$\begin{aligned} X_{11} + \cdots + X_{nn} &= I \bullet X \\ &\leq n. \end{aligned}$$

Since $X \succeq 0$, each X_{ii} is nonnegative. It follows that $X_{ii} \leq n$ for every $1 \leq i \leq n$. $\square$

Let y be a nonnegative vector. Let X be a matrix. Define

$$\langle X \rangle_y = y_1 X_{11} + \cdots + y_n X_{nn}. \tag{3.8}$$

Clearly $\langle X \rangle_y$ is nonnegative for any matrix $X \succeq 0$. Define $\mu(y)$ to be the minimum $\langle X \rangle_y$ over all feasible matrices X, i.e., let

$$\mu(y) = \min\{\langle X \rangle_y : L \bullet X = 1, X \succeq 0\}. \tag{3.9}$$

The following lemma gives another lower bound on any feasible value.

Lemma 10 A feasible value is at least $\mu(y)/y \cdot \vec{1}$ for any nonnegative vector y.

Proof By the feasibility of (X, λ) we know that $\lambda y \cdot \vec{1} \geq \langle X \rangle_y$. Since $\langle X \rangle_y \geq \mu(y)$ by definition of $\mu(y)$, the lemma follows. $\square$

The following lemma ensures that there is always a rank-one feasible matrix that achieves $\mu(y)$.

Lemma 11 $\mu(y) = \min\{\langle uu^T \rangle_y : L \bullet (uu^T) = 1\}$.

Proof Let μ be the RHS of the equality. Since $uu^T \succeq 0$ for any vector u, we know that uu^T is feasible if $L \bullet (uu^T) = 1$. Therefore $\mu(y) \leq \mu$ by definition of $\mu(y)$. We show $\mu(y) \geq \mu$. Suppose X is a feasible matrix such that $\langle X \rangle_y = \mu(y)$. Since X is positive semidefinite, X can be written as $\sum_{u \in U} uu^T$, where U is a set of vectors. Let $c_u = L \bullet (uu^T)$ for every $u \in U$. By the feasibility of X we know $\sum_{u\in U} c_u = 1$, and thus $\sum_{u\in U} |c_u| \geq 1$. Therefore

$$\begin{aligned}\langle X\rangle_y &= \sum_{u\in U}\langle uu^T\rangle_y \\ &= \sum_{u\in U}|c_u|\langle\frac{uu^T}{|c_u|}\rangle_y \\ &\geq \sum_{u\in U}|c_u|\mu \\ &\geq \mu.\end{aligned}$$

□

3.3.3 Relaxed optimality conditions

Let ϵ be a positive precision parameter. Let (X, λ) be a feasible solution. Let $\tilde{X}$ be a feasible matrix such that

$$\langle\tilde{X}\rangle_y \leq (1+\epsilon/13)\mu(y). \quad (3.10)$$

We define the following *relaxed optimality conditions*, which are adapted from [122].

(P1) $(1-\epsilon)\lambda y\cdot\vec{1} \leq \langle X\rangle_y$

(P2) $\langle X\rangle_y - \langle\tilde{X}\rangle_y \leq \epsilon(\langle X\rangle_y + \lambda y\cdot\vec{1})$

The following two lemmas are adapted from [122]. The first lemma shows that (P1) and (P2) together ensure the near optimality of (X, λ). The second lemma shows that (P1) always holds if α is sufficiently large.

Lemma 12 Let y be a nonnegative vector. Let $\tilde{X}$ be a matrix such that $\langle\tilde{X}\rangle_y \leq (1+\epsilon/13)\mu(y)$. If $\epsilon \leq 1/13$, then (P1) and (P2) imply that (X, λ) is 4ϵ-optimal.

Lemma 13 Suppose $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2n\epsilon^{-1})$. Let (X, λ) be a feasible solution. Let y be a nonnegative vector defined as $y_i = e^{\alpha X_{ii}}$. Then (P1) is satisfied.

For completeness of the thesis, we include the proofs of the above two lemmas in Appendix A.

3.3.4 The algorithm

Let C be the cost matrix for the given graph G. Let ϵ be the given positive precision parameter. We give the basic algorithm that produces an ϵ-optimal

Algorithm $\textsc{VectorMaxCut}(C, \epsilon)$

Input: C is the matrix such that C_{ij} is the weight of edge ij of the input graph.

Output: an ϵ-optimal solution matrix X to the semidefinite relaxation of MAXCUT for the input graph.

1. Scale C such that the sum of the absolute values of the edge costs is one, and then compute L from C.
2. Let $(X, \lambda) = \textsc{Initial}(C)$.
3. Let $\epsilon' = 2/3$.
4. While $\epsilon' > \epsilon$ do
   (a) Let $\epsilon' = \epsilon'/2$.
   (b) Let $(X, \lambda) = \textsc{Improve}(X, \lambda, \epsilon'/4)$.
5. Return X.

Figure 3.1: Algorithm $\textsc{VectorMaxCut}$

solution to (3.7). The algorithm starts with finding a $\frac{2}{3}$-optimal initial solution (X, λ), i.e., a solution that achieves $\frac{3}{5}$ of the optimum. It then iteratively updates (X, λ) until it is ϵ-optimal. In each iteration a procedure $\textsc{Improve}$ is called to improve the precision of (X, λ). Specifically, after the k^{th} iteration (X, λ) is guaranteed to be $O(2^{-k})$-optimal with high probability. The algorithm $\textsc{VectorMaxCut}(C, \epsilon)$ is given in Figure 3.1.

The procedure $\textsc{Initial}(C)$ obtains an initial solution (X, λ) from a greedy solution to the MAXCUT problem. Using the greedy method by Sahni and Gonzalez [137], which is shown in Figure 2.1, one can obtain a $\frac{2}{3}$-optimal solution to the MAXCUT problem in time linear in the number of edges. Moreover, such a greedy cut has value at least one half of the sum of edge costs. Let u be the characteristic vector for the greedy solution to the MAXCUT problem, where $u_i = 1$ for every node i on one side of the cut, and $u_j = -1$ for every node j on the other side of the cut. Clearly uu^T is the corresponding feasible solution to (3.4). Let $c = \bar{L} \bullet (uu^T)$. We know $\frac{1}{4}c \geq \frac{1}{2}$, since the sum of the absolute values of the edge costs is scaled to be one. Hence $c \geq 2$. Let the return value (X, λ) of $\textsc{Initial}(C)$ be $(\frac{uu^T}{c+1}, \frac{1}{c+1})$. It follows from Lemma 5 that (X, λ) is feasible to (3.7). Since $c \geq 2$, $\lambda \leq 1/3$. Therefore the initial solution (X, λ) is 2/3-optimal by Lemma 8.

Algorithm IMPROVE(X, λ_0, ϵ)

Input: X is the current solution whose value is λ_0.

Output: a solution matrix X whose value is ϵ-optimal.

1. Let $\epsilon = \min\{\epsilon, 1/13\}$.
   Let $\lambda = \lambda_0$.
   Let $\alpha = 10\epsilon^{-1}\ln(2n\epsilon^{-1})$.
   Let $\sigma = \epsilon/(4\alpha n)$.
2. Repeat
   (a) Let $y_i = e^{\alpha X_{ii}}$ for every $i = 1, \ldots, n$.
   (b) Let $\tilde{X} = $ DIRECTION$(y, \epsilon/13)$.
   (c) If (P2) is not satisfied then
   let $X = (1-\sigma)X + \sigma\tilde{X}$.
   let $\lambda = \max\{X_{11}, \ldots, X_{nn}\}$.
   else
   return (X, λ).

Figure 3.2: Algorithm IMPROVE

The procedure IMPROVE(X, λ_0, ϵ) uses a positive vector y defined as a function of the current solution (X, λ) such that (P1) is always satisfied. The procedure proceeds iteratively to update the current solution until (P2) is also satisfied. In each iteration a procedure DIRECTION$(y, \epsilon/13)$ is called to obtain a feasible solution $(\tilde{X}, \tilde{\lambda})$ such that (3.10) holds with high probability. If (P2) is satisfied, the current solution (X, λ) is returned by the procedure. If the given parameter ϵ is no more than $1/13$, by Lemma 12 we know that the return value of the procedure IMPROVE is 4ϵ-optimal as long as $\tilde{X}$ satisfies (3.10). If (P2) is not satisfied, then the current solution is moved towards $\tilde{X}$ by a small amount σ. Namely let $X = (1-\sigma)X + \sigma\tilde{X}$, and let λ be the maximum diagonal element of the new X. One can easily verify that the new (X, λ) is still feasible. In the following subsection we will show that the procedure always terminates in a finite number of iterations. The procedure IMPROVE(X, λ_0, ϵ) is given in Figure 3.2.

We will prove the following lemma in §3.3.8.

Lemma 14 The procedure DIRECTION(y, ϵ) produces a feasible matrix uu^T in time $O(\epsilon^{-1} m \ln n)$ such that $\langle uu^T \rangle_y \leq (1+\epsilon)\mu(y)$ holds with high probability.

3.3.5 Correctness

By Lemma 8 we know $\lambda^{-1} \leq 5$ for each feasible solution (X, λ). Since α is set to be $10\epsilon^{-1}\ln(2n\epsilon^{-1})$ in $\textsc{Improve}(X, \lambda, \epsilon)$, we know $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2n\epsilon^{-1})$ always holds, and thus (P1) holds by Lemma 13, during the execution of $\textsc{Improve}(X, \lambda, \epsilon)$. Therefore the correctness of the algorithm relies on the termination of $\textsc{Improve}$. Clearly $e^{\alpha\lambda^*} \leq y \cdot \vec{1} \leq ne^{\alpha\lambda}$ where λ^* is the optimal value and λ is the initial value. The following lemma, which is adapted from [122], ensures that $\textsc{Improve}(X, \lambda, \epsilon)$ terminates in a finite number of iterations.

Lemma 15 Suppose $0 < \epsilon \leq 1$. Let (X, λ) be a feasible solution. Let α, σ, y be as defined in $\textsc{Improve}(X, \lambda, \epsilon)$. Let $\tilde{X}$ be a feasible matrix (not necessarily satisfying (3.10)). Suppose (P2) is not satisfied by (X, λ), $\tilde{X}$ and y. Define a new feasible matrix $\hat{X}$ to be $(1 - \sigma)X + \sigma\tilde{X}$. Let $\hat{y}$ be defined by $\hat{y}_i = e^{\alpha\hat{X}_{ii}}$. Then $y \cdot \vec{1} - \hat{y} \cdot \vec{1} \geq \alpha\sigma\epsilon\lambda y \cdot \vec{1}$.

(For the completeness of the thesis, we include the proof in Appendix A.) We then have the following lemma.

Lemma 16 The algorithm $\textsc{VectorMaxCut}(C, \epsilon)$ obtains an ϵ-optimal solution to (3.7) with high probability.

3.3.6 Time complexity

Without loss of generality we may assume that $m \geq n$. Clearly the time required by each call to $\textsc{Improve}(X, \lambda, \epsilon)$ is dominated by the time required by obtaining the directions $\tilde{X}$. By Lemma 14 each $\tilde{X}$ can be obtained in time $O(m\epsilon^{-1}\ln n)$. We will show that the number of iterations required by each call to $\textsc{Improve}(X, \lambda, \epsilon)$ is $O(\epsilon^{-3}n\ln n)$. Moreover if (X, λ) is already $O(\epsilon)$-optimal, the number of iterations required by each call to $\textsc{Improve}(X, \lambda, \epsilon)$ is reduced to $O(\epsilon^{-2}n\ln n)$. Note that the precision parameter ϵ given to $\textsc{Improve}(X, \lambda, \epsilon)$ is reduced by a factor of two in each iteration of $\textsc{VectorMaxCut}$. Therefore the result of the previous call to $\textsc{Improve}$ is an $O(\epsilon)$-optimal initial solution for the current call to $\textsc{Improve}$ with high probability. It follows that the running time of $\textsc{VectorMaxCut}$ is $\tilde{O}(\epsilon^{-3}mn)$.

Lemma 17 Let λ^* be the value of the optimal solution to (3.7). Then the number of iterations required by $\textsc{Improve}(X, \lambda_0, \epsilon)$ is $O((1 + \frac{\lambda_0/\lambda^* - 1}{\epsilon})\epsilon^{-2}n\ln n)$.

Proof Since $\sigma = \frac{\epsilon}{4\alpha n}$ and $\lambda \geq \lambda^*$, we know that $\alpha\sigma\epsilon\lambda \geq \frac{\epsilon^2\lambda^*}{4n}$. Let N be the number of iterations required by the algorithm. Note that during the execution of the algorithm we have $e^{\alpha\lambda^*} \leq y \cdot \vec{1} \leq ne^{\alpha\lambda_0}$. It follows from Lemma 15 that

$$ne^{\alpha\lambda_0}(1 - \frac{\epsilon^2\lambda^*}{4n})^N \geq e^{\alpha\lambda^*}.$$

Therefore

$$\begin{aligned} N &\leq \frac{(\lambda^* - \lambda)\alpha - \ln n}{\ln(1 - \epsilon^2\lambda^*/(4n))} \\ &= O(\frac{(\lambda - \lambda^*)\epsilon^{-1}\ln n + \ln n}{\epsilon^2\lambda^*/n}). \end{aligned}$$

The lemma follows. $\square$

We already show that the initial solution (X, λ) obtained by INITIAL has value at most $1/3$, so its corresponding $\langle X\rangle_y$ is at most $ne^{\alpha/3}$. Lemma 15 ensures that $\langle X\rangle_y$ never increases during the execution of the algorithm. Therefore the inequality

$$e^{\alpha\lambda} \leq \langle X\rangle_y \leq ne^{\alpha/3}$$

always holds for any current solution (X, λ), which implies that $\lambda \leq 1/2$. It follows from Lemma 8 that $\lambda_0/\lambda^* = O(1)$. Hence Lemma 17 ensures that the number of iterations required by each call to IMPROVE(X, λ, ϵ) is $O(\epsilon^{-3}n \ln n)$. Moreover if $\lambda_0/\lambda^* = 1 + O(\epsilon)$, then the number of iterations is reduced to $O(\epsilon^{-2}n \ln n)$. Therefore we have the following lemma.

Lemma 18 The time complexity of the algorithm VECTORMAXCUT(C, ϵ) is $\tilde{O}(\epsilon^{-3}mn)$.

3.3.7 Space complexity

As described in §3.1.1, the algorithm IMPROVE does not maintain every element of the current solution matrix in the work space. One can see that the only elements of the current solution matrix required during the execution of IMPROVE are the diagonal elements. Therefore the algorithm just maintains the diagonal elements of the current solution in the work space. For every rank-one direction uu^T output by DIRECTION, the algorithm simply stores the vector u on the file server. Therefore the required work space is only $O(m)$, which is dominated by the space required by the matrices L and L'. Since the number of iterations is $\tilde{O}(\epsilon^{-2}n)$, as shown in the previous subsection, the space complexity of VECTORMAXCUT is $\tilde{O}(\epsilon^{-2}n^2)$.

If we are only interested in the value of the semidefinite relaxation of MAX-CUT for the given graph, we then do not have to spend the space on those vectors u. Similarly, if we are interested in only $O(m)$ elements of the resulting solution matrix, and do not care about obtaining a matrix H such that $H^T H$ equals the final solution matrix, we do not have to spend the space on those vectors u, either. For both cases the space complexity is only $O(m)$. In §3.4 and §5 we will see the cases in which we need only $O(m)$ elements of the resulting solution matrix output by VectorMaxCut. We summarize the above discussion in the following lemma.

Lemma 19 The space complexity of VectorMaxCut is $O(\epsilon^{-2}n^2 \log n)$, but the work space required by VectorMaxCut is only $O(m)$. Moreover, the space complexity is reduced to $O(m)$ if

- VectorMaxCut has to output only the value of the final solution matrix, or
- VectorMaxCut has to output only $O(m)$ elements of the final solution matrix.

3.3.8 The power method

It remains to describe Direction(y, ϵ), which produces a feasible matrix uu^T such that $\langle uu^T \rangle_y \leq (1+\epsilon)\mu(y)$ holds with high probability. We show that it is in fact an eigenvector problem to find a feasible matrix uu^T such that $\langle uu^T \rangle_y = \mu(y)$. The eigenvector problem can be approximately solved by a well-known numerical algorithm called the *power method* (see, e.g., §5.3 of [41]), which is one of the oldest methods for computing the dominant eigenpair of a matrix. Let L' be a matrix defined as

$$L'_{ij} = \frac{L_{ij}}{\sqrt{y_i y_j}}.$$

Since $L \succeq 0$, one can easily verify that $L' \succeq 0$. Since L contains $O(m)$ nonzero entries, so does L'. Let v be defined by $v_i = u_i \sqrt{y_i}$. Note that y is positive, so L' and v are both well-defined. Clearly $L \bullet (uu^T) = L' \bullet (vv^T)$ and $\langle uu^T \rangle_y = v \cdot v$. It follows from Lemma 11 that

$$\mu(y) = \min\{v \cdot v : L' \bullet (vv^T) = 1\}.$$

Therefore

$$1/\mu(y) = \max\{L' \bullet (ww^T) : w \cdot w = 1\}.$$

By Rayleigh-Ritz's theorem (see, e.g., Theorem 4.2.2 of [79]) we know $1/\mu(y)$ is the maximum eigenvalue of L'. A unit eigenvector w in the eigenspace of L' corresponding to $1/\mu(y)$ is a vector that maximizes $L' \bullet (ww^T)$ over all unit vectors. Let $v = \mu(y)w$, and let u be obtained from v and y by letting $u_i = v_i/\sqrt{y_i}$. By the above discussion we know that uu^T is a feasible solution such that $\langle uu^T \rangle_y = \mu(y)$. Clearly if w is close to the eigenspace of L' corresponding to $1/\mu(y)$, then the corresponding $L \bullet (uu^T)$ would be close to $\mu(y)$. Specifically if w is a unit vector such that

$$L' \bullet (ww^T) \geq \frac{1}{(1+\epsilon/13)\mu(y)}, \tag{3.11}$$

then the corresponding uu^T is a feasible matrix $\tilde{X}$ that satisfies (3.10). Therefore the problem of finding a feasible matrix $\tilde{X}$ that satisfies (3.10) is reduced to the following problem:

> Let λ be the maximum eigenvalue of a positive semidefinite matrix A, which has $O(m)$ nonzero entries. Find a unit vector x such that $A \bullet (xx^T) \geq (1+\epsilon)^{-1}\lambda$.

The power method consists of a series of k iterations, where k is a parameter to be determined. Each iteration consists of a matrix-vector multiplication. In particular, initialize by choosing a random unit vector $x_{(0)}$, and then compute

$$x_{(k)} = \frac{A^k x_{(0)}}{\|A^k x_{(0)}\|_2}$$

using the recurrence

$$x_{(i+1)} = \frac{Ax_{(i)}}{\|Ax_{(i)}\|_2}.$$

It is known that if $k = \Omega(\epsilon^{-1} \ln n)$ then $A \bullet (x_{(k)} x_{(k)}^T) \geq (1+\epsilon)^{-1}\lambda$ holds with high probability [102]. (For completeness of the thesis, we include an easier-to-read proof in Appendix B.) Since each iteration of the power method requires a matrix-vector multiplication, which takes time $O(m)$, Lemma 14 follows.

We would like to point out that Lanczos method (see, e.g., §9 of [77]) is an alternative to the power method for the above eigenvector problem. It is observed that Lanczos method outperforms the power method in practice (see, e.g., §9.1.5 of [77]). Based on exact arithmetic, Lanczos method requires only $O(\epsilon^{-1/2} \ln n)$ matrix-vector multiplications to obtain a vector x such that $A \bullet (xx^T) \geq (1+\epsilon)^{-1}\lambda$ [102]. Therefore if we use Lanczos method to implement Direction, then the running time of VectorMaxCut would be reduced by a factor of $\epsilon^{-1/2}$ in exact arithmetic. In practice, however, Lanczos method is not as stable as the power method. Therefore Lanczos method involves expensive reorthogonalization during its execution. Some researchers [47] argued that reorthogonalization is not really necessary for Lanczos method. Since this is not a general belief, we choose the power method to implement DIRECTION for a solid running-time analysis.

3.4 The Basic Algorithm for the Semidefinite Relaxation of COLORING

In this section we show how to obtain an $O(\epsilon)$-optimal solution to (3.5), the semidefinite relaxation of COLORING, which we restate here for convenience.

$$\begin{array}{lll} \min & \lambda & \\ \text{s.t.} & X_{ii} = 1 & \text{for every } 1 \leq i \leq n \\ & X_{ij} \leq \lambda & \text{for every edge } ij \text{ of } G \\ & X \succeq 0. & \end{array} \tag{3.12}$$

3.4.1 Definitions

A matrix Y is a *cost matrix for G* if Y is a nonnegative matrix such that $Y_{ij} = 0$ for every ij that is not an edge of G. Let $\mu(Y) = \min Y \bullet X$ over all feasible matrices X. Since (I, λ) is feasible and $Y \bullet I = 0$, we know that $\mu(Y) \leq 0$.

3.4.2 Bounding the values of feasible solutions

The following lemmas bound the value of any feasible solution, including the value of an optimal solution.

Lemma 20 Let (X, λ) be a feasible solution. Then $\lambda \leq 1$.

Proof By the feasibility of (X, λ) we know $X \succeq 0$, which implies that $u^T X u \geq 0$ for any vector u. If $X_{ij} > 1$ for some edge ij of G, then $u^T X u < 0$, where u is defined by

$$u_k = \begin{cases} 1 & \text{if } k = i \\ -1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

The lemma thus follows. $\square$

Lemma 21 Let Y be a cost matrix for G. Let (X, λ) be a feasible solution. Then $\lambda \geq \mu(Y)/Y \bullet J$.

Proof If ij is an edge of G, then $X_{ij} \leq \lambda$. If ij is not an edge of G, then $Y_{ij} = 0$. It follows that $\lambda Y \bullet J \geq Y \bullet X$. By definition of $\mu(Y)$ we know that $Y \bullet X \geq \mu(Y)$. Therefore $\lambda Y \bullet J \geq \mu(Y)$. Since Y is nonnegative, $Y \bullet J \geq 0$. The lemma thus holds. $\square$

3.4.3 Relaxed optimality conditions

Let ϵ be a positive precision parameter. Let (X, λ) be a feasible solution. Let $\tilde{X}$ be a feasible matrix such that

$$Y \bullet \tilde{X} \leq (1+\epsilon)^{-1}\mu(Y). \tag{3.13}$$

We define the following *relaxed optimality conditions*, which are adapted from [122].

(C1) $(1+\epsilon)\lambda Y \bullet J \leq Y \bullet X$

(C2) $Y \bullet X - Y \bullet \tilde{X} \leq -\epsilon(Y \bullet X + \lambda Y \bullet J)$

The following two lemmas are adapted from [122]. The first lemma shows that (C1) and (C2) together ensure the near optimality of (X, λ). The second lemma shows that (C1) is always satisfied if α is sufficiently large.

Lemma 22 Let Y be a cost matrix for G. Let $\tilde{X}$ be a matrix satisfying (3.13). Then (C1) and (C2) imply that (X, λ) is 4ϵ-optimal.

Lemma 23 Suppose $\alpha \geq -2\lambda^{-1}\epsilon^{-1}\ln(4n^2\epsilon^{-1})$. Let (X, λ) be a feasible solution. Let Y be a cost matrix for G defined as $Y_{ij} = e^{\alpha X_{ij}}$ for every edge ij of G. Then (C1) is satisfied.

For completeness of the thesis, we give the proof of the above two lemmas in Appendix A.

3.4.4 The algorithm

We give the basic algorithm that produces an ϵ-optimal solution to the semidefinite relaxation of COLORING. The algorithm starts with finding an initial solution (X, λ). It then iteratively updates (X, λ) until it is ϵ-optimal. In each iteration a procedure ImproveC is called to improve the quality of (X, λ). Specifically each call to ImproveC either reduces the value of the current solution by

Algorithm VECTORCOLORING(G, ϵ)

Input: G is the input graph

Output: an ϵ-optimal solution matrix X to G's semidefinite relaxation for graph COLORING.

1. Let $(X, \lambda) = $ INITIALC(G).
2. Let $\epsilon' = 2$.
3. While $\epsilon' > \epsilon$ do
   (a) Let $\epsilon' = \epsilon'/2$.
   (b) While (X, λ) is not ϵ'-optimal do
   Let $(X, \lambda) = $ IMPROVEC$(X, \lambda, \epsilon'/4)$.
4. Return X.

Figure 3.3: Algorithm VECTORCOLORING

a factor of two or yields a solution that is near-optimal with respect to the precision parameter given to IMPROVEC. The algorithm VECTORCOLORING(G, ϵ) is given in Figure 3.3.

The procedure INITIALC(G) obtains an initial solution (X, λ) as follows:

$$X_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\frac{1}{n} & \text{if } ij \text{ is an edge of } G \\ 0 & \text{otherwise.} \end{cases}$$

Clearly $X \succeq 0$ (see, e.g., Geršgorin's theorem in §10.6 of [103]). Thus the initial solution $(X, -\frac{1}{n})$ is feasible.

The procedure IMPROVEC(X, λ_0, ϵ) uses a nonnegative matrix Y defined as a function of the current solution (X, λ) such that (C1) is always satisfied. The procedure proceeds iteratively to update the current solution until (C2) is also satisfied. In each iteration a procedure DIRECTIONC(Y, ϵ) is called to obtain a feasible solution $(\tilde{X}, \tilde{\lambda})$ such that (3.13) holds with high probability. If (C2) is satisfied, the current solution (X, λ) is returned by the procedure. By Lemma 22 we know that the return value of the procedure is 4ϵ-optimal, as long as (3.13) holds. If (C2) is not satisfied, then the current solution is moved towards $\tilde{X}$ by a small amount σ. Namely let $X = (1 - \sigma)X + \sigma\tilde{X}$, and let λ be the new maximum X_{ij} over all edges ij of G. One can easily verify that the new (X, λ) is still feasible. In the following subsection we will show that the procedure always terminates in a finite number of iterations. The procedure

Algorithm IMPROVEC(X, λ_0, ϵ)

Input: X is the current solution matrix whose value is λ_0.

Output: a solution matrix X whose value is either ϵ-optimal or at most one-half of λ_0.

1. Let $\lambda = \lambda_0$.
   Let $\alpha = -4\epsilon^{-1}\lambda_0^{-1}\ln(4n^2\epsilon^{-1})$.
   Let $\sigma = \epsilon/(4\alpha)$.
2. Repeat
   (a) Let $Y_{ij} = e^{\alpha X_{ij}}$ for every edge ij of G.
   (b) Let $\tilde{X} = $ DIRECTIONC(Y, ϵ).
   (c) If $\lambda \leq 2\lambda_0$ or (C2) is satisfied then
   Return (X, λ).
   else
   Let $X = (1-\sigma)X + \sigma\tilde{X}$.
   Let $\lambda = \max\{X_{ij} : ij \text{ is an edge of } G\}$.
3. Return (X, λ).

Figure 3.4: Algorithm IMPROVEC

IMPROVEC(X, λ_0, ϵ) is given in Figure 3.4.

It remains to describe DIRECTIONC(Y, ϵ), which produces a feasible matrix $\tilde{X}$ that satisfies (3.13). It is interesting that finding a feasible solution (X, λ) that achieves $\mu(Y)$ is equivalent to solving the semidefinite relaxation of MAXCUT for some graph with nonnegative edge weights. The following lemma shows how to use the VECTORMAXCUT procedure to implement DIRECTIONC.

Lemma 24 Suppose G is k-colorable, where $k \geq 2$. Let $(\tilde{X}, \tilde{\lambda})$ be the output of VECTORMAXCUT$(Y, \frac{\epsilon}{6k})$, where $\epsilon \leq 1$. Then $\tilde{X}$ satisfies (3.13) with high probability.

Proof By Lemma 16 and Lemma 6 we know that with high probability $(\tilde{X}, \tilde{\lambda})$ is an $\frac{\epsilon}{3k}$-optimal solution to the semidefinite relaxation of MAXCUT for the graph specified by the cost matrix Y. Assume $(\tilde{X}, \tilde{\lambda})$ is $\frac{\epsilon}{3k}$-optimal. We show that $\tilde{X}$ satisfies (3.13).

Let (X, λ) be an optimal solution to the semidefinite relaxation of COLORING, i.e., the mathematical program (3.12). Since G is k-colorable, it follows from the results in [86] that

$$\begin{aligned}\lambda &\leq \frac{1}{1-k} \\ &\leq -\frac{1}{k}.\end{aligned}$$

By the feasibility of (X, λ) we know that

$$Y \bullet X \leq -\frac{1}{k} Y \bullet J. \tag{3.14}$$

Let X^* be an optimal solution to MAXCUT's semidefinite relaxation (3.4) corresponding to the cost matrix Y. Clearly $\mu(Y) = Y \bullet X^*$. Since X is also feasible for (3.4), it follows that

$$-Y \bullet X^* \geq -Y \bullet X. \tag{3.15}$$

By the assumption we know

$$Y \bullet (J - \tilde{X}) \geq (1 + \frac{\epsilon}{3k})^{-1} Y \bullet (J - X^*),$$

and thus

$$\begin{aligned}-(1 + \frac{\epsilon}{3k}) Y \bullet \tilde{X} &\geq -\frac{\epsilon}{3k} Y \bullet J - Y \bullet X^* \\ &\geq -(1 - \frac{\epsilon}{3}) Y \bullet X^*,\end{aligned}$$

using (3.14) and (3.15). Since $\epsilon \leq 1$, we know that

$$(1 - \frac{\epsilon}{3})(1 + \frac{\epsilon}{3k})^{-1} \geq (1 + \epsilon)^{-1}.$$

Therefore $-Y \bullet \tilde{X} \geq -(1+\epsilon)^{-1} Y \bullet X^*$. The lemma follows because $Y \bullet X^* = \mu(Y)$. $\square$

3.4.5 Correctness

We know λ is always no less than $2\lambda_0$ during the execution of ImproveC(X, λ_0, ϵ). Note that α is set to be $-4\epsilon^{-1}\lambda_0^{-1} \ln(4n^2\epsilon^{-1})$ in ImproveC(X, λ_0, ϵ). It follows that $\alpha \geq -2\lambda^{-1}\epsilon^{-1} \ln(4n^2\epsilon^{-1})$ always holds, and thus (C1) holds by Lemma 23, during the execution of ImproveC(X, λ_0, ϵ). Therefore the correctness of the algorithm relies on the termination of ImproveC. Clearly $e^{\alpha\lambda^*} \leq Y \bullet J \leq n^2 e^{\alpha\lambda_0}$, where λ^* is the optimal value and λ_0 is the initial value. The following lemma, which is an easy adaption of Lemma 3.3 in [122], ensures that ImproveC(X, λ_0, ϵ) terminates in a finite number of iterations.

Lemma 25 Suppose $0 < \epsilon \leq 1$. Let (X, λ) be a feasible solution. Let α, σ, Y be as defined in $\textsc{Improve}C(X, \lambda_0, \epsilon)$. Let $\tilde{X}$ be a feasible matrix (not necessarily satisfying (3.13)). Suppose (C2) is not satisfied by (X, λ), $\tilde{X}$ and Y. Define a new feasible matrix $\hat{X}$ to be $(1-\sigma)X + \sigma\tilde{X}$. Let $\hat{Y}$ be defined by

$$Y_{ij} = \begin{cases} e^{\alpha X_{ij}} & \text{if } ij \text{ is an edge of } G \\ 0 & \text{otherwise.} \end{cases}$$

Then $Y \bullet J - \hat{Y} \bullet J \geq -\alpha\sigma\epsilon\lambda Y \bullet J$.

(For completeness of the thesis, we include the proof of the above lemma in Appendix A.) We have the following lemma.

Lemma 26 Algorithm $\textsc{VectorColoring}(G, \epsilon)$ obtains an ϵ-optimal solution to (3.12) with high probability.

3.4.6 Time complexity

Clearly the time required by each call to $\textsc{Improve}C(X, \lambda_0, \epsilon)$ is dominated by the time required by obtaining the directions $\tilde{X}$. It follows from Lemma 18 that each $\tilde{X}$ can be obtained in time $\tilde{O}(mn\epsilon^{-3}k^3)$ if G is k-colorable. We show that the number of iterations required by each call to $\textsc{Improve}C(X, \lambda_0, \epsilon)$ is $\tilde{O}(\epsilon^{-2})$. Since ϵ' is reduced by a factor of two, each time the current solution is ϵ'-optimal with high probability. It follows that the running time required by all calls to $\textsc{Improve}$ is dominated by that required by the last call. Therefore the running time required by $\textsc{VectorColoring}(G, \epsilon)$ is $\tilde{O}(mn\epsilon^{-5}k^3)$.

Lemma 27 Let λ^* be the value of the optimal solution to (3.12). Then the number of iterations required by $\textsc{Improve}C(X, \lambda_0, \epsilon)$ is $O((1 + \frac{1-\lambda'/\lambda_0}{\epsilon})\epsilon^{-2}\ln n)$, where $\lambda' = \max\{\lambda^*, \frac{\lambda_0}{2}\}$.

Proof By Lemma 25 we know that $e^{\alpha\lambda} \leq n^2 e^{\alpha\lambda_0}$. One can easily verify that $\lambda \leq \lambda_0$. Since $\sigma = \frac{\epsilon}{4\alpha}$, it follows that $-\alpha\sigma\epsilon\lambda \geq \frac{-\epsilon^2\lambda_0}{4}$. Let N be the number of iterations required by the algorithm. Note that during the execution of the algorithm we have $e^{\alpha\lambda'} \leq Y \bullet J \leq n^2 e^{\alpha\lambda_0}$. It follows from Lemma 25 that

$$n^2 e^{\alpha\lambda_0}(1 + \frac{\epsilon^2\lambda_0}{4})^N \geq e^{\alpha\lambda'}.$$

Therefore

$$\begin{aligned} N &\leq \frac{(\lambda' - \lambda_0)\alpha - \ln 2n}{\ln(1 + \epsilon^2\lambda_0/4)} \\ &= O(\frac{(\lambda_0 - \lambda')\epsilon^{-1}\ln n + \ln n}{-\lambda_0\epsilon^2}). \end{aligned}$$

The lemma follows. $\square$

Note that the algorithm starts with letting $\epsilon' = O(1)$. In the first $O(\ln n)$ iterations, each call to IMPROVEC reduces the current value λ by a factor of two in time $O(n \ln n)$ by Lemma 27. From then on the initial solution is $O(\epsilon')$-optimal with high probability. It follows that each call to IMPROVEC(X, λ_0, ϵ) requires $\tilde{O}(\epsilon^{-2})$ iterations. Therefore we have the following lemma.

Lemma 28 The running time required by the algorithm VECTORCOLORING(G, ϵ) is $\tilde{O}(\epsilon^{-5}k^3mn)$ if G is k-colorable.

3.4.7 Space complexity

As explained in the previous subsection, the algorithm VECTORCOLORING makes $\tilde{O}(\epsilon^{-2})$ calls to the algorithm VECTORMAXCUT. For the same reason as we explained in §3.3.7, we store all the intermediate data, i.e. the output of each call to VECTORMAXCUT, on the file server. During the execution of VECTORCOLORING, we maintain only the ij-entry of the current solution matrix, for every edge ij of the graph, in the work space. Although the required space is $\tilde{O}(\epsilon^{-4}k^2n^2)$, the required work space is only $O(m)$. Moreover, if we are interested in only the value of a near-optimal solution to the semidefinite relaxation of COLORING, then exactly m elements of the solution matrix need be returned by each call to VECTORMAXCUT. Therefore the required space is reduced to $O(m)$ by Lemma 19. In Chapter 5 we will see the cases in which we are interested only the value of the near-optimal solution to COLORING's semidefinite relaxation. We summarize the space complexity of VECTORCOLORING in the following lemma.

Lemma 29 The space complexity of VECTORCOLORING is $O(\epsilon^{-4}n^2k^2\log^2 n)$, but the work space required by VECTORCOLORING is only $O(m)$. Moreover, the space complexity is reduced to $O(m)$ if VECTORCOLORING has to output only the value of the final solution.

Part II

Extensions

Chapter 4

Applications to the Approximation Algorithms of MAXCUT and COLORING

In this chapter we show how our basic algorithms can be applied to the best known approximation algorithms for MAXCUT and COLORING.

4.1 Overview

4.1.1 Application to the approximation algorithm of MAXCUT

Goemans and Williamson's approximation algorithm for MAXCUT has the following three steps:

1. Obtain an ϵ-optimal solution X to the semidefinite relaxation of MAXCUT.

2. Compute a Cholesky decomposition for X, i.e., obtain a matrix H such that $H^T H = X$. Let $h^{(i)}$ be the i^{th} column of H.

3. Randomly choose a vector z, and then obtain a cut as follows: if $z \cdot h^{(i)} \geq 0$, put the i^{th} node on the left-hand side; otherwise, put the i^{th} node on the right-hand side.

Goemans and Williamson show that the resulting cut is $0.878/(1+\epsilon)$-optimal with high probability. Clearly our basic algorithm for the semidefinite relaxation of MAXCUT is only for the first step. In this chapter, however, we show

how to modify our basic algorithm so that the above three steps can be done incrementally. Roughly speaking, our basic algorithm determines one component of each vector in $\{z, h^{(1)}, \ldots, h^{(n)}\}$ in each iteration. Therefore each dot-product $z \cdot h^{(i)}$ is known up to the j^{th} term after j iterations. As soon as our basic algorithm finds an ϵ-optimal solution to the semidefinite relaxation of MAXCUT, all the dot-products $z \cdot h^{(i)}$ are computed. A near-optimal cut can thus be easily obtained. As a result, our basic algorithm does not have to store those rank-one directions on the file server. Therefore the space complexity is reduced to $O(m)$, which is the same as the required work space.

4.1.2 Application to the approximation algorithm of COLORING

Karger, Motwani, and Sudan's approximation algorithm for COLORING has the following three steps, where the first two steps are similar to the first two steps of Goemans and Williamson's algorithm.

1. Obtain an ϵ-optimal solution X to the semidefinite relaxation of COLORING.

2. Compute a Cholesky decomposition for X, i.e. obtain a matrix H such that $H^T H = X$. Let $h^{(i)}$ be the i^{th} column of H, which corresponds to the *vector color* assigned to the i^{th} node of the input graph.

3. Randomly choose t vectors $z^{(1)}, \ldots, z^{(t)}$, and then obtain an assignment of colors as follows: color the i^{th} node with color s if $z^{(s)}$ is the vector among $z^{(1)}, \ldots, z^{(t)}$ that has the largest dot-product with $h^{(i)}$.

Suppose the input graph is k-colorable. Let Δ be the maximum degree of the input graph. Karger, Motwani, and Sudan show that if $t = \Delta^{1-\frac{2}{k(1+\epsilon)}+o(1)}$, then the resulting color assignment is a legal t-coloring with high probability. Although our basic algorithm for the semidefinite relaxation of COLORING is only for the first step, it can be modified such that the above three steps are done incrementally. Note that our basic algorithm for COLORING's semidefinite relaxation repeatedly call our basic algorithm for MAXCUT's semidefinite relaxation. Roughly speaking, we determine one component of each vector in $\{z^{(1)}, \ldots, z^{(t)}\} \cup \{h^{(1)}, \ldots, h^{(n)}\}$ in each iteration of our basic algorithm for MAXCUT's semidefinite relaxation. Therefore each dot-product $z^{(s)} \cdot h^{(i)}$ is known up to the j^{th} term after j iterations. As soon as a near-optimal solution to COLORING's semidefinite relaxation is obtained, all the dot-products $z^{(s)} \cdot h^{(i)}$ are computed. A color assignment can therefore be easily obtained. As a result, our basic algorithm does not have to store those rank-one directions on the file server. The space complexity is thus reduced to $O(m + nt)$. The required work space is still $O(m)$.

4.2 The Approximation Algorithm for MAXCUT

We show in the previous chapter that our basic algorithm for MAXCUT's semidefinite relaxation starts with finding an initial rank-one matrix uu^T, and then proceeds iteratively to improve its quality. A rank-one update $\tilde{u}\tilde{u}^T$ is applied to the current matrix in each iteration. Therefore the resulting ϵ-optimal solution X can be written as

$$X = u^{(1)}u^{(1)^T} + \cdots + u^{(r)}u^{(r)^T},$$

where $u^{(i)}u^{(i)^T}$ is some scalar times the rank-one matrix obtained from the i^{th} iteration, and r is the number of executed iterations. Let U be the matrix whose i^{th} column is $u^{(i)}$. One can easily verify that

$$X = UU^T.$$

As mentioned in §4.1.1, the approximation algorithm of Goemans and Williamson requires that we compute a matrix H such that $H^T H$ equals X. An evident choice of H is U^T. It follows that our basic algorithm obtains the j^{th} row of the matrix H in the j^{th} iteration.

Once the matrix H is obtained, the algorithm of Goemans and Williamson assigns the i^{th} column $h^{(i)}$ of H to the i^{th} node of the input graph. The resulting near-optimal cut is determined by the signs of the dot-products $z \cdot h^{(i)}$, where z is a random vector obtained from the r-dimensional unit sphere. It is known that z can be obtained by independently choosing each component z_i from the standard normal distribution and normalizing the resulting vector z (see, e.g., [99, p. 130]). For our purpose, however, there is no need to normalize the resulting vector z. Therefore we can determine z_j in the j^{th} iteration of our basic algorithm. Since the j^{th} row of H is determined in the j^{th} iteration, the j^{th} component of each vector $h^{(i)}$ is known in the j^{th} iteration. It follows that we can compute each dot-product $z \cdot h^{(i)}$ up to the j^{th} term after j iterations.

Algorithm NearOptCut(C, ϵ)

Input: C is the matrix such that C_{ij} is the weight of edge ij of the input graph.

Output: a cut of the input graph.

1. Let $t = 1$.
2. Scale C such that the sum of the absolute values of the edge costs is one, and then compute L from C.
3. Let $(X, \lambda) = (uu^T, \lambda) = \text{Initial}(C)$.
   Let $p = \text{Projection}(u, t)$.
4. Let $\epsilon' = 2/3$.
5. While $\epsilon' > \epsilon$ do
   (a) Let $\epsilon' = \epsilon'/2$.
   (b) Let $(X, \lambda, p) = \text{ModifiedImprove}(X, \lambda, \epsilon'/4, p, t)$.
6. Return the set $\{$the i^{th} node of the input graph : $p_i \geq 0\}$.

Figure 4.1: Algorithm NearOptCut

Since the resulting dot-products are all we need to obtain a near-optimal cut, our basic algorithm does not need to store all those rank-one directions on the file server. Therefore the required space is reduced to $O(m)$, which is also the required work space. The time complexity is still $O(\epsilon^{-3} mn \log^2 n)$.

We give the algorithm for finding a near-optimal cut in Figure 4.1, Figure 4.2, and Figure 4.3. Note that all the matrices X and $\tilde{X}$ in NearOptCut and ModifiedImprove keep only their diagonal entries.

4.3 The Approximation Algorithm of COLORING

We show in the previous chapter that our basic algorithm for COLORING's semidefinite relaxation starts with finding a rank-one matrix solution uu^T, and then proceeds iteratively. Each iteration a subroutine DirectionC is called to obtain a direction matrix $\tilde{X}$, which is used to guide the current solution towards the optimal solution. The subroutine DirectionC can be implemented by VectorMaxCut. Since the resulting near-optimal solution matrix X is a linear combination of the initial matrix uu^T and those direction matrices $\tilde{X}$, we can write X as

$$X = u^{(1)} u^{(1)^T} + \cdots + u^{(r)} u^{(r)^T},$$

Algorithm PROJECTION(u, t)

Input: u corresponds to one row of H, where each column $h^{(i)}$ of H corresponds to the vector assigned to a node.

Output: The s^{th} column of p corresponds to one term of the dot-products $z^{(s)} \cdot h^{(1)}, \ldots, z^{(s)} \cdot h^{(n)}$.

1. For every $s = 1, \ldots, t$ do
   (a) Let the s^{th} column of p be γu, where γ is a random number obtained from the standard normal distribution.

Figure 4.2: Algorithm PROJECTION

Algorithm MODIFIEDIMPROVE$(X, \lambda_0, \epsilon, p, t)$

Input: (X, λ_0) is the current solution, and p keeps the current dot-products $z^{(s)} \cdot h^{(i)}$, for every $1 \leq s \leq t$, $1 \leq i \leq n$.

Output: (X, λ, p), where (X, λ) is an ϵ-optimal solution matrix X to the semidefinite relaxation of MAXCUT, and p maintains the current dot-products.

1. Let $(\epsilon, \lambda, \alpha, \sigma) =$ SETPARAMETERS(λ_0, ϵ), which is exactly the first step of IMPROVE.
2. Repeat
   (a) Let $y_i = e^{\alpha X_{ii}}$ for every $i = 1, \ldots, n$.
   (b) Let $\tilde{X} = uu^T =$ DIRECTION$(y, \epsilon/13)$.
   (c) If (P2) is not satisfied then
         let $X = (1-\sigma)X + \sigma\tilde{X}$.
         let $\tilde{p} =$ PROJECTION(u, t).
         let $p = \sqrt{1-\sigma}p + \sqrt{\sigma}\tilde{p}$.
         let $\lambda = \max\{X_{11}, \ldots, X_{nn}\}$.
       else
         return (X, λ, p).

Figure 4.3: Algorithm MODIFIEDIMPROVE

where r is the total number of subroutine calls to IMPROVE in all those subroutine calls to VECTORMAXCUT. Let U be the matrix whose i^{th} column is $u^{(i)}$. Clearly

$$X = UU^T.$$

The algorithm of Karger, Motwani, and Sudan also requires that we compute a matrix H such that $X = H^T H$. Clearly we can let $H = U^T$. Therefore in the j^{th} iteration of those subroutine calls to VECTORMAXCUT, the j^{th} rows of the matrix H is obtained.

Once the matrix H is obtained, the algorithm of Karger, Motwani, and Sudan assigns the i^{th} column $h^{(i)}$ of the matrix H to the i^{th} node of the input graph. The algorithm then computes t random vectors $z^{(1)}, \ldots, z^{(t)}$ from r-dimensional space. The i^{th} node is assigned color s if $z^{(s)}$ is the vector in $\{z^{(1)}, \ldots, z^{(t)}\}$ that has the largest dot-product with $h^{(i)}$. As described in [86], each $z^{(s)}$ can be obtained by independently choosing each of its components from the standard normal distribution (without normalization). Therefore our basic algorithm can determine the j^{th} component of each $z^{(s)}$ in the j^{th} iteration. Since the j^{th} row of H is known in the j^{th} iteration, we can compute each dot-product $z^{(s)} \cdot h^{(i)}$ up to the j^{th} term in the j^{th} iteration. Since the dot-products are all we need to obtain a near-optimal coloring at the completion of execution, our basic algorithm does not have to store those rank-one directions on the file server. During the execution of our algorithm, only m elements of the current solution matrix matter. Therefore the space complexity is reduced to $O(m+tn)$, where $O(tn)$ is used to maintain the dot-products. Since $t = O(\Delta)$, where Δ is the maximum degree of the input graph, the space complexity is $O(m + n\Delta)$. The time complexity is still $O(\epsilon^{-5} k^3 mn \log^3 n)$.

Note that we do not have to keep those dot-products $z^{(s)} \cdot h^{(i)}$ in the work space. Instead, we can store them on the file server, as we did to those rank-one direction matrices in the previous chapter. At the end of each iteration, the dot-products can be updated in t iterations. Specifically, the values of $z^{(s)} \cdot h^{(1)}, \ldots, z^{(s)} \cdot h^{(n)}$ are updated in the s^{th} iteration. That way we keep the work space small. Therefore the work space of the algorithm is still $O(m)$.

We give the algorithm for finding a near-optimal coloring in Figure 4.4, Figure 4.5, and Figure 4.6. Note that all the matrices X and $\tilde{X}$ in NEAROPT-COLORING, MODIFIEDIMPROVEC, and MODIFIEDDIRECTIONC keep only their ij-entries, for every edge ij of the graph.

Algorithm NearOptColoring(G, ϵ)

Input: G is the input k-colorable graph. Let Δ be the maximum degree of G. Let t be $\Delta^{1-\frac{3}{k(1+\epsilon)}+o(1)}$.

Output: a t-color assignment for the nodes of G. The color assignment is likely to be a legal t-coloring of the input graph.

1. Let $(X, \lambda) = (uu^T, \lambda) = \text{InitialC}(G)$.
   Let $p = \text{Projection}(u, t)$.
2. Let $\epsilon' = 2$.
3. While $\epsilon' > \epsilon$ do
   (a) Let $\epsilon' = \epsilon'/2$.
   (b) While (X, λ) is not ϵ'-optimal do
   Let $(X, \lambda, p) = \text{ModifiedImproveC}(X, \lambda, \epsilon'/4, p, t)$.
4. For every $i = 1, \ldots, n$ do
   (a) Assign color s to the i^{th} node of the input graph if $p_i^{(s)} \geq p_i^{(s')}$ for every $1 \leq s' \leq t$.

Figure 4.4: Algorithm NearOptColoring

Algorithm ModifiedImproveC($X, \lambda_0, \epsilon, p, t$)

Input: (X, λ_0) is the current solution. p keeps the current dot-products $z^{(s)} \cdot h^{(i)}$ for every $1 \leq s \leq t$, $1 \leq i \leq n$.

Output: (X, λ, p), where X is the resulting matrix whose value λ is either ϵ-optimal to the semidefinite relaxation of COLORING or at most $2\lambda_0$, and p keeps the current dot-products.

1. Let $(\lambda, \alpha, \sigma) = \text{SetParamtersC}(\epsilon, \lambda_0)$, which is exactly the first step of ImproveC.
2. Repeat
   (a) Let $Y_{ij} = e^{\alpha X_{ij}}$ for every edge ij of G.
   (b) Let $(\tilde{X}, \tilde{p}) = \text{ModifiedDirectionC}(Y, \epsilon/(6k), t)$.
   (c) If $\lambda \leq 2\lambda_0$ or (C2) is satisfied then
   Return (X, λ, p).
   else
   Let $X = (1-\sigma)X + \sigma\tilde{X}$.
   Let $p = \sqrt{1-\sigma}p + \sqrt{\sigma}\tilde{p}$.
   Let $\lambda = \max\{X_{ij} : ij \text{ is an edge of } G\}$.

Figure 4.5: Algorithm ModifiedImproveC

Algorithm ModifiedDirectionC(C, ϵ, t)

Input: C is the matrix such that C_{ij} is the weight of edge ij of the input graph.

Output: (X, p), where X is an ϵ-optimal direction for ModifiedImproveC, and p, which has t columns, keeps the corresponding dot-products.

1. Execute Step 2–5 of NearOptCut(C, ϵ).
2. Return $(X/\lambda, p/\sqrt{\lambda})$.

Figure 4.6: Algorithm ModifiedDirectionC

Chapter 5

Applications to Some Perfect-graph Algorithms

In this chapter we show how our basic algorithms for COLORING's semidefinite relaxation can be applied to reduce the time and space complexity of the only known polynomial-time algorithms for some optimization problems on perfect graphs, if the chromatic number of the given graph is a constant.

The only known polynomial-time algorithms for some optimization problems on perfect graphs are based on the observation that Lovász's theta function [106] can be computed in polynomial time [66]. It follows from a result in the journal version of [86] that the clique number of a perfect graph is strongly related to the optimal value of its semidefinite relaxation of COLORING. Therefore we can use our basic algorithm for COLORING's semidefinite relaxation to reduce the time and space complexity of these only known polynomial-time algorithms for perfect graphs if the chromatic number of the given perfect graph is a constant.

5.1 Lovász's theta function

Given a graph G with n nodes and m edges, A matrix A is *feasible* for G if

- A is a real and symmetric $n \times n$ matrix,
- $A_{ii} = 1$ for every $i = 1, \ldots, n$, and
- $A_{ij} = 1$ for every $ij \notin G$.

Since a feasible matrix A for G is symmetric, every eigenvalue of A is real. We use $\lambda_{\max}(A)$ to denote the largest eigenvalue of matrix A. Clearly $\lambda_{\max}(A)$ is well-defined for every feasible matrix A for G. Lovász's theta function $\vartheta(G)$, can be defined as follows.

$$\vartheta(G) = \min\{\lambda_{\max}(A) : A \text{ is feasible for } G\}.$$

In fact there are five equivalent definitions for Lovász's theta function. The above definition is the definition $\vartheta_2(G)$, as described in [68, Equation (9.3.9) in page 287] and [100, Definition (6.3) in page 9]. Lovász [106] introduced this function to approximate the *Shannon capacity* [140, 135, 115] of a graph.

Clearly the theta function of G can be written as a semidefinite program as follows.

$$\begin{array}{lll} \min & \lambda & \\ \text{s.t.} & A_{ii} = 1 & \text{for every } i = 1, \ldots, n \\ & A_{ij} = 1 & \text{for every } ij \notin G \\ & \lambda I - A \succeq 0. & \end{array}$$

Therefore $\vartheta(G)$ is a polynomial-time computable function. Let $\bar{G}$ be the complement graph of G. Clearly $\vartheta(\bar{G})$ is also a polynomial-time computable function.

Let $\omega(G)$ be the size of the maximum clique in G. Let $\chi(G)$ be the chromatic number of G. Lovász [106] showed that

$$\omega(G) \leq \vartheta(\bar{G}) \leq \chi(G),$$

for any graph G, which is called the *Sandwich Theorem* by Lovász [107]. A graph G is *perfect* if

$$\omega(G') = \chi(G'),$$

for any induced subgraph G' of G [22, 23]. Therefore both $\omega(G)$ and $\chi(G)$ are polynomial-time computable, if G is a perfect graph.

5.2 Two Polynomial-time Algorithms for Perfect Graphs

5.2.1 MAXCLIQUE

Let v be node of G. Let G_v be the *induced subgraph* of G obtained from G by removing v and all the edges that are incident on v. Based on the observation that $\omega(G') = \vartheta(G')$ for any induced subgraph of a perfect graph G, Grötschel, Lovász, and Schrijver [66] gave the only known polynomial-time algorithm for obtaining a maximum clique in a perfect graph. We show the algorithm in Figure 5.1.

Algorithm $\textsc{MaxClique}(G)$

Input: a perfect graph G

Output: a maximum clique of G

1. Let $C = \{\}$.
2. Compute $\omega = \omega(G)$.
3. For every node v in G
   (a) Compute $\omega' = \omega(G_v)$.
   (b) If $\omega = \omega'$ then
         Let $G = G_v$,
       else
         Let $C = C \cup \{v\}$.
4. Return C.

Figure 5.1: Algorithm MaxClique

When Grötschel, Lovász, and Schrijver [67] presented the above algorithm and other only known polynomial-time algorithms for perfect graphs, the only choice for solving semidefinite programs was the ellipsoid method, whose time complexity is high. Therefore they wrote

> Our analysis of these problems should be viewed as a theoretical contribution showing that certain programming problems for perfect graphs are indeed polynomially solvable. Future research should be directed toward finding practically good algorithms for these problems. These algorithms should have a more combinatorial nature and should not suffer from the numerical instability (due to our present-day computer technology of fixed precision arithmetic)...

However, recent development of the interior-point algorithms for solving semidefinite programs has made the above algorithm more practical than it was. Suppose G has n nodes and m edges. Note that in each of those n iterations of MaxClique, a $\vartheta(G')$ is computed, where the number of edges in G', which is an induced subgraph of G, is at most m. By the above discussion, we know

Algorithm MaxStableSet(G)

Input: a perfect graph G

Output: a maximum stable set of G

1. Return MaxClique($\bar{G}$).

Figure 5.2: Algorithm MaxStableSet

$\omega(G') = \vartheta(\bar{G}')$ is equal to the optimum value of the following semidefinite program.

$$\begin{array}{lll} \min & \lambda & \\ \text{s.t.} & A_{ii} = 1 & \text{for every } i = 1, \ldots, n \\ & A_{ij} = 1 & \text{for every } ij \in G' \\ & \lambda I - A \succeq 0. & \end{array}$$

With simple change of variables, we can rewrite the program as

$$\begin{array}{lll} \min & \lambda & \\ \text{s.t.} & X_{ii} = \lambda - 1 & \text{for every } i = 1, \ldots, n \\ & X_{ij} = -1 & \text{for every } ij \in G' \\ & X \succeq 0. & \end{array}$$

Since G' has at most n nodes and at most m edges, the above semidefinite program can be approximately solved in time $\tilde{O}(\sqrt{n}m^3)$ using interior-point methods. Since there are n iterations in MaxClique, a maximum clique in a perfect graph G with n nodes and m edges can be found in time $\tilde{O}(n^{1.5}m^3)$ using interior-point methods. Based on the above observation, Alizadeh [4, 5] developed a parallel algorithm for finding a maximum clique in perfect graphs.

5.2.2 MAX STABLE SET

One can easily see that finding a maximum clique in G is equivalent to finding a maximum stable set in $\bar{G}$. The algorithm MaxStableSet(G) is simply a function call to MaxClique($\bar{G}$), as shown in Figure 5.2. The number of edges in $\bar{G}$ could be as large as $\Theta(n^2)$ even if G is sparse. Therefore the running time required by MaxStableSet is $\tilde{O}(n^{7.5})$.

5.3 Reducing the Complexity

In this section we show how our basic algorithm for COLORING's semidefinite program can be used to reduce the complexity required by MaxClique and MaxStableSet.

First of all, we observe that if $k = o(n)$, then MaxClique can be improved by performing some binary searches. Specifically, suppose G' is an induced subgraph of a perfect graph G. If $\omega(G) = \omega(G')$, then a maximum clique of G' is also a maximum clique of G. If $\omega(G') < \omega(G)$, then every maximum clique of G has at least a node in $G \setminus G'$. Therefore after computing $\omega(G')$ for $\log n$ induced subgraphs G' of G, we can locate a single node v which is guaranteed to be in a maximum clique of G. Now we can reduce the problem to finding a maximum clique of size $\omega(G) - 1$ in the induced subgraph of G on the nodes that are adjacent to v. Then we can repeat the above procedure to find another node that is guaranteed to be in a maximum clique in the induced subgraph, which is therefore in a maximum clique in the original graph.

Clearly the procedure repeats at most k times. Therefore the algorithm computes $\omega(G')$ for at most $k \log n$ induced subgraphs G' of G. We give the algorithm in Figure 5.3, where G_i is the induced subgraph of G on nodes $i, \ldots, n$, where n is the number of nodes in G. Clearly the algorithm BetterMaxStableSet(G) is just BetterMaxClique($\bar{G}$), as shown in Figure 5.4.

If we use interior-point methods to compute those $\omega(G')$ in BetterMaxClique, the running time required by BetterMaxClique is $\tilde{O}(k\sqrt{n}m^3)$. Therefore the running time required by BetterMaxStableSet is $\tilde{O}(kn^{6.5})$. If k is a constant, then we can significantly improve the running time of both algorithms using our basic approximation algorithm for the semidefinite relaxation of COLORING.

5.3.1 Time complexity

Let $\lambda(G)$ be the optimal value of G's semidefinite relaxation for COLORING. It is shown in the journal version of [86] that $\lambda(G) = 1/(1-\omega(G))$. If the given perfect graph is k-colorable, then $\omega(G) = \chi(G) \leq k$ by definition of perfect graphs. It follows that an $O(1/k)$-optimal approximation to $\lambda(G)$ can be used to determine $\omega(G)$, since $\omega(G)$ is an integer. Therefore our VectorColoring gives a way to implement BetterMaxClique and BetterMaxStableSet more efficiently. Specifically the running time required by BetterMaxClique

Algorithm BetterMaxClique(G)

Input: a perfect graph G

Output: a maximum clique of G

1. Let $C = \{\}$.
2. For $\omega = \omega(G)$ downto 1 do
   (a) Let $\ell = 1$.
   (b) Let r be the number of nodes in G.
   (c) Repeat
      i. Let $i = \lceil \frac{\ell+r}{2} \rceil$.
      ii. Compute $\omega' = \omega(G_i)$.
      iii. If $\omega' < \omega$
            Let $r = i - 1$.
         else
            Let $\ell = i$.
      Until $\ell = r$.
   (d) Insert node ℓ of G into the set C.
   (e) Change G to be the induced subgraph of G on the neighbors of node ℓ in G.
3. Return C.

Figure 5.3: Algorithm BetterMaxClique

Algorithm BetterMaxStableSet(G)

Input: a perfect graph G

Output: a maximum stable set of G

1. Return BetterMaxClique($\bar{G}$).

Figure 5.4: Algorithm BetterMaxStableSet

is $\tilde{O}(k^9mn)$; and the running time required by BetterMaxStableSet is $\tilde{O}(k^9n^3)$. One can see that the improvement is particularly striking when k is a constant.

5.3.2 Space complexity

Note that we are only interested in the value of the solution returned by each call to VectorColoring. It follows from Lemma 29 that the space required by each call to VectorColoring is only $O(m)$. Therefore the space complexity of BetterMaxClique is only $O(m)$.

Since $\bar{G}$ could be dense, the space complexity of MaxStableSet is still $O(n^2)$.

Chapter 6

Compact Algorithms

6.1 Overview

In this chapter we give algorithms for obtaining low-rank approximate solutions to the semidefinite relaxations of MAXCUT and COLORING. The objective is to obtain the low-rank solutions in the least possible space.

We would like to point out that the running-time analysis of our compact algorithms is based on exact arithmetic. In other words, we assume that there is no error in our computations. In general this is not a practical assumption. Therefore the results presented in this chapter work only in theory. The assumption on exact arithmetic is due to the difficulty in one step of our algorithm, finding a nonzero solution to a homogeneous linear system with $O(n)$ variables and $O(n)$ constraints. Because we can afford only $O(n^{1.5})$ space for this step, we resort to an iterative method, the conjugate gradient method, which requires only $O(n)$ space. The time complexity is $O(n^3)$ in exact arithmetic. Without the assumption of exact arithmetic, the running time depends linearly on the square root of the condition number of the linear system, because of the error in the computation. However, the space we can afford is more than the space we have used by a factor of $O(\sqrt{n})$. Therefore the computation in this step can be done in $O(\sqrt{n})$-bit precision. As a consequence, the computation should have less error. Perhaps this could reduce the dependence of the running time on the condition number of the linear system, and thus reduces the running time required by this step in practice.

6.1.1 MAXCUT

In Chapter 3 we show that our basic algorithm for MAXCUT's semidefinite relaxation starts with finding a rank-one feasible solution, and then proceeds iteratively to improve its quality. In each iteration a rank-one update is applied to the current solution, so the rank of the current solution is increased by at most one. Since the number of iterations could as large as $\tilde{O}(\epsilon^{-2}n)$, the resulting solution is likely to have full rank. However, it follows from Pataki's results that no matter how high the rank of the current solution is, there always exists a feasible solution of rank $O(\sqrt{n})$ that has the same quality. Therefore our compact algorithm is simply the basic algorithm augmented with a number of *rank-reduction* steps, which was first proposed by Pataki [119]. Specifically, whenever the rank of the current solution is one more than the bound given by Pataki, we replace the current solution matrix by a new matrix with the same quality, whose rank is one less than the current solution matrix. That way we keep the rank of the current solution low throughout the execution. As a result, we reduce the space required by approximating the semidefinite relaxation of MAXCUT to $O(m + n^{1.5})$, where $O(m)$ comes from the input.

6.1.2 COLORING

In Chapter 3 we show that our basic algorithm for COLORING's semidefinite relaxation runs in $\tilde{O}(\epsilon^{-2})$ iterations. In each iteration an approximate solution to a MAXCUT's semidefinite relaxation is obtained. The resulting near-optimal solution to COLORING's semidefinite relaxation is a linear combination of the solutions obtained from all iterations. If we resort to our compact algorithm for MAXCUT's semidefinite relaxation in each of those iterations, then we are guaranteed to find an approximate solution to COLORING's semidefinite relaxation whose rank is $\tilde{O}(\epsilon^{-2}\sqrt{n})$. This gives an approximation algorithm for COLORING's semidefinite relaxation that runs in space $O(m)+\tilde{O}(\epsilon^{-2}n^{1.5})$. It is interesting to observe that Pataki's results only guarantee a rank-$O(\sqrt{m})$ solution to COLORING's semidefinite relaxation. Hence the rank of the solution output by our algorithm is lower than the bound given by Pataki when ϵ is a constant and the graph is dense.

6.1.3 Outline

Here is the structure for the rest of the chapter. We first give some preliminaries in Section 6.2. In Section 6.3 we give and analyze the compact algorithm for MAXCUT's semidefinite relaxation. In Section 6.4 we show the compact algorithm for COLORING's semidefinite relaxation.

6.2 Preliminaries

We need the following lemmas in this chapter.

Lemma 30 Let X be a symmetric matrix. Suppose $X = UDU^T$, where the columns of U are linearly independent, and D is $k \times k$ and diagonal.

- If the rank of X is k, then $D_{ii} \neq 0$ for every $1 \leq i \leq k$.
- $X \succeq 0$ if and only if $D_{ii} \geq 0$ for every $1 \leq i \leq k$.

Proof Let u_i be the i^{th} column of U. We know $X = \sum_{1\leq i\leq k} D_{ii} u_i u_i^T$. If $D_{ii} = 0$, then X is the sum of at most $k-1$ rank-one matrices. Therefore the rank of S is at most $k-1$. The first statement holds.

If every D_{ii} is nonnegative, then

$$\begin{aligned} v^T X v &= \sum_{1\leq i\leq k} D_{ii} v^T u_i u_i^T v \\ &= \sum_{1\leq i\leq k} D_{ii} (v \cdot u_i)^2 \\ &\geq 0, \end{aligned}$$

for any vector v. Therefore $X \succeq 0$. The if-part of the second statement holds.

Assume $X \succeq 0$. We show $D_{ii} \geq 0$ for every $1 \leq i \leq k$. Let L_1 be the linear space spanned by $\{u_1, \ldots, u_k\}$. Let L_2 be the linear space spanned by $\{u_1, \ldots, u_k\} \setminus \{u_i\}$. Since $u_1, \ldots, u_k$ are linearly independent, the dimension of L_1 is one more than the dimension of L_2. Therefore the dimension of $L_2^\perp$ is one more than the dimension of $L_1^\perp$. It follows that there exists a nonzero vector v in $L_2^\perp \setminus L_1^\perp$. Namely $v \cdot u_i \neq 0$; and $v \cdot u_j = 0$ for every $j \neq i$. Hence

$$\begin{aligned} D_{ii}(v \cdot u_i)^2 &= v^T X v \\ &\geq 0, \end{aligned}$$

and thus $D_{ii} \geq 0$. $\square$

Lemma 31 Let S and $\hat{S}$ be two symmetric $k \times k$ matrices, where $S \succeq 0$ and $\hat{S} \not\succeq 0$. Then

$$\hat{\xi} = \max\{\xi : S + \hat{\xi}\hat{S} \succeq 0\}.$$

is well-defined. Moreover, the rank of $S + \hat{\xi}\hat{S}$ is at most $k - 1$.

Proof Since $S \succeq 0$, it follows from Theorem 8.7.1 of [77] that there exists a nonsingular $k \times k$ matrix R such that $S = RDR^T$ and $\hat{S} = R\hat{D}R^T$, where D and $\hat{D}$ are diagonal. Since R is nonsingular, we know the columns of R are linearly independent. Since $S + \xi\hat{S} = R(D + \xi\hat{D})R^T$, it follows from Lemma 30 that $S + \xi\hat{S} \succeq 0$ if and only if all (diagonal) elements of $D + \xi\hat{D}$ are nonnegative. By the second statement of Lemma 30 we know $D_{ii} \geq 0$ for every i and $\hat{D}_{jj} < 0$ for some j. Therefore $\hat{\xi}$ is well-defined as

$$\hat{\xi} = \min\{-D_{ii}/\hat{D}_{ii} : \hat{D}_{ii} < 0\}.$$

Suppose $\hat{\xi} = -D_{jj}/\hat{D}_{jj}$ for some $1 \leq j \leq k$. Clearly the j^{th} diagonal element of $D + \hat{\xi}\hat{D}$ is zero. It follows from Lemma 30 that the rank of $S + \hat{\xi}\hat{S}$ is at most $k - 1$. $\square$

Lemma 32 Let X, U, and S be three nonzero matrices such that $X = USU^T$. The following statements hold.

- The rank of X is no more than the rank of S.
- X is positive semidefinite if S is positive semidefinite.

Proof Suppose the rank of S is k. We know S can be written as RDR^T, where D is a $k \times k$ diagonal matrix. Let $W = UR$. Therefore $X = WDW^T$, which implies that X can be written as the sum of k rank-one matrices. Therefore the rank of X is no more than k. The first statement holds.

Suppose S is positive semidefinite. We prove the second statement by showing that $u^T X u \geq 0$ for any vector u. Let w be the vector such that $w = U^T u$. Clearly $u^T X u = w^T S w \geq 0$, since S is positive semidefinite. $\square$

6.2.1 Compact representation of low-rank matrices

A procedure for the following problem is required by our compact algorithms.

> Given an $n \times n$ symmetric matrix X of rank k, where k is unknown, but $k \leq \ell$ for some known $\ell \leq n$, compute an $n \times k$ matrix U and a symmetric $k \times k$ matrix S such that $X = USU^T$.

Since the rank of X is k, there exists a $k \times k$ diagonal matrix D and an $n \times k$ matrix V, whose columns are orthonormal, such that $X = VDV^T$. Let v_i be the i^{th} column of V. Namely $X = \sum_{1 \leq i \leq k} \lambda_i v_i v_i^T$. In order to solve the above problem, we first compute ℓ vectors $w_1, \ldots, w_\ell$, where each w_i is obtained by multiplying X by a random vector r_i. Namely $w_i = Xr_i$, for every $1 \leq i \leq \ell$. We have the following lemma.

Lemma 33 The linear space spanned by $v_1, \ldots, v_k$ is equal to the linear space spanned by $w_1, \ldots, w_\ell$ with probability one.

Proof Since

$$
\begin{aligned}
w_j &= Xr_j \\
&= \sum_{1 \leq i \leq k} \lambda_i w_i w_i^T r_j \\
&= \sum_{1 \leq i \leq k} (\lambda_i w_i^T r_j) w_i,
\end{aligned}
$$

each w_j is in the linear space spanned by $v_1, \ldots, v_k$. It suffices to show that each v_i is in the linear space spanned by $w_1, \ldots, w_\ell$ with probability one, which can be proved by showing $w_1, \ldots, w_k$ are linearly independent with probability one.

Assume $\alpha_1 w_1 + \cdots + \alpha_k w_k = 0$ for some coefficients $\alpha_1, \ldots, \alpha_k$. Therefore

$$X(\alpha_1 r_1 + \cdots + \alpha_k r_k) = 0.$$

Since $r_1, \ldots, r_k$ are randomly chosen, the dimension of the linear space L_1 spanned by them is k with probability one. Let L_2 be the linear space spanned by $v_1, \ldots, v_k$. The dimension of $L_2^\perp$ is $n - k$. Clearly $L_1 \cap L_2^\perp = \{0\}$ holds with probability one. Since $L_2^\perp$ is exactly the null space of X, we know $\alpha_1 r_1 + \cdots + \alpha_k r_k = 0$ with probability one. Since $r_1, \ldots, r_k$ are linearly independent with probability one, we know $\alpha_1 = \cdots = \alpha_k = 0$ with probability one. $\square$

Now we orthonormalize $w_1, \ldots, w_\ell$ using the Gram-Shmidt procedure (see, e.g., Chapter 5 of [77]), and get a set of orthonormal vectors $u_1, \ldots, u_k$ such that the linear space spanned by $w_1, \ldots, w_\ell$ is equal to the linear space spanned by $u_1, \ldots, u_k$. Let U be the $n \times k$ matrix whose columns are $u_1, \ldots, u_k$. Since the linear space spanned by $u_1, \ldots, u_k$ is equal to the linear space spanned by $v_1, \ldots, v_k$, we know there exists a $k \times k$ matrix R such that $V = UR$. It follows that $X = URDR^T U^T$. Therefore we know there exists a symmetric $k \times k$ matrix $S = RDR^T$ such that $X = USU^T$. In fact $S = U^T XU$, since the columns of U are orthonormal. We then have a USU^T factoring for the given matrix X.

If the given matrix X is in the form of $\bar{U}\bar{S}\bar{U}^T$, where $\bar{U}$ is $n \times \bar{k}$ and $\bar{S}$ is $\bar{k} \times \bar{k}$, then the rank of X is at most $\bar{k}$ by Lemma 32. One can easily verify from the above that the USU^T decomposition of X can be found in time $O(n\bar{k}^2)$ and space $O(n\bar{k})$.

6.2.2 Pataki's rank bound on the basic solutions of semidefinite programs

Consider the following set of symmetric positive-semidefinite matrices

$$Q = \{X \succeq 0 : A^{(i)} \bullet X = b_i,\ i = 1, \ldots, m\},$$

where X and $A^{(1)}, \ldots, A^{(m)}$ are $n \times n$ matrices. Pataki [117] shows that if Q is nonempty, then Q contains a matrix of rank k, for some k such that $k(k+1) \leq 2m$. The low-rank matrix is called a *basic solution*, by analogy to the notion of a basic solution to a linear program [46, 138, 19]. His proof relies on the facial structure of the positive-semidefinite cone, which requires background from convex analysis [134, 36]. Based on the existence of basic solutions, Pataki [119] also shows how to obtain a basic solution from a feasible solution by performing a sequence of rank-reduction procedure.

In the subsection we explain Pataki's rank-reduction procedure. The key for the rank-reduction procedure is the following straightforward observation.

> A homogeneous linear system has a nonzero solution if the number of variables is more than the number of constraints.

Pataki's bound on the rank of a basic solution follows inductively from the following lemma.

Lemma 34 Suppose Q contains a matrix X whose rank is k, where $k(k+1) \geq 2m+1$. Then Q contains a matrix X' of rank at most $k-1$.

Proof The proof is constructive, which follows Pataki's rank-reduction procedure. Since the rank of X is k, it follows from §6.2.1 that $X = USU^T$ for some $k \times k$ symmetric matrix S and $n \times k$ matrix U, where the columns of U are orthonormal. Since $X \succeq 0$ and $S = U^T X U$, we know $S \succeq 0$ by the second statement of Lemma 32. Suppose we can find a $k \times k$ symmetric matrix $\hat{S}$ such that

1. $\hat{S} \not\succeq 0$, and

2. $A^{(i)} \bullet (U\hat{S}U^T) = 0$ for every $i = 1, \ldots, m$.

It follows from Lemma 31 that the rank of $S + \hat{\xi}\hat{S}$ is at most $k - 1$, where $\hat{\xi} = \max\{\xi : S + \xi\hat{S}\}$. Let $S' = S + \hat{\xi}\hat{S}$. Let $X' = US'U^T$. It follows from Lemma 32 that $X' \succeq 0$ and the rank of X' is at most $k - 1$. Since $A^{(i)} \bullet (U\hat{S}U^T) = 0$ for every $i = 1, \ldots, m$, we know $A^{(i)} \bullet X' = b_i$ for every $i = 1, \ldots, m$. Therefore X' is a matrix in Q whose rank is at most $k - 1$.

It remains to show the existence of a $k \times k$ symmetric matrix $\hat{S}$ with the required properties. Let us focus on the second property first. Note that each

$$A^{(i)} \bullet (U\hat{S}U^T) = 0$$

is a homogeneous linear constraint on $\hat{S}$. Since $\hat{S}$ is symmetric and $k \times k$, it has to satisfy $\frac{k^2-k}{2}$ other homogeneous linear constraints. As a result, we have a homogeneous linear system with k^2 variables and $m + \frac{k^2-k}{2}$ constraints. Clearly

$$\begin{aligned} k^2 - (m + \frac{k^2 - k}{2}) &= \frac{k^2 + k}{2} - m \\ &\geq 1. \end{aligned}$$

The number of variables is at least one more than the number of constraints. Therefore the homogeneous linear system has a nonzero solution $\check{S}$. Clearly $-\check{S}$ is also a nonzero solution to the system. At least one of $\check{S}$ and $-\check{S}$ is not positive semidefinite. Therefore at least one of them satisfies properties 1 and 2 above. $\square$

6.3 The Compact Algorithm for the Semidefinite Relaxation of MAXCUT

Let G be a graph composed of n nodes and m edges with positive weights. Let C be the matrix such that C_{ij} is the weight of edge ij. In §3.3.4 we give the algorithm VectorMaxCut(C, ϵ) for solving the following semidefinite program to within relative error ϵ.

$$
\begin{array}{lll}
\min & \lambda & \\
\text{s.t.} & X_{ii} \leq \lambda & \text{for every } 1 \leq i \leq n \\
& L \bullet X = 1 & \\
& X \succeq 0. &
\end{array}
$$

At the completion of execution, the Cholesky decomposition of an ϵ-optimal solution is reported. The space required by VectorMaxCut is $\tilde{O}(n^2)$. In this section we show how to modify that algorithm and obtain an algorithm solving the same problem whose space requirement is only $O(m + n^{1.5})$, where $O(m)$ comes from the input.

6.3.1 The algorithm

CompactVectorMaxCut(C, ϵ), the compact version of the approximation algorithm, is given in Figure 6.1. The algorithm starts with finding an initial rank-one matrix (X, λ). It then iteratively calls the subroutine CompactImprove to improve the quality of the current solution. The only thing that CompactVectorMaxCut differs from the original version VectorMaxCut is that the new subroutine CompactImprove runs in space $O(n^{1.5})$ and always outputs a matrix X in the form of USU^T, where U is $n \times k$ and S is $k \times k$, for some $k = O(\sqrt{n})$.

The subroutine CompactImprove(X, λ_0, ϵ) is given in Figure 6.2. The subroutine Direction is described in §3.3.8, which runs in time $\tilde{O}(m\epsilon^{-1})$ and space $O(n)$. It always outputs a rank-one matrix in the form of $\tilde{u}\tilde{u}^T$ for some vector $\tilde{u}$.

The only thing that CompactImprove differs from the original version is that a new procedure RankReduction is called in each iteration after the rank-one update is applied to the current matrix. The purpose is to ensure that the rank k of the current matrix X satisfies that $k(k+1) \leq 2n+2$. Specifically, RankReduction$(\bar{X})$ computes the USU^T decomposition of $\bar{X}$ using the procedure described in §6.2.1, and therefore determines the rank k of $\bar{X}$. If $k(k+1) \geq 2n+3$, then it uses Pataki's rank-reduction procedure described in the proof of Lemma 34 to find a matrix $X' \in Q_{\bar{X}}$ whose rank is at most $k-1$, where $Q_{\bar{X}}$ is the following set of matrices

$$\{X \succeq 0 : L \bullet X = L \bullet \bar{X},\, X_{ii} = \bar{X}_{ii},\, 1 \leq i \leq n\}.$$

Algorithm $\textsc{CompactVectorMaxCut}(C, \epsilon)$

Input: C is the matrix such that C_{ij} is the weight of edge ij of the input graph.

Output: an ϵ-optimal solution matrix X to the semidefinite relaxation of graph MAXCUT for the input graph.

1. Scale C such that the sum of the absolute values of edge weights is one, and then compute L from C.
2. Let $(X, \lambda) = \textsc{Initial}(C)$.
3. Let $\epsilon' = 2/3$.
4. While $\epsilon' > \epsilon$ do
   (a) Let $\epsilon' = \epsilon'/2$.
   (b) Let $(X, \lambda) = \textsc{CompactImprove}(X, \lambda, \epsilon'/4)$.
5. Return X.

Figure 6.1: Algorithm CompactVectorMaxCut

Algorithm $\textsc{CompactImprove}(X, \lambda_0, \epsilon)$

Input: X is the current solution whose value is λ_0.

Output: a ϵ-optimal solution (X, λ).

1. Let $(\epsilon, \lambda, \alpha, \sigma) = \textsc{SetParameters}(\lambda_0, \epsilon)$, which is exactly the first step of Improve.
2. Repeat
   (a) Let $y_i = e^{\alpha X_{ii}}$ for every $i = 1, \ldots, n$.
   (b) Let $\tilde{X} = \textsc{Direction}(y, \epsilon/13)$.
   (c) If (P2) is not satisfied then
       let $X = (1-\sigma)X + \sigma\tilde{X}$.
       Let $X = \textsc{RankReduction}(X)$.
       let $\lambda = \max\{X_{11}, \ldots, X_{nn}\}$.
   else
       return (X, λ).

Figure 6.2: Algorithm CompactImprove

Algorithm RankReduction($\bar{X}$)

Input: the current solution matrix $\bar{X}$

Output: a rank-k solution matrix $X' = US'U^T$ where $k(k+1) \le 2n+2$.

1. Use the procedure given in §6.2.1 to compute the USU^T decomposition of $\bar{X}$ where the columns of U are orthonormal, and S is a $k \times k$ full-rank matrix.
2. If $k(k+1) \le 2n+2$, then return $\bar{X}$ in the form of USU^T. Otherwise,
   (a) Find a symmetric $k \times k$ matrix $\hat{S}$ such that $\hat{S} \not\succeq 0$, $L \bullet (U\hat{S}U^T) = 0$ and the diagonal elements of $U\hat{S}U^T$ are all zero.
   (b) Find a nonsingular matrix R and two diagonal matrices D and $\hat{D}$ such that $S = RDR^T$ and $\hat{S} = R\hat{D}R^T$.
   (c) Compute $\hat{\xi} = \min\{-D_{ii}/\hat{D}_{ii} : \hat{D}_{ii} < 0\}$. Let $S' = S + \hat{\xi}\hat{S}$. Let $X' = US'U^T$.
   (d) Return X' in the form of $US'U^T$.

Figure 6.3: Algorithm RankReduction

Since $\bar{X} \in Q_{\bar{X}}$, we know $Q_{\bar{X}}$ is not empty. Lemma 34 guarantees the existence of X', which is also the output of RankReduction($\bar{X}$).

In order to claim that CompactVectorMaxCut is a compact algorithm, it remains to show how to implement RankReduction to run in space $O(m+n^{1.5})$.

6.3.2 The rank-reduction step

We give RankReduction($\bar{X}$) in Figure 6.3.

The input matrix $\bar{X}$ is $(1-\sigma)$ times the output of the previous iteration plus a rank-one update $\sigma\tilde{u}\tilde{u}^T$. Therefore it can be put in the form $\bar{U}\bar{S}\bar{U}^T$, where U is $n \times \bar{k}$ and $\bar{S}$ is $\bar{k} \times \bar{k}$ for some $\bar{k} = O(\sqrt{n})$. It follows from §6.2.1 that the first step can be done in time $O(n^2)$ and space $(n^{1.5})$.

Step 2-(b) can be done in time $O(k^3)$ and space $O(k^2)$ using, for example, Algorithm 8.7.1 in [77].

The real challenge comes from Step 2-(a), which requires that we find a nonzero solution to the underconstrained homogeneous linear system $Ax = 0$ with $O(n)$ variables and $O(n)$ constraints. Note that $L \bullet (U\hat{S}U^T)$ is equal to

$(U^T LU) \bullet \hat{S}$, where $U^T LU$ is a $k \times k$ matrix obtainable in time $O(mk + nk^2)$. Also note that the i^{th} diagonal element of $U\hat{S}U^T$ is exactly $u_i^T \hat{S} u_i$, where u_i is the i^{th} column of U^T. Therefore the corresponding matrix A can be represented sparsely in space $O(n^{1.5})$, although it is $O(n) \times O(n)$.

A direct method such as Gaussian Elimination cannot be applied here to find a nonzero solution to $Ax = 0$, since it would take $O(n^2)$ space. Therefore we have to resort to iterative methods which can find an exact solution in space $O(n)$.

6.3.3 Solving a homogeneous linear system in exact arithmetic

Let $\bar{A}x = b$ be a linear system, where $\bar{A}$ is a nonsingular $n \times n$ matrix, and b is nonzero. Then a solution to the system can be found in time $O(n^3)$ in exact arithmetic and space $O(n)$ using iterative methods like the *conjugate gradient method* [141]. It can be shown that in our homogeneous system $Ax = 0$, the number of variables is $O(\sqrt{n})$ more than the number of constraints. Therefore we can throw in $O(\sqrt{n})$ linear constraints with random coefficients to make the system nonsingular and nonhomogeneous, and then use iterative methods to find a solution of the new system, which also clearly gives a nonzero solution to the original homogeneous system. Note that throwing in those $O(\sqrt{n})$ constraints requires $O(n^{1.5})$ extra space, which fortunately does not exceed the space bound we are aiming for. As for exactly how many extra constraints have to be added to make the system nonsingular but still consistent, we can use binary search, which adds an $O(\log n)$ factor to the required running time.

Therefore we can find a nonzero solution to our homogeneous linear system $Ax = 0$ in time $O(n^3 \log n)$ and space $O(n^{1.5})$ using exact arithmetic. In practice, however, the running time of iterative methods depends polynomially on the condition number of the matrix A, i.e., the largest ratio of the absolute values of two eigenvalues of A. This is due to the accumulation of the error in the computation. Researchers have developed various kinds of preconditioning techniques to bring down the condition number of the given linear system [71, 37, 91, 136]. As explained in §6.1, however, we can afford to perform the computation in $O(\sqrt{n})$-bit precision. This might reduce the running time required by the conjugate gradient method in practice.

6.3.4 Overall complexity

As shown in §3.4.4 the total number of iterations required by

$$\textsc{CompactVectorMaxCut}(C, \epsilon)$$

is $O(\epsilon^{-2} n \log(n\epsilon^{-1}))$. The running time of each iteration is dominated by the time required by solving a homogeneous linear system, which is $O(n^3 \log n)$ using exact arithmetic. It follows that the time complexity of our compact algorithm is $\tilde{O}(\epsilon^{-2} n^4)$ using exact arithmetic. As shown in the previous subsections, the required space is $O(m + n^{1.5})$. The required work space is also $O(m + n^{1.5})$.

6.4 The Compact Algorithm for the Semidefinite Relaxation of COLORING

If the given graph is k-colorable, in §3.4.4 we show that an ϵ-optimal solution to COLORING's semidefinite relaxation can be found by making $O(\epsilon^{-2} \log(n\epsilon^{-1}))$ subroutine calls to VectorMaxCut, each of which finds an $\frac{\epsilon}{6k}$-optimal solution to a semidefinite relaxation for some graph MAXCUT. The resulting ϵ-optimal solution is simply a linear combination of the approximate solutions reported by those $\tilde{O}(\epsilon^{-2})$ subroutine calls to VectorMaxCut. Clearly we can replace VectorMaxCut by CompactVectorMaxCut. We therefore have an algorithm

$$\textsc{CompactVectorColoring}(G, \epsilon),$$

which runs in space $O(m + n^{1.5} \epsilon^{-2} \log(n\epsilon^{-1}))$. Based on exact arithmetic, the time complexity of CompactVectorColoring is $O(\epsilon^{-4} k^2 n^4)$.

Since the matrix output by CompactVectorMaxCut has rank at most $\sqrt{2n+2}$, the rank of the solution output by CompactVectorColoring has rank $\tilde{O}(\epsilon^{-2}\sqrt{n})$. It is interesting to note that if ϵ is a constant, then the rank of the ϵ-optimal solution output by CompactVectorColoring is asymptotically lower than Pataki's rank bound $O(\sqrt{m})$ on the basic solutions.

Part III

Experiments and Conclusion

Chapter 7

Implementation

Many researchers have implemented general interior-point methods for solving semidefinite programs [148, 33, 131, 130, 56]. In order to evaluate the practical performance of our algorithms, we have done some preliminary numerical experiments on our basic algorithm for MAXCUT. The programs are written in Matlab. We use SPARC station 10, which has 32 MB main memory and 105 MB swap space, to perform the experiments. For comparison, we use Rendl's code [130], which is also written in Matlab. Rendl's code implements an interior-point algorithm for a very special kind of semidefinite programs. We slightly modify Rendl's code, based on the guidelines shown in the document, to make the code fully exploit the special structure of MAXCUT's semidefinite program.

7.1 Overview of Results

We ran both programs on three kinds of sparse graphs:

- Random graphs with nonnegative edge weights — The degree of each graph is roughly four. Therefore the number of edges is roughly two times the number of nodes. Each nonzero edge weight is uniformly distributed between 0 and 1.

- Random graphs from Spin Glasses — The number of edges is exactly two times the number of nodes in each graph. There are edges of positive and negative weights in each graph.

- Non-random graphs obtained from TSPLIB — The number of edges is roughly ten times the number of nodes in each graph. Each edge has a nonnegative weight.

The experiments show that our program outperforms Rendl's program when $\epsilon \geq 0.25$ and $n \geq 200$. Moreover, Rendl's program runs out of memory on graphs with more than 1,000 nodes, but our program can handle sparse graphs with 10,000 nodes.

7.2 Some Tricks in the Implementation

A naive implementation of our algorithm VectorMaxCut turns out to be an extremely slow program. This is due to the non-negligible constant factors and a bad dependence on the required precision ϵ. After using the following tricks in our implementation, however, our program becomes significantly faster. When ϵ is large enough, e.g., $\epsilon \geq 0.25$, our program is almost always better than Rendl's program in our experiments.

1. Our experiments show that the value of α given in VectorMaxCut significantly affects the number of iterations required by Improve. Specifically, the number of iterations required by Improve depends roughly linearly on the value of α. Therefore it would be nice to keep α as small as possible. In the implementation we increase the value of α only when the relaxation optimality condition (P1) does not hold. This trick speeds up our implementation significantly.

2. In the analysis for VectorMaxCut we show that the reduction of potential in each iteration is determined by the step size σ. We show a lower bound for σ, and thus guarantee that the potential is decreased by a significant factor in each iteration of Improve. In practice, we can use binary search to look for the step size which decreases the potential the most. Namely, once the new direction $\tilde{X}$ is determined, we look for the step size σ such that the potential of $(1-\sigma)X + \sigma(\tilde{X})$ is minimized. This trick also helps to improve the performance of our implementation significantly.

3. The most expensive part of our algorithm is the power iterations. The power method is notorious in its slow convergence. It takes a lot of iterations to converge the iterate to a dominant eigenvector. Although our analysis guarantees that $\tilde{O}(1/\epsilon)$ iterations are sufficient, the power

iterations are still very time-consuming. Fortunately the DIRECTION procedure does not always require that many iterations. Namely, the power iteration can stop as long as the current iterate gives a direction $\tilde{X}$ such that the relaxed optimality condition (P2) does not hold. In other words, we can stop the power iteration as soon as the direction $\tilde{X}$ guarantees to decrease the potential of the current solution by a significant amount. This trick works well especially when $\epsilon \geq 0.25$. Our experiments show that when $\epsilon \geq 0.25$ it roughly takes two power iterations to find a sufficiently good direction.

4. In the description of VECTORMAXCUT we show that the current solution is guaranteed to be $O(\epsilon)$-optimal if (P1) and (P2) hold. In the experiments, however, we observe that our program ran a lot of unnecessary iterations to satisfy (P1) and (P2). Usually the current solution is already $O(\epsilon)$-optimal, but (P2) still does not hold. The question is then how to guarantee the near-optimality of the current solution besides (P1) and (P2). The answer is in Lemma 8. In Lemma 8 we show that the value of any feasible solution is at least 0.2. Therefore we can stop our program as soon as the current value is no more than $0.2 * (1 + \epsilon)$. This trick also helps a lot when $\epsilon \geq 0.25$.

5. The reason we perturb the matrix $\bar{L}$ is to upper-bound the width by n. In the experiments, however, we observe that the perturbation does not seem to be necessary. Therefore we do not have to waste the precision of our approximation by a factor of two. This trick turns out to reduce the running time by roughly a factor of ten.

7.3 Preliminary Numerical Experiments

7.3.1 Random graphs

In this subsection we show our experiments on sparse random graphs with only positive edge weights. In order to demonstrate the effect of limited work space, we let our program keep every element of the current solution matrix in the memory. We will see that both Rendl's program and our program have the memory-swapping problem when $600 \leq n \leq 900$, where n is the number of nodes. When $n \geq 1,000$, both programs run out of memory.

		$\epsilon = 0.5$	$\epsilon = 0.25$	$\epsilon = 0.125$	$\epsilon = 0.0625$
$n = 100$	ours	00:00:06	00:00:36	00:06:36	00:41:48
	Rendl's	00:00:38	00:00:49	00:00:49	00:00:54
$n = 200$	ours	00:00:28	00:02:47	00:35:30	03:13:40
	Rendl's	00:04:23	00:05:54	00:05:54	00:06:39
$n = 300$	ours	00:01:03	00:04:09	01:30:42	09:59:16
	Rendl's	00:21:43	00:26:55	00:26:55	00:29:30
$n = 400$	ours	00:02:52	00:11:06	03:19:07	23:33:48
	Rendl's	01:14:02	01:21:20	01:21:20	01:38:48
$n = 500$	ours	00:04:33	00:18:18	05:21:35	47:55:45
	Rendl's	02:08:00	02:40:46	02:40:46	02:57:26

Table 7.1: Comparison for sparse random graphs with no more than 500 nodes. We use n to denote the number of nodes. The number of edges in each graph is roughly $2n$. The weight of each edge is uniformly distributed between 0 and 1. Each running time is specified as “hh:mm:ss”.

Table 7.1 and Figure 7.1 show the running times of both codes on sparse random graphs with no more than 500 nodes. The number of edges in each graph is roughly two times the number of nodes. All edge weights are nonnegative, uniformly distributed between 0 and 1. A typical graph of this kind is shown in Figure 7.2, where each dot corresponds to an edge. Each running time is specified in the form of “hh:mm:ss”, where “hh”, “mm”, and “ss”, are the numbers of hours, minutes, and seconds, respectively. Clearly our program performed better than Rendl's program when $\epsilon = 0.5$ and $\epsilon = 0.25$. Rendl's program is faster than our program when $\epsilon \leq 0.125$.

Table 7.2 and Figure 7.3 show the running times of both codes on sparse random graphs with $600 - 900$ nodes. The number of edges in each graph is again roughly two times the number of nodes. All edges weights are nonnegative, uniformly distributed between 0 and 1. Due to the limited amount of main memory in our machines, both programs start to suffer from memory swapping for these larger graphs.

Clearly our program performed better than Rendl's program in both CPU time and elapsed time when $\epsilon = 0.5$. However, when $\epsilon = 0.25$, the elapsed time of Rendl's program starts to be better than that of ours when $n \geq 800$, even though the CPU time of our program is still much better than that of Rendl's program.

As we explained in §1.2.1, the work space is relatively small compared to

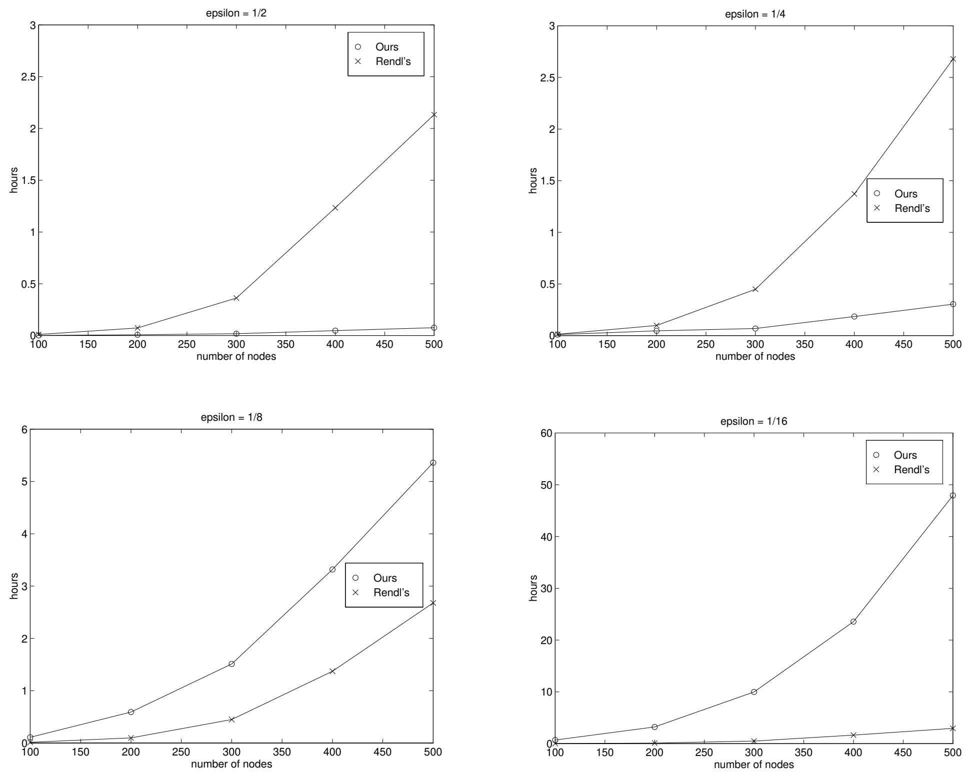

Figure 7.1: Some illustrations for Table 7.1

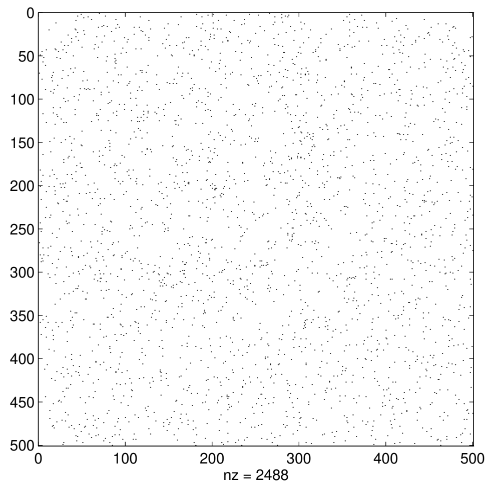

Figure 7.2: A typical random graph. Each dot corresponds to an edge in the graph.

		$\epsilon = 0.5$		$\epsilon = 0.25$		$\epsilon = 0.125$	
		CPU	elapsed	CPU	elapsed	CPU	elapsed
$n = 600$	ours	00:08:19	00:23:21	00:31:56	01:52:23	09:56:29	25:25:03
	Rendl's	03:21:32	03:49:56	04:22:27	04:59:32	04:22:27	04:59:32
$n = 700$	ours	00:14:19	01:25:09	00:52:48	06:13:50		
	Rendl's	05:21:50	06:21:05	07:00:53	08:20:41	07:50:18	09:22:20
$n = 800$	ours	00:16:32	03:01:19	01:24:02	20:21:20		
	Rendl's	07:49:07	11:00:10	10:12:18	14:27:54	11:22:29	16:04:01
$n = 900$	ours	00:12:16	04:39:01	01:03:31	24:21:02		
	Rendl's	out of memory					

Table 7.2: Comparison for sparse random graphs with 600 – 900 nodes. We use n to denote the number of nodes. The number of edges in each graph is roughly $2n$. The weight of each edge is uniformly distributed between 0 and 1. Each running time is specified as "hh:mm:ss".

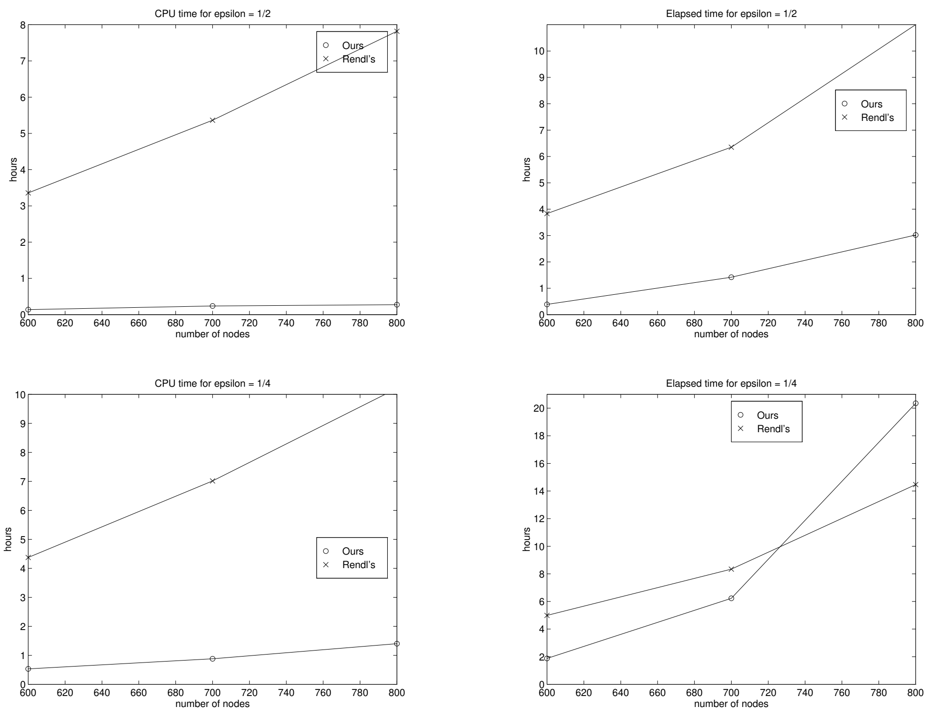

Figure 7.3: Some illustrations for Table 7.2

the file server. Therefore storing all the intermediate data on the file server can avoid the memory-swapping problem for our program, assuming the file server has sufficient disk space. That way our program does not suffer from the memory-swapping problem even for very large n, e.g., $n = 10,000$. Moreover, as described in Chapter 4, our algorithm does not have to store those intermediate data, if it is applied to the approximation algorithm of Goemans and Williamson for MAXCUT.

The observation that our program suffers from the memory-swapping problem more seriously is understandable. It is simply because our program requires a lot of iterations and each iteration requires some matrix-matrix addition. The interior-point methods, however, requires no more than 15 iterations in general, and each iteration requires constant numbers of matrix operations. If we compute only the value of the near-optimal solution instead of the near-optimal solution itself, then our program can do a lot better, since our program does not suffer from the memory-swapping problem even for very large n. Out of curiosity we have performed an experiment on a sparse random graphs with $10,000$ nodes and $20,000$ edges. If $\epsilon = 0.5$, our program runs in roughly ten hours. For the same input graph, Rendl's code would require at least one year, even if it had enough main memory to avoid the memory-swapping problem.

We also ran an experiment to show the number of potential-reduction iterations required to achieve a certain precision of the approximation. The input graph has 400 nodes and 802 edges. The result is shown in Table 7.3 and illustrated in Figure 7.4. The experiment shows that the number of iterations is roughly proportional to $\epsilon^{-2.5}$.

For the rest of the chapter we show only the experiments that compute the values of MAXCUT's semidefinite programs instead of the solutions.

7.3.2 Graphs from Spin Glasses

We performed some experiments on the instances obtained from an application of MAXCUT. Specifically, we ran our program on graphs which come from Spin Glasses [14, 142, 143, 132]. Jünger offers five weighted sparse graphs at

`http://www.informatik.uni-koeln.de/ls_juenger/projects/spinglass/index.html`

These graphs have 9, 64, 100, 400, and 2500 nodes. The number of edges in each graph is two times the number of nodes. There are edges of positive and negative weights in these graphs. A typical graph of this kind is shown in

	number of iterations
$\epsilon = 0.83$	32
$\epsilon = 0.69$	70
$\epsilon = 0.58$	112
$\epsilon = 0.48$	161
$\epsilon = 0.40$	253
$\epsilon = 0.33$	386
$\epsilon = 0.28$	566
$\epsilon = 0.23$	838
$\epsilon = 0.19$	1337
$\epsilon = 0.16$	2123
$\epsilon = 0.13$	4385
$\epsilon = 0.11$	9499
$\epsilon = 0.093$	15706
$\epsilon = 0.078$	23660
$\epsilon = 0.065$	39163

Table 7.3: The relation between the number of iterations and the precision of the approximation. The input graph has 400 nodes and 802 edges.

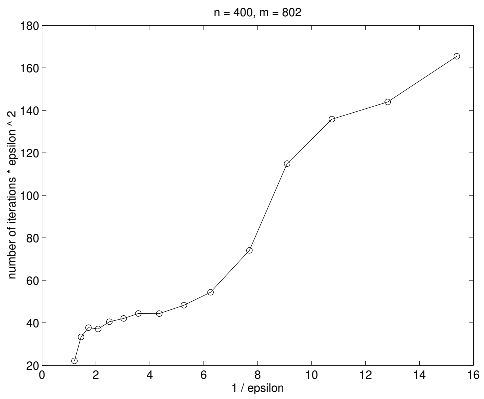

Figure 7.4: An experiment on a graph with 400 nodes and 802 edges. We show the relation between the precision and the number of iterations. The x-axis is $1/\epsilon$, and the y-axis is the number of iterations times ϵ^2.

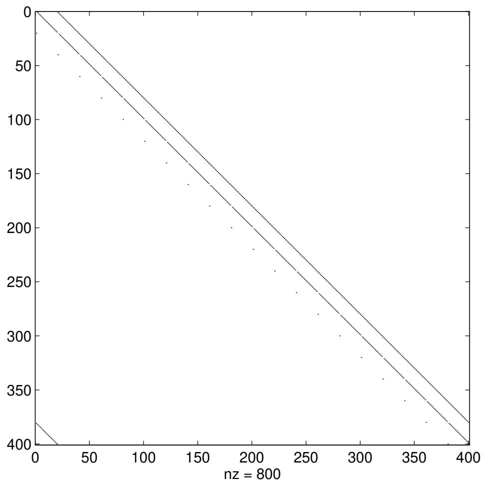

Figure 7.5: A typical graph from Spin Glasses. Each dot corresponds to an edge in the graph.

Figure 7.5, where each dot corresponds to an edge. We also ran Rendl's code on the same instances for comparison. The result is shown in Table 7.4, and illustrated in Figure 7.6.

7.3.3 Sparse non-random graphs

Since random graphs usually behave nicely, we also ran experiments on some non-random graphs. We took some graph instances from TSPLIB [129, 26]. They are all complete graphs, so we obtained a sparse graph from each of them by keeping the heaviest ten edges from each node. A typical graph of this kind is shown in Figure 7.7, where each dot corresponds to an edge. As shown in Table 7.5 and Figure 7.8, our program outperformed Rendl's program for graphs with more than five hundred nodes even when $\epsilon = 0.0625$.

7.4 Can We Do Better?

We mentioned in §7.2 that our program usually ran a lot of unnecessary iterations to satisfy (P1) and (P2). Therefore we use the lower bound obtained from Lemma 8 to stop the program early. However, that lower bound works well only for low precision, e.g., when $\epsilon \geq 0.25$. When $\epsilon \leq 0.125$, that lower bound does not let the program stop early in general. To illustrate this phenomenon, we performed an experiment on a graph with 400 nodes and 800 edges. The

		$\epsilon = 0.5$	$\epsilon = 0.25$	$\epsilon = 0.125$	$\epsilon = 0.0625$
$n = 9$	ours	00:00:00	00:00:00	00:00:03	00:00:11
	Rendl's	00:00:00	00:00:00	00:00:00	00:00:00
$n = 64$	ours	00:00:01	00:00:15	00:01:37	00:08:18
	Rendl's	00:00:08	00:00:10	00:00:10	00:00:13
$n = 100$	ours	00:00:06	00:00:31	00:04:07	00:25:10
	Rendl's	00:00:12	00:00:24	00:00:24	00:00:29
$n = 400$	ours	00:00:33	00:02:00	00:26:20	02:35:31
	Rendl's	00:45:17	00:53:32	01:01:49	01:10:05
$n = 2500$	ours	00:04:28	00:24:07	06:02:22	34:05:46
	Rendl's	out of memory			

Table 7.4: Experiments on sparse graphs obtained from Spin Glasses. We use n to denote the number of nodes. The number of edges in each graph is exactly $2n$. There are edges with positive and negative weights in these graphs. Each running time is specified as "hh:mm:ss".

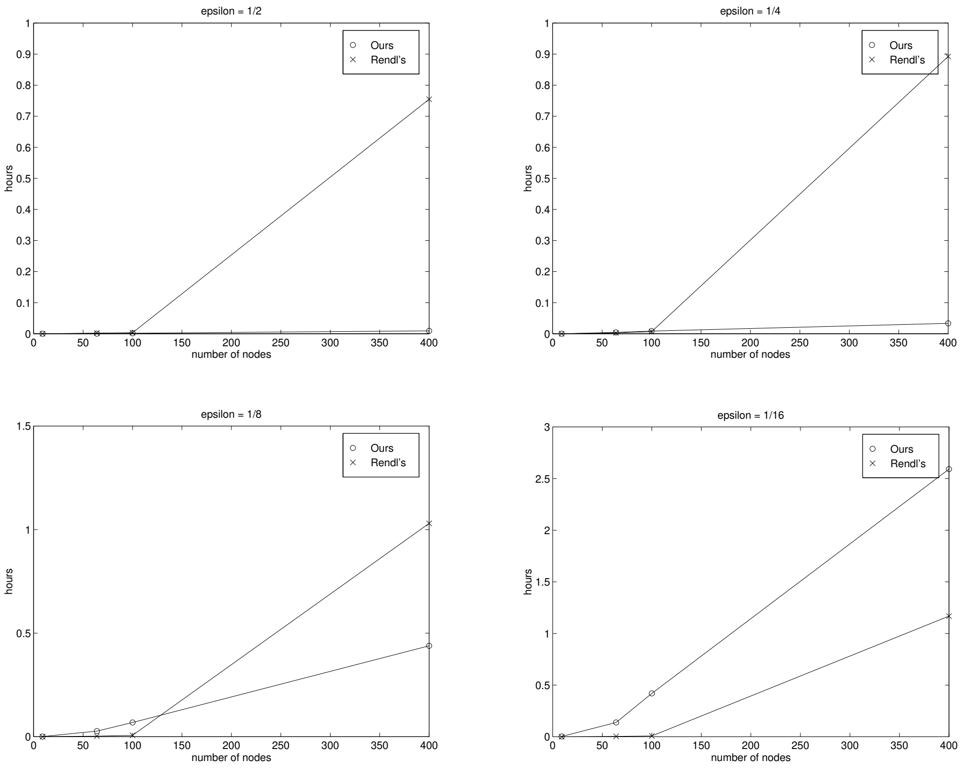

Figure 7.6: Some illustrations for Table 7.4

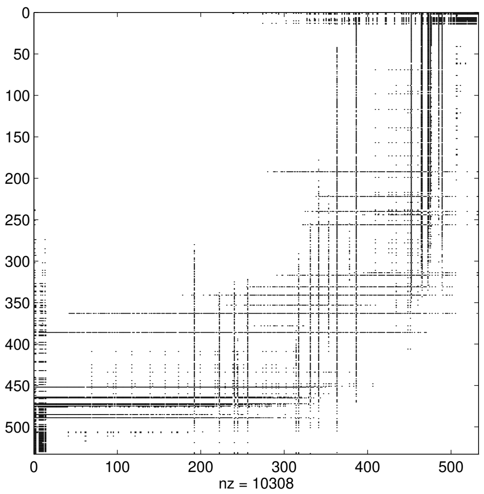

Figure 7.7: A typical non-random sparse graph obtained from TSPLIB. Each dot corresponds to an edge in the graph.

		$\epsilon = 0.5$	$\epsilon = 0.25$	$\epsilon = 0.125$	$\epsilon = 0.0625$
dantzig42	ours	00:00:04	00:00:18	00:00:56	00:02:55
($n = 42$)	Rendl's	00:00:02	00:00:02	00:00:03	00:00:03
gr120	ours	00:00:23	00:01:44	00:04:13	00:11:20
($n = 120$)	Rendl's	00:00:46	00:00:54	00:01:03	00:01:03
kroA200	ours	00:01:15	00:03:04	00:07:47	00:19:20
($n = 200$)	Rendl's	00:03:36	00:05:06	00:05:50	00:05:50
att532	ours	00:03:11	00:07:34	00:24:24	01:16:29
($n = 532$)	Rendl's	02:39:35	03:20:25	03:40:50	03:40:50
dsj1000	ours	00:08:59	00:22:44	00:53:52	02:52:40
($n = 1000$)	Rendl's	more than 10 hours			

Table 7.5: Comparison for some sparse non-random graphs obtained from TSPLIB. We use n to denote the number of nodes. The number of edges in each graph is roughly $10n$. The weight of each edge is uniformly distributed between 0 and 1. Each running time is specified as “hh:mm:ss”.

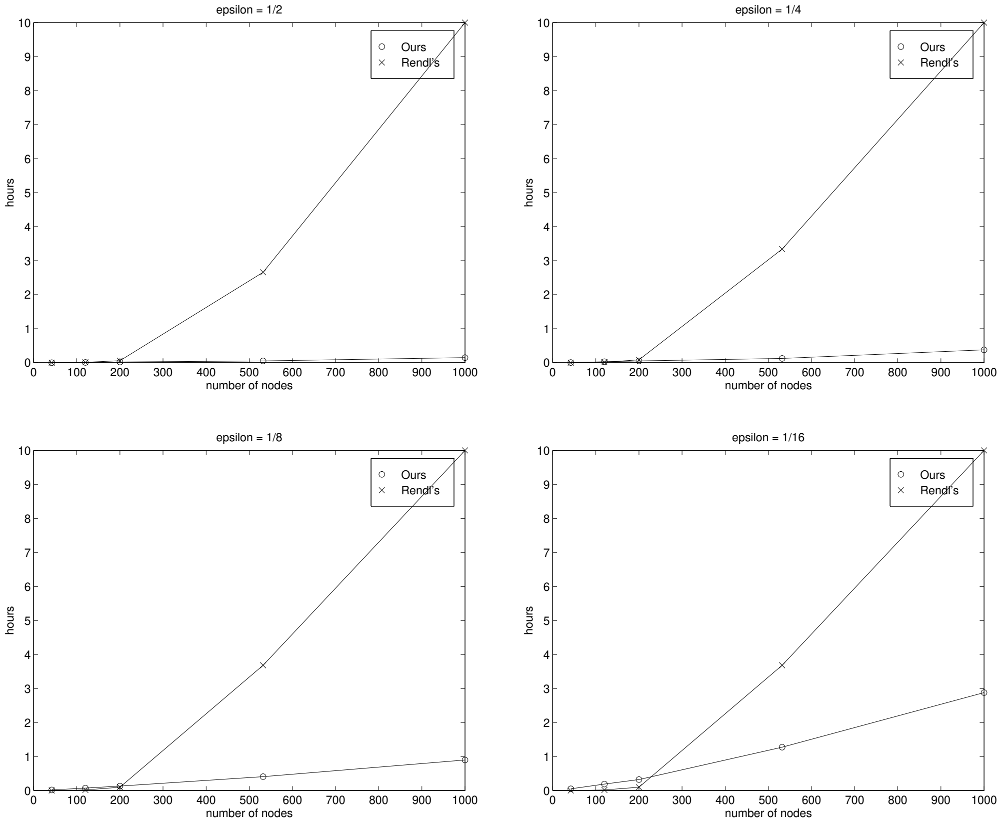

Figure 7.8: Some illustrations for Table 7.5

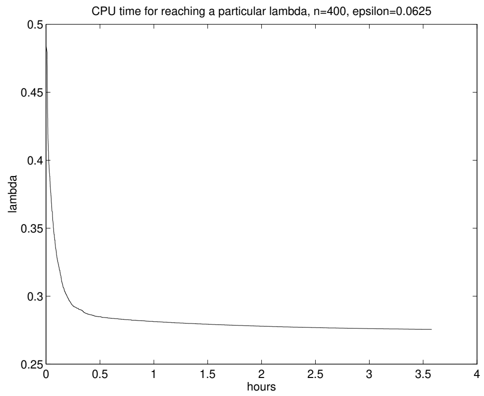

Figure 7.9: An experiment on a graph with 400 nodes and 800 edges. The precision ϵ is 0.0625.

precision ϵ is set to 0.0625. The result is shown in Figure 7.9. The y-axis is the value λ of the current solution. The x-axis is the running time. This picture shows the running time spent to bring down the current solution to a certain value. As we can see in the picture, the value decreases quickly at first. The convergence is extremely slow at the end. Our programs takes 3.5 hours to satisfy (P1) and (P2). The resulting value is 0.2756. When we ran Rendl's code, however, we find out that the optimal value is in fact at least 0.2682. Therefore our solution was already 0.0625-optimal when the value is 0.2850, which was obtained at the 29^{th} minute. Although Rendl's code requires only 2.5 hours to get a solution with 6-digit precision, it takes 1.6 hours to get a 0.0625-optimal solution. This observation suggests that we might be able to do better if we have a good lower bound on the optimal value.

Chapter 8

Concluding Remarks

8.1 Our Contribution

We have shown the following results.

- We give algorithms for obtaining approximations to the semidefinite relaxations of MAXCUT and COLORING. As a result, we speed up the currently best known approximation algorithms for MAXCUT and COLORING if the approximation precision is a constant. Our algorithms have the ability to exploit the sparsity of the given graph. The required work space is only linear in the size of the input. Therefore our algorithms enable the best known approximation algorithms for MAXCUT and COLORING to handle much larger graphs. Although our algorithms perform well only for obtaining rough approximations, they can work on larger instances that previous algorithms cannot even touch.

- We show an interesting relation between the semidefinite relaxations of MAXCUT and COLORING. Specifically, the semidefinite relaxation of MAXCUT can be regarded as a subroutine for solving the semidefinite relaxation for COLORING.

- Our algorithms adapt the framework of approximation algorithms for convex programs given by Plotkin, Shmoys, and Tardos. Previous applications of their framework are all to linear programs. Our algorithms are the first applications to nonlinear programs.

- We give compact approximation algorithms for MAXCUT and COLORING's semidefinite relaxations on sparse graphs. As a result, we reduce the space complexity required by obtaining a near-optimal solution to the semidefinite relaxations of MAXCUT and COLORING.

- Our basic algorithm for COLORING's semidefinite relaxation can be applied to reduce the time or space complexity of the only known polynomial-time algorithm for solving MAXCLIQUE and MAX STABLE SET on a perfect graph if its chromatic number is a constant.

8.2 Future Work

We propose the following future work.

- There are plenty of other semidefinite programs [106, 108, 63, 55, 51, 52, 150, 113, 32]. It would be nice if our algorithms could be generalized to work on some of them.

- Our compact algorithms are still far from practical due to their time complexity. It would be nice if one can come up with compact algorithms with better running times. We propose to perform the conjugate gradient method in $O(\sqrt{n})$-bit precision. We hope that this can reduce the time complexity of our compact algorithms.

- We observe in the numerical experiments that our algorithms usually find a near-optimal solution long before the relaxed optimality conditions guarantee the near-optimality of the solution. If we can provide a good lower bound from the dual program, then our algorithms can terminate much earlier. Therefore we propose to look more carefully into the dual problem in the framework of Plotkin, Shmoys, and Tardos. We hope that the complexity, especially the dependency on the precision of the approximation, can be reduced both in theory and in practice.

- Our algorithm for solving COLORING's semidefinite relaxation works only for the unweighted case. We propose to generalize our results to the weighted case. This might reduce the time complexity for computing the weighted version of Lovász's theta function for a graph whose chromatic number is a constant. We also propose to apply our results to reduce the complexity for solving COLORING on perfect graphs.

- As briefly mentioned in §3.3.8, Lanczos method is a promising alternative to the power method. We propose to replace the power method with

Lanczos method in our algorithms. We hope that the time complexity of all our algorithms shown in the thesis can be reduced by a factor of $O(\sqrt{1/\epsilon})$.

Appendix A

The Proofs of Six Lemmas

We include the proofs of six lemmas here. All these proofs are adapted from the corresponding proofs in the paper by Plotkin, Shmoys, and Tardos [122].

A.1 The Proof of Lemma 12

Lemma 12 Let y be a nonnegative vector. Let $\tilde{X}$ be a matrix such that $\langle \tilde{X} \rangle_y \leq (1+\epsilon/13)\mu(y)$. If $\epsilon \leq 1/13$, then (P1) and (P2) imply that (X, λ) is 4ϵ-optimal.

Proof By (P2) and (P1) we know that

$$\begin{aligned}
\langle \tilde{X} \rangle_y &\geq (1-\epsilon)\langle X \rangle_y - \epsilon\lambda y \cdot \vec{1} \\
&\geq (1-2\epsilon)\lambda y \cdot \vec{1} - \epsilon\lambda y \cdot \vec{1} \\
&\geq (1-3\epsilon)\lambda y \cdot \vec{1}.
\end{aligned}$$

It follows that

$$\begin{aligned}
\lambda &\leq (1-3\epsilon)^{-1}\langle \tilde{X} \rangle_y / y \cdot \vec{1} \\
&\leq (1-3\epsilon)^{-1}(1+\epsilon/13)\mu(y)/y \cdot \vec{1} \\
&\leq (1-3\epsilon)^{-1}(1+\epsilon/13)\lambda^* \\
&\leq (1+r\epsilon)\lambda^*,
\end{aligned}$$

where the second-to-last inequality is by Lemma 10, and the last inequality is by $\epsilon \leq 1/13$. □

A.2 The Proof of Lemma 13

Lemma 13 Suppose $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2n\epsilon^{-1})$. Let (X, λ) be a feasible solution. Let y be a nonnegative vector defined as $y_i = e^{\alpha X_{ii}}$. Then (P1) is satisfied.

Proof We introduce the following localized version of (P1):

($\hat{\mathbf{P}}$1) For each $1 \leq i \leq n$, either $(1-\epsilon/2)\lambda \leq X_{ii}$ or $y_i \leq \frac{\epsilon}{2n} y \cdot \vec{1}$.

We first show that ($\hat{\text{P}}$1) implies (P1). Let $K = \{i : (1-\epsilon/2)\lambda \leq X_{ii}\}$. We have that

$$\begin{aligned}
\lambda y \cdot \vec{1} &= \lambda \sum_{i \in K} y_i + \lambda \sum_{i \notin K} y_i \\
&\leq \frac{1}{1-\epsilon/2} \sum_{i \in K} y_i X_{ii} + \lambda \sum_{i \notin K} \frac{\epsilon}{2n} y \cdot \vec{1}. \\
&\leq \frac{1}{1-\epsilon/2} \langle X \rangle_y + \frac{\epsilon}{2} \lambda y \cdot \vec{1}.
\end{aligned}$$

Thus

$$\begin{aligned}
\langle X \rangle_y &\geq (1-\epsilon/2)^2 \lambda y \cdot \vec{1} \\
&\geq (1-\epsilon) \lambda y \cdot \vec{1}.
\end{aligned}$$

It remains to prove that $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2n\epsilon^{-1})$ implies that ($\hat{\text{P}}$1) is satisfied. Since λ is a diagonal element of X, we know that $y \cdot \vec{1} \geq e^{\alpha\lambda}$. Consider any index i such that $X_{ii} < (1-\epsilon/2)\lambda$. Clearly $y_i < e^{(1-\epsilon/2)\alpha\lambda}$. Therefore

$$\begin{aligned}
\frac{y_i}{y \cdot \vec{1}} &< \frac{e^{(1-\epsilon/2)\alpha\lambda}}{e^{\alpha\lambda}} \\
&= e^{-\epsilon\alpha\lambda/2} \\
&\leq \frac{\epsilon}{2n},
\end{aligned}$$

where the last inequality is by the assumption that $\alpha \geq 2\lambda^{-1}\epsilon^{-1}\ln(2n\epsilon^{-1})$. $\square$

A.3 The Proof of Lemma 15

Lemma 15 Suppose $0 < \epsilon \leq 1$. Let (X, λ) be a feasible solution. Let α, σ, y be as defined in $\textsc{Improve}(X, \lambda, \epsilon)$. Let $\tilde{X}$ be a feasible matrix (not necessarily satisfying (3.10)). Suppose (P2) is not satisfied by (X, λ), $\tilde{X}$ and y. Define a new feasible matrix $\hat{X}$ to be $(1-\sigma)X + \sigma\tilde{X}$. Let $\hat{y}$ be defined by $\hat{y}_i = e^{\alpha \hat{X}_{ii}}$. Then $y \cdot \vec{1} - \hat{y} \cdot \vec{1} \geq \alpha\sigma\epsilon\lambda y \cdot \vec{1}$.

Proof By Lemma 9, $|X_{ii} - \tilde{X}_{ii}| \le n$ for every $1 \le i \le n$. Since $\sigma = \frac{\epsilon}{4\alpha n}$, we have

$$\begin{aligned} \alpha\sigma|X_{ii} - \tilde{X}_{ii}| &\le \epsilon/4 \\ &\le 1/4. \end{aligned}$$

Using the second-order Taylor theorem, we see that if $|\delta| \le \epsilon/4 \le 1/4$ then for all z, $e^{z+\delta} \le e^z + \delta e^z + \frac{\epsilon}{2}|\delta|e^z$. Letting $\delta = \alpha\sigma(X_{ii} - \tilde{X}_{ii})$, we see that

$$\hat{y}_i \le y_i + \alpha\sigma y_i(\tilde{X}_{ii} - X_{ii}) + \frac{\epsilon}{2}\alpha\sigma y_i|\tilde{X}_{ii} - X_{ii}|.$$

Therefore

$$y_i - \hat{y}_i \ge \alpha\sigma y_i(X_{ii} - \tilde{X}_{ii}) - \alpha\sigma\epsilon y_i(X_{ii} + \tilde{X}_{ii})/2,$$

since $|\tilde{X}_{ii} - X_{ii}| \le (\tilde{X}_{ii} + X_{ii})$. It follows that

$$\begin{aligned} y \cdot \vec{1} - \hat{y} \cdot \vec{1} &\ge \alpha\sigma(\langle X\rangle_y - \langle\tilde{X}\rangle_y) - \frac{\alpha\sigma\epsilon}{2}(\langle X\rangle_y + \langle\tilde{X}\rangle_y) \\ &\ge \alpha\sigma\epsilon(\langle X\rangle_y + \lambda y \cdot \vec{1}) - \alpha\sigma\epsilon\langle X\rangle_y \\ &= \alpha\sigma\epsilon\lambda y \cdot \vec{1}, \end{aligned}$$

where the second inequality is because (P2) is not satisfied. The lemma is proved. □

A.4 The Proof of Lemma 22

Lemma 22 Let Y be a cost matrix for G. Let $\tilde{X}$ be a matrix satisfying (3.13). Then (C1) and (C2) imply that (X, λ) is 4ϵ-optimal.

Proof By (C2) and (C1) we know that

$$\begin{aligned} Y \bullet \tilde{X} &\ge (1+\epsilon)Y \bullet X + \epsilon\lambda Y \bullet J \\ &\ge (1+\epsilon)^2\lambda Y \bullet J + \epsilon\lambda Y \bullet J \\ &= (1+3\epsilon)\lambda Y \bullet J. \end{aligned}$$

Let λ^* be the optimal value. It follows that

$$\begin{aligned} \lambda &\le (1+3\epsilon)^{-1} Y \bullet \tilde{X}/Y \bullet J \\ &\le (1+3\epsilon)^{-1}(1+\epsilon)^{-1}\mu(Y)/Y \bullet J \\ &\le (1+4\epsilon)^{-1}\lambda^*, \end{aligned}$$

where the second inequality is by (3.13), and the last inequality is by Lemma 21.

□

A.5 The Proof of Lemma 23

Lemma 23 Suppose $\alpha \geq -2\lambda^{-1}\epsilon^{-1}\ln(4n^2\epsilon^{-1})$. Let (X, λ) be a feasible solution. Let Y be a cost matrix for G defined as $Y_{ij} = e^{\alpha X_{ij}}$ for every edge ij of G. Then (C1) is satisfied.

Proof We introduce the following localized version of (C1):

($\hat{\mathbf{K}}$1) For each edge ij of G, either $X_{ij} \geq (1+\epsilon/2)\lambda$ or $X_{ij}Y_{ij} \geq \frac{\epsilon}{2n^2}Y \bullet J$.

We first show that ($\hat{\mathrm{K}}$1) implies (C1). Note that by ($\hat{\mathrm{K}}$1) we know that

$$X_{ij}Y_{ij} \geq (1+\frac{\epsilon}{2})\lambda Y_{ij} + \frac{\epsilon}{2n^2}\lambda Y \bullet J$$

for every edge ij of G. Therefore

$$\begin{aligned} Y \bullet X &\geq (1+\epsilon/2)\lambda Y \bullet J + (\epsilon/2)\lambda Y \bullet J \\ &= (1+\epsilon)\lambda Y \bullet J. \end{aligned}$$

It remains to prove that $\alpha \geq -2\lambda^{-1}\epsilon^{-1}\ln(4n^2\epsilon^{-1})$ implies that ($\hat{\mathrm{K}}$1) is satisfied. Since there is an edge ij of G such that $X_{ij} = \lambda$, we know that $Y \bullet J \geq e^{\alpha\lambda}$. Consider any edge ij such that $X_{ij} < (1+\epsilon/2)\lambda$. We show $X_{ij}Y_{ij} \geq \frac{\epsilon}{2n^2}Y \bullet J$. Clearly $\alpha X_{ij} \leq \alpha\lambda \leq -1$. Note that a function $f(x) = xe^{cx}$ is monotonically nonincreasing if $cx \leq -1$. It follows that

$$\begin{aligned} X_{ij}Y_{ij} &= X_{ij}e^{\alpha X_{ij}} \\ &\geq (1+\frac{\epsilon}{2})\lambda e^{(1+\frac{\epsilon}{2})\lambda}. \end{aligned}$$

Therefore

$$\begin{aligned} \frac{X_{ij}Y_{ij}}{Y \bullet J} &\geq (1+\frac{\epsilon}{2})\lambda e^{\alpha\epsilon\lambda/2} \\ &\geq 2\lambda e^{\alpha\epsilon\lambda/2} \\ &\geq \frac{\epsilon}{2n^2}, \end{aligned}$$

where the last inequality is by the assumption that $\alpha \geq -2\lambda^{-1}\epsilon^{-1}\ln(4n^2\epsilon^{-1})$.
□

A.6 The Proof of Lemma 25

Lemma 25 Suppose $0 < \epsilon \leq 1$. Let (X, λ) be a feasible solution. Let α, σ, Y be as defined in $\textsc{ImproveC}(X, \lambda, \epsilon)$. Let $\tilde{X}$ be a feasible matrix (not necessarily satisfying (3.13)). Suppose (C2) is not satisfied by (X, λ), $\tilde{X}$ and Y. Define a new feasible matrix $\hat{X}$ to be $(1-\sigma)X + \sigma\tilde{X}$. Let $\hat{Y}$ be defined by

$$Y_{ij} = \begin{cases} e^{\alpha X_{ij}} & \text{if } ij \text{ is an edge of } G \\ 0 & \text{otherwise.} \end{cases}$$

Then $Y \bullet J - \hat{Y} \bullet J \geq -\alpha\sigma\epsilon\lambda Y \bullet J$.

Proof By Lemma 20, $|X_{ij} - \tilde{X}_{ij}| \leq 1$ for every edge ij of G. Since $\sigma = \frac{\epsilon}{4\alpha}$, we have $\alpha\sigma|X_{ij} - \tilde{X}_{ij}| \leq \epsilon/4 \leq 1/4$. Using the second-order Taylor theorem, we see that if $|\delta| \leq \epsilon/4 \leq 1/4$ then for all z, $e^{z+\delta} \leq e^z + \delta e^z + \frac{\epsilon}{2}|\delta|e^z$. Letting $\delta = \alpha\sigma(X_{ij} - \tilde{X}_{ij})$, we see that $\hat{Y}_{ij}$ is at most

$$Y_{ij} + \alpha\sigma Y_{ij}(\tilde{X}_{ij} - X_{ij}) + \frac{\epsilon}{2}\alpha\sigma Y_{ij}|\tilde{X}_{ij} - X_{ij}|.$$

Therefore $Y_{ij} - \hat{Y}_{ij}$ is at least

$$\alpha\sigma Y_{ij}(X_{ij} - \tilde{X}_{ij}) + \frac{1}{2}\alpha\sigma\epsilon Y_{ij}(X_{ij} + \tilde{X}_{ij}),$$

since $|\tilde{X}_{ij} - X_{ij}| \leq -(\tilde{X}_{ij} + X_{ij})$. It follows that

$$\begin{aligned} Y \bullet J - \hat{Y} \bullet J &\geq \alpha\sigma Y \bullet (X - \tilde{X}) + \frac{\alpha\sigma\epsilon}{2} Y \bullet (X + \tilde{X}) \\ &\geq \alpha\sigma\epsilon(-Y \bullet X - \lambda Y \bullet J) + \alpha\sigma\epsilon Y \bullet X \\ &= -\alpha\sigma\epsilon\lambda Y \bullet J, \end{aligned}$$

where the second inequality is follows from that fact that (C2) is not satisfied. The lemma is proved. $\square$

Appendix B

An Analysis of the Power Method

Let A be an $n \times n$ symmetric positive-semidefinite matrix. Let λ be the largest eigenvalue of A. Let $x_{(0)}$ be a randomly chosen unit vector. Define

$$x_{(k)} = \frac{A^k x_{(0)}}{\|A^k x_{(0)}\|_2}.$$

In this appendix we prove the following lemma.

Lemma 35 Let $k = \frac{1}{\epsilon}\ln(\frac{2n^{c+1}}{\epsilon})$. Then

$$A \bullet (x_{(k)} x_{(k)}^T) \geq (1+\epsilon)^{-1}\lambda$$

holds with probability at least $1 - n^{-c}$.

Since $A \succeq 0$, it has n orthonormal eigenvectors $w_{(1)}, \ldots, w_{(n)}$. Let $\lambda_1, \ldots, \lambda_n$ be the corresponding eigenvectors, and suppose they are indexed so that $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$. It is known that A can be written as

$$\lambda_1 w_{(1)} w_{(1)}^T + \cdots + \lambda_n w_{(n)} w_{(n)}^T.$$

Since $w_{(1)}, \ldots, w_{(n)}$ are orthonormal, every arbitrary unit vector x can be written as $\alpha_1 w_{(1)} + \cdots + \alpha_n w_{(n)}$, where $\alpha_1^2 + \cdots + \alpha_n^2 = 1$. Note that

$$\begin{aligned} A^k x &= \sum_{1 \leq i \leq n} \alpha_i \lambda_i^k w_{(i)} \\ \|A^k x\|_2 &= \sqrt{\sum_{1 \leq i \leq n} \alpha_i^2 \lambda_i^{2k}} \end{aligned}$$

for any $k \geq 0$. It is not hard to see that as k grows larger, $x_{(k)}$ gets closer to $u_{(1)}$. The following lemma gives an upper on the k that is required for $x_{(k)}$ to be a unit vector x such that $A \bullet (xx^T) \geq (1+\epsilon)^{-1}\lambda_1$.

Lemma 36 Let $x_{(k)}$ be as defined above. Suppose $\alpha_1 \geq 0$. If $k \geq \epsilon^{-1}\ln(\epsilon^{-1}\alpha_1^{-2})$, then $A \bullet (x_{(k)}x_{(k)}^T) \geq (1+\epsilon)^{-1}\lambda_1$.

Proof Assume $A \bullet (x_{(k)}x_{(k)}^T) < \lambda_1/(1+\epsilon)$. We show $k < \epsilon^{-1}\ln(\epsilon^{-1}\alpha_1^{-2})$. Note that

$$A \bullet (x_{(k)}x_{(k)}^T) = \frac{\sum_{1\leq i\leq n} \alpha_i^2\lambda_i^{2k+1}}{\sum_{1\leq i\leq n} \alpha_i^2\lambda_i^{2k}}.$$

It follows from the assumption that

$$(1+\epsilon)\sum_{1\leq i\leq n} \alpha_i^2\lambda_i^{2k+1} < \lambda_1 \sum_{1\leq i\leq n} \alpha_i^2\lambda_i^{2k}.$$

Let $\delta_i = \lambda_1 - (1+\epsilon)\lambda_i$ for every $2 \leq i \leq n$. The above inequality can be rewritten as follows by moving the first terms of both summations to the left, and the remaining terms to the right.

$$\epsilon\alpha_1^2\lambda_1^{2k+1} < \sum_{2\leq i\leq n} \alpha_i^2\lambda_i^{2k}\delta_i. \tag{B.1}$$

Clearly $\delta_2 \leq \cdots \leq \delta_n$. If δ_n were less than or equal to zero, then the RHS of (B.1) would be less than or equal to zero, which contradicts the fact that the LHS of (B.1) is nonnegative. Therefore we know $\delta_n > 0$. Let t be the smallest index such that $\delta_t > 0$. It follows that

$$\begin{aligned} \epsilon\alpha_1^2\lambda_1^{2k+1} &< \sum_{t\leq i\leq n} \alpha_i^2\lambda_i^{2k}\delta_i \\ &\leq \sum_{t\leq i\leq n} \alpha_i^2\lambda_t^{2k}\lambda_1 \\ &\leq \lambda_t^{2k}\lambda_1. \end{aligned}$$

Therefore $\epsilon\alpha_1^2(\lambda_1/\lambda_t)^{2k} < 1$. Since $\delta_t > 0$, we know that $\lambda_1/\lambda_t > (1+\epsilon)$. Thus $\epsilon\alpha_1^2(1+\epsilon)^{2k} < 1$. Since $\ln(1+\epsilon) \geq \epsilon/2$ for any $0 \leq \epsilon \leq 1$, it follows that $k < \epsilon^{-1}\ln(\epsilon^{-1}\alpha_1^{-2})$. □

Since the number of iterations required by the power method is determined by the α_1^2 of $x_{(0)}$, it suffices to show how to choose the initial vector x such that its α_1^2 is sufficiently large. It turns out that randomly choosing $x_{(0)}$ uniformly from the surface of the n-dimensional unit sphere is sufficient. Note that the α_1 of x is exactly $x \cdot w_{(1)}$, the projection of x on $w_{(1)}$. The following lemma provides a lower bound on the probability that $\alpha_1^2 = \Omega(n^{-c-1})$ for any $c \geq 0$.

Lemma 37 Suppose $n \geq 2$. Let x' be a fixed unit vector in n-dimensional space. Let x be a random vector that is uniformly distributed on the surface of the n-dimensional unit sphere. Then the probability that $(x \cdot x')^2 \leq \frac{1}{2n^{c+1}}$ is at most n^{-c} for any $c \geq 0$.

Proof Let $a = \frac{1}{2n^{c+1}}$. Clearly $\Pr((x \cdot x')^2 \leq a^2) = \Pr(|x \cdot x'| \leq a)$. We show $\Pr(|x \cdot x'| \leq a) \leq n^{-c}$.

Let $\hat{\theta} = \arccos a$. It is known by multivariable analysis (see, e.g. §60 of [126]) that

$$\begin{aligned} \Pr(|x \cdot y| \leq a) &= \frac{1}{S} \int_{\hat{\theta}}^{\pi/2} \sin^{n-2}\theta \, d\theta \\ &\leq \frac{1}{S}(\frac{\pi}{2} - \hat{\theta}), \end{aligned}$$

where $S = \displaystyle\int_0^{\pi/2} \sin^{n-2}\theta \, d\theta$. Note that $\theta \leq 2\sin\theta$, for any $\theta \leq \frac{\pi}{2}$. It follows that

$$\begin{aligned} \frac{\pi}{2} - \hat{\theta} &\leq 2\sin(\frac{\pi}{2} - \hat{\theta}) \\ &= 2\cos\hat{\theta} \\ &= 2a \\ &= n^{-c-1}. \end{aligned}$$

Therefore it suffices to show that $S \geq \frac{1}{n}$.

It is known (see, e.g., (131) of §60 in [126]) that

$$S = \frac{\Gamma(\frac{n-1}{2}+1)n\pi^{n/2}}{2(n-1)\pi^{(n-1)/2}\Gamma(\frac{n}{2}+1)}$$

for any $n \geq 2$. Therefore

$$S \geq \frac{\Gamma(\frac{n}{2})}{2\Gamma(\frac{n}{2}+1)} = \frac{1}{n}.$$

□

A uniformly distributed random vector x on the surface of the n-dimensional unit sphere can be obtained by randomly generating n values $x_1, \ldots, x_n$ independently from the standard normal distribution, and then normalizing the resulting vector (see, e.g., page 130 of [99].) The standard normal distribution can be simulated using the uniform distribution between 0 and 1 (see, e.g., page 117 of [99].) Let $x = x_{(k)}$, where $k = \lceil \frac{1}{\epsilon} \ln(\frac{2n^{c+1}}{\epsilon}) \rceil$. It follows from Lemma 36 and Lemma 37 that the probability that $A \bullet (xx^T) \geq (1+\epsilon)^{-1}\lambda_1$ holds is at least $1 - n^{-c}$. Lemma 35 is proved.

Bibliography

[1] *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, Montréal, Québec, Canada, 23–25 May 1994.

[2] *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, Las Vegas, Nevada, 29 May–1 June 1995.

[3] *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, Philadelphia, Pennsylvania, 22–24 May 1996.

[4] F. Alizadeh. A sublinear-time randomized parallel algorithm for the maximum clique problem in perfect graphs. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 188–194, San Francisco, California, 28–30 Jan. 1991.

[5] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[6] N. Alon and N. Kahale. Approximating the independence number via the θ-fucntion. Unpublished manuscript, Nov. 1994.

[7] K. Appel and W. Haken. Every planar map is four colorable. Part I: Discharging. *Illinois Journal of Mathematics*, 21:429–490, 1977.

[8] K. Appel and W. Haken. Every planar graph is four colorable. *Contemporary Mathematics*, 98:1–741, 1989.

[9] K. Appel, W. Haken, and J. Koch. Every planar map is four colorable. Part II: Reducibility. *Illinois Journal of Mathematics*, 21:491–567, 1977.

[10] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of $\mathcal{NP}$-hard problems. In ACM [2], pages 284–293.

[11] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science*, pages 14–23, Pittsburgh, Pennsylvania, 24–27 Oct. 1992. IEEE.

[12] E. Balas and J. Xue. Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs. *SIAM Journal on Computing*, 20(2):209–221, Apr. 1991.

[13] E. Balas and J. Xue. Addendum: Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs. *SIAM Journal on Computing*, 21(5):1000, Oct. 1992.

[14] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A*, 15:3241–3253, 1982.

[15] F. Barahona. The Max Cut problem in graphs not contractible to K_5. *Operations Research Letters*, 2:107–111, 1983.

[16] F. Barahona. On some applications of the Chinese Postman Problem. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, Flows, and VLSI-Layout*, volume 9 of *Algorithms and Combinatorics*, pages 1–16. Springer-Verlag, 1990.

[17] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical optimization and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

[18] A. I. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete and Computational Geometry*, 13:189–202, 1995.

[19] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley & Sons, second edition, 1990.

[20] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability—towards tight results. In *36th Annual Symposium on Foundations of Computer Science* [82], pages 422–431.

[21] M. Bellare and M. Sudan. Improved non-approximability results. In ACM [1], pages 184–193.

[22] C. Berge. Färbung von Graphen, deren sämtliche bzw, ungerade Kreise starr sind (Zusammenfassung). In *Wissenschaftliche Zeitschrift*, pages 114–115. Martin Luther Universität Halle-Wittenberg, Mathematisch-Naturwissenschaftliche Reihe, 1961.

[23] C. Berge. Sur une conjecture relative au problème des codes optimaux. In *Communication.* 13ème Assemblée Générale de 1'URSI, Tokyo, 1962.

[24] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam and London, 1973.

[25] D. Bertsimas and R. Vohra. Linear programming relaxations, approximation algorithms and randomization; A unified view of covering problems. *Draft*, 1994.

[26] B. Bixby and G. Reinelt. TSPLIB — A library of travelling salesman and related problem instances. http://nhse.cs.rice.edu/softlib/catalog/tsplib.html, Dec. 1992.

[27] A. Blum. An $\tilde{O}(n^{0.4})$-approximation algorithm for 3-coloring (and improved approximation algorithm for k-coloring). In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 535–542, Seattle, Washington, 15–17 May 1989.

[28] A. Blum. Some tools for approximate 3-coloring (extended abstract). In *31st Annual Symposium on Foundations of Computer Science*, volume II, pages 554–562, St. Louis, Missouri, 22–24 Oct. 1990. IEEE.

[29] A. Blum. New approximation algorithms for graph coloring. *Journal of the ACM*, 41(3):470–516, May 1994. The preliminary versions are [27] and [28].

[30] A. Blum and D. Karger. An $\tilde{O}(n^{3/14})$-coloring for 3-colorable graphs. Submitted to *Information Processing Letters*, Jan. 1996.

[31] I. Borosh and L. B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proceedings of the American Mathematical Society*, 55:299–304, 1976.

[32] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix Inequalities in System and Control Theory.* SIAM, 1994.

[33] S. Boyd and S. Wu. SDPSOL: A parser/solver for semidefinite programs with matrix structure. ftp://isl.stanford.edu/pub/boyd/semidef_prog/sdpsol, 1996.

[34] P. Briggs, K. D. Cooper, K. Kennedy, and L. Torczon. Coloring heuristics for register allocation. In *the SIGPLAN'89 Conference on Programming Language Design and Implementation*, pages 257–274, 1989.

[35] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428–455, May 1994.

[36] A. Brøndsted. *An Introduction to Convex Polytopes.* Springer-Verlag, New York Heidelberg Berlin, 1983.

[37] A. M. Bruaset. *A Survey of Preconditioned Iterative Methods.* Longman Scientific & Technical, 1995.

[38] G. J. Chaitin. Register allocation and spilling via graph coloring. In *the SIGPLAN'82 Conference on Compiler Construction*, pages 98–101, 1982.

[39] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, 1981.

[40] K. C. Chang and D. H. Du. Efficient algorithms for layer assignment problems. *IEEE Transactions on Computer-Aided Design*, CAD-6(1):67–78, 1987.

[41] F. Chatelin. *Eigenvalues of Matrices.* John Wiley & Sons, 1993.

[42] R. Chen, Y. Kajitani, and S. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Transactions on Circuits and Systems*, CAS-30(5):284–299, 1983.

[43] B. Chor and M. Sudan. A geometric approach to betweenness. In *Proceedings of the Third Annual European Symposium Algorithms*, pages 227–237, Corfu, Greece, 25–27 Sept. 1995.

[44] F. C. Chow and J. L. Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems*, 12(4):501–536, Oct. 1990.

[45] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[46] V. Chvátal. *Linear Programming.* Freeman, New York, 1983.

[47] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1 Theory.* Birkhäuser, Boston, Basel, Stuttgart, 1985.

[48] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, Cowles Commission Monograph No. 13, pages 339–347. Wiley, New York, NY, 1951.

[49] W. F. de la Vega. MAXCUT has a randomized approximation scheme in dense graphs. unpublished manuscript, Oct. 1994.

[50] D. den Hertog. *Interior Point Approach to Linear, Quadratic and Convex Programming.* Kluwer Academic Publishers, 1994.

[51] U. Feige and M. X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.

[52] R. Fletcher. Semi-definite matrix constraints in optimization. *SIAM Journal on Control and Optimization*, 23(4):493–513, 1985.

[53] R. M. Freund. Complexity of an algorithm for finding an approximate solution of a semi-definite program with no regularity assumption. Technical Report OR-302-94, Operations Research Center, M.I.T., 1994.

[54] A. Frieze and M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. In *Proceedings of the Sixth IPCO Conference on Integer Programming and Combinatorial Optimization*, LNCS 920, pages 1–13. Springer, 1996. The journal version is [55].

[55] A. Frieze and M. Jerrum. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. *Algorithmica*, (to appear). The preliminary version is [54].

[56] K. Fujisawa and M. Kojima. SDPA (Semidefinite Programming Algorithm) users manual. Technical Report B-308, Department Mathematica and Computer Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152, Japan, 1995.

[57] H. N. Gabow and O. Kariv. Algorithms for edge coloring bipartite graphs and multigraphs. *SIAM Journal on Computing*, 11(1):117–129, Feb. 1982.

[58] M. R. Garey and D. S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[59] M. R. Garey, D. S. Johnson, and H. C. So. An application of graph coloring to printed circuit testing (working paper). In *16th Annual Symposium on Foundations of Computer Science*, pages 178–183, The University of California, Berkeley, 13–15 Oct. 1975. IEEE.

[60] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, Feb. 1976.

[61] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2):180–187, June 1972.

[62] M. X. Goemans and D. P. Williamson. .878-approximation algorithms for MAX CUT and MAX 2SAT. In ACM [1], pages 422–431.

[63] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, Nov. 1995. The preliminary version is [62].

[64] R. E. Gomory. An algorithm for integer solutions to linear programs. *Recent Advances in Mathematical Programming*, pages 269–302, 1963.

[65] M. D. Grigoriadis and L. G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. Technical Report DCS-TR-273, Rutgers University, New Brunswick, NJ, 1991.

[66] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[67] M. Grötschel, L. Lovász, and A. Schrijver. Polynomial algorithms for perfect graphs. *Annal of Discrete Mathematics*, 21:325–357, 1984.

[68] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer Verlag, 1988.

[69] M. Grötschel and W. R. Pulleyblank. Weakly bipartite graphs and the max-cut problem. *Operations Research Letters*, 1:23–27, 1981.

[70] R. Gupta, M. L. Soffa, and D. Ombres. Efficient register allocation via coloring using clique separators. *ACM Transactions on Programming Languages and Systems*, 16(3):370–386, May 1994.

[71] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations.* Springer-Verlag, 1994.

[72] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, Sept. 1975.

[73] D. J. Haglin and S. M. Venkatesan. Approximation and intractability results for the maximum cut problem and its variant. *IEEE Transactions on Computers*, 40:110–113, 1991.

[74] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.

[75] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, May 1996.

[76] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, Aug. 1982.

[77] G. H. Holub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, second edition, 1989.

[78] S. Homer and M. Peinado. A highly parallel algorithm to approximate MaxCut on distributed memory architechtures. In *Proceedings of the 9th International Symposium on Parallel Processing (IPPS'95)*, pages 113–117. IEEE Computer Society Press, Apr. 1995.

[79] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

[80] W.-L. Hsu. The coloring and maximum independent set problems on planar perfect graphs. *Journal of the ACM*, 35(3):535–563, July 1988.

[81] IEEE. *30th Annual Symposium on Foundations of Computer Science*, Research Triangle Park, North Carolina, 30 Oct.–1 Nov. 1989.

[82] IEEE. *36th Annual Symposium on Foundations of Computer Science*, Milwaukee, Wisconsin, 23–25 Oct. 1995.

[83] F. Jarre. An interior-point method for minimizing the maximum eigenvalue of a linear combination of matrices. *SIAM Journal on Control and Optimization*, 31(5):1360–1377, 1993.

[84] T. R. Jensen and B. Toft. *Graph Coloring Problems.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, New York, 1995.

[85] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, Dec. 1974.

[86] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. In *35th Annual Symposium on Foundations of Computer Science*, pages 2–13, Santa Fe, New Mexico, 20–22 Nov. 1994. IEEE.

[87] D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In ACM [2], pages 18–25.

[88] H. Karloff. How good is the Goemans-Williamson MAX CUT algorithm? In ACM [3], pages 427–434.

[89] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311, Washington, D.C., 1984.

[90] R. M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, NY, 1972.

[91] C. T. Kelly. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, 1995.

[92] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nuak SSSR*, 244:1093–1096, 1979.

[93] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *the Second Israeli Symposium on Theory and Computing Systems*, pages 250–260, 1992.

[94] P. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAXCUT and COLORING. In ACM [3], pages 338–347.

[95] P. Klein and H.-I. Lu. Space-efficient approximation algorithms for MAXCUT and COLORING semidefinite programs. *Submmitted to SODA'97*, 1996.

[96] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23(3):466–487, June 1994. The preliminary version is [97].

[97] P. Klein, C. Stein, and É. Tardos. Leighton-Rao might be practical: Faster approximation algorithms for concurrent flow with uniform capacities. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 310–321, Baltimore, Maryland, 14–16 May 1990.

[98] J. Kleinberg and M. Goemans. The Lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM Journal on Discrete Mathematics*, to appear.

[99] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1973.

[100] D. E. Knuth. The sandwich theorem. *The Electronic Journal of Combinatorics*, 1(A1):1–48, 1994.

[101] M. Kojima, S. Shindoh, and S. Hara. Interior–point methods for the monotone linear complementarity problem in symmetric matrices. Technical Report B-282, Department of Information Sciences, Tokyo Institute of Technology, 1994.

[102] J. Kuczyński and H. Woźniakowski. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM Journal on Matrix Analysis and Applications*, 13(4):1094–1122, Oct. 1992.

[103] P. Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, 1985.

[104] M. Laurent, S. Poljak, and F. Rendl. Connections between semidefinite relaxations of the max-cut and stable set problems. Technical Report BS-R9502, Department of Operations Research, Statistics, and System Theory, Centrum voor Wiskunde en Informatica, 1995.

[105] T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 101–111, New Orleans, Louisiana, 6–8 May 1991.

[106] L. Lovász. On the Shannon Capacity of a graph. *IEEE Transactions on Information Theory*, IT-25:1–7, 1979.

[107] L. Lovász. Stable sets and polynomials. *Discrete Mathematics*, to appear.

[108] L. Lovász and A. Schrijver. Cones of Matrices and Setfunctions, and 0-1 Optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[109] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems (extended abstract). In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 286–293, San Diego, California, 16–18 May 1993.

[110] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, Sept. 1994. The preliminary version is [109].

[111] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *36th Annual Symposium on Foundations of Computer Science* [82], pages 162–169.

[112] Y. Nesterov and A. Nemirovskii. *Self-Concordant Functions and Polynomial Time Methods in Covex Programming.* Central Economic and Mathematical Institute, USSR Academy of Science, Moscow, 1989.

[113] Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming: Theory and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, 1994.

[114] G. I. Orlova and Y. G. Dorfman. Finding the maximal cut in a graph. *Engineering Cybernetics*, 10(3):502–506, 1972.

[115] W. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[116] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, Dec. 1991.

[117] G. Pataki. On the facial structure of cone-LP's and semi-definite programs. Technical Report MSRR-595, Graduate School of Industrial Administration, Carnegie–Mellon University, 1994.

[118] G. Pataki. On cone-LP's and semi-definite programs: Facial structure, basic solutions and the simplex method. Draft, 1995.

[119] G. Pataki. Cone-LP's and semidefinite programs: Geometry and a simplex-type method. In *Proceedings of the Fifth IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 162–174, Vancouver, B.C., 3–5 June 1996. The preliminary version is [118].

[120] R. Y. Pinter. Optimal layer assignment for interconnect. *Journal of VLSI and Computer Systems*, 1:123–137, 1984.

[121] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *32nd Annual Symposium on Foundations of Computer Science*, pages 495–504, San Juan, Puerto Rico, 1–4 Oct. 1991. IEEE. The journal version is [122].

[122] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, to appear. The preliminary version is [121].

[123] S. Poljak. Integer linear programs and local search for max-cut. *SIAM Journal on Computing*, 24(4):822–839, Aug. 1995.

[124] S. Poljak and D. Turzík. A polynomial algorithm for constructing a large bipartite subgraph. *Canadian Journal of Mathematics*, 34:519–524, 1982.

[125] S. Poljak and Z. Tuza. Maximum cuts and largest bipartite subgraphs. In W. Cook, L. Lovász, and P. Seymour, editors, *Combinatorial Optimization*, volume 20 of *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, pages 181–244. Americal Mathematical Society, 1995.

[126] G. B. Price. *Multivariable Analysis.* Springer-Verlag, 1984.

[127] P. Raghavan. Probabilistic construction of deterministic algorithms approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, Oct. 1988.

[128] P. Raghavan and C. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[129] G. Reinelt. TSPLIB — A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[130] F. Rendl. A Matlab toolbox for semidefinite programming. The program can be found at ftp://orion.uwaterloo.ca/pub/henry/teaching/co769g/, 1994.

[131] F. Rendl, R. Vanderbei, and H. Wolkowicz. A primal-dual interior-point method for the max-min eigenvalue problem. Technical report, University of Waterloo, Dept. of Combinatorics and Optimization, 1993.

[132] H. Rieger, L. Santen, U. Blasum, M. Jünger, and M. Diehl. The critical exponents of the two-dimensional Ising spin glass revisited: Exact ground state calculations and Monte Carlo simulations. *Journal Physica A*, to appear.

[133] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In ACM [3], pages 571–575.

[134] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, New Jersey, 1970.

[135] M. Rosenfeld. On a problem of Shannon. *Proceedings of the American Mathematical Society*, 18:315–319, 1967.

[136] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, 1996.

[137] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3):555–565, July 1976.

[138] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[139] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, Apr. 1990.

[140] C. E. Shannon. The zero-error capacity of a noisy channel. *IRE Transaction on Information Theory*, IT-2(3):8–19, 1956.

[141] J. R. Shewchuk. An introduction to the Conjugate Gradient Method without the agonizing pain. Technical Report CMU-CS-94-125, Shool of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, Mar. 1994.

[142] C. D. Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. *Journal of Statistical Physics*, 80:487–496, 1995.

[143] C. D. Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 84(5/6), 1996.

[144] G. Strang. *Linear Algebra and Its Applications.* Academic Press, New York, second edition, 1980.

[145] A. Tucker and D. Wilson. An $O(N^2)$ algorithm for coloring perfect planar graphs. *Journal of Algorithms*, 5(1):60–68, Mar. 1984.

[146] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets (extended abstract). In *30th Annual Symposium on Foundations of Computer Science* [81], pages 338–343.

[147] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication (extended abstract). In *30th Annual Symposium on Foundations of Computer Science* [81], pages 332–337.

[148] L. Vandenberghe and S. Boyd. SP: Software for semidefinite programming. ftp://isl.stanford.edu/pub/boyd/semidef_prog/, 1994.

[149] L. Vandenberghe and S. Boyd. A primal-dual potential reduction method for problems involving matrix inequalities. *Mathematical Programming, Series B*, 69:205–236, 1995.

[150] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, Mar. 1996.

[151] P. M. Vitányi. How well can a graph be n-colored? *Discrete Mathematics*, 34:69–80, 1981.

[152] A. Wigderson. A new approximate graph coloring algorithm. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 325–329, San Francisco, California, 5–7 May 1982.

[153] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM*, 30(4):729–735, Oct. 1983. The preliminary version is [152].

[154] D. C. Wood. A technique for coloring a graph applicable to large-scale optimization problems. *Computer Journal*, 12:317, 1969.

[155] M. Yannakakis. Node- and edge-deletion NP-complete problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing*, pages 253–264, San Diego, California, 1–3 May 1978.

[156] Y. Ye. An $O(n^3L)$ potential reduction algorithm for linear programming. *Mathematical Programming*, 50:239–258, 1991.

[157] A. Yoshise. An optimization method for convex programs—interior-point method and analytical center. *Systems, Control and Information*, 38(3):155–160, Mar. 1994.

[158] N. E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, San Fransico, California, 22–24 Jan. 1995.