11. An Introduction to Semidefinite Programming and its Applications to Approximation Algorithms*

Thomas Hofmeister, Martin Hühne,

Informatik 2, Universität Dortmund, 44221 Dortmund

11.1 Introduction

$\mathcal{NP}$-hard problems are often tackled using the *relaxation method* approach. The idea of this approach is to add or remove conditions to or from the original problem in such a way that the solution space is enlarged. The hope is that the new solution space will be more smooth in the sense that it allows a more efficient, i.e., polynomial-time, computation of the optimum.

One possibility of applying this method lies in computing upper or lower bounds within a branch-and-bound approach. Let us assume that we are dealing with a maximization problem. Since the modified solution space contains the original solution space as a subset, the optimal solution to the relaxed problem is an upper bound to the optimum of the original problem.

Known examples for problems which can be treated in this way include the KNAPSACK problem. Given n objects with size c_j and value t_j, we want to fill a knapsack such that its value is maximized. This is an $\mathcal{NP}$-hard problem, but if we are allowed to take fractional parts of the objects, then we can solve the problem in polynomial time, using a greedy algorithm. The greedy solution provides us with an upper bound for the original KNAPSACK problem.

One can also try to apply some transformation (often called "rounding") to a solution of the relaxed problem in order to obtain a good solution for the original problem. The quality of the so obtained solution can then be estimated by comparing it with the upper bound provided by the relaxed solution. There are cases, like "totally unimodular" linear programs, where relaxations even yield the optimal solutions, as is the case with the weighted matching problem.

The relaxation method can also be combined with *semidefinite programs*. Semidefinite programming is a generalization of linear programming. It has been studied for quite some time. For example, it is known that semidefinite programs can be solved in polynomial time. The attention of algorithm designers has turned to semidefinite programming only recently. This is due to a paper by

* This article will appear as a chapter of a book on the PCP theorem and approximation algorithms in the Springer-Verlag. References to chapters within this report refer to chapters within that book.

Goemans and Williamson [GW95], where for several $\mathcal{NP}$-hard problems efficient approximation algorithms based on semidefinite programming were presented. One of these problems is MaxCut, where the set of vertices of a given graph has to be partitioned into two parts such that the number of edges between the parts is maximized. (For a definition of the other problems considered in this chapter, see the end of this introduction.) Goemans and Williamson describe a randomized polynomial-time algorithm that computes a cut of size at least 0.878 times the size of the maximum cut.

The idea is to first formulate the MaxCut-problem as an integer quadratic program. The relaxation considers not only integer variables as possible candidates for the solution, but n-dimensional unit vectors. By this relaxation, the solution space becomes much larger, but the resulting problem can be solved in polynomial time with the help of semidefinite programming. From the solution obtained in this way, an integer solution can be constructed by randomly selecting a hyperplane and identifying the vectors with either 1 or -1, depending on which side of the plane they are. It can be shown that the integer solution so obtained has an expectation for the objective function which is at least 0.878 times the relaxed solution. Thus, Goemans and Williamson have provided a *randomized* $1/0.878 \approx 1.139$-approximation algorithm for MaxCut, improving upon the previously best approximation ratio of 2. In the conference version of their paper, they also sketched how this method might be derandomized into a deterministic algorithm. However, their derandomization contains a flaw, which is reported in the journal version of that paper. A correct but complex derandomization procedure has been given by Mahajan and Ramesh [MR95a].

It should be noted that (unless $\mathcal{P}$=$\mathcal{NP}$) there can be no polynomial-time approximation algorithm with an approximation ratio of $1+\varepsilon$ for arbitrarily small ε, since the by now famous PCP-theorem has as one of its consequences that there is a constant $\varepsilon_0 > 0$ such that computing $1+\varepsilon_0$-approximations for MaxCut is $\mathcal{NP}$-hard. In fact, this constant has recently been made more explicit by Håstad in [Hås97] who showed that (for all $\varepsilon > 0$) already computing a $(17/16-\varepsilon)$-approximation is $\mathcal{NP}$-hard.

The article by Goemans and Williamson has started a series of research activities. Several authors have studied the question whether semidefinite programming can also be applied to other graph problems or, more general, classes of problems. The papers by Karger, Motwani and Sudan [KMS94], Klein and Lu [KL96] as well as Bacik and Mahajan [BM95] go into this direction. Karloff investigates in [Kar96] whether the estimations in [GW95] are sharp and also shows that additional linear constraints can not lead to better approximation ratios.

The essence of all these papers is that semidefinite programming is a very versatile and powerful tool. Besides MaxCut, it has been applied to obtain better approximation algorithms for several other important $\mathcal{NP}$-hard problems. Among these are the problems MaxSat, Max$k$Sat, MaxEkSat (especially for $k = 2$) and MaxDiCut. In the following table, the approximation ratios α obtained in

[GW95] are listed. The table also lists $1/\alpha$, since that value is more commonly used in that paper and its successors.

Problem	$1/\alpha$	α
MaxDiCut	0.796	1.257
Max2Sat	0.878	1.139
MaxSat	0.755	1.324
MaxSat	0.758	1.319

(In this chapter, the approximation ratios are rounded to three decimal digits.)

The first of the two MaxSat-algorithms was given in the conference version and the second one in the final version. Before, the best known approximation ratio for MaxDiCut was 4 [PY91]. For MaxE2Sat, a 1.334-approximation algorithm was known [Yan92, GW94].

Tuning the approach of semidefinite programming, several authors studied improvements of the approximation algorithms for these problems. At present, polynomial-time algorithms with the following approximation ratios are known:

Problem	$1/\alpha$	α	Reference
MaxDiCut	0.859	1.165	Feige/Goemans [FG95]
MaxE2Sat	0.931	1.075	Feige/Goemans [FG95]
MaxSat	0.765	1.307	Asano/Ono/Hirata [AOH96]
MaxSat	0.767	1.304	Asano/Hori/Ono/Hirata [AHOH96]
MaxSat	0.770	1.299	Asano [Asa97]

We will discuss some of the improvements in this chapter. However, our emphasis will be on *methods* rather than on results. I.e., we will not present the very last improvement on the approximation of these problems. Instead, we present the basic methods and techniques which are needed. More precisely, we will look at five basic methods used in the approximation algorithms for MaxDiCut and MaxSat:

1. The modeling of asymmetric problems such as MaxDiCut and MaxE2Sat.
2. A method for handling long clauses in MaxSat instances.
3. Combining different approximation algorithms for MaxEkSat yields better approximations for MaxSat.
4. Adding linear restrictions to the semidefinite program can improve the approximation ratio for MaxDiCut and MaxSat. However, there are no linear restrictions which improve the MaxCut approximation from [GW95].
5. Nonuniform rounding of the solution of the semidefinite program is a way to take advantage of the modeling of asymmetric problems. This improves the approximation ratio. The ratio is further improved if nonuniform rounding is combined with method 4.

Note that there are several other approaches for the approximation of MaxCut and MaxSat. For example, in Chapter 12, the "smooth integer programming" technique is described, which can be used if the instances are dense. For additional references on semidefinite programming and combinatorial optimization, we refer to [Ali95, VB96, Goe97].

Our survey will be organized as follows: First, we describe the mathematical tools necessary to understand the method of semidefinite programming (Sections 11.2–11.4). Since algorithms for solving semidefinite programs are rather involved, we only give a sketch of the so-called interior-point method. We describe a "classical" approximation of MaxCut, which is not based on semidefinite programming (Section 11.5.1). We then show how Goemans and Williamson applied semidefinite programming to obtain better approximation algorithms for MaxCut (Section 11.5.2) and analyze the quality of their algorithm. In Section 11.6 we describe the modeling of the asymmetric problems MaxDiCut and MaxE2Sat. A method for modeling long clauses in a semidefinite program is described in Section 11.7.1. We review some classical approaches for MaxSat and explain how different MaxSat algorithms can be combined in order to improve the approximation ratio (Section 11.7.2). Section 11.8 describes the effect of additional constraints and of a nonuniform rounding technique.

We give a formal definition of the problems considered in this chapter.

MaxCut

Instance: Given an undirected graph $G = (V, E)$.
Problem: Find a partition $V = V_1 \cup V_2$ such that the number of edges between V_1 and V_2 is maximized.

We will assume that $V = \{1, \ldots, n\}$.

MaxDiCut

Instance: Given a directed graph $G = (V, E)$.
Problem: Find a subset $S \subseteq V$ such that the number of edges $(i, j) \in E$ with tail i in S and head j in $\overline{S}$ is maximal.

MaxSat

Instance: Given a Boolean formula Φ in conjunctive normal form.
Problem: Find a variable assignment which satisfies the maximal number of clauses in Φ.

Max$k$Sat, $k \in \mathbb{N}$

Instance: Given a Boolean formula Φ in conjunctive normal form with at most k literals in each clause.
Problem: Find a variable assignment which satisfies the maximal number of clauses in Φ.

MaxEkSat, $k \in \mathbb{N}$

Instance: Given a Boolean formula Φ in conjunctive normal form with exactly k literals in each clause.

Problem: Find a variable assignment which satisfies the maximal number of clauses in Φ.

The variables of the Boolean formula are denoted by $x_1, \ldots, x_n$, the clauses are $C_1, \ldots, C_m$. Note that each algorithm for Max$k$Sat is also an algorithm for MaxEkSat. An important special case is $k = 2$.

Sometimes, weighted versions of these problems are considered.

weighted MaxCut

Instance: Given an undirected graph $G = (V, E)$ and positive edge weights.

Problem: Find a partition $V = V_1 \cup V_2$ such that the sum of the weights of edges between V_1 and V_2 is maximized.

weighted MaxDiCut

Instance: Given a directed graph $G = (V, E)$ and nonnegative edge weights.

Problem: Find a subset $S \subseteq V$ such that the total weight of the edges $(i, j) \in E$ with tail i in S and head j in $\overline{S}$ is maximal.

weighted MaxSat

Instance: Given a Boolean formula $\Phi = C_1 \wedge \ldots \wedge C_m$ in conjunctive normal form and nonnegative clause weights.

Problem: Find a variable assignment for Φ which maximizes the total weight of the satisfied clauses.

All results mentioned in this chapter hold for the unweighted and for the weighted version of the problem. The approximation ratio for the weighted version is the same as for the corresponding unweighted version. Since the modification of the arguments is straightforward, we restrict ourselves to the unweighted problems.

11.2 Basics from Matrix Theory

In this section, we will recall a few definitions and basic properties from Matrix Theory. The reader interested in more details and proofs may consult, e.g., [GvL86]. Throughout this chapter, we assume that the entries of a matrix are real numbers.

An $n \times m$ matrix is called *square* iff $n = m$. A matrix A such that $A = A^T$ is called *symmetric*. A matrix A such that $A^T A$ is a diagonal matrix is called *orthogonal*. It is *orthonormal* if in addition $A^T A = Id$, i.e., $A^T = A^{-1}$.

Definition 11.1. *An* eigenvector *of a matrix A is a vector $x \neq 0$ such that there exists a $\lambda \in \mathbb{C}$ with $A \cdot x = \lambda \cdot x$. The corresponding λ is called an* eigenvalue. *The* trace *of a matrix is the sum of the diagonal elements of the matrix.*

It is known that the trace is equal to the sum of the eigenvalues of A and the determinant is equal to the product of the eigenvalues.

Note that zero is an eigenvalue of a matrix if and only if the matrix does not have full rank.

Another characterization of eigenvalues is as follows:

Lemma 11.2. *The eigenvalues of matrix A are the roots of the polynomial $\det(A - \lambda \cdot Id)$, where λ is the free variable and Id is the identity matrix.*

It is known that if A is a symmetric matrix, then it possesses n real eigenvalues (which are not necessarily distinct). For such an $n \times n$- matrix, the canonical numbering of its n eigenvalues is $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_n$. The number of times that a value appears as an eigenvalue is also called its *multiplicity.*

Definition 11.3. *The* inner product *of two matrices is defined by*

$$A \bullet B := \sum_{i,j} A_{i,j} \cdot B_{i,j} \quad = \quad trace(A^T \cdot B).$$

Theorem 11.4 (Rayleigh-Ritz). *If A is a symmetric matrix, then*

$$\lambda_1(A) = \max_{||x||=1} x^T \cdot A \cdot x \qquad \textit{and} \qquad \lambda_n(A) = \min_{||x||=1} x^T \cdot A \cdot x.$$

Semidefiniteness

Having recalled some of the basic matrix notions and lemmata, let us now turn to the definitions of definiteness.

Definition 11.5. *A square matrix A is called*

$$\left.\begin{array}{lll} \text{positive semidefinite} & \textit{if} & x^T \cdot A \cdot x \geqslant 0 \\ \text{positive definite} & \textit{if} & x^T \cdot A \cdot x > 0 \end{array}\right\} \textit{for all } x \in \mathbb{R}^n \setminus \{\underline{0}\}$$

In this chapter, we will abbreviate "positive semidefinite" also by "PSD."

As a corollary of Theorem 11.4, we obtain that a symmetric matrix is PSD if and only if its smallest eigenvalue $\lambda_n \geqslant 0$.

Examples of PSD matrices. It should be noted that it is not sufficient for a matrix to contain only positive entries to make it PSD. Consider the following two matrices

$$A := \begin{pmatrix} 1 & 4 \\ 4 & 1 \end{pmatrix} \qquad B := \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix}.$$

The matrix A is not PSD. This can be seen by observing that $(1,-1) \cdot A \cdot (1,-1)^T = -6$. The eigenvalues of A are 5 and -3. On the other hand, the symmetric matrix B with negative entries is PSD. The eigenvalues of B are approximately 4.30 and 0.697.

For a diagonal matrix, the eigenvalues are equal to the entries on the diagonal. Thus, a diagonal matrix is PSD if and only if all of its entries are nonnegative.

Also by the definition, it follows that whenever B and C are matrices, then the matrix

$$A = \begin{pmatrix} B & 0 \\ 0 & C \end{pmatrix}$$

is PSD if and only if B and C are PSD. This means that requiring some matrices $A_1, \dots, A_r$ to be PSD, is equivalent to requiring *one* particular matrix to be PSD.

Also, the following property holds: If A is PSD, then the $k \times k$-submatrix obtained from A by eliminating rows and columns $k+1$ to n is also PSD.

As a final example, consider the symmetric matrix

$$\begin{pmatrix} 1 & a & \dots & a \\ a & 1 & \dots & a \\ \dots & \dots & \dots & \dots \\ a & a & \dots & 1 \end{pmatrix}$$

with ones on the diagonal and an a everywhere else. A simple calculation reveals that its eigenvalues are $1 + a \cdot (n-1)$ (with multiplicity 1) and $1 - a$ (with multiplicity $n-1$). Hence, the matrix is PSD for all $-1/(n-1) \leqslant a \leqslant 1$.

Further properties. In most places in this chapter, we will only be concerned with matrices A which are symmetric. For symmetric matrices, we have the following lemma:

Lemma 11.6. *Let A be an $n \times n$ symmetric matrix. The following are equivalent:*

- *A is PSD.*
- *All eigenvalues of A are nonnegative.*
- *There is an $n \times n$-matrix B such that $A = B^T \cdot B$.*

Note that given a symmetric PSD matrix A, the decomposition $A = B^T \cdot B$ is not unique, since the scalar product of vectors is invariant under rotation, so any rotation of (the vectors given by) matrix B yields another decomposition. As a consequence, we can always choose B to be an upper triangular matrix in the decomposition.

Goemans-Williamson's approximation algorithm uses a subroutine which computes the "(incomplete) Cholesky decomposition" of a given PSD symmetric matrix A. The output of the subroutine is a matrix B such that $A = B^T \cdot B$. This decomposition can be performed in time $\mathcal{O}(n^3)$, see e.g. [GvL86].

As an example, consider the following Cholesky decomposition:

$$A = \begin{pmatrix} 1 & -1 \\ -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & \sqrt{3} \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 \\ 0 & \sqrt{3} \end{pmatrix}$$

Remark 11.7. The problem of deciding whether a given square matrix A is PSD can be reduced to the same problem for a symmetric matrix A'. Namely, define A' to be the symmetric matrix such that $A'_{i,j} := \frac{1}{2} \cdot (A_{i,j} + A_{j,i})$. Since

$$x^T \cdot A \cdot x \;=\; \sum_{i,j} A_{i,j} \cdot x_i \cdot x_j ,$$

it holds that $x^T \cdot A' \cdot x = x^T \cdot A \cdot x$ and A' is PSD if and only if A is PSD.

11.3 Semidefinite Programming

One problem for a beginner in semidefinite programming is that in many papers, semidefinite programs are defined differently. Nevertheless, as one should expect, most of those definitions turn out to be equivalent. Here is one of those possible definitions:

Definition 11.8. *A* semidefinite program *is of the following form. We are looking for a solution in the real variables $x_1, \ldots, x_m$. Given a vector $c \in \mathbb{R}^m$, we want to minimize (or maximize) $c \cdot x$. The feasible solution space from which we are allowed to take the x-vectors is described by a symmetric matrix SP which as its entries contains linear functions in the x_i-variables. Namely, an x-vector is feasible iff the matrix SP becomes PSD if we plug the components of x into the corresponding positions.*

Here is an example of a semidefinite program:

$$\begin{array}{ll} \textbf{maximize} & x_1 + x_2 + x_3 \\ \textbf{such that} & \begin{pmatrix} 1 & x_1 - 1 & 2x_1 + x_2 - 1 \\ x_1 - 1 & -x_1 & x_2 + 3 \\ 2x_1 + x_2 - 1 & x_2 + 3 & 0 \end{pmatrix} \text{ is PSD.} \end{array}$$

Again, by Remark 11.7, when we are modeling some problem, we need not restrict ourselves to symmetric matrices only.

The matrix that we obtain by plugging into SP the values of vector x will also sometimes be written as $SP(x)$.

Given an ε and a semidefinite program which has some finite, polynomially bounded solution, the program can be solved in polynomial time, up to an error term of ε. In general, an error term cannot be avoided since the optimal solution to a semidefinite program can contain irrational numbers.

Semidefinite programming is a generalization of linear programming. This can be seen as follows. Let the linear inequalities of the linear program be of the form $a_1x_1 + \ldots + a_mx_m + b \geqslant 0$. Given r linear inequalities $IN_i \geqslant 0$, $(1 \leqslant i \leqslant r)$, we can define the matrix SP for the semidefinite program to be of the following diagonal form:

$$\begin{pmatrix} IN_1 & 0 & \cdots & 0 \\ 0 & IN_2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & IN_r \end{pmatrix}$$

Since a diagonal matrix is PSD if and only if all of its entries are nonnegative, we have that the feasible solution spaces for the linear program and the semidefinite program are equal. Hence, linear programming can be seen as the special case of semidefinite programming where the given symmetric matrix SP is restricted to be diagonal.

The feasible solution space which is defined by a semidefinite program is convex. This can be seen as follows:

Given two feasible vectors x and y and its corresponding matrices $SP(x)$ and $SP(y)$, all vectors "between" x and y are also feasible, i.e., for all $0 \leqslant \lambda \leqslant 1$,

$$SP\left((1-\lambda)x + \lambda y\right) = (1-\lambda)SP(x) + \lambda SP(y)$$

is also PSD, as can easily be seen: For all v,

$$v^T \cdot SP((1-\lambda)x + \lambda y) \cdot v = v^T \cdot (1-\lambda) \cdot SP(x) \cdot v + v^T \cdot \lambda \cdot SP(y) \cdot v \geqslant 0.$$

This means that the feasible solution space which we can describe with the help of a semidefinite program is a convex space. As a consequence, we are not able to express a condition like "$x \geqslant 1$ or $x \leqslant -1$" in one dimension.

For a beginner, it is hard to get a feeling for what can and what cannot be formulated as a semidefinite program. One of the reasons is that - contrary to linear programs - it makes a big difference whether an inequality is of the type "$\leqslant$" or "$\geqslant$", as we shall soon see.

In the following, we show that a large class of quadratic constraints can be expressed in semidefinite programs.

11.3.1 Quadratically constrained quadratic programming

Quadratically constrained quadratic programming can be solved using semidefinite programming, as we shall see now. Nevertheless, Vandenberghe and Boyd [VB96] state that as far as efficiency in the algorithms is concerned, one should better use interior-point methods particularly designed for this type of problems. A quadratically constrained quadratic program can be written as

$$\begin{array}{ll} \textbf{minimize} & f_0(x) \\ \textbf{such that} & f_i(x) \leqslant 0, \quad i = 1, \ldots, L \end{array}$$

where the f_i are convex quadratic functions $f_i(x) = (A_i x + b_i)^T (A_i x + b_i) - c_i^T x - d_i$. The corresponding semidefinite program looks as follows:

$$\begin{array}{ll} \textbf{minimize} & t \\ \textbf{such that} & \begin{pmatrix} I & A_0 x + b_0 \\ (A_0 x + b_0)^T & c_o^T x + d_0 + t \end{pmatrix} \text{ is PSD and} \\ & \forall i : \begin{pmatrix} I & A_i x + b_i \\ (A_i x + b_i)^T & c_i^T x + d_i \end{pmatrix} \text{ is PSD.} \end{array}$$

As we have seen earlier, the AND-condition of matrices being PSD can easily be translated into a semidefinite program.

As an example, consider the following problem in two dimensions:

$$\begin{array}{ll} \textbf{minimize} & -x + \dfrac{y}{2} \\ \textbf{such that} & y \geqslant x^2 \text{ and } y \leqslant \dfrac{x}{3} + \dfrac{1}{2} \end{array} \tag{11.1}$$

The space of feasible solutions consists of all points located "between" the parabola and the straight line shown in Figure 11.1. We leave it as an exercise to the reader to compute the optimal solution (Exercise 11.2).

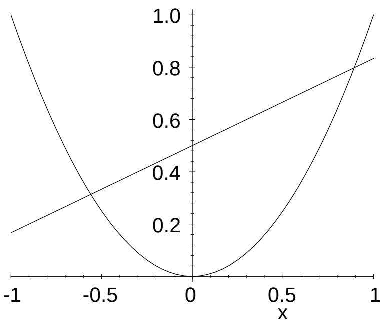


Fig. 11.1. Graphical representation of the program (11.1).

Since the feasible space is convex, we are not able to express a condition like $y \leqslant x^2$ in a semidefinite program. This shows that one has to be careful with the direction of the inequalities, in contrast to linear programs.

As another example, consider the symmetric 2×2-matrix

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix}.$$

This matrix is PSD if and only if $a \geqslant 0, c \geqslant 0$, and $b^2 \leqslant ac$. Consequently,

$$\begin{pmatrix} x_1 & 2 \\ 2 & x_2 \end{pmatrix}$$

allows us to describe the feasible solution space $x_1 \cdot x_2 \geqslant 4, x_1 \geqslant 0$.

Finally, we remark that in some papers, a semidefinite program is defined by

$$\textbf{minimize} \;\; C \bullet X \;\; \textbf{such that} \;\; A_i \bullet X = b_i \;\; (1 \leqslant i \leqslant m) \text{ and } X \text{ is PSD}$$

where X, C, A_i are all symmetric matrices. In fact, this is the dual (see Section 11.4) of the definition we gave.

11.3.2 Eigenvalues, Graph Theory and Semidefinite Programming

There are interesting connections between eigenvalues and graph theory, let us for example state without proof the "Fundamental Theorem of Algebraic Graph Theory" (see, e.g., [MR95b, p. 144]).

Theorem 11.9. *Let $G = (V, E)$ be an undirected (multi)graph with n vertices and maximum degree Δ. Then, under the canonical numbering of its eigenvalues λ_i for the adjacency matrix $A(G)$, the following holds:*

1. *If G is connected, then $\lambda_2 < \lambda_1$.*
2. *For $1 \leqslant i \leqslant n$, $|\lambda_i| \leqslant \Delta$.*
3. *Δ is an eigenvalue if and only if G is regular.*
4. *G is bipartite if and only if for every eigenvalue λ there is an eigenvalue $-\lambda$ of the same multiplicity.*
5. *Suppose that G is connected. Then, G is bipartite if and only if $-\lambda_1$ is an eigenvalue.*

Another connection is described by the so-called *Lovász number.* Given an adjacency matrix A, we obtain the matrix A' by changing A as follows: Put 1's on the main diagonal, and replace every 0 at a position (i,j) by the variable $x_{i,j}$.

The Lovász number of a graph is defined by

$$\theta(G) = \min_{\underline{x}\in\mathbb{R}^n} \lambda_1(A'(\underline{x}))$$

The Lovász number of a graph can be used to obtain an upper bound for the maximum clique-size $\omega(G)$ in a graph G, namely: $\omega(G) \leqslant \theta(G)$.

We sketch the proof of this property: Without loss of generality, the largest clique of size k is on the first k vertices. This means that the k-th "principal submatrix" contains ones only. From matrix theory, it is known that the largest eigenvalue of a matrix is at least the value of the largest eigenvalue of any principal submatrix. Since k is an eigenvalue of the k-th principal submatrix, we obtain

$$\theta(G) \geqslant \lambda_1(A') \geqslant k = \omega(G).$$

The largest eigenvalue of a symmetric matrix can be computed with the help of a semidefinite program. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of a matrix A. Then the eigenvalues of $z \cdot Id - A$ are $z - \lambda_1, \ldots, z - \lambda_n$. Hence, the solution to the following semidefinite program yields the largest eigenvalue of A:

$$\textbf{minimize} \ \ z \ \ \textbf{such that} \ \ z \cdot Id - A \ \ \text{is PSD}$$

It is now obvious that we can also compute the Lovász number of a graph with the help of semidefinite programming.

11.4 Duality and an Interior-Point Method

Duality is an important concept in mathematical programming. It allows one to prove optimality of a given solution in a simple fashion. Furthermore, it plays a crucial role in algorithms which employ the interior-point method to solve such mathematical programs.

In this section, we will sketch very roughly the interior-point method by Ye which can be employed in solving a semidefinite program. The reader interested in details is referred to [Ye90].

Let us first recall the concept of duality from linear programming. A linear program (LP) consists (without loss of generality) of the following:

$$\textbf{minimize} \ c^T x \ \ \textbf{such that} \ \ Ax = b \ \ \text{and} \ \ x \geqslant 0.$$

As an example,

$$\textbf{minimize } x_1 + 3x_2 + x_3 \textbf{ such that } x_1 + x_2 = 3, \; x_3 - x_1 = 4, \; x_1, x_2, x_3 \geqslant 0.$$

Without solving the linear program explicitly, we can get bounds from the linear equalities. For example, since the objective function $x_1 + 3x_2 + x_3$ is at least x_2+x_3, and adding the first two linear equalities yields the condition $x_2+x_3 = 7$, we know that the optimal solution cannot be smaller than seven. One can now see that taking any linear combination of the equalities yields a bound whenever it is smaller than the objective function. Of course, we want to obtain a bound which is as large as possible. This yields the following problem.

$$\textbf{maximize } b^T \cdot y \textbf{ such that } A^T \cdot y \leqslant c.$$

This is another linear program which is called the *dual* of the original one, which is called the *primal.* Our arguments above can be used to show that any solution to the dual yields a lower bound for the primal.

This principle is known as *weak duality*: The optimum value of the dual is not larger than the optimum value of the primal. Even more can be shown, namely that for linear programming, *strong* duality holds, i.e., both values agree:

If the primal or the dual is feasible, then their optima are the same.

This shows that duality can be used to prove optimality of some solution easily: We just provide two vectors x and y which are feasible points in the primal and dual, respectively. If $b^T y = c^T x$, we know that x must be an optimal point in the primal and dual problem.

It also makes sense to measure for an arbitrary point x how far away it is from the optimum. For this purpose, we solve the primal and dual simultaneously. Given a primal-dual pair x, y in between, one defines the *duality gap* as $c^T \cdot x - b^T \cdot y$. The smaller the gap, the closer we are to the optimum.

Similar concepts hold for semidefinite programming. Before exhibiting what the correspondences are, we want to sketch how duality can be used in an interior-point method for solving a linear program.

Khachian was the first to come up with a polynomial-time algorithm for solving linear programs. His method is known as the *ellipsoid algorithm.* Later, Karmarkar found another, more practical, polynomial-time algorithm which founded a family of algorithms known as *the interior-point method.*

On a very abstract level, an interior-point algorithm proceeds as follows: Given a point x in the interior of the polyhedron defined by the linear inequalities, it maps the polyhedron into another one in which x is "not very close to the boundaries." Then, the algorithm moves from x to some x' in the transformed space, and it maps x' back to some point in the original space. This step is repeated until some potential function tells us that we are very close to the optimal solution. Then, we can either stop and be happy with the good enough approximation, or we can apply another procedure which from this point obtains an optimal solution.

In defining the potential function, the size L of a linear program is needed which measures how many bits we need to store the program. We do not want to give a formal definition, since it is very close to the intuition. As one consequence, it is valid that every vertex of a linear program has rational coordinates where the numerator and denominator can be written using L bits only. (A vertex is a feasible point x such that there is no vector $y \neq 0$ with $x+y$ and $x-y$ feasible.)

There is the following lemma:

Lemma 11.10. *If x_1 and x_2 are vertices of $Ax = b, x \geqslant 0$, then either $c^T x_1 - c^T x_2 = 0$ or $c^T x_1 - c^T x_2 > 2^{-2L}$.*

This means that we only need to evaluate the objective function with an error less than 2^{-2L} since whenever we have reached a point which has distance less than 2^{-2L} to the optimum, then it must be the optimal point.

The first step in Ye's interior-point algorithm consists of *affine scaling*. Given a current solution pair x, y for the primal and dual, a scaling transformation (which depends on x and y) is applied to the LP. This affine scaling step does not change the duality gap.

The potential function is defined as follows:

$$G(x,y) := q \cdot \ln(x^T \cdot y) - \sum_{i=1}^{n} \ln(x_i y_i) \qquad \text{for} \quad q = n + \lceil \sqrt{n}\, \rceil.$$

From the definition of the potential function G, it follows that if $G(x,y) \leqslant -k\sqrt{n}L$, then $x^T \cdot y$ is small enough to imply that x is an optimal solution to the primal problem. The high-level description of Ye's algorithm now looks as follows:

<u>Ye's algorithm</u>
while $G(x,y) > -2\sqrt{n}L$
  do affine scaling
  do change either x or y according to some rule
end

It can be shown that the "rule" above can be chosen such that in every step of the while-loop, the potential function decreases by a constant amount, say $\frac{7}{120}$. It can also be shown that one can obtain an initial solution which has a potential of size $\mathcal{O}(\sqrt{n}L)$.

Altogether, this guarantees that the while loop will be executed $\mathcal{O}(\sqrt{n}L)$ many times, and it can be shown that every step of the operation can be performed in time $\mathcal{O}(n^3)$.

Let us now try to sketch very roughly what this algorithm looks like in the context of semidefinite programs: The primal and dual problems look as follows:

Primal	**Dual**
maximize $b^T y$	**minimize** $C \bullet X$
such that $C - \sum_{i=1}^{m} y_i A_i$ is PSD.	**such that** $A_i \bullet X = b_i, i = 1 \dots m$ and X is PSD.

Here is a proof of weak duality:

Lemma 11.11. *Let X be a feasible matrix for the dual and y be any feasible vector for the primal. Then $C \bullet X \geqslant b^T y$.*

Proof. $$C \bullet X - \sum_{i=1}^{m} b_i y_i \; = \; C \bullet X - \sum_{i=1}^{m} (A_i \bullet X) y_i \; = \; (C - \sum_{i=1}^{m} y_i A_i) \bullet X \; \geqslant \; 0.$$

The last inequality holds since the inner product of two PSD matrices is non-negative. ∎

One of the things that make semidefinite programming more complex than linear programming is that the strong duality of a problem pair cannot be proved in a fashion as simple as for linear programming.

Nevertheless, it can be shown that whenever a polynomial *a priori bound* on the size of the primal and dual feasible sets are known, then the primal-dual pair of problems can be transformed into an equivalent pair for which strong duality holds.

A primal-dual solution pair now is a pair of matrices X and Y, and the potential function is defined as follows:

$$G(X, Y) = q \cdot \ln(X \bullet Y) - \ln \det(XY).$$

Like in Ye's algorithm, one now proceeds with a while-loop where in every single step the potential function is reduced by a constant amount, and after at most $\mathcal{O}(\sqrt{n}\ |\log \varepsilon|\ )$ executions of the while-loop a solution with duality gap at most ε can be found. Nevertheless, the details are much more difficult and beyond the scope of this survey. The interested reader may find some of those details in [Ali95] or in [Ye90].

As Alizadeh [Ali95] notes, the remarkable similarity between Ye's algorithm for linear programming and its version for semidefinite programming suggests that other LP interior-point methods, too, can be turned into algorithms for semidefinite programming rather mechanically.

Why an a priori bound makes sense. Whereas for a linear program, we have a bound of 2^L on the size of the solutions given the size L of the linear program, the situation is different for semidefinite programs. Consider the following example from Alizadeh:

$$\textbf{minimize } x_n \quad \textbf{such that} \quad x_1 = 2 \text{ and } x_i \geqslant x_{i-1}^2$$

The optimum value of this program is of course 2^{2^n}. This is a semidefinite program since we have already seen earlier that the condition $x_i \geqslant x_{i-1}^2$ can be expressed. Just outputting this solution would take us time $\Omega(2^n)$.

11.5 Approximation Algorithms for MaxCut

We recall the definition of MaxCut.

<u>MaxCut</u>
Instance: Given an undirected graph $G = (V, E)$.
Problem: Find a partition $V = V_1 \cup V_2$ such that the number of edges between V_1 and V_2 is maximized.

MaxCut is an $\mathcal{NP}$-hard problem. There is also a weighted version in which there is a positive weight for every edge and we are asked to compute a partition where the sum of the edge weights in the cut is maximized.

11.5.1 MaxCut and Classical Methods

It is known that the weight of the largest cut is at least 50 percent of the graph weight. A simple probabilistic argument works as follows: Scan the vertices v_1 to v_n and put each independently with probability $1/2$ into V_1 and with probability $1/2$ into V_2. The probability of an edge being in the cut is $1/2$. By linearity of expectation, the expected value is at least 50% of the graph weight, hence there exists a cut of this weight. As one might guess, this simple argument can be derandomized, leading to a simple greedy strategy which is a 2-approximation algorithm.

It has also been shown by Ngoc and Tuza (see, e.g., the survey paper by Poljak and Tuza [PT95]) that for every $0 < \varepsilon < 1/2$, it is $\mathcal{NP}$-complete to decide whether the largest cut of a graph has size at least $(1/2 + \varepsilon) \cdot |E|$.

Before the work of Goemans and Williamson, progress has only been on improving additional terms. We sketch one of those results from [HL96] here:

Theorem 11.12. *Every graph G has a cut of size at least $\frac{w(G)+w(M)}{2}$, where M is a matching in G. (Here, $w(G)$ and $w(M)$ denote the sum of weights of the edges in those subgraphs.) Given M, such a cut can be computed in linear time.*

We give a proof sketch of the existence. We process the edges of the matching consecutively. For an edge $e = \{v, w\}$ of the matching, we either add v to V_1 and w to V_2, or we add v to V_2 and w to V_1, each with probability 1/2. Remaining vertices are distributed independently to either V_1 or V_2, each with probability 1/2. It can now be seen that the edges in the matching appear in the cut, and other edges appear with probability 1/2, which means that the expected value of the cut is $\frac{w(G)+w(M)}{2}$. (The reader is asked to derandomize this experiment in Exercise 11.3).

By Vizing's Theorem, the edges of every graph can be partitioned into at most $\Delta + 1$ matchings (where Δ denotes the maximum degree in a graph). Thus, one of them has size at least $|E|/(\Delta + 1)$ yielding that every graph has a cut of size at least $(|E|/2) \cdot (1 + 1/(\Delta + 1))$.

11.5.2 MaxCut as a Semidefinite Program

We first observe that the optimum value of MaxCut can be obtained through the following mathematical program:

$$\textbf{maximize} \quad \sum_{\{i,j\} \in E} \frac{1 - y_i y_j}{2} \quad \textbf{such that} \quad y_i \in \{-1, 1\} \quad \text{for all } 1 \leqslant i \leqslant n.$$

The idea is that $V_1 = \{i \mid y_i = 1\}$ and $V_2 = \{i \mid y_i = -1\}$ constitute the two classes of the partition for the cut. The term $(1 - y_i y_j)/2$ contributes 1 to the sum iff $y_i \neq y_j$ and 0 otherwise.

We want to relax this program into such a form that we can use semidefinite programming. If we relax the solution space from one dimension to n dimensions, we obtain the following mathematical program:

$$\textbf{maximize} \quad \sum_{\{i,j\} \in E} \frac{1 - \underline{y_i}\,\underline{y_j}}{2} \quad \textbf{such that} \quad ||\underline{y_i}|| = 1, \underline{y_i} \in \mathbb{R}^n.$$

(Note that in this section only, vector variables are underlined in order to distinguish them more clearly from integer variables.)

This is not yet a semidefinite program, but by introducing new variables, we can cast it as a semidefinite program:

$$\textbf{max} \quad \sum_{\{i,j\} \in E} \frac{1 - y_{i,j}}{2} \quad \textbf{such that} \quad \begin{pmatrix} 1 & y_{1,2} & y_{1,3} & \cdots & y_{1,n} \\ y_{1,2} & 1 & y_{2,3} & \cdots & y_{2,n} \\ y_{1,3} & y_{2,3} & 1 & \cdots & y_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{1,n} & y_{2,n} & y_{3,n} & \cdots & 1 \end{pmatrix} \quad \text{is PSD.}$$

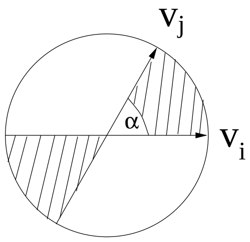


Fig. 11.2. The angle $\alpha = \arccos(\underline{v_i} \cdot \underline{v_j})$ between the vectors $\underline{v_i}$ and $\underline{v_j}$ is invariant under rotation.

The reason is the following: By the third equivalence condition in Lemma 11.6, the matrix defines a solution space in which every variable $y_{i,j}$ can be written as the product of some $\underline{v_i} \cdot \underline{v_j}$. The diagonal guarantees that $||\underline{v_i}|| = 1$ for all i.

This is a relaxation since any solution in one dimension (with the other components equal to zero) yields a feasible solution of this semidefinite program with the same objective function value.

Now, assume that we are given an (optimal) solution to this semidefinite program. We proceed as follows in order to obtain a partition :

- Use Cholesky decomposition to compute $\underline{v_i}$, $i = 1, \ldots, n$ such that $y_{i,j} = \underline{v_i} \cdot \underline{v_j}$. This can be done in time $\mathcal{O}(n^3)$.
- Choose randomly a hyperplane. (This can be done by choosing some random vector $\underline{r}$ as the normal of the hyperplane.) For this purpose, use the rotationally symmetric distribution.
- Choose V_1 to consist of all vertices whose vectors are on one side of the hyperplane (i.e., $\underline{r} \cdot \underline{v_i} \leqslant 0$) and let $V_2 := V \setminus V_1$.

The process of turning the vectors $\underline{v_i}$ into elements from $\{-1, 1\}$ by choosing a hyperplane is also called the "rounding procedure." It remains to show that the partition so obtained leads to a cutsize which is at least 87% of the optimum cutsize. For this purpose, we consider the expected value of the cutsize obtained. Because of linearity, we only need to consider for two given vectors $\underline{v_i}$ and $\underline{v_j}$ the probability that they are on different sides of the hyperplane. Since the product $\underline{v_i} \cdot \underline{v_j}$ is invariant under rotation, we can consider the plane defined by the two vectors, depicted in Figure 11.2.

First, we note that the probability that the hyperplane H chosen randomly is equal to the considered plane is equal to zero.

In the other cases, H and the plane defined by $\underline{v_i}$ and $\underline{v_j}$ intersect in a straight line. If the two vectors are to be on different sides of the hyperplane H, then this intersection line has to lie "between" $\underline{v_i}$ and $\underline{v_j}$ (i.e., fall into the shaded part of the figure) which happens with probability

$$\frac{2\alpha}{2\pi} = \frac{\alpha}{\pi} = \frac{\arccos(\underline{v_i} \cdot \underline{v_j})}{\pi}.$$

Hence, the expected value of the cut is equal to

$$\mathbf{E}[cutsize] \quad = \quad \sum_{\{i,j\} \in E} \frac{\arccos(\underline{v_i} \cdot \underline{v_j})}{\pi}. \tag{11.2}$$

We remark that one can also arrive at this expression by observing that

$$\mathbf{E}[\operatorname{sgn}(\underline{r} \cdot \underline{v_i}) \cdot \operatorname{sgn}(\underline{r} \cdot \underline{v_j})] = \frac{1}{2\pi} \cdot \int_{\varphi=0}^{2\pi} \operatorname{sgn}(\cos\varphi) \cdot \operatorname{sgn}(\cos(\varphi - \alpha))\, d\varphi = 1 - 2 \cdot \frac{\alpha}{\pi}.$$

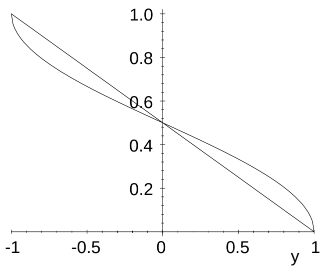
Whenever we solve a concrete semidefinite relaxation of the MaxCut problem, we obtain an upper bound on the MaxCut solution. The quality of the outcome of the rounding procedure can then directly be measured by comparing it with this upper bound.

Nevertheless, one is of course interested in how good one can guarantee the expected solution to be in the worst case. The next two subsections will analyze this worst case behavior. One should keep in mind, though, that experiments indicate that in "most" cases, the quality of the solutions produced is higher than in the worst case.

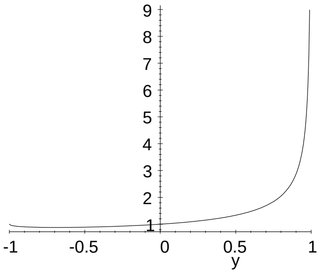
Analyzing the Quality Coarsely. We compare the expected value (11.2) with the value of the relaxed program. One way to do so is to compare every single term $\arccos(\underline{v_i} \cdot \underline{v_j})/\pi$ with $(1 - \underline{v_i} \cdot \underline{v_j})/2$.
Solving a simple calculus exercise, one can show that in the range $-1 \leqslant y \leqslant 1$,

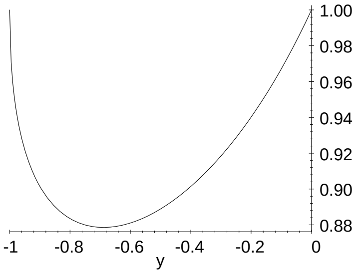
$$\frac{\arccos(y)}{\pi} \geqslant 0.87856 \cdot \frac{1-y}{2}.$$

(Let us denote by $\alpha_{GW} = 0.87856\ldots$ the maximum constant we could have used.)
A sketch of the involved functions may also be quite helpful. The following figure shows the function $\arccos(y)/\pi$ as well as the function $(1-y)/2$.



The following two figures show the quotient $\frac{\arccos(y)}{\pi} / \frac{1-y}{2}$ in the range $-1 \leqslant y \leqslant 1$ and $-1 \leqslant y \leqslant 0$, respectively. (At first sight, it is rather surprising that the quotient is larger than 1 for $y > 0$, since our randomized procedure then yields terms which are larger than the terms in the relaxed program.)





Let $Relax_{opt}$ denote the optimal value of the semidefinite program. The conclusion of the analysis above is that the expected value of the cutsize produced by the rounding procedure is at least $0.878 \cdot Relax_{opt}$, hence the quality of the solution is also not larger than $1/0.878 \approx 1.139$.

We also see that the quality of the solution obtained could be improved if we could achieve that $\underline{v_i} \cdot \underline{v_j}$ is "far away" from approximately -0.689.

Thus, one might hope that extra constraints could improve the quality of the solution. In this direction, Karloff [Kar96] has shown that the quality of the rounding procedure cannot be improved by adding *linear* inequality constraints to the semidefinite program. For MaxCut, the ratio can be improved for some families of graphs, but not in the general worst case. However, additional linear inequality constraints can improve the approximation ratio in the case of MaxDiCut and MaxE2Sat, see Section 11.8.

Analyzing the Quality More Precisely. In the above subsection, we have analyzed the quality of our solution term by term. We can improve our estimations by taking into account the value of the solution our semidefinite program has produced.

For this purpose assume that the optimum is assumed for some values of $y_{i,j}$, and let $Y_{sum} := \sum_{\{i,j\} \in E} y_{i,j}$. Then,

$$Relax_{opt} := \sum_{\{i,j\} \in E} \frac{1 - y_{i,j}}{2} \quad = \quad \frac{|E|}{2} - \frac{1}{2} \cdot Y_{sum}$$

depends only on Y_{sum} and not on each individual value $y_{i,j}$.

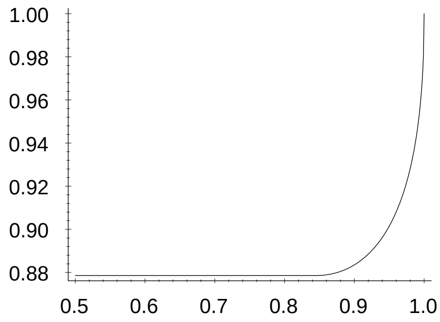
If $Relax_{opt}$ is known, then $Y_{sum} = |E| - 2 \cdot Relax_{opt} \leqslant 0$ is known. We can ask for which individual values of $y_{i,j}$ summing to Y_{sum}, $S := \sum_{i,j} \arccos(y_{i,j})$ attains a minimum. Observe that $\arccos(y)$ is convex on $[-1, 0]$ and concave on $[0, 1]$ (the derivative of $\arccos y$ is $-1/\sqrt{1-y^2}$).

In Exercise 11.4, we ask the reader to verify the following: If Y_{sum} is fixed, then the $y_{i,j}$ for which $\sum_{i,j} \arccos(y_{i,j})$ attains a minimum, fulfill the following property:
There is a value $Y \leqslant 0$ and an r with $|E|/2 \leqslant r \leqslant |E|$ such that r of the $y_{i,j}$ are equal to Y and $|E| - r$ of the $y_{i,j}$ are equal to one.

Then, $\sum_{\{i,j\}\in E} \arccos(y_{i,j}) = r \cdot \arccos(\frac{Y_{sum}-|E|+r}{r})$ and the quotient $\frac{\sum_{\{i,j\}} \arccos(y_{i,j})}{Relax_{opt}}$ can be estimated. We obtain that

$$\begin{aligned}
\frac{\mathbf{E}[\text{cutsize}]}{Relax_{opt}} &\geqslant \min_{|E|/2\leqslant r\leqslant |E|} \frac{r \cdot \arccos(\frac{Y_{sum}-|E|+r}{r})}{\pi \cdot Relax_{opt}} \\
&= \min_r \frac{\arccos(\frac{r-2\cdot Relax_{opt}}{r})}{\pi \cdot Relax_{opt}/r} \\
&= \frac{\arccos(1-2x)}{\pi \cdot x} =: q(x),
\end{aligned}$$

where we have substituted $x := Relax_{opt}/r$ for the r where the minimum is attained. We have obtained the following: Given a graph G with optimal cutsize MC, and "relative" cutsize $y := MC/|E|$, it holds that $y \leqslant x$ and thus the "rounding quality" $\mathbf{E}[\text{cutsize}]/Relax_{opt}$ obtained for graph G is at least $\min_{x\geqslant y} q(x)$. Below, we give a plot of $Q(y) := \min_{x\geqslant y} q(x)$.



As we can see, the quality of the rounding procedure becomes better when the relative maximum cutsize of the input graph becomes larger.

The above analysis only considers the quality of the rounding procedure. Karloff has shown in [Kar96] that there is a family of graphs for which the quality of the rounding procedure is arbitrarily close to α_{GW}. Nevertheless, it might be possible to obtain a better approximation ratio than $1/\alpha_{GW} \approx 1.139$ by using a different rounding procedure, since for the family of graphs constructed by Karloff, the optimum cutsize is equal to $Relax_{opt}$.

Thus, it is also a natural question to ask how much larger $Relax_{opt}$ can be compared to the optimum MAXCUT solution MAX_{opt}.

Karloff [Kar96] calls this the "integrality ratio", and remarks that from his paper, nothing can be concluded about this "integrality ratio."

On the other hand, Goemans and Williamson mention that for the input graph "5-cycle", $MAX_{opt}/Relax_{opt} = 0.88445\ldots$ holds which gives a graph where the relaxation is relatively far away from the original best solution.

It is clear that for these graphs, even a better rounding procedure would not lead to better results.

The above considerations suggest that it would be nice if we could keep a substantial subset of all products $v_i \cdot v_j$ away from ≈ -0.689, where the worst case approximation $0.878\ldots$ is attained. One might hope that adding extra constraints might lead to better approximation ratios.

Implementation Remarks. For practical purposes, it is probably not a good idea to derandomize the probabilistic algorithm given by Goemans and Williamson, since it seems very likely that after only a few rounds of choosing random hyperplanes, one should find a good enough approximation, and the quality of the approximation can also be controlled by comparing with the upper bound of the semidefinite program.

Nevertheless, it is an interesting theoretical problem to show that semidefinite programming also yields a deterministic approximation algorithm.

In the original proceedings paper by Goemans and Williamson, a derandomization procedure was suggested which later turned out to have a flaw. A new suggestion was made by Mahajan and Ramesh [MR95a], but their arguments are rather involved and technical which is why we omit them in this survey. One can only hope that a simpler deterministic procedure will be found.

Just one little remark remains as far as the implementation of the randomized algorithm is concerned. How do we draw the vector r, i.e., how do we obtain the rotationally symmetric distribution? For this purpose, one can draw n values r_1 to r_n independently, using the standard normal distribution. Unit length of the vector can be achieved by a normalization.

For the purposes of the MAXCUT-algorithm, a normalization is not necessary since we are only interested in the sign of $r \cdot v_i$. The standard normal distribution can be simulated using the uniform distribution between 0 and 1, for details see [Knu81, pp. 117 and 130].

11.6 Modeling Asymmetric Problems

In the following, we describe approximation algorithms for MaxDiCut and Max2Sat/MaxE2Sat. The algorithms are based on semidefinite programming. All results in this section (as well as many in the following section) are due to Goemans and Williamson [GW95].

The general approach which yields good randomized approximation algorithms for MaxDiCut and MaxE2Sat is very similar to the approach used to approximate MaxCut with ratio 1.139, cf. Section 11.5.2. We summarize the three main steps of this approach:

1. **Modeling.** First, model the problem as an integer quadratic problem over some set $Y = \{y_1, \ldots, y_n\}$ of variables. The objective function is linear in $\{y_i y_j : y_i, y_j \in Y\}$. The only restrictions on the variables are $y_i \in \{-1, 1\}$, $y_i \in Y$.
2. **Relaxation.** Consider the semidefinite relaxation of the integer quadratic problem. The relaxed problem has $|Y|^2$ variables $y_{i,j}$. The objective function is linear in these variables. In the relaxation, the restriction is that the matrix $(y_{i,j})$ is PSD and that all entries on the main diagonal of this matrix are one.

   An optimal solution of the semidefinite program can be computed in polynomial time with any desired precision ε.
3. **Rounding.** By a Cholesky decomposition, the solution of the relaxation can be expressed by n vectors $v_i \in \mathbb{R}^n$. Use these vectors in a probabilistic experiment, i.e., choose a hyperplane at random and assign $+1$ and -1 to the variables y_i. The result is a (suboptimal) solution of the integer quadratic problem.

In order to approximate MaxDiCut and MaxE2Sat, step 2 will be exactly the same as for MaxCut. Step 3, the rounding, will be almost the same. However, the modeling in step 1 is different.

The difference in the modeling stems from an asymmetry which is inherent in the problems MaxDiCut and MaxE2Sat. MaxCut is a symmetric problem, since switching the role of the "left" and the "right" set of the partition does not change the size of the cut. Thus, the objective function of the integer quadratic program of MaxCut models only whether the vertices of an edge are in different sets:

$$\begin{array}{ll} \textbf{maximize} & \displaystyle\sum_{\{i,j\}\in E} \frac{1 - y_i \cdot y_j}{2} \\ \textbf{such that} & y_i \in \{-1, 1\} \quad \forall i \in \{1, \ldots, n\} \end{array}$$

Of course, for any set S of vertices, the value of the objective function for the cuts $(S, \overline{S})$ and $(\overline{S}, S)$ is the same. In contrast, for a directed cut the direction of an edge does make a difference. The objective function must distinguish the directed cuts $(S, \overline{S})$ and $(\overline{S}, S)$.

We describe an approach to model these problems by a semidefinite program.

11.6.1 Approximating MaxDiCut

As before, we want the values of the variables y_i of the integer quadratic program to be restricted to 1 or -1. The reason is that this simplifies the relaxation. The objective function must be linear in the product $y_i \cdot y_j$ of any pair (y_i, y_j) of variables. However, we cannot distinguish edge (i, j) from edge (j, i) by just looking at the product $y_i \cdot y_j$.

The trick Goemans and Williamson use to model the direction of an edge is to introduce a new variable y_0. Like other variables, y_0 may be -1 or 1. Now, the graph-theoretical property

$$i \in S$$

is expressed in the integer quadratic program by the property

$$y_i \cdot y_0 = 1.$$

In this way, the term

$$(1 + y_i y_0) \cdot (1 - y_j y_0) \;/\; 4 \quad = \quad (1 + y_i y_0 - y_j y_0 - y_i y_j) \;/\; 4 \tag{11.3}$$

is equal to 1 if $y_i = y_0 = -y_j$. Otherwise, it is 0. Thus, (11.3) represents the contribution of edge (i, j) to the objective function. The value of the whole directed cut, i.e., the objective function for MaxDiCut, is the sum of term (11.3) over all edges in the graph. In this way, we get

$$\begin{array}{ll} \textbf{maximize} & \displaystyle\sum_{(i,j)\in E} \frac{1 + y_i y_0 - y_j y_0 - y_i y_j}{4} \\ \textbf{such that} & y_i \in \{-1, 1\} \;\; \forall i \in \{0, \ldots, n\} \end{array} \tag{11.4}$$

as an integer quadratic program modeling the problem MaxDiCut.

Remark 11.13. An equivalent view of (11.4) is that an isolated vertex $0 \notin V$ has been added to the graph. For any cut, the objective function counts exactly those edges (i, j) where the tail i is in the same set as 0 and where the head j is in the other set.

As in the MaxCut approximation, step 2 of the MaxDiCut approximation consists of relaxing to

$$\begin{array}{ll} \textbf{maximize} & \displaystyle\sum_{(i,j)\in E} \frac{1 + y_i y_0 - y_j y_0 - y_i y_j}{4} \\ \textbf{such that} & y_i \in \mathbb{R}^{n+1}, ||y_i|| = 1 \ \ \forall i \in \{0, \ldots, n\}. \end{array} \tag{11.5}$$

Note that in this case the relaxation is to the $(n+1)$-dimensional vector space.

Letting $y_{i,j} = y_i \cdot y_j$, we obtain the semidefinite program

$$\begin{array}{ll} \textbf{maximize} & \displaystyle\sum_{(i,j)\in E} \frac{1 + y_{i,0} - y_{j,0} - y_{i,j}}{4} \\ \textbf{such that} & \text{the matrix } (y_{i,j}) \text{ is PSD,} \\ & y_{i,i} = 1 \ \ \forall i \in \{0, \ldots, n\}. \end{array} \tag{11.6}$$

By solving (11.6) and computing the Cholesky decomposition, we get vectors $v_0, v_1, \ldots, v_n$ such that $(y_{i,j}) = (v_0, v_1, \ldots, v_n)^T (v_0, v_1, \ldots, v_n)$ maximizes the value of the objective function (within the desired precision).

The next step consists of rounding the vectors $v_0, v_1, \ldots, v_n$. Because of the special role of v_0, this step slightly differs from the rounding step for MaxCut. In a certain sense, the inner product $v_i \cdot v_0$ measures the probability of vertex i being in cut S of the output. If $v_i \cdot v_0 = 1$, then $i \in S$ with probability 1, if $v_i \cdot v_0 = -1$, then $i \in S$ with probability 0. More formally, with uniform distribution, choose a unit vector $r \in \mathbb{R}^{n+1}$ and let H_r be the hyperplane through the origin which has normal r. Let

$$S := \{i : \operatorname{sgn}(v_i \cdot r) = \operatorname{sgn}(v_0 \cdot r)\}$$

be the set of those vertices i where v_i and v_0 are on the same side of H_r. The randomized procedure outputs this set S.

The probability of edge (i,j) occurring in the directed cut is exactly

$$\text{Prob}_{r \in \mathbb{R}^{n+1}, ||r||=1}[\operatorname{sgn}(v_i \cdot r) = \operatorname{sgn}(v_0 \cdot r) \neq \operatorname{sgn}(v_j \cdot r)].$$

By arguments from spherical geometry (see [GW95] for details), for any unit vectors $v_i, v_j, v_0 \in \mathbb{R}^{n+1}$, this probability is at least

$$0.796 \cdot \frac{1 + v_i v_0 - v_j v_0 - v_i v_j}{4}.$$

This yields

Theorem 11.14 ([GW95]). MaxDiCut *can be approximated by a randomized polynomial-time algorithm with approximation ratio 1.257, i.e., the expected value of the computed solution is at least 0.796 times the optimal solution.*

11.6.2 Approximating MAX2SAT

We turn to the approximation of MAX2SAT. Of course, the results in this section also hold for MAXE2SAT.

As MAXDICUT, MAX2SAT is an asymmetric problem: In general, switching the truth values in an assignment for the variables in a Boolean formula will change the number of clauses which are satisfied.

As before, we will use an additional variable y_0 when modeling MAX2SAT. ("The value of y_0 will determine whether -1 or 1 will correspond to 'true' in the MAX2SAT instance" [GW95].) Let $v(C) \in \{0,1\}$ denote the contribution of clause C to the objective function. Thus, the contribution of a singleton clause x_i (or $\overline{x}_i$) is

$$v(x_i) = \frac{1 + y_i \cdot y_0}{2} \tag{11.7}$$

(or

$$v(\overline{x}_i) = \frac{1 - y_i \cdot y_0}{2}, \tag{11.8}$$

respectively). For clauses of length two, say $x_i \vee \overline{x}_j$, we have

$$\begin{aligned} v(x_i \vee \overline{x}_j) &= 1 - v(\overline{x}_i) \cdot v(x_j) \\ &= 1 - \frac{1 - y_i \cdot y_0}{2} \cdot \frac{1 + y_j \cdot y_0}{2} \\ &= \frac{1 + y_i \cdot y_0}{4} + \frac{1 - y_j \cdot y_0}{4} + \frac{1 + y_i \cdot y_j}{4}. \end{aligned} \tag{11.9}$$

In this way, the value of all clauses of length up to two can be expressed as nonnegative linear combinations of $1 + y_i \cdot y_j$ and $1 - y_i \cdot y_j$. Thus, we can compute nonnegative $a_{i,j}$, $b_{i,j}$ such that

$$\sum_{i=1}^{m} v(C_i) = \sum_{i,j} a_{i,j}(1 + y_i \cdot y_j) + b_{i,j}(1 - y_i \cdot y_j).$$

The MAX2SAT instance is thus modelled by the integer quadratic program

$$\begin{aligned} \textbf{maximize} \quad & \sum_{i,j} a_{i,j}(1 + y_i \cdot y_j) + b_{i,j}(1 - y_i \cdot y_j) \\ \textbf{such that} \quad & y_i \in \{-1,1\} \quad \forall i \in \{0,\ldots,n\} \end{aligned} \tag{11.10}$$

and the semidefinite formulation of the relaxation is

$$\begin{aligned} \textbf{maximize} \quad & \sum_{i,j} a_{i,j}(1 + y_{i,j}) + b_{i,j}(1 - y_{i,j}) \\ \textbf{such that} \quad & \text{the matrix } (y_{i,j}) \text{ is PSD,} \\ & y_{i,i} = 1 \quad \forall i \in \{0,\ldots,n\}. \end{aligned} \tag{11.11}$$

In the rounding procedure, those variables x_i are set to true where the vectors v_i and v_0 of the Cholesky decomposition are on the same side of the random hyperplane.

By an argument that is similar to the one used in the analysis of MAXCUT, the expected value of the computed solution is at least 0.878 times the value of an optimal solution. I.e., we have

Theorem 11.15 ([GW95]). MAX2SAT *(and, thus,* MAXE2SAT*) can be approximated by a randomized polynomial-time algorithm with approximation ratio 1.139.*

11.7 Combining Semidefinite Programming with Classical Approximation Algorithms

Prior to the work in [GW95], at least three different approaches to approximate MAXEkSAT/MAXSAT have been proposed: Johnson's algorithm, an algorithm based on network flow, and an algorithm based on linear programming. To distinguish these approaches from the semidefinite programming technique, we call them *classical* algorithms.

Johnson's algorithm. Johnson's algorithm [Joh74] is the derandomization of the probabilistic algorithm where the truth assignment is uniformly chosen among all possible assignments. This algorithm is described in Chapter 2 and derandomized in Chapter 3. MAXEkSAT is solved with approximation ratio $1/(1-2^{-k})$. The approximation ratio for MAXSAT is at least 2. (Poljak and Turzík describe another MAXSAT algorithm that achieves the same approximation ratio.)

Network flow. Yannakakis [Yan92] describes an approximation algorithm for MAXSAT which is based on computing maximal flows in networks. Yannakakis' algorithm is rather involved. Asano et al. [AHOH96] give an outline of this algorithm. The main idea is to construct several networks. The maximum flow in these networks is used to partition the set of variables in the Boolean formula into three classes. The variables in the first (second, third) class are independently set to be true with probability 0.5 (0.555, 0.75, respectively). The expected value of the approximation ratio of this algorithm is shown to be 1.334. The algorithm can be derandomized.

Linear programming. Chapter 2 contains a description of a MAXSAT-algorithm due to Goemans and Williamson [GW94] which solves the linear relaxation of an integer program modeling MAXSAT. The solution of the linear program determines for each variable the probability with which this variable is set to true. This algorithm solves MAXkSAT with approximation ratio $1/(1-(1-1/k)^k)$ and MAXSAT with approximation ratio $e/(e-1) = 1.582$. It can be derandomized.

k	Johnson's algorithm	Network flow	Linear programming
1	0.5	0.75	1.0
2	0.75	0.75	0.75
3	0.875	0.75	0.703
4	0.937	0.765	0.683
5	0.968	0.762	0.672
6	0.984	0.822	0.665
7	0.992	0.866	0.660

Table 11.1. For three basic MaxSat algorithms, a lower bound on the probability that the computed solution satisfies a fixed clause of length k is given, $k = 1, \ldots, 7$.

For all of these algorithms, the approximation ratio depends on the concrete MaxSat instance. If the clauses in the formula are rather long, then Johnson's algorithm is very good. If almost all of the clauses are very short, then the algorithm based on linear programming might be better. However, the algorithm based on network flows is the only one of these algorithms where the analysis guarantees approximation ratio 1.334. Thus, among these algorithms there is no "best" one. Table 11.1 summarizes (lower bounds on) the approximation ratio of these algorithms on instances of MaxEkSat for different k, $1 \leqslant k \leqslant 7$.

In Chapter 2, we have seen that the merits of different MaxSat algorithms can be combined. The combination of two different algorithms may yield a better approximation of MaxSat. Namely, another 1.334-approximation is obtained by combining Johnson's algorithm and the algorithm based on linear programming. On any input, the combined algorithm calls both of these algorithms. This yields two truth assignments. The combined algorithm outputs that truth assignment which satisfies the larger number of clauses.

In the rest of this section we use a variant of this technique where the algorithms are combined in a probabilistic way. In our examples, three different algorithms will be combined.

First, we describe MaxSat approximations that are based on semidefinite programming (Section 11.7.1). Then, these algorithms are combined with a classical algorithm (Section 11.7.2).

11.7.1 Handling Long Clauses

The algorithm described in Section 11.6.2 handles only clauses of length at most two. In the following, we describe how Goemans and Williamson treat longer clauses.

Let $\Phi = C_1 \wedge \ldots \wedge C_m$ be a Boolean formula in conjunctive normal form. For any clause C_j let $\ell(C_j)$ denote the length of the clause and let I_j^+ (or I_j^-) denote the set of non-negated (negated, respectively) variables in C_j.

Recall Goemans and Williamson's formulation of Φ as a *linear program* ((2.1) in Chapter 2):

$$\begin{array}{lrcll} \textbf{maximize} & \multicolumn{3}{c}{\sum_{j=1}^m z_j} & \\ \textbf{such that} & \sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) & \geqslant & z_j & \forall j \in \{1, \ldots, m\} \\ & 0 \leqslant y_i & \leqslant & 1 & \forall i \in \{1, \ldots, n\} \\ & 0 \leqslant z_j & \leqslant & 1 & \forall j \in \{1, \ldots, m\} \end{array} \tag{11.12}$$

In a very similar way, by using the function v introduced in (11.7)–(11.9), MaxSat is modelled by

$$\begin{array}{lrcll} \textbf{maximize} & \multicolumn{3}{c}{\sum_{j=1}^m z_j} & \\ \textbf{such that} & \sum_{i \in I_j^+} v(x_i) + \sum_{i \in I_j^-} v(\overline{x}_i) & \geqslant & z_j & \forall j \in \{1, \ldots, m\} \\ & v(C_j) & \geqslant & z_j & \forall j \in \{1, \ldots, m\}, \ell(C_j) = 2 \\ & y_i \cdot y_i & = & 1 & \forall i \in \{0, \ldots, n\} \\ & 0 \leqslant z_j & \leqslant & 1 & \forall j \in \{1, \ldots, m\}. \end{array} \tag{11.13}$$

To obtain the corresponding semidefinite formulation, replace each vector product $y_i \cdot y_k$ by a new variable $y_{i,k}$. As restrictions, we get that the matrix $A_1 := (y_{i,k})_{i,k \in \{0,\ldots,n\}}$ is PSD and that a system of linear inequalities in the variables $y_{i,k}$, $i, k \in \{0, \ldots, n\}$, and z_j, $j \in \{1, \ldots, m\}$ is satisfied. As noted in Section 11.3, the linear inequalities can be transformed into a matrix $A_2 := SP(y_{0,0}, \ldots, y_{n,n}, z_1, \ldots, z_m)$ which is PSD if and only if all inequalities are satisfied. Let A be the matrix

$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

which is PSD if and only if A_1 and A_2 are PSD. Then

$$\begin{array}{ll} \textbf{maximize} & \displaystyle\sum_{j=1}^m z_j \\ \textbf{such that} & \text{matrix } A \text{ is PSD} \\ & y_{i,i} = 1 \quad \forall i \in \{0, \ldots, n\} \end{array} \tag{11.14}$$

is the corresponding semidefinite relaxation.

Let $\hat{y}_{i,k}$ and $\hat{z}_j$ denote the value of the variables in a solution of (11.14).

By **A1**, we denote the algorithm which computes the Cholesky decomposition of the matrix $(\hat{y}_{i,k})$ and applies Goemans-Williamson's rounding procedure to the solution (as described in Section 11.6.2). For each clause C_j of length one or two,

$$\sum_{i\in I_j^+} v(x_i) + \sum_{i\in I_j^-} v(\overline{x}_i) \quad\geqslant\quad v(C_j),$$

i.e., $v(C_j) \geqslant z_j$ implies $\sum_{i\in I_j^+} v(x_i) + \sum_{i\in I_j^-} v(\overline{x}_i) \geqslant z_j$. Thus, algorithm **A1** satisfies instances of MAX2SAT with probability at least 0.878.

By **A2** we denote the following algorithm: Independently for each variable, set x_i to be true with probability $(1+\hat{y}_{i,0})/2$. Then, by the same argument as in the analysis of (11.12) (cf. Chapter 2), the probability that clause C_j is satisfied is at least

$$\left(1-\left(1-\frac{1}{\ell(C_j)}\right)^{\ell(C_j)}\right)\ \hat{z}_j.$$

11.7.2 Approximating MAXSAT

Let (p_1,p_2,p_3) be a probability vector, i.e., $0 \leqslant p_j \leqslant 1$ and $p_1+p_2+p_3=1$. Consider the following combination of three algorithms:

1. With probability p_1, execute algorithm **A1**.
2. With probability p_2, execute algorithm **A2**.
3. With probability p_3, execute Johnson's algorithm.

The expected value of the solution of these algorithms is

$$\sum_{j:\ell(C_j)\leqslant 2} 0.878\ \hat{z}_j,\quad \sum_{j=1}^{k}\left(1-\left(1-\frac{1}{\ell(C_j)}\right)^{\ell(C_j)}\right)\ \hat{z}_j,\quad \text{or} \sum_{j=1}^{k}\left(1-\frac{1}{2^{\ell(C_j)}}\right),$$

respectively. Thus, the solution of the combined algorithm has expected value

$$p_1 \sum_{j:\ell(C_j)\leqslant 2} 0.878\ \hat{z}_j + p_2 \sum_{j=1}^{k}\left(1-\left(1-\frac{1}{\ell(C_j)}\right)^{\ell(C_j)}\right)\ \hat{z}_j + p_3 \sum_{j=1}^{k}\left(1-\frac{1}{2^{\ell(C_j)}}\right).$$

If we choose $p_1 = 0.0430$ and $p_2 = p_3 = 0.4785$, then by a numerical computation it can be checked that this term is at least 0.755 times the optimum of (11.14). I.e., we have

Theorem 11.16 ([GW95]). MAXSAT *can be approximated by a randomized polynomial-time algorithm with approximation ratio 1.324.*

Note that in a practical implementation we would use a deterministic combination of **A1**, **A2**, and Johnson's algorithm. The combined algorithm outputs that truth assignment which satisfies the larger number of clauses. The approximation ratio of this algorithm is also at least 1.324.

Using a similar approach, where long clauses are handled in a slightly different way, Goemans and Williamson also achieve an approximation ratio of 1.319 for MaxSat.

Asano et al. have proposed another method for handling long clauses by semidefinite programs [AOH96]. They obtain an algorithm with approximation ratio 1.307. Tuning the latter method, Asano et al. [AHOH96] have obtained an 1.304-approximation algorithm for MaxSat. However, the best known approximation for MaxSat is an algorithm by Asano [Asa97] which is based on a refinement of Yannakakis' algorithm. This algorithm has approximation ratio 1.299.

11.8 Improving the Approximation Ratio

In this section we describe two techniques due to Feige and Goemans [FG95] and to Karloff [Kar96] which improve the approximation ratio of the semidefinite programs for MaxE2Sat and MaxDiCut. The idea is to add linear restrictions to the semidefinite program and to modify the rounding procedure. This improves the approximation ratio for MaxE2Sat to 1.075 and the approximation ratio for MaxDiCut to 1.165.

11.8.1 Adding Constraints

Note that we can add any (polynomial) number of linear restrictions to a semidefinite program whilst keeping the running time of the approximation algorithm polynomial. The constraints considered here are valid for all truth assignments and all cuts, respectively. (In particular, the constraints do not depend on the instance of the problem.) However, they do not hold for all the vectors in the relaxation. Thus, they may change the solution of the relaxation. This may improve the approximation ratio of the algorithm.

Feige and Goemans [FG95] discuss the use of two sets of linear inequality constraints which sharpen the restrictions of MaxE2Sat and MaxDiCut. Their first set contains for all $i, j, k \in \{0, \ldots, n\}$ the inequalities

$$\begin{aligned} y_i \cdot y_j + y_j \cdot y_k + y_k \cdot y_i &\geqslant -1 \\ -y_i \cdot y_j - y_j \cdot y_k + y_k \cdot y_i &\geqslant -1 \\ -y_i \cdot y_j + y_j \cdot y_k - y_k \cdot y_i &\geqslant -1 \\ y_i \cdot y_j - y_j \cdot y_k - y_k \cdot y_i &\geqslant -1, \end{aligned}$$

These inequalities are valid, since they hold for any assignment of y_i, y_j, y_k with -1 and 1. For MAXE2SAT it expresses the *tertium non datur*. For MAXDICUT and MAXCUT it expresses the fact that any cut partitions the set of vertices into two sets.

The second set of constraints is a subset of the first one. It contains for all $i, j \in \{0, \ldots, n\}$ the inequalities

$$\begin{aligned} y_i \cdot y_0 + y_j \cdot y_0 + y_i \cdot y_j &\geqslant -1 \\ -y_i \cdot y_0 - y_j \cdot y_0 + y_i \cdot y_j &\geqslant -1 \\ -y_i \cdot y_0 + y_j \cdot y_0 - y_i \cdot y_j &\geqslant -1 \\ y_i \cdot y_0 - y_j \cdot y_0 - y_i \cdot y_j &\geqslant -1. \end{aligned}$$

Consider Goemans-Williamson's formulation of MAXE2SAT described in Section 11.6.2. For an instance with one clause, say $x_1 \vee x_2$, the objective function is $(3 + y_1 \cdot y_0 + y_2 \cdot y_0 - y_1 \cdot y_2)/4$. In the relaxation of the original formulation, there are vectors v_0, v_1, v_2 where $v_1 \cdot v_0 = v_2 \cdot v_0 = -v_1 \cdot v_2 = 0.5$, i.e., where the objective function is $9/8$. Thus, there is an instance where the approximation ratio is at most 0.888. If the second set of constraints is added to the program, then the value of the objective function is at most 1 for any feasible solution of this instance.

11.8.2 Nonuniform Rounding

Feige and Goemans [FG95] introduced the concept of *nonuniform rounding*, a technique which improves Goemans-Williamson's MAXDICUT and MAXE2SAT approximation.

The idea is to modify step 3, the rounding, in a way that takes advantage of the special role of vector v_0.

First, Feige and Goemans consider those outputs of the semidefinite program which give rise to the worst case of the approximation ratio. Consider a clause in a MAXE2SAT instance, say $\overline{x}_i \vee \overline{x}_j$. Its contribution to the objective function is $(3 - y_i \cdot y_0 - y_j \cdot y_0 - y_i \cdot y_j)/4$. Let v_0, v_i, v_j be the vectors computed by the approximation algorithm. In Section 11.6.2 we have seen that this clause is satisfied with probability at least 0.878 times its contribution in the objective function, if the "uniform rounding" procedure of Goemans and Williamson is used. It can be shown that the worst case ≈ 0.878 is attained exactly if two of the inner products $v_i \cdot v_0$, $v_j \cdot v_0$, $v_i \cdot v_j$ are approximately -0.689 and the other product is 1, cf. also the analysis of the MAXCUT algorithm in Section 11.5.2. Thus, in the worst case, either $v_i \cdot v_0 \leqslant 0$ or $v_j \cdot v_0 \leqslant 0$. (Note that the triple of worst case vectors is not excluded by the additional constraints given in Section 11.8.1.)

Consider the algorithm which assigns x_i to true if $v_i \cdot v_0 \geqslant 0$ and to false otherwise. (Call this *crude rounding.*) If the triple of vectors v_0, v_i, v_j is a worst

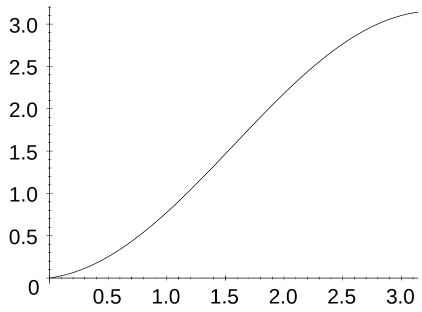


Fig. 11.3. Plot of the function f used for nonuniform rounding in the approximation of MaxE2Sat.

case for the uniform rounding step, then crude rounding ensures that the clause $\overline{x}_i \vee \overline{x}_j$ is always satisfied. Thus, for the worst case vectors of the uniform rounding procedure, crude rounding is better. Of course, it is not a good idea to use crude rounding, since it is much worse in many other cases.

We want to skew the uniform rounding procedure a little bit. We would like to use a rounding procedure which combines uniform rounding with a pinch of crude rounding. That is, if $v_i \cdot v_0 > 0$, then the probability of x_i being true should be increased. If $v_i \cdot v_0 < 0$, then the probability of x_i being true should be decreased.

Technically, this effect is obtained in the following way. Let ϑ be the angle between v_0 and v_i. Fix some function $f : [0, \pi] \to [0, \pi]$. Then, map vector v_i to that vector v_i' which is on the same hyperplane as the vectors v_i and v_0 and where the angle between v_0 and v_i' is $f(\vartheta)$. Finally, perform the uniform rounding procedure using the vectors v_0 and $v_1', \ldots, v_n'$.

The choice of $f(\vartheta) = \vartheta$ corresponds to uniform rounding. We require $f(\pi - \vartheta) = \pi - f(\vartheta)$ in order to ensure that negated and unnegated literals are handled in the same way.

Among other functions, Feige and Goemans study the effect of

$$f(\vartheta) \;=\; \vartheta + 0.806765\left[\frac{\pi}{2}(1-\cos\vartheta) - \vartheta\right] \tag{11.15}$$

on the approximation ratio in nonuniform rounding, cf. Figure 11.3. Using this function, they obtain the following approximation ratio.

Theorem 11.17 ([FG95]). MaxE2Sat *can be approximated by a randomized polynomial-time algorithm with approximation ratio 1.075, i.e., the expected value of the computed solution is at least 0.931 times the optimal solution.*

Sketch of Proof. By numerical computation, analyze the approximation ratio of the following modification of the algorithm described in Section 11.6.2:

- Add the constraints listed in Section 11.8.1.
- Use function (11.15) for nonuniform rounding.

In the analysis, first discretize the space of all vectors v_0, v_i, v_j that satisfy the additional constraints. In fact, it is sufficient to discretize the space of all possible angles between v_0, v_i and v_j. Then, numerically compute the ratio between the probability that a clause is satisfied and the contribution of that clause to the objective function. Check that this ratio is at least 0.931 for all these vectors. ∎

The importance of the additional constraints is stressed by the fact that, for nonuniform rounding using this function f, the worst case of the performance ratio is attained for a triple of vectors where $v_i \cdot v_0 + v_j \cdot v_0 + v_i \cdot v_j = -1$.

Using the same method with a different function f, Feige and Goemans obtain

Theorem 11.18 ([FG95]). MaxDiCut *can be approximated by a randomized polynomial-time algorithm with approximation ratio 0.859.*

11.9 Modeling MaximumIndependentSet as a Semidefinite Program?

Just for didactical purposes, we consider how one might turn MaximumIndependentSet into a semidefinite program. We leave to the interested reader the investigation whether this can be used to obtain better approximation algorithms for MaximumIndependentSet than are currently known.

MaximumIndependentSet

Instance: Given an undirected graph $G = (V, E)$.
Problem: Find a subset $V_1 \subseteq V$ such that V_1 does not contain any edge and $|V_1|$ is maximized.

We first claim that the size of a maximum independent set in a graph (and the corresponding independent set) can be obtained by finding a solution to the following mathematical program:

$$\textbf{max} \;\; x_1 + \cdots + x_n - \sum_{\{i,j\} \in E} x_i x_j \qquad \textbf{such that } \forall i: \; 0 \leqslant x_i \leqslant 1 \tag{11.16}$$

Then, the following mathematical program also yields the optimum:

$$\textbf{max} \;\; x_1^2 + \cdots + x_n^2 - \sum_{\{i,j\}\in E} x_i x_j \qquad \textbf{such that } \forall i: \; 0 \leqslant x_i \leqslant 1 \qquad (11.17)$$

This follows from the observation that the objective function considered in one variable only (others fixed) is a parabola which is open into the positive y-direction. Hence, we have a guarantee that for either $x_i = 0$ or $x_i = 1$, the optimum is obtained.

Now, let us show that (11.16) represents MAXIMUMINDEPENDENTSET. Given an independent set I and setting $x_i = 1$ for $i \in I$ (and $x_i = 0$ otherwise), we see that the optimal solution of (11.16) is at least the size of a maximum independent set.

Assume that we are given a solution of (11.16). Since the objective function is linear in every variable, we can assume that all x_i are either 0 or 1. We construct an independent set I with the same value as the objective function as follows: If the set $I := \{i \mid x_i = 1\}$ is not an independent set, we proceed as follows: If $x_i = x_j = 1$ and $\{i, j\}$ is an edge in the graph, then we set $x_i = 0$ which means that we have not decreased the objective function. (One vertex was eliminated, but at least one edge was also removed.)

Hence, the optimal solution of (11.16) or (11.17) yields the optimum cardinality of an independent set. Since by Theorem 4.12 MAXIMUMINDEPENDENTSET cannot be approximated within a factor n^ε (unless $\mathcal{P} = \mathcal{NP}$), it is unlikely that we can solve this mathematical program with the help of semidefinite programming, although it looks very much like the problem formulation for MAXCUT.

Consider for example the following semidefinite program P:

$$\textbf{maximize} \;\; y_{1,1} + \cdots + y_{n,n} - \sum_{\{i,j\}\in E} y_{i,j} \quad \textbf{such that} \;\; Y \text{ is PSD}$$

We may also add the linear inequalities $0 \leqslant y_{i,j} \leqslant 1$ to P. Since this semidefinite program is a relaxation of (11.17), we have obtained a method to compute an upper bound, unfortunately, this upper bound is useless, as the value of the semidefinite program always is n, since one can choose Y to be the identity matrix. The question remains whether there are more suitable relaxation approaches for MAXIMUMINDEPENDENTSET.

Exercises

Exercise 11.1. Given the following sets M_i,

- $M_1 = \{(x_1, x_2) : x_1 \geqslant x_2, x_1 \geqslant 1\}$,
- $M_2 = \{(x_1, x_2) : x_1 x_2 \geqslant 1, x_1 \geqslant 0, x_2 \geqslant 0\}$,

- $M_3 = \{(x_1, x_2) : x_1^2 + x_2^2 \leqslant 1\}$
- $M_4 = \{(x_1, x_2) : x_1^4 + x_2^4 \leqslant 1\}$

construct matrices F_i^0, F_i^1 and F_i^2 such that

$$M_i = \{(x_1, x_2) : \text{The matrix } F_i(x_1, x_2) := F_i^0 + x_1 \cdot F_i^1 + x_2 \cdot F_i^2 \text{ is PSD}\}.$$

Exercise 11.2. Find an optimal solution to the semidefinite program (11.1) (parabola and line).

Exercise 11.3. Derandomize the approximation algorithm for MaxCut which is described in Section 11.5.1.

Exercise 11.4. Let $-1 \leqslant y_i \leqslant 1, i = 1, \ldots, T$ and let $Y_{sum} := \sum_{i=1}^{T} y_i \leqslant 0$. Prove the following: If Y_{sum} is fixed, then the y_i for which $\sum_i \arccos(y_i)$ attains a minimum, fulfill the following property:
There is a value $Y \leqslant 0$ and an r with $T/2 \leqslant r \leqslant T$ such that r of the y_i are equal to Y and $T - r$ of the y_i are equal to one.

Bibliography

[AHOH96] T. Asano, K. Hori, T. Ono, and T. Hirata. Approximation algorithms for MaxSat: Semidefinite programming and network flows approach. Technical report, 1996.

[Ali95] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[AOH96] T. Asano, T. Ono, and T. Hirata. Approximation algorithms for the maximum satisfiability problem. *Nordic Journal of Computing*, 3:388–404, 1996.

[Asa97] T. Asano. Approximation algorithms for MaxSat: Yannakakis vs. Goemans-Williamson. In *Proceedings of the 5th Israel Symposium on the Theory of Computing and Systems*, 1997.

[BM95] R. Bacik and S. Mahajan. Semidefinite programming and its applications to $\mathcal{NP}$ problems. In *Proceedings of the 1st Computing and Combinatorics Conference (COCOON)*. Lecture Notes in Computer Science 959, Springer Verlag, 1995.

[FG95] U. Feige and M. Goemans. Approximating the value of two prover proof systems, with applications to Max2Sat and MaxDiCut. In *Proceedings of the 3rd Israel Symposium on the Theory of Computing and Systems*, pages 182–189, 1995.

[Goe97] M.X. Goemans. Semidefinite programming in combinatorial optimization. In *Proceedings of the 16th International Symposium on Mathematical Programming*, 1997.

[GvL86] G.H. Golub and C.F. van Loan. *Matrix Computations*. North Oxford Academic, 1986.

[GW94] M.X. Goemans and D.P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[Hås97] J. Håstad. Some optimal in-approximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[HL96] T. Hofmeister and H. Lefmann. A combinatorial design approach to MaxCut. *Random Structures and Algorithms*, 9(1-2):163–175, 1996.

[Joh74] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[Kar96] H.J. Karloff. How good is the Goemans-Williamson MaxCut algorithm? In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 427–434, 1996.

[KL96] P. Klein and H.-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MaxCut and Coloring. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 338–347, 1996.

[KMS94] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1994.

[Knu81] D.E. Knuth. *The Art of Computer Programming*, volume 2: *Seminumerical Algorithms*. Addison-Wesley, 2nd edition, 1981.

[MR95a] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 162–169, 1995.

[MR95b] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[PT95] S. Poljak and Z. Tuza. Maximum cuts and large bipartite subgraphs. In *Combinatorial Optimization*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 181–244, 1995.

[PY91] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[VB96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.

[Yan92] M. Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.

[Ye90] Y. Ye. An $\mathcal{O}(n^3 L)$ potential reduction algorithm for linear programming. In *Contemporary Mathematics*, volume 114, pages 91–107, 1990.