

Chapter 2

Semidefinite Programming

Semidefinite programming means optimizing a linear function (of the matrix entries) over the set of all positive definite $n \times n$ -matrices subject to finitely many linear inequalities. It is a powerful technique in the design of approximation algorithms. Here, we will illustrate it by means of one application, the Goemans-Williamson approximation algorithm for the MAXCUT problem [GW95]. Our discussion is based on Chapter 7 of the lecture notes [BG] and on the survey article [Lov03].

2.1 Positive Semidefinite Matrices and Semidefinite Programs

We first recall the definition of positive definite matrices:

Definition 2.1. Let $A = [a_{ij}]_{i,j=1}^n \in \mathbb{R}^{n \times n}$ be a symmetric matrix (i.e., $A^T = A$, or in other words, $a_{ij} = a_{ji}$ for all i, j). A is called *positive semidefinite*, denoted by $A \succeq 0$, if the quadratic form defined by A is nonnegative, i.e., if $x^T A x \geq 0$ for all $x \in \mathbb{R}^n$ (where \cdot^T denotes the transpose, here turning a column vector into a row vector, so that matrix multiplication makes sense).¹

We recall from linear algebra that a symmetric real $n \times n$ -matrix A has n real eigenvalues $\lambda_1, \dots, \lambda_n \in \mathbb{R}$ with corresponding eigenvectors u_1, \dots, u_n , $Au_i = \lambda_i u_i$, that form an orthonormal basis of \mathbb{R}^n , i.e., the u_i are pairwise orthogonal, of euclidean length $\|u_i\|_2 = 1$. Equivalently, there is an *orthogonal* matrix $U \in \mathbb{R}^{n \times n}$, i.e., one that satisfies $U^T U = I$, such that $U^T A U$ is the diagonal matrix with entries $\lambda_1, \dots, \lambda_n$.

¹Observe that the requirement that A be symmetric is not a real restriction: For any square matrix A , $x^T A x$ is a real number, so $x^T A x = (x^T A x)^T = x^T A^T x$. Thus, if $x^T A x \geq 0$ for all x , where $\tilde{A} := \frac{1}{2}(A + A^T)$ is the symmetrization of A

Lemma 2.2. Let $A = [a_{ij}] \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then the following statements are equivalent:

- (i) $A \succeq 0$.
- (ii) All eigenvalues of A are nonnegative: $\lambda_1, \dots, \lambda_n \geq 0$.
- (iii) There is a matrix $B \in \mathbb{R}^{d \times n}$, for some $d \geq 0$, such that $A = B^T B$. In other words, there are vectors $b_1, \dots, b_n \in \mathbb{R}^d$ (the columns of B) such that $a_{ij} = \langle b_i, b_j \rangle$ for all i, j .

We leave the proof of this lemma as an exercise. The matrix B in (iii) is called a *Gram matrix* for A , and the factorization $A = B^T B$ a *Gram decomposition*. Observe that n vectors always span a linear subspace of dimension at most n , so we may assume $d \leq n$. A special kind of Gram decomposition is the *Cholesky decomposition* that is studied in numerical analysis: For every symmetric positive semidefinite matrix $A \in \mathbb{R}^{n \times n}$, there is a *unique* upper triangular matrix $U \in \mathbb{R}^{n \times n}$ (i.e., $u_{ij} = 0$ for $j > i$) such that $A = U^T U$. If the input matrix A is positive definite and has rational entries, the matrix U can be computed efficiently, i.e., in time polynomial in n and in the bit sizes of the entries of A . To be more exact, the entries of the matrix U may be irrational² but can be computed efficiently to arbitrary precision (see [SB02], for instance).

A *semidefinite program* (SDP) is an optimization problem of the following form: Optimize (maximize or minimize) a linear function $\sum_{i,j} c_{ij} x_{ij}$ of the entries of the matrix $X = [x_{ij}]$ over all symmetric positive semidefinite matrices $X \in \mathbb{R}^{n \times n}$ that satisfy a finite number of linear inequalities in the matrix entries, i.e., finitely many inequalities $\sum_{i,j} b_{ijk} x_{ij} \leq \beta_k$, $1 \leq k \leq m$. That is, the numbers c_{ij} , b_{ijk} and β_k are given and specify the semidefinite program. There are many possible equivalent forms to write an SDP, for instance as

$$\begin{aligned} & \text{minimize} && c^T x = c_1 x_1 + \dots + c_m x_m \\ & \text{subject to} && x_1 A_1 + \dots + x_m A_m - B \succeq 0, \end{aligned} \tag{2.1}$$

where A_1, \dots, A_m, B are given symmetric $n \times n$ matrices and $c \in \mathbb{R}^m$ is a given vector. (It is a useful exercise to convince yourself that every SDP can be brought into this form.) Observe that in the special case that A_1, \dots, A_m, B are diagonal matrices, the SDP is equivalent to a linear program.

Fact 2.3. *Semidefinite programs can be solved efficiently. More precisely, there is an algorithm that, given an SDP with rational coefficients b_{ijk} and β_k and an error*

²For instance, if A is the diagonal matrix with 2's on the diagonal, then U is a diagonal matrix with $\sqrt{2}$ on the diagonal. The standard Cholesky decomposition algorithm requires the computation of n square roots and $O(n^3)$ additions and multiplications.

parameter $\varepsilon > 0$, computes a rational positive definite $n \times n$ -matrix Y such that $\sum_{i,j} b_{ijk} y_{ij} \leq \beta_k$ for all k and

$$\sum_{i,j} c_{ij} y_{ij} \leq \text{opt} + \varepsilon,$$

where opt is the optimal value of the SDP, i.e., the infimum of $\sum_{i,j} c_{ij} x_{ij}$ over all positive definite matrixes X satisfying all constraints of the SDP.³ The runtime of the algorithm is polynomial in n , $\log(1/\varepsilon)$, and in the bitsizes of the coefficients β_{ijk} and b_k .

We will not prove this or discuss the algorithm (the relevant keyword is “interior point methods”) but refer to the book [NN94] and the article [Ali95]. In the following, we will discuss an application, due to Goemans and Williamson, that exemplifies how semidefinite programming can be used to design approximation algorithms for hard optimization problems.

2.2 An Approximation Algorithm for MAXCUT

A *cut* in a graph $G = (V, E)$ is a partition $V = S \cup (V \setminus S)$ of the vertex set into two disjoint parts. The *size* or *weight* of the cut is the total number of edges across the cut, i.e., $|E(S, V \setminus S)|$, where $E(S, V \setminus S) = \{\{u, v\} \in E : u \in S, v \in V \setminus S\}$.

The well-known MINCUT problem is the combinatorial optimization problem of finding a cut of minimum size in a given graph G . This is closely related to finding maximum flows in networks, and there are a variety of algorithms to solve the MINCUT problem efficiently (in polynomial time).

In the MAXCUT problem, by contrast, the goal is to find a cut of *maximum* size. It has applications, for instance, in statistical physics and in VLSI design [BGJR88]. The MAXCUT problem (more precisely, its decision version⁴) was one of the first problems to be shown to be NP-complete [Kar72].

³To be precise, for this statement to be correct, one has to assume that the SDP is feasible (i.e., that there exists at least one positive semidefinite matrix X satisfying all constraints) and bounded, i.e., that $\text{opt} > -\infty$. Both assumptions will be satisfied in our applications, and in general, an arbitrary SDP can be transformed into one that is feasible and bounded, and such that solving the new SDP allows one to decide whether the original one was unbounded or infeasible. We also remark that, even for a feasible and bounded SDP, the optimum need not be attained. For example, the if we want to minimize x_1 subject to the condition that

$$\begin{bmatrix} x_1 & 1 \\ 1 & x_2 \end{bmatrix} \succeq 0$$

(this is an SDP in the form (2.1)), the semidefiniteness boils down to $x_1, x_2 \geq 0$ and $x_1 x_2 \geq 1$. Thus, $\text{opt} = 0$, but this is not attained.

⁴Given a graph G and an integer k , decide whether there is a cut in G of size at least k .

There is a simple randomized algorithm that achieves an expected approximation ratio of $1/2$: simply partition the vertices of the graph randomly, by deciding for each vertex independently in which of the two parts to put it (each part has probability $1/2$). Here, we present a (randomized) polynomial-time approximation algorithm, due to Goemans and Williamson [GW95], that uses semidefinite programming and achieves an (expected) approximation ratio of roughly 0.878 . That is, given a graph G , the algorithm computes a cut, making some random choices along the way, such that the expected number of edges across the computed cut is at least 0.878 times the maximum size of any cut in G .

It will be convenient to work with the following “arithmetization” of the problem. Suppose we are given a graph $G = (V, E)$. Without loss of generality, assume that $V = \{1, 2, \dots, n\}$. Define coefficients a_{ij} , $1 \leq i < j \leq n$ by $a_{ij} = 1$ if $\{i, j\} \in E$ and $a_{ij} = 0$ if $\{i, j\} \notin E$. Then we can rephrase the problem of finding a maximum cut in G as the following integer quadratic optimization problem:

$$\text{maximize } \sum_{1 \leq i < j \leq n} a_{ij} \frac{1 - x_i x_j}{2}, \quad x_i \in \{+1, -1\}, 1 \leq i \leq n \quad (2.2)$$

To see that the optimal solution to (2.2) corresponds to a maximum cut in G , observe that there is a bijection between subsets $S \subseteq V$ and ± 1 -vectors $x = (x_1, \dots, x_n) \in \{+1, -1\}^n$, namely $x \leftrightarrow S = \{i : x_i = +1\}$. Moreover, $\frac{1 - x_i x_j}{2}$ is equal to 1 if x_i and x_j have different signs, and 0 otherwise. Thus, under the above bijection,

$$\sum_{1 \leq i < j \leq n} a_{ij} \frac{1 - x_i x_j}{2} = \sum_{\{i,j\} \in E} \frac{1 - x_i x_j}{2} = |E(S, V \setminus S)|.$$

Observe that this immediately gives a very simple randomized approximation algorithm: Choose random signs X_i independently at random with $\Pr[X_i = +1] = \Pr[X_i = -1] = 1/2$. Then $\Pr[X_i X_j = -1] = \Pr[X_i X_j = 1] = 1/2$, by independence, and therefore the expected size of the resulting random cut equals

$$\mathbf{E} \left[\sum_{1 \leq i < j \leq n} a_{ij} \frac{1 - X_i X_j}{2} \right] = \sum_{1 \leq i < j \leq n} a_{ij} \frac{1 - \mathbf{E}[X_i X_j]}{2} = \frac{|E|}{2}.$$

Since $|E(S, V \setminus S)| \leq |E|$ for any S , this expectation is at least $1/2$ times the size of a maximum cut, i.e., we have a randomized approximation algorithm with expected approximation ratio at least $1/2$.

We will now use geometry and semidefinite programming to get a better approximation ratio. One way of motivating the improved algorithm is as

follows: We can think of the choices $+1$ and -1 as 1-dimensional unit length vectors. We relax (2.2) by allowing unit vectors that live in higher dimensions (and replacing the product of real numbers by the scalar product):

$$\text{maximize } \sum_{1 \leq i < j \leq n} a_{ij} \frac{1 - \langle u_i, u_j \rangle}{2}, \quad u_i \in \mathbb{S}^{n-1}, 1 \leq i \leq n, \quad (2.3)$$

where $\mathbb{S}^{n-1} = \{u \in \mathbb{R}^n : \|u\|_2 = 1\}$ is the set of unit vectors in \mathbb{R}^n , i.e., the $(n-1)$ -dimensional unit sphere centered at the origin. Observe that there is no need to consider unit vectors in higher dimensions,⁵ since the linear span of any set of n vectors is always of dimension at most n . Also note that the optimum of (2.3) is always at least as large as that of (2.2), since we can interpret any collection of n signs $x_1, \dots, x_n \in \{+1, -1\}$ as unit vectors $u_i = (x_i, 0, \dots, 0) \in \mathbb{S}^{n-1}$, $1 \leq i \leq n$ by simply filling up the remaining coordinates with zeros.

We remark that one can think of (2.3) in physical terms as the problem of placing the vertices of the graph on the unit sphere \mathbb{S}^{n-1} , where there is a repulsive force between adjacent vertices that grows linearly with the distance (for instance, the edges could be springs that want to expand), and we want to minimize the “energy” $\mathcal{E} = -\sum_{\{i,j\} \in E} \|u_i - u_j\|^2$.

For our purposes, however, it is more relevant that (2.3) can be rewritten as semidefinite program. Namely, given $u_1, \dots, u_n \in \mathbb{S}^{n-1}$, we can consider the positive semidefinite matrix $Y = [y_{ij}]$ given by $y_{ij} = \langle u_i, u_j \rangle$ (in particular with diagonal entries $y_{ii} = 1$), and conversely, given such a matrix Y , we can retrieve unit vectors u_i by computing a Gram decomposition for Y . The Gram decomposition is not unique, but we only care about the scalar products among the u_i , and these are uniquely determined by Y .

Thus, we get the following SDP:

$$\text{maximize } \sum_{1 \leq i < j \leq n} a_{ij} \frac{1 - y_{ij}}{2}, \quad Y \succcurlyeq 0, y_{ii} = 1, 1 \leq i \leq n. \quad (2.4)$$

We could ignore the constant terms in the objective function (since they sum up to $|E|/2$) and instead consider the problem of minimizing the linear function $\sum_{i < j} a_{ij} y_{ij}$ to bring the SDP closer to a standard form, but this is immaterial. Note also that the constraints are exactly of the required form: Positive semidefiniteness of the matrix Y , together with some linear inequalities on the matrix entries ($y_{ii} = 1$ can be expressed by two inequalities $y_{ii} \leq 1$ and $y_{ii} \geq 1$).

⁵One could consider, however, an intermediate problem, where the u_i are required to lie in some \mathbb{S}^{d-1} , $1 < d < n$. For $d = 1$, the problem is equivalent to MAXCUT, and hence NP-hard. Lovász [Lov03] mentions that he is not aware of any specific hardness results, but that he expects the problem to be NP-hard for any fixed d .

As mentioned above, we can solve this SDP and compute a Gram decomposition $Y = U^T U$ in polynomial time (more precisely, we can do this to arbitrary precision, an issue which we are going to ignore in the sequel). The constraints $y_{ii} = 1$ imply that the columns u_i of the matrix U are unit vectors⁶, and so an optimal solution of (2.4) yields an optimal solution of (2.3).

Thus, we have found a set of unit vectors $u_i \in \mathbb{S}^{n-1}$ that is “spread out” in the sense that on average, adjacent u_i and u_j are “far” from each other. How do we use this to obtain a large cut in the original graph? The idea is very simple: Partition the points u_i by a random hyperplane. Equivalently, choose $v \in \mathbb{S}^{n-1}$ uniformly at random and define $x_i := \text{sgn}(\langle u_i, v \rangle)$, where $\text{sgn}(t) = +1$ if $t \geq 0$ and $\text{sgn}(t) = -1$ if $t < 0$.

Thus, the approximation algorithm for MAXCUT consists of the following steps:

1. Given a graph G , compute an optimal solution Y to the SDP (2.4).
2. Compute a Gram decomposition $Y = U^T U$. The column vectors $u_1, \dots, u_n \in \mathbb{S}^{n-1}$ form an optimal solution for (2.3). As mentioned above, $\sum_{i < j} a_{ij} \frac{1 - \langle u_i, u_j \rangle}{2}$ is at least as large as the maximum size of a cut in G .
3. Use “random rounding” by a random vector $v \in \mathbb{S}^{n-1}$, i.e., define $x_i := \text{sgn}(\langle u_i, v \rangle)$. It will follow from Lemma 2.4 below that the resulting cut has size at least 0.878 times $\sum_{i < j} a_{ij} \frac{1 - \langle u_i, u_j \rangle}{2}$, hence at least 0.878 times the size of a maximum cut.

In order to make the description of the algorithm absolutely complete and precise, one would also need to specify how to choose a uniformly random unit vector $v \in \mathbb{S}^{n-1}$ *algorithmically*. Since the entries of v will typically be irrational, we can again only do this approximately (to arbitrary precision). One of the simplest ways may be to choose a vector $w \in \mathbb{R}^n$ according to the standard normal distribution (this is simple (again up to arbitrary precision) since we can choose the entries of w as independent 1-dimensional standard normal random variables) and then set $v = \frac{w}{\|w\|}$. In fact, one could skip the renormalization and work directly with w , since all we will need is that the distribution from which we pick the vector is rotationally symmetric.

Lemma 2.4. *Let $u \neq u' \in \mathbb{S}^{n-1}$, and let $v \in \mathbb{R}^n$ be a random vector chosen to a fixed rotationally symmetric (with respect to the origin) probability distribution (for instance, the uniform distribution on the unit sphere, or the standard normal distribution).⁷*

⁶Observe also that the SDP is feasible (take any set of unit vectors and the resulting matrix of scalar products) and bounded: since the u_i 's are unit vectors, we have $|y_{ij}| = |\langle u_i, u_j \rangle| \leq 1$ for all i, j , hence $|\sum_{i < j} a_{ij} y_{ij}| \leq |E| < \infty$.

⁷Rotational symmetry means that if $A \subseteq \mathbb{R}^d$ is a measurable set, and if ρ is a rotation of \mathbb{R}^d

Set $x := \text{sgn}(\langle v, u \rangle)$ and $x' := \text{sgn}(\langle v, u' \rangle)$. Then

$$\mathbf{E} \left[\frac{1 - xx'}{2} \right] = \Pr[xx' = -1] = \frac{\arccos\langle u, u' \rangle}{\pi} \geq \alpha \frac{1 - \langle u, u' \rangle}{2},$$

where $\alpha := \inf_{-1 \leq s < 1} \frac{2 \arccos(s)}{\pi(1-s)} > 0.87856$.

Proof. The equality $\mathbf{E} \left[\frac{1 - xx'}{2} \right] = \Pr[xx' = -1]$ is just the definition of the expectation of a 0/1-valued random variable. The inequality $\frac{\arccos\langle u, u' \rangle}{\pi} \geq \alpha \frac{1 - \langle u, u' \rangle}{2}$ follows directly from the definition of α , and the estimate for α is the subject of Exercise 11.

It remains to prove the middle equation $\Pr[xx' = -1] = \frac{\arccos\langle u, u' \rangle}{\pi}$. Consider first the case that $n = 2$, i.e., that u and u' are unit vectors on the unit circle in the plane. We have $xx' = -1$ iff the line orthogonal to v separates u and u' iff v lies in the double wedge formed by the two lines orthogonal to u and u' , respectively, see Figure 2.1. If $\beta = \arctan(\langle u, u' \rangle)$ is the angle between u and u' , then this double wedge covers an angle of 2β . Since the distribution of v is rotationally symmetric, the probability that v falls into this double wedge equals $\frac{2\beta}{2\pi} = \beta/\pi$, as desired.

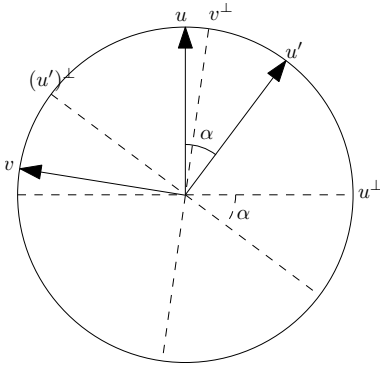


Figure 2.1: The probability that a random line separates two vectors equals twice the angle between them over 2π .

In the general case $u, u' \in \mathbb{S}^{n-1}$, let $F \subset \mathbb{R}^n$ be the 2-dimensional plane spanned by u and u' . Let \bar{v} be the orthogonal projection of v onto the plane F . Note that $\langle u, \bar{v} \rangle = \langle u, v \rangle$ and $\langle u', \bar{v} \rangle = \langle u', v \rangle$. Moreover, the distribution of \bar{v} is again rotationally symmetric. Thus, the general statement follows from the 2-dimensional case. \square

(an orthogonal linear transformation with determinant 1), then $\Pr[v \in A] = \Pr[v \in \rho A]$. We also assume that the origin has mass 0, i.e., $\Pr[v = \mathbf{0}] = 0$.

2.3 Exercises

Exercise 10. Let $A = [a_{ij}]$ be a symmetric $(n \times n)$ -matrix. A is called positive semidefinite, written $A \succcurlyeq 0$, if $x^T A x \geq 0$ holds for all $x \in \mathbb{R}^n$ (where \cdot^T denotes the transpose).

- Show that the set SDF_n of positive semidefinite symmetric $(n \times n)$ -matrices is a convex cone, i.e., if $A, B \in SDF_n$ and if $\alpha, \beta \in \mathbb{R}_{\geq 0}$, then $\alpha A + \beta B \in SDF_n$.
- Show that the following statements are equivalent:
 - $A \succcurlyeq 0$.
 - All eigenvectors of A are nonnegative.
 - For some dimension d , there exist vectors $u_1, \dots, u_n \in \mathbb{R}^d$ such that A is the Gram matrix of the u_i 's, in the sense that $a_{ij} = \langle u_i, u_j \rangle$ for $1 \leq i, j \leq n$. In other words, $A = U^T U$ for some $U \in \mathbb{R}^{d \times n}$. (A bonus question: What is the minimal d (as a well-known function of A) such that A can be written as such a Gram matrix?)
 - A can be written as a nonnegative linear combination of matrices of the form vv^T , $v \in \mathbb{R}^n$.

Exercise 11. Define

$$\alpha := \inf_{-1 \leq s < 1} \frac{2 \arccos(s)}{\pi(1-s)} = \inf_{0 < t \leq \pi} \frac{2t}{\pi(1 - \cos(t))}.$$

Show that $\alpha > 0.87856$.

(To avoid ambiguities, we mean the function $\arccos : [-1, 1] \rightarrow [0, \pi]$ that is the inverse of the cosine function $\cos : [0, \pi] \rightarrow [-1, 1]$.)

Exercise 12. The topic of this exercise is another application of semidefinite programming, this time to MAX-2-SAT. Recall that a 2-CNF formula in the variables x_1, \dots, x_n is a formula in conjunctive normal form $\varphi = C_1 \wedge \dots \wedge C_m$, where each clause $C_k = (\ell_i \vee \ell_j)$ is the disjunction of two literals $\ell_i \in \{x_1, \dots, x_n\} \cup \{\neg x_1, \dots, \neg x_n\}$, e.g. example,

$$\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3).$$

MAX-2-SAT is the following problem: given a 2-CNF formula φ , find an assignment $x_i \mapsto \{\text{true}, \text{false}\}$ of truth values to the variables that satisfies as many clauses as possible.

- Describe a randomized approximation algorithm that, in expectation, achieves an approximation ratio of $1/2$.

- b Reformulate MAX-2-SAT as a quadratic programming problem in variables $y_i \in \{+1, -1\}$. Hint: In order to circumvent certain technical difficulties, introduce an additional variable y_0 and interpret $x_i = \mathbf{true}$ as $y_i = y_0$, and $x_i = \mathbf{false}$ as $y_i \neq y_0$, and proceed as for MAXCUT. (In order to appreciate the hint, you might first want to try it without y_0 .)
- c Define a SDP relaxation of the above quadratic program and show how to use it to obtain an $(0.878 - \varepsilon)$ -approximation-algorithm for MAX-2-SAT.