CSC2411 - Linear Programming and Combinatorial Optimization*
Lecture 9: Semi-Definite Programming Combinatorial Optimization

Notes taken by Michael Mathioudakis

March 15th, 2007

Summary: This lecture consists of two main parts. In the first one, we revisit Semi-Definite Programming (SDP). We show its equivalence to Vector Programming, we prove it has efficient membership and separation oracles and finally state a theorem that shows why Ellipsoid can be used to acquire an approximate solution of a semi-definite program. In the second part, we make a first approach to Combinatorial Optimization. We introduce the concepts of Relaxation, Exact Relaxation and Totally Unimodular matrices and show how the latter are used to characterize a relaxation as exact.

1 Semi-Definite Programming

1.1 Introduction

In the previous lecture [Mos07], we introduced Semi-Definite Programming as a generalization of Linear Programming. In SDP, our feasible domain is the cone of Positive Semi-Definite Matrices and the restrictions of the problem are given by linear inequalities on the elements of a square matrix. In this lecture, we'll see that SDP has membership and separation oracles. However, SDP lacks the underlying structure of a polytope with rational vertices, that ensured (and helped us determine) the rational solutions of an LP. Therefore, we can use the Ellipsoid Algorithm together with the oracles, only to acquire an approximate solution of an SDP in polynomial time.

Below, we give the formal definition of a positive semi-definite matrix, the standard formulation of an SDP and a useful theorem that was proved in the previous lecture.

Definition 1.1. A matrix A is said to be Positive Semi-Definite (PSD) iff $x^T Ax \geq 0$ for every $x \in \mathbb{R}^n$.

* Lecture Notes for a course given by Avner Magen, Dept. of Computer Sciecne, University of Toronto.

In the following, we denote the symmetric matrices with $n \times n$ dimensions by M_n and we write $X \succeq 0$ to declare that X is positive semi-definite. In addition, we use the Frobenius inner product of matrices, denoted $A \bullet B$, in order to express the constraints of the program.

$$A \bullet B = \sum_{i=1}^{n} \sum_{i=1}^{n} a_{ij} b_{ij} = tr(A^T \cdot B)$$

Definition 1.2. The standard formulation of a Semi-Definite Program (SDP) is

$$\begin{gathered} \min C \bullet X \\ A_k \bullet X = b_k, \;\; 1 \leq k \leq m \\ X \in M_n \\ X \succeq 0 \end{gathered}$$

with C, X, $A_k \in R^{n \times n}$.

The following theorem provides alternative ways to perceive positive semi-definite matrices.

Theorem 1.3. *If $A \in \mathbb{R}^{n \times n}$ is symmetric, then the following are equivalent:*

1. *A is PSD.*
2. *If λ is an eigenvalue of A, then $\lambda \geq 0$.*
3. *There is a matrix $W \in \mathbb{R}^{n \times n}$, such that $A = W^T W$.*

1.2 Semi-Definite Programming and Vector Programming

Definition 1.4. A Vector Program is a program that is linear with respect to inner products between n vectors of size n. More specifically, a Vector Program can be formulated as follows.

$$\min \textstyle\sum_{1 \leq i,j \leq n} c_{ij} v_i^T v_j$$

$$\sum_{1 \leq i,j \leq n} a_{ij}^{(k)} v_i^T v_j = b_k, 1 \leq k \leq m$$

$$v_i \in \mathbb{R}^n, 1 \leq i \leq n.$$

From Theorem 1.3, we know that if we have a square matrix X with $X \in M_n$ and $X \succeq 0$, then $X = W^T W$ for some square matrix W and therefore, every entry of X can be expressed as vector inner product $X(i,j) = (W_i)^T \cdot W_j$, where W_i denotes the i-th column of W.

Now, let's assume that we have an SDP S. We define the corresponding Vector Program V as follows. We replace the restrictions $X \in M_n$ and $X \succeq 0$ with the demand for existence of n vectors $\{W_1, W_2, \ldots, W_n\}$ of size n, and we replace every occurrence of the variable entry $X(i,j)$ with the inner product $(W_i)^T \cdot W_j$. Obviously, for every feasible solution X there is a feasible solution $\{W_1, W_2, \ldots, W_n\}$ of n-vectors

for V, with the same objective value. Inversely, if $\{w_1, w_2, \ldots, w_n\}$ is a solution to V, then $X = W^T W$, with $W = [w_1|w_2|\ldots|w_n]$ is a feasible solution of S with the same objective value. Therefore, we have shown the next theorem.

Theorem 1.5. *Semi-definite programming is equivalent to vector programming.*

1.3 SDP and Ellipsoid Algorithm

Unlike Linear Programs, Semi-Definite Programs can only be solved approximately within an additive error ϵ. This can be achieved by using the Ellipsoid Algorithm (see [GLS88]) or the Interior Points Method (see [Ali95]).

As we know, Ellipsoid uses membership and separation oracles and is oracle-polynomial. In Claim 1.6, we prove that Semi-Definite Programs have efficient (polynomial with respect to the input size) membership and separation oracles (see [Vaz03] for a slightly different proof).

Claim 1.6. *Let S be a semi-definite program and $A \in \mathbb{R}^{n \times n}$. Then, there is a polynomial algorithm that either*

1. *determines that A is feasible for S, or*
2. *provides a separating hyperplane.*

Proof *We can check in polynomial time whether A is not symmetric. If it isn't, then we can find i, j for which $A(i,j) > A(j,i)$. Then $A(i,j) \leq A(j,i)$ provides a separating hyperplane.*

If A is symmetric, then, using Cholesky decomposition, we can bring it in the form

$$A = U^T D U \text{ , } D \text{ diagonal, } U \text{ nonsingular}$$

in polynomial time. We can show that A is PSD iff D is PSD. Indeed, if D is PSD, then for every $w \in \mathbb{R}^n$

$$w^T A w = w^T U^T D U w = (Uw)^T D (Uw) \geq 0$$

and therefore A is PSD, too. Vice versa, if D is not PSD, then it has a negative (diagonal) entry. For that entry, say D_{ii}, we have that $e_i^T D e_i = D(i,i) < 0$. If we take $z = U^{-1} e_i$, then $z^T A z = z^T U^T D U z = e_i^T D e_i < 0$ and therefore, A is not PSD.

So, in order to check whether A is PSD it suffices to check whether D is PSD. This can be done in polynomial time, since D is PSD iff all its diagonal entries are non-negative. If the entry $D(i,i)$ is negative, then for $z = U^{-1} e_i$ we have that $z^T A z = z^T U^T D U z = e_i^T D e_i < 0$ and therefore $(z z^T) \bullet A \geq 0$ provides a separation hyperplane.

If A is both symmetric and PSD, then we check if any of the linear constraints is violated. This takes polynomial time w.r.t. the size of the input. If there is not a violated constraint, then A is feasible. Otherwise, the violated constraint immediately gives a separating hyperplane.

The fact that Ellipsoid can be actually used to efficiently approximate the solution of an SDP is guaranteed by Theorem 1.8 (see [GLS88]).

Definition 1.7. Let $K \in \mathbb{R}^n$ be a convex body.

$$S(K, \epsilon) = \{x | d(x, K) \leq \epsilon\}$$

$$S(K, -\epsilon) = \{x | d(x, K^{-c}) \geq \epsilon\} = (S(K^c, \epsilon))^c$$

Theorem 1.8. *Let K be a convex body, such that $B(C, r) \subset K \subset B(O, R)$, for some $K, O \in \mathbb{R}^n$, $0 < r < R$, supplied with efficient membership oracle. Then the following approximation can be achieved using Ellipsoid, in time $poly(n, |c|, \log\left(\frac{R}{r}\right), \log\left(\frac{1}{\epsilon}\right))$.*

1. *Find $y \in S(K, \epsilon)$, so that $c^T y \leq c^T x + \epsilon$ for all $x \in S(K, -\epsilon)$, or*
2. *Assert $S(K, -\epsilon) = \emptyset$*

1.4 SDP and Interior Points Methods

Alizadeh in [Ali95] presents an interior point algorithm that converges to the solution of an SDP in polynomial time. The approach he follows is an extension of Ye's projective method for LP ([Ye90]). Alizadeh's algorithm uses the following barrier function.

$$\Phi(X) = -\log(\det(X)) = -\log(\prod_i \lambda_i(x)) = -\sum_i \log(\lambda_i(x))$$

2 Combinatorial Optimization

2.1 Introduction

In Combinatorial Optimization problem settings, we try to find optimal solutions over a well defined discrete space. In general, combinatorial optimization problems are given as pairs of a set of restrictions that define the problem space and an objective function we wish to optimize. In the rest of the course, we'll mainly look into problems that are NP-hard and can be formulated as Integer Programs.

Definition 2.1. The standard form of an Integer Program is

$$\begin{gathered} \min c^T x \\ Ax = b \\ x \geq 0 \\ x \in \mathbb{Z}^n. \end{gathered}$$

2.2 From Integer Programs to LP

Consider the well known problem of Vertex Cover.

Definition 2.2. Minimum Vertex Cover Given a graph $G = (V, E)$, find a set of vertices S, such that for every edge $uv \in E$, either $u \in S$ or $v \in S$.

Vertex Cover can be easily formulated into an Integer Program as follows. Notice also that there might be more than one integer programs to a combinatorial problem.

Min Vertex Cover

$$\min \sum_{i=1}^{n} x_i$$
$$x_i + x_j \geq 1, \forall i, j \in E$$
$$x \in \{0,1\}^n$$

By looking at the integer program above, we observe that it actually suffices to demand $x \geq 0$, $x \in \mathbb{Z}$ instead of $x \in \{0,1\}^n$. This leads to the formulation of the following integer program.

Min Vertex Cover

$$\min \sum_{i=1}^{n} x_i$$
$$x_i + x_j \geq 1, \forall i, j \in E$$
$$x \geq 0$$
$$x \in \mathbb{Z}$$

Unfortunately, Vertex Cover is NP-hard and therefore it doesn't have a polynomial time algorithm unless P=NP. (Notice that this shows Integer Programming is NP-hard). It would be nice, though, if we were able to approximate it. One idea towards this end is to drop the restriction $x \in \mathbb{Z}$, which brings us to a problem we know well how to deal with.

Min Vertex Cover Relaxation

$$\min \sum_{i=1}^{n} x_i$$
$$x_i + x_j \geq 1, \forall i, j \in E$$
$$x \geq 0$$

The program above is a linear program and we'll say it is a *relaxation* of Vertex Cover. More formally, we give the following definition.

Definition 2.3. We say that a linear program R is a **relaxation** to a combinatorial problem P, if every feasible solution of P is also a feasible solution of R and every infeasible solution of P is infeasible for R. Furthermore, a relaxation is said to be **exact**, if all the vertices of R are integral.

It's easy to see that when an LP relaxation is exact, it achieves the same optimum with the integer program. But how far can we fall from the optimum value of the integer program? To characterize the 'quality' of a relaxation we define the concept of the *integrality gap*. The integrality gap of a relaxation that is exact for all instances of a problem is equal to 1.

Definition 2.4. The **integrality gap** between a minimization integer program P and its relaxation R_P is defined as

$$\sup_{I \; instance \; of \; P} \frac{OPT_P(I)}{OPT_{R_P}(I)}.$$

2.3 Exact Relaxations and Totally Unimodular Matrices

As another example, consider the problem of Maximum Weighted Matching.

Definition 2.5. Maximum Weighted Matching Given a graph $G(V, E)$ and weights w_e on the edges, find a matching M with maximum $\sum_{e \in M} w_e$.

The corresponding integer program is given below.

Maximum Weighted Matching

$$\max \sum w_e x_e$$
$$\sum_{e \text{ touches } v} x_e \leq 1, \forall v \in V$$
$$x_e \in \{0, 1\}$$

Similarly as with Vertex Cover, we get the following relaxation of the problem.

Maximum Weighted Matching Relaxation

$$\max \sum w_e x_e$$
$$\sum_{e \text{ touches } v} x_e \leq 1, \forall v \in V$$
$$x_e \geq 0$$

We'll prove that *when the graph $G(V, E)$ is bipartite, the relaxation above is exact*. Equivalently, we need to show that the polytope of the relaxation has integral vertices. As we can see from Claim 2.7, Totally Unimodular matrices are tightly bound with this property (see also [GLS88] and [PS98]). As it follows from their definition, a totally unimodular matrix can have only +1, -1 or 0 as elements.

Definition 2.6. A matrix A is called Totally Unimodular if every square submatrix M of it has $\det(M) \in \{+1, -1, 0\}$.

Claim 2.7. *If A is Totally Unimodular and b is integral, then the vertices of $P = \{x | Ax = b, x \geq 0\}$ are integral.*
Proof *We know from LP that A has a submatrix A' with full row rank so that each solution x^* of $A'x = b'$, where b' is the corresponding subvector of b, is a solution to $Ax = b$. Without loss of generality, assume $A' = (A_1, A_2)$, with A_1 non-singular. Let $M_j = [A_{1_1}, \ldots, A_{1_{j-1}}, b', A_{1_{j+1}}, \ldots, A_{1_m}]$ be the square matrix that we get by substituting the j-th column of A_1 with b'. Then, by Cramer's rule, we have that the j-th basic variable is given by the formula*

$$x_j^* = \frac{\det(M_j)}{\det(A_1)}.$$

Since A_1 is unimodular and M_j is integral, we have that $x_j^* \in \mathbb{Z}$.

This result can be extended to other matrices related with A.

Claim 2.8. *If A is Totally Unimodular and b is integral, then the vertices of $P = \{x | Ax \leq b, x \geq 0\}$ are integral.*
Proof. *It suffices to prove that if A is Totally Unimodular, then $A' = (A|I)$ is also Totally Unimodular. Let C be a square submatrix of A' covering parts of both A and I. By permuting its rows, C can be written as*

$$C = \left(\begin{array}{c|c} B & O \\ \hline D & I_k \end{array}\right).$$

B can be a square submatrix of A or a singular matrix (with a zero column). It's clear that

$$\det(C) = \det(B) \in \{+1, -1, 0\}.$$

What still remains to be proved, is that if G is a bipartite graph, then its incidence matrix is Totally Unimodular. This follows directly from the next claim. The proof provided here is for an undirected graph G. Please consult the notes of Lecture 10 for a different proof related with directed graphs.

Claim 2.9. *An integer matrix A with $a_{ij} \in \{0, 1\}$ is Totally Unimodular if no more than two 1-entries appear in any column and if the rows of A can be partitioned into two sets I_1 and I_2 such that if a column has two 1-entries, their rows are in different sets.*

Proof *We perform induction on the size of the (square) matrices. For single matrix entries, the claim is true by hypothesis. Now let B be any submatrix of size k.*

- *If B has a zero column, then* $\det(B) = 0$.
- *If B has no zero columns and has a column with one 1-entry, then we can expand its determinant along that column and the result follow from the induction hypothesis.*
- *If B has only columns with two 1-entries, from hypothesis we have that*

  $$\sum_{i \in I_1} a_{ij} = \sum_{i \in I_2} a_{ij} = 1, \textit{ for every } j$$

  and therefore, we can find a linear combination of the rows that is equal to zero. Hence, $\det(C) = 0$.

From Claim 2.8 and Claim 2.9, it follows that **the relaxation we gave for Maximum Weighted Matching is exact for bipartite graphs**.

References

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

[GLS88] M. Grotschel, L. Lovasz, and A. Schrijver. Geometric algorithms and combinatorial optimization. *Springer, Heidelberg*, 1988.

[Mos07] S. Mostafavi. Linear programming and combinatorial optimization - lecture 8: Interior points method and semi definite programming. *CSC2411, University Of Toronto*, 2007.

[PS98] C. Papadimitriou and I. Steiglitz. Combinatorial optimization: algorithms and complexity. *Dover*, 1998.

[Vaz03] V. Vazirani. Approximation algorithms. *Springer*, 2003.

[Ye90] Yinyu Ye. A class of projective transformations for linear programming. *SIAM Journal on Computing*, 19(3):457–466, 1990.