CSC2411 - Linear Programming and Combinatorial Optimization*
Lecture 11: Solving SDPs and Applying LP to Combinatorial Optimization Problems

Notes taken by Vinod Nair

April 6, 2005

Summary: The two main topics of this lecture are:

1. Solution methods for semidefinite programs: We look at using the Ellipsoid algorithm and interior point methods for computing an approximate solution for semidefinite programs in polynomial time.
2. Application of linear programming to combinatorial optimization problems: We consider a number of examples of combinatorial optimization problems. Initially these problems are expressed as integer programs, which are then "relaxed" to obtain a related linear program. The relationship between the integer program and its corresponding linear program is discussed.

1 Algorithms for Solving Semidefinite Programs

We have already seen how the Ellipsoid algorithm and interior point methods can be used to solve linear programs. These two methods are applicable to problems more general than just LP. Here we look at how they can be used to solve semidefinite programs (SDPs).

1.1 Ellipsoid Algorithm for SDP

Recall that the Ellipsoid algorithm is a procedure for finding a point in the feasible set of the optimization problem. Roughly speaking, it starts off by putting a large enough ellipsoid around the feasible set. It tests whether the centre of the ellipsoid is in the feasible set. If the centre point is not feasible, the algorithm finds a hyperplane that defines a half-space separating the feasible set from the ellipsoid's centre. Then a

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

new, smaller, ellipsoid that contains the intersection of this half-space and the original ellipsoid is computed, and the same procedure is repeated.

A key step in this procedure is to test whether a given point is in the feasible set, and if not, compute a hyperplane that separates that point from the feasible set. We need to design a separation oracle that can carry out this step in polynomial time if we want to solve an SDP with the ellipsoid algorithm in polynomial time. In the case of SDP, a separation oracle is a procedure that efficiently tests whether a symmetric matrix A is positive semidefinite, and if not, computes a hyperplane that separates A from the convex cone corresponding to the set of all positive semidefinite matrices (of the same dimension as A).

An efficient separation oracle for positive semidefinite matrices works as follows. Any symmetric matrix A can be written in the form

$$A = U^t D U$$

where U and D are square matrices of the same dimension, U is nonsingular, and D is a diagonal matrix that is positive semidefinite if and only if A is. Since the diagonal entries of D are its eigenvalues, they must all be nonnegative for it to be positive semidefinite. This condition provides an easy test for whether A is positive semidefinite.

If D has a negative entry, then we want to compute a separating hyperplane that places A on one side and all positive semidefinite matrices on the other side. In other words, we want to compute a matrix H such that $H \cdot A < 0$ ($\cdot$ stands for matrix inner product) and $H \cdot Z \geq 0$ for any positive semidefinite matrix Z.

Without loss of generality, let $D_{11} < 0$. Then $e_1^t D e_1 = D_{11} < 0$, where e_1 is a vector with the first element set to 1 and all others set to 0. Let $w = U^{-1} e_1$. We get

$$w^t A w = e_1^t (U^{-1})^t A U^{-1} e_1 = e_1^t D e_1 < 0.$$

Therefore, $w^t A w = (w w^t) \cdot A < 0$. For any positive semidefinite matrix Z, by definition, $(w w^t) \cdot Z = w^t Z w \geq 0$. That means the matrix $w w^t$ is a separating hyperplane for A. All computations needed to test positive semidefiniteness and to find the separating hyperplane can be done in polynomial time.

Now that we have an efficient separation oracle, we can use the Ellipsoid algorithm to solve an SDP in polynomial time. However, unlike in LP, it may not be possible to compute an exact solution in polynomial time. For example, in assignment 3 we saw that an SDP can have an irrational solution, even if the SDP itself consists of only rational numbers. In such cases, the best we can hope for is an approximate solution. It can be shown that the Ellipsoid algorithm can approximate the true optimal solution within ϵ in a running time that is polynomial in $\frac{1}{\epsilon}$.

Let $K \in \Re^n$ be a convex body. Define $S(K, \epsilon) = \{x \in \Re^n \mid d(K, x) \leq \epsilon\}$, where d stands for Euclidean distance. This is the set of points that are within ϵ distance from the boundary of K (for small ϵ, these are points in the vicinity of K's boundary). Also define $S(K, -\epsilon) = \{x \in \Re^n \mid d(\overline{K}, x) \geq \epsilon\}$, where $\overline{K}$ is the complement of K. This is the set of points inside K that are more than ϵ away from its boundary (i.e., points that are deep inside K). Figure 1 shows an example of these sets. Now we state a theorem that says the Ellipsoid algorithm can compute an approximate optimum of a rational, linear objective function over a convex body in polynomial time.

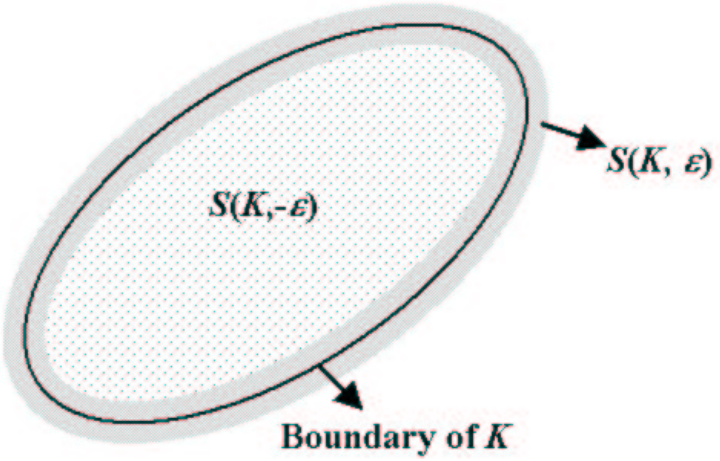

Figure 1: $S(K, \epsilon)$ is the set of points that are within ϵ of the boundary of K. $S(K, -\epsilon)$ is the set of points inside K that are more than ϵ away from its boundary.

Theorem 1.1. *(Adapted from [GLS93].) Let K be a convex body in $\Re^n$ such that $B(0, r) \subseteq K \subseteq B(0, R)$, where $R > r > 0$. Assume that an efficient separation oracle exists for K. Let the objective function we want to minimize be $\langle c, x \rangle$, where $c \in \Re^n$ is a rational vector. For a given (rational) $\epsilon > 0$, the Ellipsoid algorithm can*

1. *find a rational vector $y \in \Re^n$ such that $y \in S(K, \epsilon)$ and $\langle c, y \rangle \leq \langle c, x \rangle + \epsilon$ for all $x \in S(K, -\epsilon)$.*

   or else,

2. *assert $S(K, -\epsilon) = \varnothing$,*

in a running time that is polynomial in n, $\log(R/r)$ and $\frac{1}{\epsilon}$.

Note there are three different approximations being made here: 1) the solution found by the Ellipsoid algorithm may only be in the vicinity of K, 2) the cost of this solution will be only within ϵ of the cost of points in K that are 3) only deep inside K. But the Ellipsoid algorithm's approximate solution can be made arbitrarily close to the true solution at the expense of more running time.

1.2 Interior Point Method for SDP

The Ellipsoid algorithm for SDP is interesting mainly as a proof of the existence of a polynomial time algorithm, because it is often too slow to be practically useful. The interior point method gives a practical algorithm for solving SDPs. Here we give a high-level overview of how the method can be applied to SDPs.

We need to define a barrier function that keeps the algorithm away from the boundary of the feasible set. We have seen that a symmetric matrix X must be positive

semidefinite for it to be feasible, i.e., its eigenvalues must be nonnegative. So the boundary of feasibility is reached when an eigenvalue becomes 0. Therefore, the barrier function should be designed so that it keeps the eigenvalues of X away from 0. This can be done by defining the barrier function to be

$$\phi(Y) = -\sum_i \log(\lambda_i(X)) = -\log\prod_i(\lambda_i(X)) = -\log(det(X)),$$

where $\lambda_i(X)$ are the eigenvalues of X. The value of this function increases as an eigenvalue approaches 0 and blows up to infinity at 0.

With this barrier function, we can apply the interior point method to SDP in much the same way as it was used for LP. It can be shown that the interior point method can approximate the optimum within some ϵ in a running time that is polynomial in $\log(\frac{1}{\epsilon})$.

2 LP in Combinatorial Optimization

Now we look at how LP can be used in combinatorial optimization problems. We begin by considering integer programming (IP). An integer program (or more specifically an integer linear program) is a linear program with the additional constraint that the optimization variable can take on only integer values:

$$\begin{aligned}
&\min\langle c, x\rangle \\
&Ax \leq b, \\
&x \geq 0, \\
&x \in Z^n.
\end{aligned}$$

Integer programming is NP-hard, so there is no known algorithm to efficiently solve it. One natural idea for solving IPs is to "relax" the integrality constraint and allow x to take on real values. The resulting problem is an LP, which we know how to solve efficiently. Clearly, the LP's optimum need not be the IP's optimum (see figure 2). We can somehow try to round the solution of the LP to a feasible integral point and take that as the solution. But determining how to do the rounding so as to obtain the true optimum can be as hard as solving the original IP itself. The next best thing we can do is develop methods that go from the optimum of the LP to points that are nearby the true optimum, and bound how far off the the true optimum is.

Next, we give examples of NP-hard combinatorial optimization problems expressed as IPs, and look at the LPs corresponding to their relaxed versions.

2.1 Minimum vertex cover

We want to find the smallest subset of vertices of an undirected graph that contains at least one endpoint of every edge in the graph. To represent this problem as an IP, first define for each vertex i an indicator variable x_i that indicates whether it is in the vertex

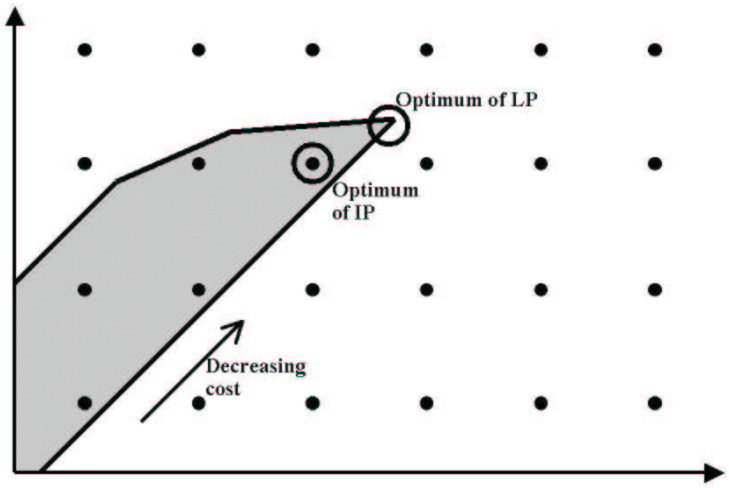

Figure 2: The optimum for an IP can be different from that of the LP given by relaxation.

subset S. So x_i = 1 if $i \in S$, and x_i = 0 otherwise. The IP is:

$$\min \sum_i x_i$$
$$\text{For every edge } e = \{i, j\},\ x_i + x_j \geq 1,$$
$$x_i = \{0, 1\}.$$

The first constraint makes sure that for every edge, at least one of the two vertices that it connects is in S.

We can relax the integrality constraint to obtain the following LP:

$$\min \sum_i x_i$$
$$\text{For every edge } e = \{i, j\},\ x_i + x_j \geq 1,$$
$$0 \leq x_i \leq 1.$$

Note that optimum of this LP can be different from the optimum of the IP. For example, for a graph with n vertices and edges between all pairs of vertices (i.e., a clique on n vertices), the LP's optimum is attained when $x_i = \frac{1}{2}$ for all i, with an optimal cost of $\frac{n}{2}$. But the optimum of the IP is $n - 1$.

2.2 Knapsack

Given n items, each with value c_i and weight w_i ($i = 1, ..., n$), we want to find the subset of items with the maximum total value, subject to the constraint that its total

weight is at most M. To write this as an IP, again define an indicator variable x_i, where x_i = 1 if item i is in the subset and 0 otherwise. So the IP is:

$$\max \sum_i c_i x_i$$
$$\sum_i w_i x_i \leq M,$$
$$x_i \in \{0, 1\}.$$

As before, rewriting the integrality constraint as $0 \leq x_i \leq 1$ gives the LP for the relaxed version.

2.3 Maximum independent set

We want to find the largest subset of vertices of a graph such that no two vertices in the subset are connected by an edge. Define, for each vertex i, an indicator variable x_i that has value 1 when i is in the subset, and 0 otherwise. Then the IP is

$$\max \sum_i x_i$$
For every edge $e = \{i, j\}$, $x_i + x_j \leq 1$,
$x_i \in \{0, 1\}$.

Again, the LP for the relaxed version replaces the integrality constraint with $0 \leq x_i \leq 1$. For a complete graph with n vertices, the LP's optimum is attained when $x_i = \frac{1}{2}$ for all i, with an optimal value of $\frac{n}{2}$. But the optimum of the IP is 1.

2.4 Satisfiability

Given a set of Boolean variables, and a Boolean expression containing conjunctions (ANDs) and disjunctions (ORs) of these variables and their negations, we want to know whether there is a setting of the variables that can make the expression true. For example, let x_1, x_2, and x_3 be Boolean variables. We want to know whether the expression

$$(x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3)$$

can be made true by setting x_1, x_2 and x_3 to either 0 (false) or 1 (true). This can be formulated as the problem of finding a feasible solution to an IP. The constraints of the IP are:

$$x_1 + x_2 \geq 1,$$
$$(1 - x_2) + x_3 \geq 1,$$
$$x_i \in \{0, 1\} \text{ for all i.}$$

(Here there is no objective function to optimize, but since the feasibility problem is equivalent to the optimization problem, it is possible to convert the above problem into

an equivalient ordinary IP.) The relaxed version of the problem replaces the integrality constraint with $0 \leq x_i \leq 1$. Note that determining how to round the solution of the relaxed problem so as to obtain the true optimum is as hard as solving the original problem itself. For example, a feasible solution for the relaxed version of the above problem is $x_1 = x_2 = x_3 = \frac{1}{2}$. But rounding this solution to get the true optimum is equivalent to solving the satisfiability problem.

2.5 Maximal weight matching in a bipartite graph

Given a bipartite graph with a nonnegative weight we assigned to each edge, we want to find the subset of the edges that has the largest total weight, subject to the constraint that no two edges in this subset share an endpoint. Let x_e be an indicator variable for edge e with value 1 when e is included in the subset, and 0 when it is not. Then we get the following IP:

$$\max \sum_e w_e x_e$$

$$\text{For every vertex } v, \sum_{\text{edges incident on } v} x_e \leq 1$$

$$x_e = \{0, 1\}.$$

In the relaxed version, the integrality constraint is changed to $x_e \geq 0$. (We don't have to explicitly enforce $x_e \leq 1$ because this is already done by the first constraint.)

We will now show that the optimum of the LP corresponding to the relaxed version of this problem is the same as that of the IP. In other words, for this problem, relaxation does not change the optimum.

Definition 2.1. The relaxation of an IP is *exact* if all vertices of the relaxed LP's polyhedron are integer vectors.

It easily follows from this definition that if a relaxation is exact, the optimum of the relaxed LP must be the same as the optimum of the IP. We can see this intuitively from figure 3.

Claim 2.2. *Relaxation for maximal weight matching in a bipartite graph is exact.*

To prove this claim, we first need the following definition.

Definition 2.3. A matrix A is *totally unimodular* (TUM) if the determinant of every square submatrix of A is either -1, 0 or 1.

This definition is useful here because it is possible to show that if A is TUM, then all vertices of the polyhedron $P = \{x \mid Ax \leq b, x \geq 0\}$ are integer vectors when b is an integer vector. Note that total unimodularity restricts the entries of A to be -1, 0 or 1, since the determinant of a 1x1 submatrix is itself.

In the maximal weight matching problem, the constraint matrix A of the relaxed LP is the incidence matrix of the bipartite graph. (The incidence matrix of an undirected graph with n vertices and m edges is an $n \times m$ matrix in which the entry at (i, j) is 1

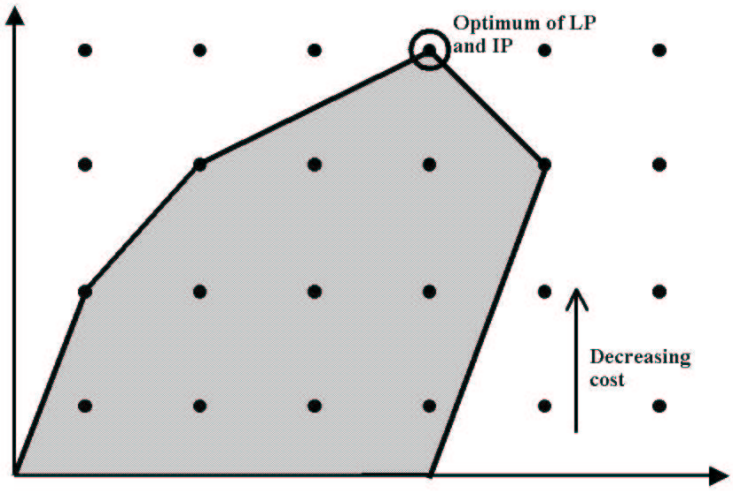

Figure 3: Both the LP and the IP have the same optimum, regardless of the objective function, when the vertices of the feasible set are integer vectors.

if vertex i is an endpoint of edge j, and 0 otherwise. For a directed graph, the entry at (i, j) is 1 if edge j goes out of vertex i, -1 if edge j goes into vertex i, and 0 otherwise.) It turns out (as we will show) that the incidence matrix of a bipartite graph is TUM. This then implies that the relaxed LP can only have an integer solution, and therefore, the relaxation is exact.

We start by proving the following claim:

Claim 2.4. *All vertices of the polyhedron $P_1 = \{x \mid Ax \leq b\}$ are integer vectors if A is TUM and b is an integer vector.*

Proof. Recall that the vertices of P_1 are solutions to $A'x = b'$, where A' is a nonsingular submatrix formed from a subset of linearly independent rows of A, and b' is the corresponding subvector of b. Using Cramer's rule to solve for x gives

$$x_i = \frac{det(M_i)}{det(A')},$$

where x_i is the i^{th} element of x, and M_i is A' with column i replaced by b'. Determinant of M_i must be an integer because all its elements are integers. Determinant of A' must be either -1 or +1 because it is a submatrix of a TUM matrix and A' is nonsingular. Therefore, each element of x must be an integer. □

We can apply this result to a polyhedron of the form $P = \{x \mid Ax \leq b, x \geq 0\}$ by rewriting the constraints in the form

$$\begin{pmatrix} A \\ -I_n \end{pmatrix} x \leq \begin{pmatrix} b \\ 0_n \end{pmatrix}$$

where n is the dimensionality of x, I_n is the $n \times n$ identity matrix, and 0_n is the n-dimensional zero vector.

All we have left to show is that the incidence matrix of a bipartite graph is TUM. But before we do that, consider the following lemma:

Lemma 2.5. *The incidence matrix A of a directed graph is TUM.*

Proof. First, note some properties of the incidence matrix of a directed graph. All of its entries are -1, 0 or 1. Every column has exactly one -1 and one +1 (i.e. every edge must go into a vertex and must come out of a vertex). So, for any square submatrix of A, a column can have at most two nonzero entries, and if there are two nonzero entries, then they must be -1 and +1.

We can do induction on the size of the submatrices of A. The base case is a 1×1 submatrix. Since the entries of A are in $\{-1, 0, 1\}$, the determinant of any 1×1 submatrix must be in $\{-1, 0, 1\}$.

Now we have to show that if the determinant of any $k \times k$ submatrix is in $\{-1, 0, 1\}$, then the same is true for any $(k+1) \times (k+1)$ submatrix. Let C be a $(k+1) \times (k+1)$ submatrix. There are three possible cases:

1. C has a column of all zeros: In this case, $det(C) = 0$.

2. C has a column with exactly one nonzero element: By expanding the determinant of C along that column, we get $det(C) = \pm 1 \times det(C_k)$, where C_k is a $k \times k$ submatrix of C. Determinant of C_k is in $\{-1, 0, 1\}$ (by the induction hypothesis), and therefore the determinant of C must also be so.

3. The only remaining case is where C has exactly one -1 and one +1 in every column. In this case, the sum of all the rows of C is the zero vector, which implies that $det(C) = 0$.

In all three cases $det(C)$ is -1, 0 or 1. Therefore, the incidence matrix of a directed graph is TUM. □

From this lemma it follows that the incidence matrix of a bipartite graph is also TUM. Too see this, first note that the incidence matrix of a bipartite graph has entries of either 0 or 1. We can then transform the graph into a directed one by assigning directions to the edges as going from the vertices on the left to the vertices on the right. The zero-entries in the original incidence matrix will still remain zero, but the one-entries may change sign as a result of this transformation. This will affect only the sign of the determinant of the incidence matrix and any of its submatrices. Therefore, the incidence matrix of the bipartite graph must also be TUM.

Putting all these results together, we finally reach the conclusion that relaxation for maximal weight matching in a bipartite graph is exact. So, *Optimum*(LP) = *Optimum*(IP).

2.6 Duality, maximal weight matching and vertex cover

The LP for the relaxed version of the maximal weight matching problem can be written as:

$$\begin{aligned} &\max\langle w, x\rangle \\ &Ax \leq \mathbf{1}, \\ &x \geq 0, \end{aligned}$$

where w is the vector of weights on the edges, x is the vector of indicator variables, **1** is the vector with all elements set to 1, and A is the incidence matrix. Its dual is

$$\begin{aligned} &\min\langle y, 1\rangle \\ &A^t y \geq w, \\ &y \geq 0. \end{aligned}$$

Restrict w to be a vector of all ones, i.e. look for the maximal matching. Then we can write the dual as:

$$\begin{aligned} &min \sum_i y_i \\ &\text{For every edge } e = \{i, j\},\ y_i + y_j \geq 1 \\ &y_i \geq 0. \end{aligned}$$

The IP corresponding to the dual LP can be obtained by imposing the constraint that $y_i \in \{0, 1\}$. This is exactly the same IP as that for the minimum vertex cover problem discussed earlier. Looking at the relationship between the different integer and linear programs, we get:

Max. matching = IP $\leq$ Relaxed LP $\leq$ Dual LP $\leq$ IP of dual LP = Vertex cover.

We have already shown that for the maximal weight matching problem, *Optimum*(IP) = *Optimum*(LP), so the first inequality above holds as an equality. From LP duality theorem we also know that *Optimum*(LP) = *Optimum*(Dual LP), which converts the second inequality into an equality. Since A is TUM, A^t (appearing in the dual LP) is also TUM because total unimodularity is invariant to transposition. Therefore, *Optimum*(Dual LP) = *Optimum*(IP of dual LP), so the last inequality is also an equality. In other words, for a bipartite graph, the size of the maximal matching is the same as the size of the minimum vertex cover. So, by using LP duality and relaxation, we were able to link together two different problems in graph theory.

References

[GLS93] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, second corrected edition edition, 1993.