6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms October 22, 2008

Lecture 13

Lecturer: Michel X. Goemans

In the last lecture, we discussed the details of the ellipsoid algorithm and described how to apply the ellipsoid algorithm to a linear program. In this lecture, we will see two applications of the ellipsoid method: linear programming without an explicit linear program, and semidefinite programming.

1 Undirected s–t Shortest Path Problem

We are given an undirected graph $G = (V, E)$ with $s, t \in V$ and cost function $c : E \to \mathbb{R}$. We assume that there is no negative cost cycle in G. In the s–t shortest path problem, we find a simple s–t path P minimizing

$$c(P) = \sum_{e \in P} c_e$$

We will apply the ellipsoid method using the following steps:

1. Modify the shortest path problem to be a minimum cost perfect matching problem.
2. Apply the ellipsoid algorithm to the minimum cost perfect matching problem.

1.1 Minimum Cost Perfect Matching Problem

We introduce the following definitions.

Definition 1 *A matching M on a graph H is a set of edges with no common endpoint.*

Definition 2 *Matching M is perfect iff $|M| = \frac{|V|}{2}$.*

The original graph $G = (V, E)$ is transformed into an auxiliary graph $H = (V', E \cup F)$. We would like the $s - t$ shortest path problem in G to be equivalent to the minimum cost perfect matching problem in H. The transformation is as follows. For each $v \in V$, one of the following cases holds.

1. $v \notin$ {s,t} and $deg(v)$ even (where $deg(v)$ is the degree of vertex v),
2. $v \in$ {s,t} and $deg(v)$ odd,
3. $v \notin$ {s,t} and $deg(v)$ odd,
4. $v \in$ {s,t} and $deg(v)$ even.

- For cases 1 and 2, graph H contains $deg(v)$ vertices corresponding to $v \in G$. Edges are added so that these vertices are fully connected (they form a *clique*). The added edges are in F, and $c_e = 0 \ \forall e \in F$.
- For cases 3 and 4, graph H contains $deg(v) + 1$ vertices corresponding to $v \in G$. Edges are added so that these vertices form a clique; they are fully connected. The added edges are in F, and $c_e = 0 \ \forall e \in F$.

So far, H contains a set of disjoint cliques, one for each vertex $v \in V$. To each original edge $e = (u, v) \in E$, we create a corresponding edge between one of the copies of u and one of the copies of v. This is done in such a way that these edges in H (which we will denote by E as they correspond to the original edges) form a matching. In other words, for every vertex v, the $deg(v)$ edges in G incident to v become now incident to distinct copies of v in H. These $|E|$ additional edges keep the cost they have in G.

Now we will show that the minimum cost perfect matching problem on H is equivalent to the shortest s–t path problem on G, whenever G does not have any negative cost cycle. First consider the case where we have a simple path P from s to t in G of cost $c(P)$. In H, these edges in P form a (not perfect) matching. We add edges of F to P to make a perfect matching: we show that $\exists N \subseteq F$ such that $M = P \cup N$. Indeed, by construction, the edges of P cover in H all but an even number of vertices corresponding to $v \in V$ (and thus these remaining vertices can be matched with edges of F of 0 cost), and this is true for any vertex $v \in V$:

1. For case 1, P uses 0 or 2 edges incident to v in G, and therefore $deg(v)$ or $deg(v) - 2$ copies of v (an even number) are unmatched in H,
2. for case 2, P uses precisely 1 edge incident to v in G, and therefore $deg(v) - 1$ (an even number) vertices remain unmatched,
3. cases 3 and 4 are similar.

By completing P into a perfect matching $P \cup N$, we get a perfect matching whose cost equals the cost of the path P, $c(P)$.

Now we show that this holds if we go in the opposite direction. Start with a perfect matching $M \subseteq E \cup F$ in H of cost $c(M)$ and consider the set of edges $J = M \cap E$ in G (one could view J as obtained from M by shrinking all cliques of F). Observe that by construction J is an s–t join:

Definition 3 *A set $J \subseteq E$ of edges is an s–t join iff, for each vertex v*

$$\left\{ \begin{array}{c} v \notin \{s,t\} \rightarrow deg_J(v) \text{ even} \\ v \in \{s,t\} \rightarrow deg_J(v) \text{ odd} \end{array} \right\}$$

We claim that from this s–t join we can derive an s–t path in G of no greater cost.

Lemma 1 *If there is no negative cost cycle in G then, for any s–t join J, there exists an s–t path P of no greater cost.*

Proof of Lemma 1: Take optimum s–t join J. Remove as many cycles as possible until J is acyclic; let C_i be these edge-disjoint cycles. Let P denote what remains:

$$J = (\cup_{i=1}^{k} C_i) \cup P.$$

P is thus an acyclic graph with odd degree at s and t, and even degree everywhere else. Therefore, P must be an s–t path (if there was a vertex of degree at least 3 in this acyclic graph, there would be at least 3 leaves (each of odd degree), and that would be a contradiction.) Furthermore,

$$c(J) = \sum c(C_i) + c(P) \geq c(P)$$

since all cycles are supposed to be non-negative cost. □

This means that by solving the minimum cost s–t join in H (and removing any 0 cost cycles in it), we obtain a minimum cost t–t path in G.

1.2 Applying the Ellipsoid Method

We apply the ellipsoid method to the problem of finding a perfect matching M of minimum total cost given a graph H and cost function $c : E \to \mathbb{R}$.

The first step is to formulate the problem as an integer program, i.e. a linear program in which variables are restricted to take integer values. To every matching M, we associate a vector $x \in \mathbb{R}^{|E|}$:

$$M \to x_e = \begin{cases} 1 & \text{for} \quad e \in M \\ 0 & \text{for} \quad \textit{otherwise} \end{cases}$$

The matching problem then becomes the following integer program:

$$\begin{aligned} \min \quad & \sum_e c_e x_e \\ s.t. \quad & x_e \in \{0,1\} \qquad \forall e \in E \\ & \sum_{e \in \delta(v)} x_e = 1 \quad \forall v \in V \end{aligned}$$

This is not a valid linear program due to the integrality constraint on x_e. We can attempt to relax this constraint, replacing $x_e \in \{0,1\}$ with $0 \leq x_e \leq 1$, to obtain a valid linear program. If this would result in a linear program in which every vertex x had only coordinates taking values 0 or 1, we would be done since we are guaranteed that the optimum of a linear program is at a vertex. However, in this case, there are vertices of this linear program which are not integer valued, and also this linear program might have a non-empty feasible region while there are no perfect matchings. Consider indeed a cycle of length 3 (thus $|V| = 3$). Clearly, there are no perfect matching but the feasible region of this linear program is non-empty since we can let $x_e = 1$ for every edge e (this is a vertex of the corresponding linear program).

To be able to use linar programming, we need to add more constraints to guarantee that the linear program has all its vertices corresponding to matchings. This can be done and is an important result due to Edmonds.

Theorem 2 (Edmonds) *All vertices of*

$$\begin{cases} \sum_{e \in \delta(v)} x_e = 1 & \forall v \in V \\ \sum_{e \in (S:\bar{S})} x_e >= 1 & \forall S \subseteq V \textit{ with } |S| \textit{ odd} \\ 0 \leq x_e \leq 1 & \forall e \in E \end{cases}$$

are incidence vectors of perfect matchings, and therefore a minimum cost perfect matching can be obtained by minimizing $c^T x$ over the above constraints.

(Recall that the cut $(S : \bar{S})$ denotes the set of edges with exactly one endpoint in S, and similarly $\delta(v) = (\{v\} : V \setminus \{v\})$. Notice that the second set of constraints are indeed satisfied by the incidence vectors of all perfect matchings. If $|S|$ is odd, no perfect matching can match the vertices together, and thus at least one of them has to be matched to a vertex outside S, leading to the validity of these constraints.

This linear program seems completely impractical because the number of constraints is

$$|V| + |E| + \frac{1}{2} 2^{|V|}.$$

This is exponential in the size of the input. However, we can use the ellipsoid method, and avoid having to explicitly list all constraints. Indeed, for the ellipsoid to work to optimize over a convex set $K \subseteq \mathbb{R}^n$, we only need to have a separation oracle for K. This is an algorithm which, given $a \in \mathbb{R}^n$,

- either claim $a \in K$,
- or output $c : \{x : c^T x < c^T a\} \supseteq K$.

If we have a separation oracle then we can easily extend the ellipsoid algorithm from finding a point in K to minimizing $c^T x$ over K. Indeed every time a point a in K is found, we can update the best point found so far and add the inequality $c^T x \leq c^T a$ and proceed with the ellipsoid algorithm. Formally, one would need to argue about the choice of R and r for the starting ellipsoid and for deciding when to stop, but this is omitted here. Since the ellipsoid algorithm makes a polynomial number of calls to the separation oracle, we can derive a polynomial-time algorithm for optimizing over K provided that the separation oracle itself can be implemented in polynomial-time.

For the linear inequality description given in Theorem 2, checking if a vector $a \in \mathbb{R}^n$ satisfies the first set of constraints is clearly polynomial (just check each of them individually), but for the second set of constraints, the separation problem can be solved by solving the *minimum odd cut problem*: Given a, find

$$\lambda = \min_{S \subseteq V : |S| odd} \sum_{(S:\bar{S})} a_e.$$

Indeed, if $\lambda \geq 1$ then a satisfies all inequalities $\sum_{(S:\bar{S})} x_e \geq 1$. Otherwise, if $\lambda < 1$, then we have found a set S for which the inequality $\sum_{(S:\bar{S})} x_e \geq 1$ is violated by a, and we can return this inequality. In the problem set, we show how the minimum odd cut problem can be solved as a sequence of a polynomial number of minimum s–t cut problems. Through the ellipsoid, this gives a polynomial-time algorithm for solving the minimum cost perfect matching problem. There is also a (much more combinatorial) polynomial-time algorithm due to Edmonds for matching that does not involve the ellipsoid algorithm, but this is a nice illustration of the power of the approach through a separation oracle.

2 Semidefinite Programming

Yet another application of the ellipsoid algorithm is in the solution of semidefinite programming problems. Semidefinite programming is a type of convex optimization in which a linear objective function is optimized over the intersection of positive semidefinite matrices with an affine space. It is more general class of problems than linear programming.

Let's denote by $\mathbb{S}^n$ the class of $n \times n$ real, symmetric matrices.

Definition 4 *For $A \in \mathbb{S}^n$ we say that A is positive definite if we have $x^T Ax > 0 \ \forall x \in \mathbb{R}^n, x \neq 0$ and that it is positive semidefinite if $x^T Ax \geq 0 \ \forall x \in \mathbb{R}^n$.*

If A is positive definite, we write that $A \succ 0$; if it is positive semidefinite, we write that $A \succeq 0$. Remember that all eigenvalues of a real symmetric matrix are real. An equivalent definition for positive definiteness is $A \succ 0$ iff $\lambda_i > 0 \ \forall$ eigenvalues λ_i of A, and that $A \succeq 0$ iff $\lambda_i >= 0$. We define the PSD cone as: PSD $= \{A \in \mathbb{S}^n : A \succeq 0\}$.

Lemma 3 *PSD is a convex cone.*

Proof of Lemma 3:

$$\forall x : x^T(\lambda A + (1-\lambda)B)x = \lambda x^T Ax + (1-\lambda)x^T Bx \geq 0.$$

We can see that the first term on the right-hand side of this equality, $\lambda x^T Ax$, must be nonnegative by our definition of positive semidefiniteness. So too must the second term (since we have $0 \leq \lambda \leq 1$); therefore, their sum must also be nonnegative. □

We need a notion of an inner product over symmetric matrices. The Frobenius inner product, $A \bullet B$, of two matrices $A, B \in \mathbb{S}^n$ is defined as

$$A \bullet B = \sum_i \sum_j A_{ij} B_{ij} = \text{Tr}(A^T B) = \text{Tr}(AB^T),$$

where Tr denotes trace. The Frobenius inner product is the component-wise inner product of the two matrices as though they were vectors.

In a semidefinite programming problem, we are minimizing over $X \in \mathbb{S}^n$; as such matrices are symmetric, there are $n(n+1)/2$ unknown and we could view this as optimizing over $\mathbb{R}^{n(n+1)/2}$. The primal form of a semidefinite programming problem is as follows. Given $C \in \mathbb{S}^n$, $A_i \in \mathbb{S}^n$, and $b_i \in \mathbb{R}$ for $i \in \{1, \cdots, m\}$, we are attempting to

$$\min C \bullet X$$

subject to the constraints

$$A_i \bullet X = b_i$$

and

$$X \succeq 0.$$

Just as with linear programming, a dual semidefinite program also exists; this will be discussed in greater detail in the next lecture.

It is important to note that the (unique) optimum solution may be irrational, so it is not clear how to even concisely output the solution. Consider, for example, the problem of

$$\min \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \bullet X$$

subject to

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \bullet X = 1$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \bullet X = 5,$$

and $X \succeq 0$. Clearly, the optimum matrix is

$$X = \begin{pmatrix} 1 & -\sqrt{5} \\ -\sqrt{5} & 5 \end{pmatrix},$$

with an irrational objective function value.

Given A_i and b_i, the semidefinite program is feasible iff $\exists X \succeq 0 : A_i \bullet X = b_i$. In fact, the problem of determining whether a semidefinite program is feasible is an open question and is not known to be in NP. As a special case, consider the question of whether, given $a_1, ..a_n, b \in \mathbb{N}$, $\sum_{i=1}^n \sqrt{a_i} < b$. This can be formulated as the feasibility question of a semidefinite program. It is easy to evaluate $\sum_{i=1}^n \sqrt{a_i}$ but it is unclear how many bits or decimal places of accuracy are sufficient to determine whether it is or not less than b. The complexity of this question is open.

With the ellipsoid algorithm, one can in polynomial time find an almost feasible solution X, which is also almost optimum (up to some ϵ). Indeed, the separation problem over

$$\{X \in \mathbb{S}^n : A_i \bullet X = b_i \text{ for } i \in \{1, \cdots, m\}, X \succeq 0\},$$

can be solved efficiently. Indeed, the linear constraints are easy to check individually while checking whether $X \succeq 0$, i.e. $X \in PSD$, can be done by a Cholevsky decomposition. In $O(n^3)$ time, this either proves that $X \succeq 0$ or provides a vector $a \in \mathbb{R}^n$, $a \neq 0$ such that $a^T X a < 0$. This means that we have found an inequality violated by our current matrix X (namely the (linear in X) inequality $a^T X a \geq 0$) and we can use it to cut our ellipsoid in half.