PRINCETON UNIV. F'23 COS 521: ADVANCED ALGORITHM DESIGN

Lecture 6: Semidefinite Programming

Lecturer: *Huacheng Yu*

Recall that a set of points K is *convex* if for every two $x, y \in K$ the line joining x, y, i.e., $\{\lambda x + (1-\lambda)y : \lambda \in [0,1]\}$ lies entirely inside K. A function $f : \Re^n \to \Re$ is *convex* if $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$ for all $x, y \in \Re^n$ and $\lambda \in [0,1]$. It is called *concave* if the previous inequality goes the other way. A linear function is both convex and concave. A *convex program* consists of a convex function f and a convex body K and the goal is to minimize $f(x)$ subject to $x \in K$. It is a vast generalization of linear programming and like LP, can be solved in polynomial time under fairly general conditions on f, K. Today's lecture is about a special type of convex program called *semidefinite programs.*

1 Positive Semidefinite matrices

Recall that a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is *positive semidefinite* (PSD) if

$$x^T A x \geq 0 \qquad \text{for all} \qquad x \in \mathbb{R}^n.$$

This property is equivalent to:

1. A has all non-negative eigenvalues.
2. A can be written as $A = U^T U$ for some $U \in \mathbb{R}^{n\times n}$, i.e., $A_{ij} = u_i^T u_j$ where u_i is the i^{th} column of U.

To denote that a matrix is PSD, we write $A \succeq 0$. $A \succeq B$ indicates that $A - B$ is PSD, or equivalently that $x^T A x \geq x^T B x$ for all $x \in \mathbb{R}^n$. The symbols $\succeq$ and $\preceq$ can be used to define an ordering on matrices, which is called the "Loewner ordering". It's a partial order: it's impossible for both $A \succeq B$ and $B \succeq A$ to hold for $A \neq B$, it could be that neither does.

Exercise 1. *Come up with a simple example where $A \not\succeq B$ and $B \not\succeq A$.*

The Loewner ordering has many useful properties. For example, $A \succeq B$ implies that $A^{-1} \preceq B^{-1}$. $A \succeq B$ also implies, that for all i, $\sigma_i(A) \geq \sigma_i(B)$, where σ_i denotes the i^{th} singular values (which is the same as the i^{th} eigenvalue for PSD matrices).[1]

You have to be careful though. For example, $A \succeq B \not\Rightarrow A^2 \succeq B^2$.

PSD matrices appear all the time in algorithmic applications, including some that we have already seen. Graph Laplacians, Hessians of convex functions, covariance matrices, and many other natural matrices are always PSD. As we will see today, PSD matrices are also very useful in formulating optimization problems.

[1] The opposite statement is not true – it can be that $\sigma_i(A) \geq \sigma_i(B)$ for all i, but $A \not\succeq B$.

2 Semidefinite programming

The goal of semidefinite programming is to solve optimization problems where the input is a matrix that is constrained to be PSD. I.e. we optimize over $X \in \mathbb{R}^{n \times n}$ where $X \in \mathcal{K}$ and:

$$\mathcal{K} = \{M \mid M \succeq 0\}.$$

$\mathcal{K}$ is a convex set: if $X \succeq 0$ and $Y \succeq 0$ are PSD then for all $\lambda \in [0,1]$, it's easy to see that $\lambda X + (1-\lambda)Y \succeq 0$ with the right definition:

Lemma 1. *The set of all $n \times n$ PSD matrices is a convex set in $\Re^{n^2}$.*

Proof. The property that $u^T A u \geq 0$ for all u is the easiest to verify:[2] Note that $u^T(\lambda M_1 + (1-\lambda)M_2)u = \lambda u^T M_1 u + (1-\lambda)u^T M_2 u \geq 0 + 0 = 0$. Therefore, $\lambda M_1 + (1-\lambda)M_2$ is PSD as well. □

This realization leads to the following convex optimization problem:

Problem 2 (Semidefinite program – SDP)**.** *Let f be a convex function and let $\langle M, N \rangle$ denote $\sum_{i,j} M_{ij} N_{ij}$. We seek to find $X \in \mathbb{R}^{n \times n}$ which solves:*

$$\begin{aligned} &\min f(X) \text{ such that:} \\ &X \succeq 0, \\ \text{for } i = 1, \ldots, k,\ &\langle A_i, X \rangle \geq b_i. \end{aligned}$$

Here $A_1, \ldots, A_k$ and $b_1, \ldots, b_k$ are input constraints. It is very common to have:

$$f(X) = \langle C, X \rangle$$

for some C. I.e. to have our objective be a linear function in X.

Problem 2 is optimizing over a convex set, since the convex PSD constraint intersected with k linear constraints forms a convex set. It can be viewed as a Linear Program with an infinite number of constraints. Specifically, our constraints are equivalent to:

$$\begin{aligned} &\min f(X) \text{ such that:} \\ \forall v \in \mathbb{R}^n\ &\left\langle vv^T, X \right\rangle \geq 0, \\ \text{for } i = 1, \ldots, k,\ &\langle A_i, X \rangle \geq b_i. \end{aligned}$$

Note that the convex objective can be replaced by $\min T$ subject to $f(X) \leq T$ and other constraints. It is not hard to verify by definition that the constraint $f(X) - T \leq 0$ induces a convex set for convex f. Thus, this is still a convex program.

The PSD constraint gives a compact way of encoding these infinite linear constraints. in this sense, SDPs are strictly stronger than linear programs.

[2] This is also a good instance of why characterization theorems are helpful: one definition happened to be trivial for this proof, whereas the others take more work.

Exercise 2. *Show that every LP can be written as an SDP. The idea is that a diagonal matrix, i.e., with off-diagonal entries are* 0*, is PSD if and only if its entries are non-negative.*

Semidefinite programs can be solved (relatively) efficiently with a variety of methods, including the ellipsoid method and specially designed interior point methods. They model a wide range of natural problems, several examples of which are outlined in [1]. One example problem is as follows:

Example 1 (Minimum Volume Ellipsoid via an SDP)**.** *Suppose we have points* $v_1, \ldots, v_k \in \mathbb{R}^n$ *and we want to find the smallest (specifically, minimum volume) ellipsoid* E *centered at* 0 *that contains these points. This problem can be formulated as a semidefinite program.*

Recall from our lecture on the Ellipsoid Method that any ellipsoid E centered at 0 can be parameterized by a PSD matrix $X \in \mathbb{R}^{n \times n}$, where a point y lies inside E if and only if:

$$y^\top X y \leq 1.$$

Also note that E's volume is proportional to $\sqrt{\det(X^{-1})} = \sqrt{\prod_{i=1}^{n} 1/\sigma_i(X)}$. With some work, it's possible to verify that $\log(\det(X^{-1}))) = -\log(\det(X))$ is a convex function in X. So to solve the minimum volume ellipsoid problem we can solve:

$$\begin{aligned} \min_X \ & -\log(\det(X)) \text{ such that:} \\ & X \succeq 0, \\ \text{for } i = 1, \ldots, k, \ & y^T X y \leq 1. \end{aligned}$$

Note that all constraints $y^T X y \leq 1$ are linear constraints on X.

2.1 Alternative View of SDP

Since any PSD matrix X can be written as $V^T V$ where $V = [v_1, \ldots, v_n]$ is a matrix in $\mathbb{R}^{n \times n}$, we can equivalently formulate the SDP problem as solving:

$$\min f(V^T V) \qquad \text{subject to} \qquad \left\langle A_i, V^T V \right\rangle \geq b_i. \tag{1}$$

This is sometimes a more useful view as one can think of the column vectors $\{v_i\}_i$ of V as the "actual variables". In this case, each entry $X_{ij} = \langle v_i, v_j \rangle$ is an inner product, and the objective is simply some convex function on the pairwise inner products, and the constraints are all linear constraints on them.

3 Maximum Cut

Just as we saw for linear programs, SDPs can be very useful in obtaining approximation algorithms for combinatorial optimization problems. In fact, it's possible to use the same "relax-and-round" framework that we saw for linear programs. Semidefinite programs allow for a richer variety of relaxation and rounding techniques.

One classic example of a combinatorial problem that can be approximated using a algorithm based on semidefinite programming is the maximum cut problem:

Problem 3 (Maximum cut)**.** *Give an undirected, unweighted graph $G = (V, E)$ with $|V| = n$, find $S \subset V$ such that $|E(S, V \setminus S)|$ is maximized. $|E(S, V \setminus S)|$ denotes the number of edges between nodes in S and nodes not in S – i.e. the size of the cut between S and $V \setminus S$.*

Denote the optimal value for this problem by $OPT_{MC} = \max_S |E(S, V \setminus S)|$.

This problem can be formulated as an integer optimization problem:

$$\max_{u_1,\ldots,u_n \in \{-1,1\}} \sum_{(i,j)\in E} \frac{1}{4}|u_i - u_j|^2. \tag{2}$$

If we set $u_i = 1$ for all $i \in S$ and -1 otherwise, then this objective function exactly captures the size of the cut between S and $V \setminus S$: $|u_i - u_j|^2 = 0$ if i, j are on the same side of the cut and $|u_i - u_j|^2 = 4$ if they're on different sides.

Unfortunately solving (2) is NP-hard. It's possible to solve approximately using a greedy algorithm or LP relaxation, but both obtain objective values of just $\frac{1}{2}OPT_{MC}$.

Our main result today is that the maximum cut problem can be approximated to much better accuracy using an algorithm based on semidefinite program:

Theorem 4 (Goemans, Williamson '94 [2])**.** *There is a randomized SDP rounding scheme that finds a cut with expected size $\geq .878 \cdot OPT_{MC}$.*

3.1 SDP Relaxation

The Goemans and Williamson approach relaxes binary variables to *continuous vectors*:

$$u_i \in \{-1, 1\} \Longrightarrow v_i \in \mathbb{R}^n \qquad \text{with} \qquad \|v_i\|_2 = 1, \quad \forall i$$

Specifically, they solve:

Problem 5 (Relaxed Maximum Cut)**.**

$$\max_{v_1,\ldots,v_n,\ \|v_i\|_2=1\ \forall i} \sum_{(i,j)\in E} \frac{1}{4}\|v_i - v_j\|_2^2. \tag{3}$$

This problem can be solved as a semidefinite program and we denote its optimal value by OPT_{SDP}.

To check that Problem 5 can be solved as an SDP, we refer to the formulation of (1). The constraint that $\|v_i\|_2 = 1$ is simply a constraint that all diagonal entries of $X = V^T V$ are 1, which can be encoded as a linear constraint. Additionally, since $\|v_i - v_j\|_2^2 = v_i^T v_i + v_j^T v_j - 2v_i^T v_j$, our objective function (3) can be written as $\langle C, X\rangle$ for some C.

To repeat the above reasoning, recall that X_{ij}, for $i, j \in [n]$ denote our n^2 variables. We have (infinitely many) linear constraints on X that ensures X is PSD, and therefore $X = V^T V$. Moreover, observe that $X_{ij} = v_i^T v_j$. Therefore, our objective function is $\sum_{(i,j)\in E}(X_{ii} + X_{jj} - 2X_{ij})/4$, and this is a convex program.

Intuitively, Problem 5 seeks to arrange vectors on the unit circle in such away that vectors corresponding to connected nodes i, j are placed as far as possible from each other.

Problem 5 is a valid relaxation of Problem 3. In particular, we have:

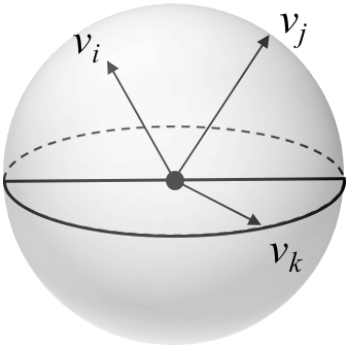

Figure 1: SDP solutions are unit vectors which are arranged so that vectors v_i and v_j are far apart when nodes i and j are connected with an edge in G.

Claim 6.

$$OPT_{MC} \leq OPT_{SDP}.$$

Proof. Given a solution $u_1, \ldots, u_n$ to Problem 3 we simply set $v_i = u_i \cdot e_1$, where $e_1 = [1, 0, \ldots, 0]^T$ is a standard basis vector. Then (3) exactly equals (2). □

3.2 Random Hyperplane Rounding

To obtain a solution to Problem 3 from an optimal solution to Problem 5 we employ the following rounding strategy:

1. Solve the semidefinite program in Problem 5 to obtain vectors $v_1, \ldots, v_n$.
2. Choose a random vector $c \in \mathbb{R}^n$ by choosing each entry to be an independent standard Gaussian random variable.
3. Set $\tilde{u}_i = \text{sign}\left(c^T v_i\right)$.

Claim 7.

$$\mathbb{E}\left[\sum_{(i,j)\in E} \frac{1}{4}|\tilde{u}_i - \tilde{u}_j|^2\right] \geq .878 \cdot \sum_{(i,j)\in E} \frac{1}{4}\|v_i - v_j\|_2^2$$

It follows that our rounded solution obtains an expected cut value $\geq .878 \cdot OPT_{SDP}$, which is $\geq .878 \cdot OPT_{MC}$ by Claim 6. Applying Markov's inequality, a few repeated trials ensures that we obtain a good approximate max cut with high probability.

Proof. Since c is *spherically symmetric* our rounding strategy corresponds to choosing a random n dimensional hyperplane through the origin. For all vectors v_i placed on one side of the hyperplane, node i belongs to S. The nodes corresponding to all vectors on the other side of the hyperplace belong to $V \setminus S$. This approach is known as **random hyperplane rounding**. It is visualized in Figure 2.

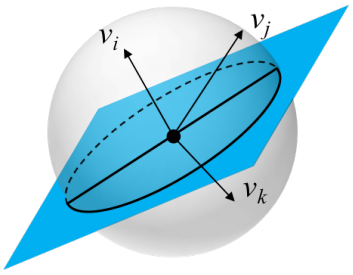

Figure 2: Our SDP solution is rounded by choosing a random hyperplane through the origin and assigning nodes to each side of the cut based on what side of the hyperplane their corresponding vector lies on. In this case, nodes i and j are placed on one side of the cut, with node k placed on the other side. In other words, $\tilde{u}_i = \tilde{u}_j = -\tilde{u}_k$.

Intuitively, since vectors corresponding to connected nodes are in general placed as far apart as possible by the SDP, it is more likely that the a random hyperplane separates connected nodes, and thus that we obtain a large cut value.

Formally, we bound the expected number of edges cut in our solution $\tilde{u}_1, \ldots, \tilde{u}_n$. Let θ_{ij} denote the angle (in radians) between vectors v_i and v_j. What is the probability that nodes i and j end up on different sides of the cut after random hyperplane rounding? This may seem a difficult n-dimensional calculation, until we realize that there is a 2-dimensional subspace defined by v_i, v_j, and all that matters is the intercept of the random hyperplane with this 2-dimensional subspace, which is a random line in this subspace.

In particular, observe that there is a two-dimensional space spanned by v_i, v_j. Observe also that picking a uniformly random hyperplane through the origin corresponds to picking a uniformly random vector in the unit sphere, and taking the hyperplane orthogonal to it. When we project a uniformly random vector in the unit sphere to this two-dimensional space, we also get a uniformly random directly in this two-dimensional space, and therefore projecting the uniformly random hyperplane into this two-dimensional space is also uniformly random. So, we just get a uniformly random line through the origin in this space, and the probability that it lies between v_i and v_j is exactly $\frac{\theta_{ij}}{\pi}$. Thus by linearity of expectations,

$$\mathbb{E}[\text{Number of edges in cut defined by } \tilde{u}_1, \ldots, \tilde{u}_n\ ] = \sum_{\{i,j\} \in E} \frac{\theta_{ij}}{\pi}. \tag{4}$$

How do we relate this to OPT_{SDP}? We use the fact that $\langle v_i, v_j \rangle = \cos\theta_{ij}$ to rewrite the SDP objective as:

$$OPT_{SDP} = \sum_{\{i,j\} \in E} \frac{1}{4}\|v_i - v_j\|^2 = \sum_{\{i,j\} \in E} \frac{1}{4}(\|v_i\|^2 + \|v_j\|^2 - 2\langle v_i, v_j \rangle) = \sum_{\{i,j\} \in E} \frac{1}{2}(1 - \cos\theta_{ij}). \tag{5}$$

To compare this objective function to (4) Goemans and Williamson observed that:

$$\frac{\theta/\pi}{\frac{1}{2}(1-\cos\theta)} = \frac{2\theta}{\pi(1-\cos\theta)} \geq 0.87856\ldots \qquad \forall \theta \in [0, \pi].$$

This is easy to verify by plotting e.g. in MATLAB.

It follows that the expected size of our cut $\geq 0.878 \cdot OPT_{SDP} \geq 0.878 \cdot OPT_{MC}$. □

The saga of 0.878**...** The GW paper came on the heels of the PCP Theorem (1992) which established that there is a constant $\epsilon > 0$ such that $(1 - \epsilon)$-approximation to MAX-CUT is NP-hard. In the ensuing few years this constant was improved. Meanwhile, most researchers hoped that the GW algorithm could not be optimal. The most trivial relaxation, the most trivial rounding, and an approximation ratio derived by MATLAB calculation: it all just didn't smell right. However, in 2005 Khot et al. showed that Khot's unique games conjecture implies that the GW algorithm cannot be improved by any polynomial-time algorithm. (Aside: not all experts believe the unique games conjecture.)

References

[1] Vandenberghe, Lieven, and Stephen Boyd. Applications of semidefinite programming. Applied Numerical Mathematics 29.3 (1999): 283-300.

[2] Goemans, Michel X., and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM (JACM) 42.6 (1995): 1115-1145.