Lecture 5: Semidefinite Programming

Lecturer: *Sahil Singla* Last updated: January 25, 2022

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 $f(\frac{x+y}{2}) \leq \frac{1}{2}(f(x) + f(y))$. 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. $A_{ij} = u_j^T u_i$ 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. Observe that $u^T(M_1/2 + M_2/2)u = u^T M_1 u/2 + u^T M_2 u/2 \geq 0 + 0 = 0$. Therefore, $(M_1 + M_2)/2$ 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 view as a Linear Program with an infinite number of constraints. Specifically, our constraints are equivalent to:

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

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

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 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 can be parameterized by a PSD matrix $X \in \mathbb{R}^{n \times n}$ and center $c \in \mathbb{R}^n$, where a point y lies inside E if and only if[2]:

$$\|Xy - c\|_2 \leq 1.$$

Also note that E's volume is proportional to $\det(X^{-1}) = \prod_{i=1}^{n} 1/\sigma_i(X)$. 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:

$$\begin{aligned} \min_{X,c} \; & -\log(\det(X)) \text{ such that:} \\ & X \succeq 0, \\ \text{for } i = 1, \ldots, k, \; & \|Xv_i - c\|_2^2 \leq 1. \end{aligned}$$

To check that the k constraints involving $v_1, \ldots, v_k$ can be written as PSD constraints, note that (exercise) for any vector z the constraint $\|z\|_2^2 \leq 1$ can be equivalently written as

$$\begin{bmatrix} I & z \\ z^T & 1 \end{bmatrix} \succeq 0.$$

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:

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

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:

[2]This ellipsoid definition might seem different from last class' definition $(x-a)^T B(x-a) \leq 1$, but both can be shown to be equivalent using the fact that for any PSD matrix B (i.e., symmetric with non-negative eigenvalues) there is a unique PSD matrix X (often called the square root) that satisfies $B = XX$.

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.

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:

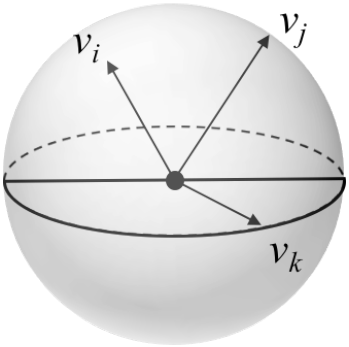

Figure 1: SDP solutions are unit vectors which are arranged so that vectors v_i and v_j are far apart when nodes i and j are connected with an edge in G.

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, 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.

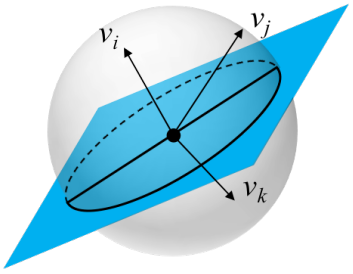

Figure 2: Our SDP solution is rounded by choosing a random hyperplane through the origin and assigning nodes to each side of the cut based on what side of the hyperplane their corresponding vector lies on. In this case, nodes i and j are placed on one side of the cut, with node k placed on the other 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.

So this probability is exactly equal to $\frac{\theta_{ij}}{\pi}$. 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.)

4 0.878-approximation for MAX-2SAT

We earlier designed approximation algorithms for MAX-2SAT using LP. The SDP relaxation gives much tighter approximation than the 3/4 we achieved back then. Given a 2CNF formula on n variables with m clauses, we can express MAX-2SAT as a quadratic optimization problem (note that this will be a different phrasing than the integer program we wrote in the LP rounding lecture). Below, the symbol y_ℓ^1 is a placeholder for the variable x_i if the first literal in clause ℓ is x_i, and a placeholder for $-x_i$ if the first literal is $\bar{x}_i$.

$$\text{Maximize} \sum_\ell 1 - (1 - y_\ell^1) \cdot (1 - y_\ell^2)/4$$
$$x_i^2 = 1, \ \forall i$$

Indeed, observe that when the clause is satisfied, one of the literal variables is 1, so the second term is 0 (and we get $1 - 0$). Otherwise, it both literal variables are -1 (and we get $-2 \cdot (-2)/4 = 1$, and then $1 - 1 = 0$). Observe also that the constraint $x_i^2 = 1$ ensures that $x_i \in \{\pm 1\}$. Observe also that these constraints *almost* look like we can just replace the variables with vectors and have an SDP. But this is tricky because of the "1" (e.g. a term like $1 - (1 - x_i)(1 - x_j)$ — what are we supposed to do with $1 \cdot x_i$?). Instead we are going to look for $n + 1$ vectors $u_0, u_1, \ldots, u_n$. The first vector u_0 is a dummy vector that stands for "1." If $u_i = u_0$ then we think of this variable being set to True and if $u_i = -u_0$ we think of the variable being set to False. Of course, in general $\langle u_i, u_0 \rangle$ need not be ± 1 in the optimum solution. So now, we update our notation so that instead of $(1 - x_i)$, we write $u_0 - u_i$ (the rest is the same, replacing x_i with u_i everywhere).

$$\text{Maximize} \sum_\ell 1 - (u_0 - y_\ell^1) \cdot (u_0 - y_\ell^2)/4$$
$$\langle u_i, u_i \rangle = 1, \ \forall i$$

For instance if the clause is $y_i \vee y_j$ then the expression is

$$1 - \frac{1}{4}(u_0 - u_i) \cdot (u_0 - u_j) = \frac{1}{4}(1 + u_0 \cdot u_j) + \frac{1}{4}(1 + u_0 \cdot u_i) + \frac{1}{4}(1 - u_i \cdot u_j).$$

This is a very Goemans-Williamson like expression, except we have expressions like $1 + u_0 \cdot u_i$ whereas in MAX-CUT we have $1 - u_i \cdot u_j$. We do a similar randomized rounding:

- Pick a random unit-vector v.
- Set $x_i := \text{sign}(\langle u_i, v\rangle \cdot \langle u_0, v\rangle)$. That is, set x_i to 1 if u_i is on the same side as u_0 of the hyperplane orthogonal to v, and -1 if not.

The key insight is that since we round to ± 1, each term $1 - u_i \cdot u_j$ (just like in GW for Max-Cut) is 2 with probability $\frac{\theta_{ij}}{\pi}$ and is 0 otherwise. Similarly, $1 + u_0 \cdot u_i$ becomes 2 with probability $1 - \theta_{ij}/\pi = \frac{\pi - \theta_{ij}}{\pi}$ and 0 else. So we care about the minimum ratio between either $\frac{2\theta}{\pi(1-\cos(\theta_{ij})}$, or $\frac{2(\pi-\theta)}{\pi(1+\cos(\theta))}$ over $\theta \in [0, \pi]$.

Observe that if we set $\gamma := \pi - \theta$, then $\cos(\gamma) = \cos(\pi - \theta) = -\cos(-\theta) = -\cos(\theta)$. Therefore, $\frac{2(\pi-\theta)}{\pi(1+\cos(\theta))} = \frac{2\gamma}{\pi(1-\cos(\gamma))}$, and is also at least 0.878. We conclude that the expected number of satisfied clauses is at least 0.878 times OPT_{SDP}.

Notes

This lecture is based on course notes from COS 521 at Princeton University.

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.