PRINCETON COS 521: ADVANCED ALGORITHM DESIGN

Semidefinite Programming and Approximating the Maximum Cut

Recall the Maximum cut problem: given a graph, find a subset of vertices S such that the number of edges going across it, $|E(S, \overline{S})|$ is maximized.

Problem 1 (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 is NP-hard; thus, unless $P = NP$, we cannot hope to solve it exactly in polynomial time. You may recall a simple algorithm that achieves a $1/2$ factor approximation: return a uniformly random cut. Until the 1990s, despite decades of effort, there was no better algorithm. In particular, the major workhorse of approximation algorithm design, *linear programming relaxations*, seemed ineffective: every relaxation researchers came up with had an integrality gap of 2 (i.e., could not beat the trivial algorithm). The reason for this was found a little later [1] and led to a theorem that (appropriately formalized) says there cannot be sub-exponential size linear programs for Max-Cut with integrality gap < 2.

[1]

Today, we will see a dramatic improvement over the $1/2$ factor approximation via a new tool: *semidefinite programming*.

Keeping an eye on the future, let's first formulate the problem as a quadratic (instead of integer linear) optimization problem:

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

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 (1) 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 1 (Goemans-Williamson 1994)**.** *There is a polynomial time algorithm that, given a graph with max-cut OPT, finds a cut of size at least $\alpha_{GW} OPT$.*

Their algorithm is based on a new kind of convex relaxation that can be seen as a (strict, in retrospect) generalization of linear programming called semidefinite programming. We will present two equivalent views of this relaxation one seen as optimization over the space of vector-valued, as opposed to integer, assignments and the other as optimization over certain PSD matrices.

SDP Relaxation: The Vector View

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

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

Specifically, they solve:

Problem 2 (Relaxed Maximum Cut)**.**

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

Intuitively, the above formulation seeks to arrange vectors on the unit circle so that vectors corresponding to connected nodes i, j are placed as far apart (i.e., close to antipodal) as possible.

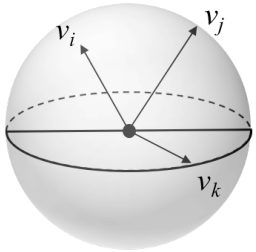

Figure 1: SDP solutions are unit vectors that are arranged so that vectors v_i and v_j are far apart when nodes i and j are connected with an edge in G.

Problem 3 is a valid relaxation of Problem 1. In particular, we have:

Claim 1.

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

Proof. Given a solution $u_1, \ldots, u_n$ to Problem 1 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 (1). □

SDP Relaxation: The PSD Matrix Optimization View

We first recall some basic definitions of positive semidefinite matrices from linear algebra.

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 "Löwner ordering". It's a partial order: both $A \succeq B$ and $B \succeq A$ can't hold for $A \neq B$; it could be that neither does.

Exercise 1. *Find a simple example where $A \not\succeq B$ and $B \not\succeq A$.*

The Löwner 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).[2]

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

PSD matrices often appear in algorithmic applications, including some we have already seen. Graph Laplacians, Hessians of convex functions, covariance matrices, and many other natural matrices are always PSD.

The *trace* of a square matrix is simply the sum of its diagonal elements. For a symmetric matrix A, $\mathsf{tr}(A)$ also equals the sum of its eigenvalues. Given two $n \times n$ square matrices A, B, observe that $\mathsf{tr}(AB) = \sum_{i,j} A_{i,j} B_{i,j}$ – that is, the inner product of the matrices seen as n^2-dimensional vectors.

We now describe a new relaxation equivalent to the vector view discussed above (here A is the adjacency matrix of the input graph).

Problem 3 (Relaxed Maximum Cut)**.**

$$\max_{X \succeq 0, X_{i,i} = 1 \forall i} 1/2 - \frac{1}{4m} \mathsf{tr}(AX). \tag{3}$$

This version is commonly called an SDP since it optimizes a linear function over the space of positive semidefinite matrices (with some additional linear constraints). Notice that this formulation takes the quadratic relaxation that *linearizes* it by writing the objective function as a linear function of a matrix X.

Why is it a relaxation? Consider $x \in \{\pm 1\}^n$ that indicates any cut in G. Then, setting $X = xx^\top$ – the outer product of x with itself – gives a valid solution: every diagonal entry of this matrix is 1, and

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

it's a rank 1 symmetric and thus also a PSD matrix (which definition would you use to verify this?).

Indeed, if we were to replace the PSDness constraint in the above program with rank 1 constraint, then it would be equivalent to the quadratic program (and thus max-cut without any relaxation) – verify this! PSDness can be thought of as a relaxation of the rank 1 constraint – unlike the space of rank 1 matrices, the space of PSD matrices is closed and convex.

Finally, to see the equivalence between the two views notice that we can go between a PSD matrix with diagonal 1 and vector assignment easily. Given a X, we can write it as $X = VV^\top$ and notice that the columns of V give unit vector assignment. On the other hand, given a vector assignment $v_1, v_2, \ldots, v_n$, we can set X so that $X = VV^\top$ for the matrix V whose columns are v_is. Notice that in this case, $X_{i,j} = X_{j,i} = \langle v_i, v_j \rangle$. Such a matrix is clearly PSD (the 3rd property!), and further, the diagonal entries are 1 since v_is are unit vectors.

Solving SDPs

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 $f : \mathbb{R}^n \to \mathbb{R}$ 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. Caution, though: not all convex programs are easy, and in fact, they can encode hard problems.

The goal of semidefinite programming is to 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:[3] 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. □

[3] 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.

This realization leads to the following convex optimization problem:

Problem 4 (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 4 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.

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 various methods, including the ellipsoid method (that we will see in the next clas) and specially designed interior point methods. They model a wide range of natural problems, several examples of which are outlined in [4].

[4]

Rounding SDP for Max-Cut: Gaussian Rounding

To obtain a solution to Problem 1 from an optimal solution to Problem 3 we employ the following rounding strategy:

1. Solve the semidefinite program in Problem 3 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 $\tilde{u}_i = \text{sign}\left(c^T v_i\right)$.

First, we informally discuss why this is a natural rounding algorithm. One argument is via symmetry. Our goal is to take vectors and convert them into ± 1 while preserving the cut value as much as possible. We will do this in two steps – one find a way to map vectors into the real line and two, apply some well-chosen function to convert the real numbers into signs. The second step will be a bit ad hoc[5]. The first step, however, is, in some sense, the right thing to do.

[5] we will simply take the sign of the numbers here and this will suffice but I cannot give you any good reason for why this should be a great idea except for the argument itself

Given a vector assignment $v_1, v_2, \ldots, v_n$, notice that $Rv_1, Rv_2, \ldots, Rv_n$ is an equally good solution since the inner products between any pair of vectors are not altered by applying the same rotation to both and the objective function only depends on pairwise inner products of v_is. We would thus like a mapping from vectors into real numbers that maintains some information about the inner products while being rotationally invariant. Such a map should not depend on individual coordinates of v_is (since these are not preserved under rotations). Taking the inner product with a Gaussian is equivalent to choosing a uniformly random direction and projecting vectors onto it.

The only property of Gaussian random vectors that we will use in the analysis is *rotation invariance* – for a Gaussian random vector g and any rotation matrix R, Rg, and g have the same distribution.

Claim 2.

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

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 1. 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.

Intuitively, since vectors corresponding to connected nodes are in general placed as far apart as possible by the SDP, it is more likely

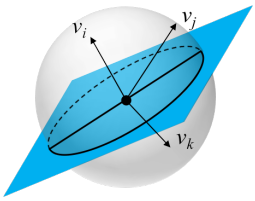

Figure 2: Our SDP solution is rounded by choosing a random hyperplane through the origin and assigning nodes to each side of the cut based on what side of the hyperplane their corresponding vector lies on. In this case, nodes i and j are placed on one side of the cut, with node k placed on the other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

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,

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

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:

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

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.)

Bibliography

[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.

[3] Kothari, Pravesh K., Meka, Raghu, and, Raghavendra, Prasad. Approximating Rectangles by Juntas and Weakly Exponential Lower Bounds for LP Relaxations of CSPs. STOC 2017, Siam Journal of Computing, 2019.