Lecture 7

Approximation algorithms for MAXCUT and MAXQP

In this lecture we apply the MMW-based algorithm introduced in the previous lecture to the MAXCUT problem. We also investigate the quality of the SDP relaxation for a more general cases of discrete quadratic programs.

7.1 Application to the MAXCUT problem

We use the MMWA to solve the MAXCUT SDP introduced a couple lectures ago. We saw that for a given undirected graph $G = (V, E)$, the size of the largest cut could be written as

$$\text{MAXCUT}(G) = \frac{|E|}{2} + \max_{x_i \in \{\pm 1\}} \frac{1}{2} \sum_{(i,j)\in E} -x_i x_j \leq \frac{|E|}{2} + \sup_{\substack{u_i \in \mathbb{R}^n \\ \|u_i\|=1}} \frac{1}{2} \sum_{i,j} -A_{i,j} u_i . u_j = \text{SDP}(G),$$

where A is the (symmetrized) adjacency matrix of the graph G, which has $1/2$ for every entry (i, j) and (j, i) associated to an edge $\{i, j\}$ and zeros elsewhere. This problem can be written in standard form in the following way:

$$\begin{aligned} \text{SDP}(G) = \sup \quad & B \bullet X \\ s.t. \quad & E_i \bullet X \leq 1 \quad \forall i = 1, \ldots, n \\ & X \succcurlyeq 0, \end{aligned}$$

where $B = \frac{d}{4}\mathbb{I} - \frac{A}{2}$ and E_i is a matrix whose i^{th} diagonal entry is 1 and the others are 0.

From now on let's assume G is d-regular (i.e. each vertex has degree exactly d). Then the adjacency matrix A has at most d non-zero entries, each equal to $1/2$, in every row, so $\|A\| \leq d/2$ and $0 \preceq B \preceq \frac{d}{2}\mathbb{I}$. Note that if α is the optimal value for the SDP we have $\frac{|E|}{2} = \frac{nd}{4} \leq \alpha \leq |E| = \frac{nd}{2}$. The first inequality follows since there is always a cut of size $|E|/2$ (a random cut will cut half the edges), and the second follows from the bound on the norm of B.

Now our goal is to design an oracle $\mathcal{O}$ that we can use for the algorithm that we introduced to solve SDP using MMW. In other words, given $X \succcurlyeq 0$ such that $\text{Tr}(X) = n$ (n plays the role of R in our algorithm), find $y \in \mathbb{R}^n_+$ such that $c^T y = \sum_i y_i \leq \alpha$ and $X \bullet (\sum_i y_i E_i - B) \geq 0$.

We design the oracle by distinguishing the following cases:

(i) If $B \bullet X \leq \alpha$, let $y_i = \frac{\alpha}{n} \geq 0$ for all i. Then $c^T y = \sum_i y_i = n\frac{\alpha}{n} = \alpha$. Moreover,

$$X \bullet (\sum_i y_i E_i - B) = \sum_i y_i X_{ii} - X \bullet B = \frac{\alpha}{n}\text{Tr}(X) - X \bullet B = \alpha - X \bullet B \geq 0,$$

and we are done.

(ii) Suppose $B \bullet X = \lambda\alpha > \alpha$ ($\lambda > 1$). We also have $\lambda \leq 2$ because $B \bullet X \leq ||B||\text{Tr}(X) = \frac{d}{2}n \leq 2\alpha$. Since $B \bullet X > \alpha$, if X is feasible then we are in case (b) for the oracle and we are done: we found a good feasible solution. Otherwise define

$$S = \{i : X_{ii} > \lambda\}, \qquad K = \sum_{i \in S} X_{ii}.$$

S is the set of indices corresponding to constraints $E_i \bullet X \leq 1$ that are violated by a large amount (since $\lambda > 1$), and K is the sum of violating diagonal entries of X. Now consider the following two cases:

- If $K > \frac{\delta\lambda n}{4}$, then let

$$y_i = \begin{cases} \frac{\lambda\alpha}{K} & \text{if } i \in S, \\ 0 & \text{if } i \notin S. \end{cases}$$

Then obviously $y \geq 0$ and

$$c^T y = \sum_i y_i = \frac{\lambda\alpha}{K}|S| \leq \frac{\lambda\alpha}{K}\frac{K}{\lambda} = \alpha,$$

where the ienquality holds since $K \geq \lambda|S|$ from the way we defined K and S. Besides,

$$X \bullet \Big(\sum_i y_i E_i - B\Big) = \sum_{i \in S} \frac{\lambda\alpha}{K} X_{ii} - X \bullet B = \frac{\lambda\alpha}{K}K - X \bullet B = \lambda\alpha - X \bullet B = 0.$$

- If $K \leq \frac{\delta\lambda n}{4}$ (only a few constraints are violated), assume without loss of generality (permuting the rows and columns if necessary) that the first $|S|$ diagonal entries of X correspond to those $i \in S$, so we can write

$$X = \begin{pmatrix} X_{S,S} & X_{S,\bar{S}} \\ X_{\bar{S},S} & X_{\bar{S},\bar{S}} \end{pmatrix}.$$

Now define a new matrix $\bar{X}$ as

$$\bar{X} = \begin{pmatrix} 0 & 0 \\ 0 & \frac{1}{\lambda}X_{\bar{S},\bar{S}} \end{pmatrix}.$$

A diagonal block extracted from a PSD matrix is PSD so $\bar{X} \succcurlyeq 0$. Besides $\bar{X}_{ii} \leq 1$ for every i, thus $\bar{X}$ is primal feasible. It remains to evaluate its objective value. Using the definition of λ,

$$\begin{aligned}\alpha - B \bullet \bar{X} &= B \bullet \Big(\frac{1}{\lambda} X - \bar{X}\Big) \\ &= \frac{1}{\lambda}\Big(B_{S,S} \bullet X_{S,S} + B_{S,\overline{S}} \bullet X_{S,\overline{S}} + B_{\overline{S},S} \bullet X_{\overline{S},S}\Big).\end{aligned}$$

Using $\mathrm{Tr}(Y^T Z) \leq \|Y\|_\infty \|Z\|_1$ and $\|Z\|_1 \leq \sqrt{n}\mathrm{Tr}(ZZ^\dagger)^{1/2}$ for any $Y, Z \in \mathbb{R}^{n\times m}$ (an application of Cauchy-Schwarz), starting from the above we can bound

$$\begin{aligned}\alpha - B \bullet \bar{X} &\leq \frac{3}{\lambda}\|B\|\sqrt{|S|}\sqrt{\sum_{i\in S}\|u_i\|^2} \\ &\overset{(1)}{=} \frac{3d}{2\lambda}\sqrt{|S|K} \\ &\overset{(2)}{\leq} \frac{3d}{2\lambda}\frac{K}{\sqrt{\lambda}} \\ &\overset{(3)}{\leq} \frac{3\delta n d}{8\sqrt{n}} \\ &\overset{(4)}{\leq} \frac{3}{2}\delta\alpha,\end{aligned}$$

Here (1) is because $\|u_i\|^2 = X_{ii}$ and $\sum_{i\in S} X_{ii} = K$. (2) uses $K > \lambda|S|$. (3) follows from our assumption $K \leq \frac{\delta\lambda}{4}n$ and (4) is because $\frac{nd}{4} \leq \alpha$. Thus finally we have $B \bullet \bar{X} \geq (1 - 3\delta/2)\alpha$.

Remark 7.1. Note that the case distinction on the values of K made above is not strictly necessary: the definition of the vector y used in the first case would in fact work for both cases. The reason we are making the distinction is that if we were to use the same y for the second case as well the oracle would turn out to have a very bad width parameter (we'll compute the width below), and as we know from last lecture this would affect the runtime.

So the oracle works. How good is it? First note that it runs very fast. We have only three cases to distinguish between, and in each one we check a linear constraint. Thus the running time is linear in the number of edges m of the graph.

Next we need to bound the width of the oracle.

$$\Big\|\sum_i y_i E_i - B\Big\| \leq \left\| \begin{pmatrix} y_1 & & 0 \\ & \ddots & \\ 0 & & y_n \end{pmatrix} \right\| + \|B\| \leq \max_i |y_i| + \frac{d}{2}.$$

In order to find $\max_i |y_i|$ we should check all of the cases. For example for the case when $K \geq \frac{\delta\lambda}{4}n$, we have

$$\lambda_i \leq \frac{\lambda\alpha}{K} \leq \frac{4}{\lambda\delta n}\frac{nd}{2} = \frac{4d}{\delta}.$$

Thus $\max_i y_i \backsim \frac{d}{\delta}$ and the width $\sigma \backsim \frac{d}{\delta}$. Recall that the MMWA algorithm for solving SDPs required $T = \frac{8\sigma^2 R^2}{\delta^2 \alpha^2} \ln(n)$ iterations. At each iteration there is one call to the oracle made. Using $R/\alpha \leq 4/d$, our running time is then $O(mT) = O(m\delta^{-4} \log n)$, which is a quasi-linear-time algorithm! This is much better than a "generic" SDP solver, which would, at best, scale like $O(n^3)$.

Unfortunately we cheated a little bit — at each iteration, we need not only call the oracle, but also perform the update for $X^{(t)}$, and this requires computing a matrix exponential...which can take time $O(n^3)$! Note however that here $X^{(t)}$ itself is never needed: the only information the oracle requires is the dot product $X^{(t)} \bullet B$, as well as the diagonal entries of X. It is possible to compute those very efficiently, in time $O(m \log n)$, using dimension reduction techniques such as the Johnson-Lindenstrauss lemma that we will see in a couple lectures. Then the overall running time remains $\tilde{O}(m)$.

7.2 General quadratic programs

Consider the following problem

$$\text{MAXQP}(A) = \sup_{\substack{x_i, y_j \in \{\pm 1\} \\ i=1,\ldots,n \\ j=1,\ldots,m}} \sum_{i,j} A_{i,j} x_i y_j,$$

where $A \in \mathbb{R}^{n \times m}$.

Exercise 1. Show that this problem is NP-hard by performing a reduction from MAXCUT.

We will see how a good approximation can be obtained in polynomial time. For this we propose the following relaxation:

$$\text{MAXQP}(G) \leq \text{SDP}(G) = \sup_{\substack{u_i, v_j \in \mathbb{R}^{m+n} \\ \|u_i\| = \|v_j\| = 1}} \sum_{i,j} A_{i,j} u_i \cdot v_j. \tag{7.1}$$

Since the u_i and v_j are different sets of vectors it may not be immediately obvious that this program is an SDP, but it is — we will show this soon. The following theorem guarantees that the above relaxation is never too bad — SDP(G) is never more than a constant factor (independent of A and its dimension) larger than MAXQP(G). Moreover we'll give a *constructive* proof of this fact, showing how any vector solution to the SDP can be "rounded" to a ± 1 solution to the original integer program.

Theorem 7.2. *Given unit vectors u_i and v_j achieving the optimum in SDP(G), there exists a polynomial-time algorithm that produces x_i and y_j in $\{\pm 1\}$ such that*

$$\sum_{i,j} A_{i,j} x_i y_j \geq C \cdot \text{MAXQP}(G),$$

where C is a universal constant.

Different proof techniques for the theorem yield different values of C. In your homework you will develop an algorithm to achieve $C \approx 0.56$. The best value for C is called Grothendieck's constant K_G and can be defined as

$$K_G = \inf \left\{ C : \forall m, n, \forall A \in \mathbb{R}^{n \times m}, \quad \sup \sum_{i,j} A_{i,j} u_i \cdot v_j \leq K_G \sup \sum_{i,j} A_{i,j} x_i y_j \right\}$$

Before proceeding let's check that SDP(A) is indeed an SDP by writing it in the primal canonical form

$$\begin{aligned} \sup \quad & B \bullet Z \\ s.t. \quad & A_i \bullet Z \leq c_i \\ & Z \succcurlyeq 0. \end{aligned}$$

Let U be a matrix whose columns are the u_i, and V whose columns are the v_j; define

$$Z = (UV)^T (UV) = \begin{pmatrix} [u_i \cdot u_j] & [u_i \cdot v_j] \\ [v_i \cdot u_j] & [v_i \cdot v_j] \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}.$$

Clearly Z is a PSD matrix (it is a Gram matrix), and its diagonal elements are the squared norms $\|u_i\|^2$ and $\|v_j\|^2$, which should be at most one. Thus we let $c_i = 1$ and $A_i = E_i$ for $i = 1, \ldots, n+m$, where E_i is a $(n+m) \times (n+m)$ matrix whose i^{th} diagonal entry is one and the others are zero. Finally for the objective value, we define

$$B = \frac{1}{2} \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}.$$

Then the corresponding SDP is equivalent to the definition of SDP(A) given earlier.

7.3 Analysis of the SDP relaxation

In this section we analyze the performance of the relaxation (7.1). The main result is that we can bound the ratio SDP(A)/MAXQP(A) by a constant factor, regardless of the choice of A.

Theorem 7.3. *There exists a universal constant $K = K_G$ such that $\forall A$,* SDP(A) $\leq K \cdot$ MAXQP(A).

Remark 7.4. (1) If one does not care about the optimal constant the result can be made "algorithmic" in the following sense: the proof we'll give in this lecture shows that given any unit vectors u_i, v_j achieving SDP(A), there is a randomized (even deterministic) polynomial-time algorithm that outputs $x_i, y_j \in \{\pm 1\}$ such that

$$\sum_{i,j} A_{ij}\, x_i y_j \geq \alpha \sum_{i,j} A_{ij} \vec{u}_i \cdot \vec{v}_j$$

for some $\alpha \geq 0.01$.

(2) One can do better in the value of α. You have a homework problem showing one can achieve $2\ln(1+\sqrt{2})/\pi \approx 0.56$ with a randomized algorithm. The best possible constant is known as Grothendieck's constant, which is strictly greater than 0.56. We do not know its precise value — only the first few digits have been pinned down.

Let's prove Theorem 7.3. Let $\vec{u}_i, \vec{v}_j \in \mathbb{R}^d$ achieving SDP(A) be given (note that technically we should assume they achieve SDP(A)$-\varepsilon$, as we don't know if the supremum is attained; for clarity we'll put this issue aside). We can always assume $d \leq m+n$ since this is the number of vectors. Observe that

$$\sum_{i,j} A_{ij} \vec{u}_i \cdot \vec{v}_j = \frac{1}{d} \sum_{k=1}^{d} \left(\sum_{i,j} A_{ij} \left(\sqrt{d}(\vec{u}_i)_k \right) \left(\sqrt{d}(\vec{v}_j)_k \right) \right).$$

This implies we can find $k \in \{1, \ldots, d\}$ such that

$$\sum_{i,j} A_{ij} \left(\sqrt{d}(\vec{u}_i)_k \right) \left(\sqrt{d}(\vec{v}_j)_k \right) \geq \sum_{i,j} A_{ij} \vec{u}_i \cdot \vec{v}_j = \text{SDP}(A) \geq \text{OPT}(A).$$

How large are the $|(\vec{u}_i)_k|$? From $\|\vec{u}_i\| \leq 1$ we know $|(\vec{u}_i)_k| \leq 1$, which implies $\sqrt{d}(\vec{u}_i)_k \leq \sqrt{d}$. This naive bound is not good enough: we are looking for an assignment of values in the range $[-1, 1]$, so we'd have to divide all coordinates by $\sqrt{d}$, but then we'd lose a factor d in the objective value.

Now, if everything was "well-behaved", i.e. the vectors are random, we would expect $|(\vec{u}_i)_k| \simeq 1/\sqrt{d}$ because $\|\vec{u}_i\| \leq 1$ — in this case barely any renormalization is needed. This heuristic suggests the approach for the proof, which is to perform a "random" rotation (we will in fact apply a deterministic transformation) that guarantees the coordinates are well-balanced, so that most of them are not too large, no larger than some constant. We'll then "truncate" the coordinates that are too large, and argue that the loss in objective value suffered through this truncation is not too large. The rotation we'll apply is based on the following claim:

Claim 7.5. *In dimension d, there exists $t = O(d^2)$ vectors $\vec{g}_1, \ldots, \vec{g}_t \in \{\pm 1\}^d$, such that the $\vec{g}_i$ are 4-wise independent.*

Recall from lecture 1 that the condition of 4-wise independence in the claim can be interpreted as follows. Consider the matrix

$$G = \begin{pmatrix} - & \vec{g}_1 & - \\ - & \vec{g}_2 & - \\ & \cdots & \\ - & \vec{g}_t & - \end{pmatrix}.$$

Then if we fix any 4 columns of G, corresponding to coordinates of the $\vec{g}_i$, each possible pattern in $\{\pm 1\}^4$ occurs exactly $t/16$ times. This property naturally implies r-wise independence for $r < 4$. That is, if we look at a single column, the number of 1's should be the same

as the number of -1's, and similar conditions hold for two and three columns. Note that if we allow exponential large t, we can just choose all the possible d-dimensional ± 1 vectors to satisfy this property. The main point here is that we can make this t only polynomial in d. A homework exercise will show you how to construct such a family of vectors based on BCH error-correcting codes. In Lecture 1 we also saw a construction that would construct the $\vec{g}_i$ from a 4-wise independent family of hash functions based on polynomials. We'll prove the theorem in the next lecture.