18.415/6.854 Advanced Algorithms 28 November 2001

Lecture 20

Lecturer: Michel X. Goemans *Scribe: David Liben-Nowell and Bill Thies*

1 Review

In the previous lecture, we were considering the LIN-2-MOD-2 problem, which is as follows. We are given a set of equations, and our goal is to satisfy as many of them as possible:

$$x_i + x_j \equiv \{0, 1\} \pmod 2 \tag{1}$$

That is, each equation states that the sum of exactly two variables is congruent (modulo 2) to either 0 or 1. Restricting our attention to the case when each sum is congruent to 1, this problem is equivalent to the following instance of MAX-CUT. Given a graph $G = (V, E)$ with a vertex v for each variable x and an edge $\langle v_i, v_j \rangle$ for each equation $x_i + x_j \equiv 1 \pmod 2$, find the set of vertices $S \subset V$ that maximizes the number of edges crossing the boundary of the set:

$$\max_{S \subset V} d(S) \tag{2}$$

where $d(S) = |\delta(S)|$ is the number of edges crossing from S to $V \setminus S$

We then established the following upper bound (UB) for MAX-CUT:

$$UB = \max \sum_{\langle i,j \rangle \in E} \frac{1 - v_i \cdot v_j}{2}$$
$$\text{s.t.} \quad \|v_i\| = 1, \quad v_i \in \mathbb{R}^p \quad (\forall i \in V). \tag{3}$$

We will refer to this maximization problem as **SDP**, which stands for "semi-definite programming" as we will see later. In the case when $p = 1$, UB is clearly equal to the value of the maximum cut; we showed that for larger p, the expression above still remains an upper bound. Finally, we used this expression to sketch out a 0.878-approximation algorithm for MAX-CUT. In this lecture, we formulate the details of this algorithm and prove its correctness.

2 A 0.878-Approximation Algorithm for Max-Cut

The 0.878-approximation algorithm that we consider is due to Goemans and Williamson [2]. An outline of the algorithms is as follows:

Algorithm for Max-Cut:

1. Solve the SDP problem (Equation 3) to obtain vectors v_i.
2. We'd like to separate the vectors into two groups that are "maximally separated" in order to determine the cut $S \subset V$. As discussed in the last lecture, we can do this with a hyperplane in $\mathbb{R}^p$, and split the vectors based on which side of the hyperplane they fall. However, since we can rotate all of the vectors v_i without changing the goodness of the solution, it is not a good idea to fix a single hyperplane to do this separation since some cases will elicit a poor split.

   Instead, we select a unit vector r uniformly at random from the p-dimensional sphere $S_{p-1} = \{x \in \mathbb{R}^p : \|x\| = 1\}$. (We denote this sphere by S_{p-1} since S_1 is a circle in two dimensions.)

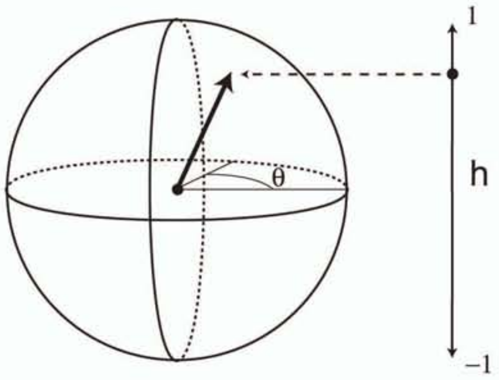

Figure 1: One can select a point uniformly at random from the surface of S_2, the three-dimensional unit sphere, by choosing uniformly at random θ from $[0, 2\pi]$ and h from $[-1, 1]$ and then projecting the point horizontally onto the surface of the sphere.

3. Select the cut S as those vectors which fall on one side of r: $S = \{i \in V : r \cdot v_i \geq 0\}$.

There are three issues in the above algorithm that we need to address. In Section 5, we present a method for solving the SDP problem as required in step 1. In Section 3 we show how to randomly select the unit vector r as required in step 2, and in Section 4 we analyze selection of the cut to show that it is, in fact, a 0.878-approximation algorithm.

3 Choosing a Random Vector

It is not entirely trivial to choose a vector that is uniformly distributed across the points of a multi-dimensional sphere. For instance, in three dimensions, choosing both latitude and longitude coordinates uniformly at random will not yield points that are uniformly distributed across the surface of the earth. This is because a given range of latitude covers a smaller and smaller portion of the earth's surface as one nears the poles–thus, if latitude were chosen uniformly, more points would end up near the poles than near the equator.

3.1 3-D Solution

For the case of three dimensions, a uniform spherical distribution can be obtained by choosing different coordinates uniformly at random. These coordinates are 1) an angle $\theta \in [0, 2\pi]$ and 2) a height $h \in [-1, 1]$. The random vector can then be obtained by projecting horizontally onto a sphere from the corresponding position on an enclosing cylinder (see Figure 1). In other words, one can take θ as the longitude and $asin(h)$ as the latitude.

3.2 General Solution

In the general case, one can generate r from S_{p-1} uniformly at random by choosing each coordinate according to $N(0, 1)$, the normal distribution with mean 0 and standard deviation 1. Using $\propto$ to denote proportionality, we have by the definition of a normal distribution that:

$$N(0,1) \propto f(x) \propto \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \tag{4}$$

We can see that this gives the desired result if we consider the normal distribution in p dimensions:

$$f(x_1, x_2, \ldots x_p) \propto c e^{-\frac{1}{2}(x_1^2 + x_2^2 + \ldots + x_p^2)} \tag{5}$$

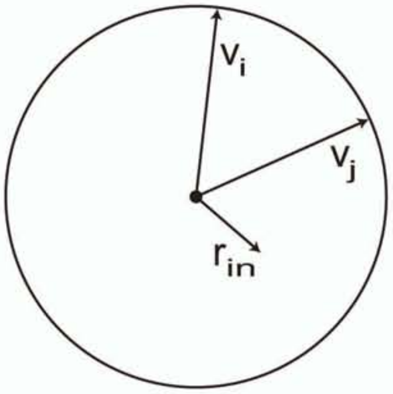

Figure 2: Two vectors v_i and v_j with r_{in}, the component of the random vector r which lies in the plane of v_i and v_j.

where c denotes some constant. Noting that the distance of any point from the origin is $d = \sqrt{x_1^2 + x_2^2 + \ldots + x_p^2}$, we can see that the distribution depends only on d instead of the position along any given axis:

$$f(d) \propto ce^{-\frac{1}{2}d^2} \tag{6}$$

Thus, selecting each coordinate from this distribution will give vectors that are uniformly distributed within the unit sphere S_{p-1}.

4 Analyzing the Algorithm

We now turn our attention to proving that our choice of $S = \{i \in V : r \cdot v_i \geq 0\}$ does yield a 0.878-approximation algorithm for MAX-CUT, given that v_i are a solution to SDP (Equation 3) and r is chosen randomly as described above. As a first step, we will need a closed form expression for the expected value of the cut, given our random choice of r. For this, we turn to the following theorem:

Theorem 1 *For any set of unit vectors v_i, $E[d(S)] = \sum_{\langle i,j\rangle \in E} \frac{acos(v_i \cdot v_j)}{\pi}$.*

Proof: Let us introduce an indicator variable I_{ij} whose value is 1 if $\langle i,j\rangle$ is in the cut and 0 otherwise. Then we have that, by the definition of d and of expectation:

$$\begin{aligned} E[d(S)] &= E[\sum_{\langle i,j\rangle \in E} I_{ij}] \\ &= \sum_{\langle i,j\rangle \in E} Pr[\langle i,j\rangle \in \delta(S)] \end{aligned}$$

Now let us consider this last probability–that exactly one of a pair of vertices i and j are contained in S. This will be the case if the unit vectors v_i and v_j were separated by the hyperplane orthogonal to our random vector r. Let us split r up into two components: 1) r_{in}, the component that is in the plane of v_i and v_j, and 2) r_{out}, the component that is orthogonal to v_i and v_j. We are not concerned with r_{out} because its dot product with v_i and v_j is zero; therefore, our decision as to whether or not to include i and j in S depends only on r_{in}.

Thus, we restrict our attention to r_{in} (see Figure 2). Because r was chosen by a spherically symmetric distribution, we know that r_{in} is uniformly distributed within a circle in the space spanned

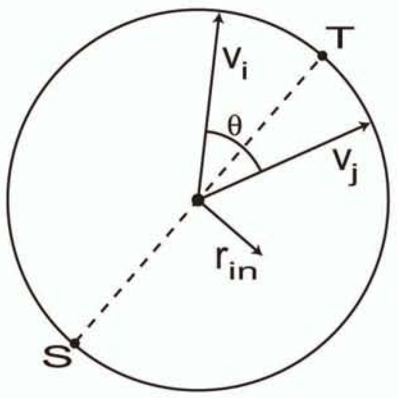

Figure 3: The segment ST is orthogonal to r_{in}, and the angle θ is measured between v_i and v_j.

by v_i and v_j. (It might not be S_1, the unit circle, since r_{in} could be shorter than 1. However, this does not affect the proof since we are only interested in the direction of r_{in}.)

Now, our cut will separate i and j if and only if v_i and v_j fall on opposite sides of the segment ST which is orthogonal to r_{in} (see Figure 3). This is a probability that is straightforward to compute. Let θ denote the angle between v_i and v_j. Since both v_i and v_j are distributed uniformly within the circle, θ is distributed uniformly within $[0, 2\pi]$. And since r_{in}, and therefore the endpoint T, is oriented at an angle with a uniform distribution between $[0, 2\pi]$, we can see that T will fall within θ with a probability of $\theta/2\pi$. But since ST will also separate v_i and v_j if S falls within the angle θ, the probability that v_i and v_j are separated is $2\theta/2\pi = \theta/\pi$. Noting that $v_i \cdot v_j = cos(\theta)$ because v_i and v_j are unit vectors, we have that $\theta = acos(v_i \cdot v_j)$ and thus $Pr[\langle i, j\rangle \in \delta(S)] = acos(v_i \cdot v_j)/\pi$ as desired. □

This yields the following corollary, which seems quite surprising:

Corollary 2 *Fixing p and allowing the v_i's to vary, define*

$$Z = \max \sum_{\langle i,j\rangle \in E} \frac{acos(v_i \cdot v_j)}{\pi} \tag{7}$$

Then Z is equal to OPT, the optimal value of MAX-CUT.

Proof: We show that $Z \leq OPT$ and $Z \geq OPT$:

- $Z \leq OPT$ because, by Theorem 1, $\sum_{\langle i,j\rangle \in E} \frac{acos(v_i \cdot v_j)}{\pi}$ is the expected value of a cut for a given set of v_i's. Thus, for the set of v_i's yielding the maximum, Z, there must be some cut with a value at least as great as the expected (average) value, and this value can be at most OPT.
- To show $Z \geq OPT$, consider the elements S of the optimal cut. Select a unit vector w in $\mathbb{R}^p$, and set $v_i = w$ if $i \in S$ and $v_i = -w$ otherwise. Then $\sum_{\langle i,j\rangle \in E} \frac{acos(v_i \cdot v_j)}{\pi} = OPT$ since v_i and v_j are opposite (with $acos(v_i \cdot v_j) = \pi$) for edges $\langle i, j\rangle$ across the cut, but they are identical (with $acos(v_i \cdot v_j) = 0$) for other edges.

□

Thus, we have seen that we could solve the MAX-CUT problem by solving Equation 7, but (as far as we know) this can't be done in polynomial time. Instead, we would like to solve the easier SDP problem in Equation 3, which can be done in (almost) polynomial time, and then use the randomized cut to obtain an approximation of the optimum. We formalize the correctness of this approach with the following theorem:

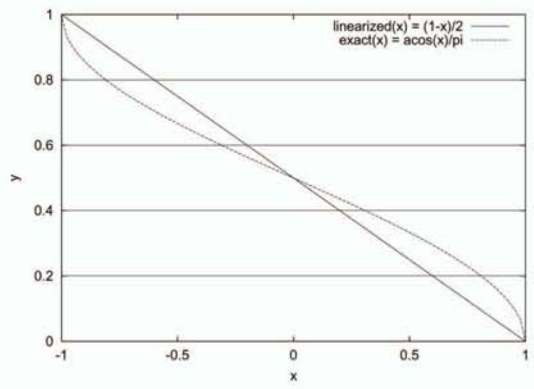

Figure 4: The linearized approximation compared to the exact objective function. The original objective function is always at least 0.878 as great as the linearized version.

Theorem 3 *For any set of vectors v_i:*

$$E[d(S)] = \textstyle\sum_{\langle i,j\rangle \in E} \frac{acos(v_i \cdot v_j)}{\pi} \geq \alpha \left(\sum_{\langle i,j\rangle \in E} \frac{1-v_i \cdot v_j}{2} \right)$$
$$where\ \alpha = min_{x\in[-1,1]} \tfrac{acos(x)/\pi}{(1-x)/2} \approx 0.87856 \tag{8}$$

Proof: The first equality is from Theorem 1 and included only for clarity. The inequality holds term-wise on each component of the sum, as we can show with simple algebraic manipulation:

$$\begin{aligned} acos(v_i \cdot v_j)/\pi &= (acos(v_i \cdot v_j)/\pi) \left(\frac{(1-v_i \cdot v_j)/2}{(1-v_i \cdot v_j)/2} \right) \\ &= \left(\frac{acos(v_i \cdot v_j)/\pi}{(1-v_i \cdot v_j)/2} \right) (1-v_i \cdot v_j)/2 \\ &\geq \left(\min_{x\in[-1,1]} \frac{acos(x)/\pi}{(1-x)/2} \right) (1-v_i \cdot v_j)/2 \\ &\geq \alpha(1-v_i \cdot v_j)/2 \end{aligned} \tag{9}$$

This inequality is shown graphically in Figure 4. □

Now, if we could solve the SDP problem, we would obtain $UB = \max \sum_{\langle i,j\rangle \in E} \frac{1-v_i \cdot v_j}{2}$. Instead consider that we solve SDP within some margin of error ϵ, since finding the optimal solution might require more than polynomial time, but we can still find a nearly-optimal solution. Then, by Theorem 3, we have that our randomized algorithm produces $E[d(S)] \geq \alpha(1-\epsilon)UB \geq 0.878 * OPT$ for sufficiently small ϵ. Thus, our algorithm is a 0.878-approximation algorithm for MAX-CUT.

As an extension, we could generalize the problem to consider any instance of LIN-2-MOD-2 (above we consider only equations congruent to 1, whereas LIN-2-MOD-2 also allows equations congruent to 0). This simply involves modifying the objective function in the SDP problem to add a $\frac{1+v_i \cdot v_j}{2}$ term for every equation $x_i + x_j \equiv 0 \mod 2$. These terms will have an expected value of $1 - acos(v_i \cdot v_j)/\pi$ instead of $acos(v_i \cdot v_j)/\pi$, but the approximation guarantee will be preserved since $1 - acos(x)/\pi$ is at least as great as $\alpha(1+x)/2$ in $[-1,1]$ (see Figure 5).

Remarks. We conclude the analysis with a few remarks about work related to the algorithm:

1. Feige and Schechtman [1] identified graphs $G_n = (V,E)$ such that $inf_n\ OPT/UB = \alpha \approx 0.87856$. This shows that our analysis of the algorithm is tight. That is, given our estimation of the upper bound UB, we cannot do better than an α-approximation algorithm on every input,

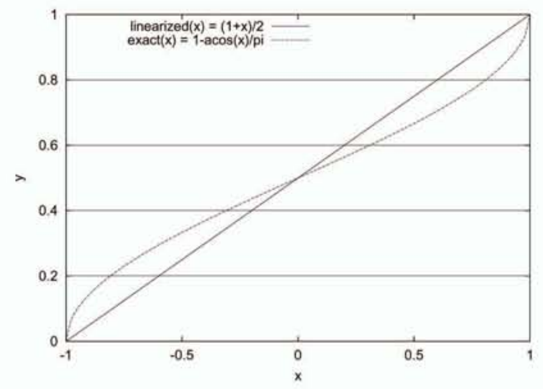

Figure 5: The linearized approximation compared to the exact objective function for equations congruent to 0 in the original LIN-2-MOD-2 problem. The original objective function is still at least 0.878 as great as the linearized version.

since otherwise there would be an instance of G_n for which we would exceed the optimum. However, this does not preclude there being tighter estimations of UB that would allow a better approximation algorithm.

2. Håstad [3] established that there is no $(16/17 + \epsilon)$-approximation algorithm for MAX-CUT for all $\epsilon > 0$ unless $P = NP$. Given that $16/17 \approx 0.942$, this leaves a gap down to 0.878 between which it is not known where the limit of approximability lies. However, the gap is smaller for LIN-2-MOD-2, for which Håstad found that there is no $(11/12 + \epsilon)$-approximation algorithm $(11/12 \approx 0.916)$.

3. The algorithm given above is a randomized one; it is messy to obtain a deterministic algorithm instead. As in the last lecture, the method of conditional expectations can be applied. However, it is more complicated because the v_i's do not have discrete coordinates; rather, they can assume continuous values instead. A technique for derandomizing the algorithm is described in [4].

5 How do we solve the SDP?

The last missing piece in the algorithm is a mechanism for solving the SDP:

$$\max \sum_{\langle i,j\rangle \in E} \frac{1 - v_i \cdot v_j}{2}$$
$$\text{s.t.} \quad \|v_i\| = 1, \quad v_i \in \mathbb{R}^p \quad (\forall i \in V). \tag{10}$$

We will sketch a solution to this problem based upon an extension of a polynomial-time algorithm for linear programming — either ellipsoid or interior point suffices. (For concreteness, we will focus on this particular program rather than the more general class of *semidefinite programs* (SDPs) that we can solve in this way.)

For a matrix M, let $M \succeq 0$ denote that M is *positive semidefinite*. Recall from linear algebra that, by definition, we have $M \succeq 0 \iff \forall a \; a^{\mathsf{T}} M a \geq 0$.

Consider a matrix $Y \in \mathbb{R}^{n \times n}$ so that $Y_{ij} = v_i \cdot v_j$ for $1 \leq i, j \leq n$. Thinking about this matrix, we can reformulate the program (10) as follows:

$$\max \sum_{\langle i,j\rangle \in E} \frac{1 - Y_{ij}}{2}$$

$$\begin{aligned}\text{s.t.}\quad & Y_{ii} = 1 \\ & Y_{ij} = Y_{ji} \\ & Y \succeq 0. \end{aligned} \tag{11}$$

(Generally, a semidefinite program has a linear objective function, linear constraints on Y, and the restriction that $Y \succeq 0$ be symmetric.)

Claim 4 *Programs (10) and (11) are equivalent.*

Proof: Given a solution $v_1, \ldots, v_n$ to (10), we produce a solution to (11) by setting $Y_{ij} = v_i \cdot v_j$. All of the constraints of (11) are satisfied:

- $Y_{ii} = v_i \cdot v_i = \|v_i\| = 1$ by the constraint of (10).
- $Y_{ij} = v_i \cdot v_j = v_j \cdot v_i = Y_{ji}$ by the commutativity of inner product.
- For any vector a,

$$\begin{aligned} a^T Y a &= \sum_i \sum_j (a_i\, a_j)\ (v_i \cdot v_j) \\ &= \sum_i \left[a_i v_i \cdot \left(\sum_j a_j v_j \right) \right] \\ &= \left(\sum_i a_i v_i \right)^2 \ \geq \ 0 \end{aligned}$$

since $\sum_i a_i v_i$ is a scalar. So $Y \succeq 0$.

Given a solution Y to (11), we produce the vectors v_i as follows. Recall from linear algebra that $Y \succeq 0$ iff there is a *Cholesky factorization* $Y = VV^T$, where $V \in \mathbb{R}^{n \times p}$, where $p = \mathsf{rank}(Y) \leq n$. Take as vector v_i^T the ith row of V. Then $v_i \cdot v_j = (VV^T)_{ij} = Y_{ij}$. By the constraint of (11) that $Y_{ii} = 1$, we have the desired condition that $\|v_i\| = 1$.

So any feasible solution to one program can be converted to a feasible solution to the other program, where $Y_{ij} = v_i \cdot v_j$, and thus the values of the objective functions are also identical. Thus the programs are equivalent. □

Cholesky factorization can be accomplished in $O(n^3)$ time by Gaussian elimination. So we now need only solve version (11) of the program. Observe that the semidefinite program is actually almost a linear program: the only constraint that doesn't fit into a linear program is $Y \succeq 0$, i.e., $a^T Y a = \sum_i \sum_j a_i a_j Y_{ij} \geq 0$ for every a. But this *is* a linear constraint for any particular a; there are just infinitely many such a's, and therefore infinitely many linear constraints.

Observe that the set of feasible solutions to the SDP is convex: if A and B are feasible, then $\lambda A + (1-\lambda)B$ is trivially symmetric and has ones on its diagonal, and, for any a,

$$a^T(\lambda A + (1-\lambda)B)a \ = \ a^T(\lambda A)a + a^T((1-\lambda)B)a \ = \ \lambda(a^T A a) + (1-\lambda)(a^T B a) \ \geq \ 0$$

so long as $\lambda \in [0,1]$, since $A, B \succeq 0$. (This is a special case of *convex programming* — a linear objective function over a convex set.)

We can solve the SDP by adapting interior point or ellipsoid:

- [interior point.] In LP, we'd added a penalty function $\mu \sum_j \log(x_j)$ to our objective function; for SDP, we add the penalty $\mu \log(\mathsf{det}(Y)) = \mu \sum_j \log(\lambda_j)$. ($Y$ is symmetric and positive semidefinite iff all of Y's eigenvalues are positive; $\mathsf{det}(Y) \geq 0$ bounds Y's eigenvalues away from 0.)

  The mantra, which is only partially false: "SDP is LP over the eigenvalues."

- [ellipsoid.] We need to be able to answer the question "is Y feasible?", and, if not, produce a violated constraint. We can do this by computing the eigenvalues of Y; if $\lambda_j < 0$, then take $a = v_j$ the corresponding eigenvector. Then $a^T Y a < 0$, violating $Y \succeq 0$.

  Finding the initial and final ellipsoids is not so simple, but it can be done.

We may lose the factor of ε in solving the SDP because the optimal solution may be irrational (unlike in LP, where there was nice form of the solution in terms of ratios of small integers); we have to give up before we actually reach optimality.

References

[1] U. Feige and G. Schechtman. On the optimality of the random hyperplane rounding technique for MAX CUT. Technical report, Weizmann Institute, Rehovot, Israel, 2000.

[2] M. X. Goemans and D. P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. Assoc. Comput. Mach.*, 42:1115–1145, 1995.

[3] J. Håstad. Some optimal inapproximability results. In Proc. 29th ACM Symp. on Theory of Computing, 1997.

[4] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 162–169, 1995.