Theoretician's Toolkit **Spring 2019**

Lecture 6: Semidefinite Programming and MAXCUT

Scribe: Ida Liu, Jiazheng Zhao 4/1/2019

6.1 Introduction

Sometimes it's not easy to capture a problem using polynmialy sized linear programming, while it's straightforward to write out the constraints and objectives in quadratic equations. This section we are going to introduce how to use quadratic programming and SDP rounding to solve the max cut problem.

6.2 Semidefinite Programming

6.2.1 Semidefinite Programming

A semidefinite program (SDP) is similar to a linear program in that there is a linear objective function and linear constraints. In addition, however, a square symmetric matrix of variables can be constrained to be positive semidefinite. This means that there will $n * n$ variables in the form of $x_i j$, for $1 \leq i, j \leq n$, where n is the dimension of the matrix X. Also, it requires the additional constraint of $x_{ij} = x_{ji}$ to make X symmetric, and the constraint $X = (x_{ij}) \succeq 0$ to make X positive semidefinite.

$$\begin{aligned} \text{min/max} \quad & \sum_{i,j} c_{i,j} x_{i,j} \\ \text{subject to} \quad & \sum_{i,j} a_{ijk} x_{ij} = b_k \quad \forall k \\ & x_{ij} = x_{ji} \\ & X = (x_{ij}) \succeq 0 \end{aligned}$$

We often express semidefinite programming in the form of vector programming. The variables of vector programs are vectors $v_i \in R_n$, where the dimension n of the space is the number of vectors in the vector program. The vector program has an objective function and constraints that are linear in the inner product of these vectors. We write the inner product of v_i and v_j as $v_i \cdot v_j$, or sometimes as $v_i^T v_j$. Below is an example of a vector program:

$$\begin{aligned} \min/\max \quad & \sum_{i,j} c_{ij}(v_i^\top v_j) \\ \text{subject to} \quad & \sum_{i,j} a_{ijk}(v_i^\top v_j) = b_k \quad \forall k \\ & v_i \in \mathbb{R}^n \qquad\qquad\qquad \forall i \end{aligned}$$

Semidefinite programming and vector programming are equivalent, because a symmetric X is PSD if and only if $X = V^T V$ for some matrix V. To see this fact, first verify that if a matrix X is symmetric, diagonalizing of X gives: $X = P\Sigma P^T = P\left(\Sigma\right)^{\frac{1}{2}}\left(\Sigma\right)^{\frac{1}{2}}P^T = P\left(\Sigma\right)^{\frac{1}{2}}\left(\Sigma\right)^{\frac{1}{2}^T}P^T = \left(P\left(\Sigma\right)^{\frac{1}{2}}\right)\left(P\left(\Sigma\right)^{\frac{1}{2}}\right)^T$. Setting V to the transpose of $P\left(\Sigma\right)^{\frac{1}{2}}$ gives us the desired form of $X = V^T V$. Also, $X = V^T V$ is PSD because for any vector y, $y^T X y = y^T V^T V y = (Vy)^T (Vy) \succeq 0$. Therefore, semidefinite programming and vector programming are equivalent because we can take the solution of one and transform it to the other.

6.2.2 Solving SDP

Given some error bound ϵ, SDP can be solved in time that is polynomial in size of input and $log(1/\epsilon)$. We are not going to discuss the details of the algorithm here, and instead assume that SDP can be solved efficiently.

6.3 Example: MAXCUT

6.3.1 Problem Formulation

In this section, we show how to use semidefinite programming to find an improved approximation algorithm for the maxcut problem. Recall that for this problem, the input is an undirected graph $G = (V, E)$, and nonnegative weights $w_{ij} \geq 0$ for each edge $(i, j) \in E$. The goal is to partition the vertex set into two parts, U and $W = V - U$, so as to maximize the weight of the edges whose two endpoints are in different parts, one in U and one in W.

Now we shall come up with a formulation of the maxcut problem. The intuition is that, if an edge crosses the cut, we will add its weight to the total sum, or else we do not add it to the total sum. Also, we want to use a variable which label each vertex in some way that indicates which group the vertex belong to. With these idea in mind, we have the following formulation:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - y_i y_j) \\ \text{subject to} \quad & y_i \in \{-1, +1\} \qquad\qquad i = 1, ..., n. \end{aligned}$$

Lemma 6.1. *The program models the maximum cut problem.*

Proof. Consider the cut $U = \{i : y_i = -1\}$ and $W = \{i : y_i = +1\}$. Note that if an edge (i, j) is in this cut, then $y_i y_j = -1$, and $1 - y_i y_j = 2$; while if the edge is not in the cut, $y_i y_j = 1$, and $1 - y_i y_j = 0$. Thus

$$\text{maximize} \quad \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - y_i y_j)$$

will include the weight of the edges that are in the cut, and exclude the weights of those that are not in the cut. Hence finding the setting of the y_i to ± 1 that maximizes this sum gives the maximum-weight cut. □

Again, this problem formulation is NP-complete. Recall that linear programming can be used to approximate Integer Linear Programs (ILP): we first relax ILP into LP, solve LP problem, and then round the result into integers using some rounding scheme. Similarly, to give an approximation of max cut, we will first relax it into vector programming and then round the result.

Now let's consider a relaxation of the problem formulation using vector programming:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j)\in E} w_{ij}(1 - v_i \cdot v_j)) & \\ \text{subject to} \quad & v_i \cdot v_i = 1, & i = 1, ..., n, \\ & v_i \in \Re^n, & i = 1, ..., n, \end{aligned}$$

This program is a relaxation of the previous formulation since we can take any feasible solution y and produce a feasible solution to this program of the same value by setting $v_i = (y_i, 0, 0, ..., 0)$: clearly $v_i \cdot v_i = 1$ and $v_i \cdot v_j = y_i y_j$ for this solution. Since any solution y is also a solution for VP, the value of an optimal solution for VP will always be at least as good as the optimal solution for max cut.

Now we shall present a rounding scheme in the next section that will achieve 0.878-approximation.

6.3.2 Rounding SDP

If we solve this SDP, we will get n length 1 vectors in $\mathbb{R}^n$, so now we need to somehow map these vectors to a $\{0, 1\}$ solution. Note that if we draw a random vector r such that each entry of r follows $\mathcal{N}(0, 1)$ and take its direction, it's uniformly distributed the n-dimensional unit sphere. We will used this vector r for our randomized rounding.

First, we sample r as we discussed above. Then for each v_i from the SDP solution: set $y_i = 1$ if $v_i^T r > 0$, and set $y_i = 0$ if $v_i^T r \leq 0$. Intuitively, this random vector r gives a hyperplane across the center of the unit sphere. and everything on one side of the plane will be assigned to one side of the cut.

Theorem 6.2. *This randomized rounding is a 0.878 approximation.*

Before we proceed to proof, here are some useful fact for the analysis:

Lemma 6.3. *Probability that any edge (i, j) falls in the cut is $\frac{arccos(v_i^T v_j)}{\pi}$.*

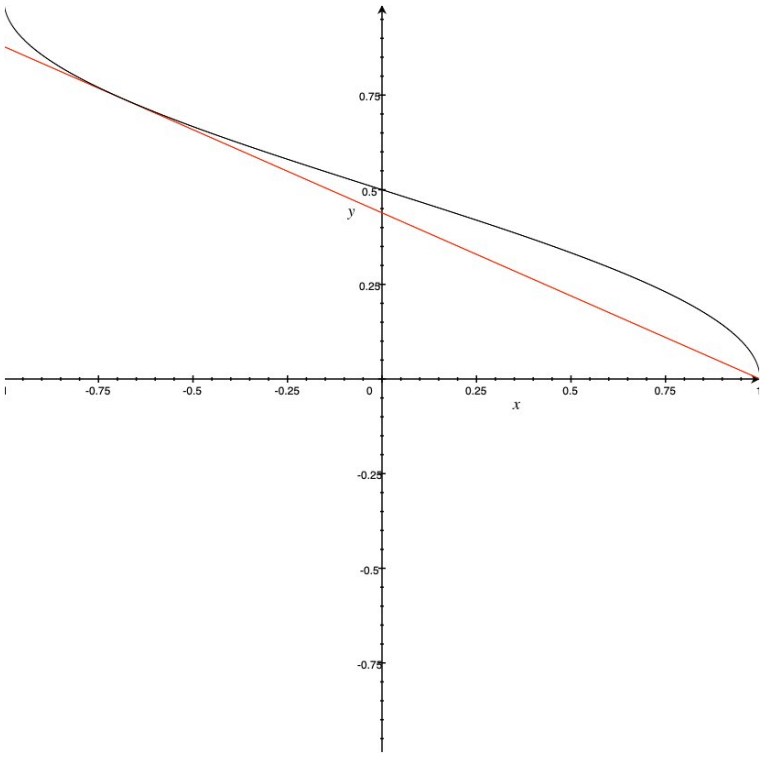

Figure 6.1: inequality for lemma 6.4

Since the angle between two unit vectors v_i and v_j is $v_i^T v_j$, the probability that the hyperplane falls in between them, which is the probability of (i, j) being in the cut, is $\frac{arccos(v_i^T v_j)}{\pi}$.

Lemma 6.4. *For* $x \in [-1, 1]$, $\frac{1}{\pi} arccos(x) \geq 0.878 \cdot \frac{1}{2}(1 - x)$

This is simply an algebraic fact. Figure 6.1 shows that the inequality holds.

Now we can proceed the analysis of approximation ratio of the randomized rounding.

Proof. let X_{ij} be an indicator random variable that is 1 if (i, j) is in the cut and vice versa. define $W = \sum_{(i,j)\in E} w_{ij} X_{ij}$. Therefore the expected value of max cut produced by this rounding algorithm is:

$$E[W] = \sum_{(i,j)\in E} w_{ij} X_{ij} = \sum_{(i,j)\in E} w_{ij} \Pr(\text{edge (i,j) in cut}) = \sum_{(i,j)\in E} w_{ij} \cdot \frac{1}{2} arccos(v_i^T v_j)$$

Now we plug in lemma 6.4:

$$E[W] \geq 0.878 \sum_{(i,j)\in E} w_{ij} \frac{1}{2}(1 - (v_i^T v_j)) = 0.878 \cdot \text{VP-OPT} \geq 0.878 \cdot OPT$$

□