CS880: Approximation and Online Algorithms	**Scribe:** Jeremy
Lecture 13: Semidefinite Programming	**Date:** 10/14/2019

13.1 Introduction

In this lecture we discuss the MaxCut problem and a general technique called Semidefinite programming can used to tackle it. Semidefinite programming is a powerful variation of linear that can be used to approximate very difficult problems.

13.2 MaxCut

Recall in the MaxCut problem, we are given a graph $G = (V, E)$ and edge costs c_e. Our goal is to find a partition $(S, V \setminus S)$ maximizing the cost of the cut, $c(\delta(S))$. Simply flipping a coin for each vertex to determine which side of the cut it will belong to yields a randomized $\frac{1}{2}$-approximation for this problem.

A possible LP for MaxCut is the following:

$$\begin{array}{ll} \max & \sum_e c_e d_e \\ \text{subject to} & d \text{ is a metric} \\ & 0 \leq d_e \leq 1 \end{array}$$

The idea is that we want distance 0 on edges that are not cut and distance 1 on edges that cross the cut. Unfortunately, the constant vector having 1 as all entries is a trivial solution to this program with large value. Thus, we cannot necessarily infer any information on the largest cut from this LP's solution. So, we need a different program formulation.

The next strategy we could take is to add more decision variables that have to give us information on the cut. Consider the following ILP where we let $v_i \in \{-1, 1\}$ denote the which side of the cut that vertex i is on, so $v_i = -1$ indicates $i \in V \setminus S$ and $v_i = 1$ indicates that $i \in S$:

$$\begin{array}{lll} \max & \sum_e c_e d_e & \\ \text{subject to} & \text{if } v_i = v_j, \text{ then } d_{ij} = 0 & \forall (i,j) \in E \\ & \text{if } v_i \neq v_j, \text{ then } d_{ij} = 1 & \forall (i,j) \in E \\ & v_i \in \{-1, 1\} & \forall i \in V \end{array}$$

Note, we can make the if's into linear constraints by using the fact that they are equivalent to the constraint $d_{ij} = \frac{1}{2}[2 - |v_i - v_j|]$, which can then be split into two linear inequalities. Though, the corresponding LP relaxation has a trivial solution of making everything 0.

We need some way of constructing a program that does not have a trivial solution. A natural

approach would be to use the constraint that $v_i^2 = 1$ to enforce $v_i \in \{-1, +1\}$. This then leads to the following *quadratic program*:

$$\begin{array}{lll} \max & \sum_e c_e d_e & \\ \text{subject to} & d_{ij} = \frac{1}{2}[2 - |v_i - v_j|] & \forall i, j \\ & v_i^2 = 1 & \forall i \end{array}$$

However, this program exactly encodes the ILP and so is NP-hard to solve. Whether the feasibility set is convex or not determines solvability of the program. In this case, the feasibility set is a collection of vectors with ± 1 entries, which is not convex.

13.3 Semidefinite Programs

A *Semidefinite Program* (SDP) is like a relaxation of a quadratic program where scalars become vectors and products become dot products. In the case of the quadratic program above, the constraint that $v_i^2 = 1$ turns into a dot product, which is equivalent to requiring that v_i is a unit vector. The resulting SDP is:

$$\begin{array}{ll} \max & \frac{1}{2} \sum_{(i,j) \in E} c_{ij}(1 - v_i v_j) \\ \text{subject to} & v_i\text{s are unit vectors} \end{array}$$

The v_is here have no limit on their dimension, so they need not be scalars, but this program is solvable. Technically, the optimal solution to a SDP can be irrational and so unrepresentable, but we can get solutions with arbitrary precision.

A real-valued square symmetric matrix A is called *positive semidefinite* (PSD), denoted $A \succcurlyeq 0$, if $\forall x, x^T A x \geq 0$. We can view SDPs as LPs that have positive definite constraints. In other words, they are LPs that have additional constraints of the from $A \succcurlyeq 0$ for some real-valued symmetric matrix, A. The entries of A may be variables or constants. Notice that this constraint is equivalent to $\forall x, \sum_{i,j} x_i x_j A_{ij} \geq 0$, which encodes infinitely many simple linear constraints.

For example, consider the constraint $\begin{pmatrix} 1 & -z \\ z & 1 \end{pmatrix} \succcurlyeq 0$. By definition, this means $x_1^2 + x_2^2 - z^2 x_1 x_2 \geq 0$, which ensures that z is not too large.

Theorem 13.3.1 *The following are equivalent:*

1. *A is PSD*
2. *All eigenvalues of A are non-negative*
3. *There is a matrix V s.t. $A = V^T V$.*

For example, V could be a matrix of scaled eigenvectors in the last statement.

If $A_{i,j} = v_i v_j$, then the constraints $A_{ii} = 1$ and $A \succcurlyeq 0$ together ensure that the v_is are unit vectors.

13.4 Geomans-Williamson Rounding

Goemans and Williamson proposed the following randomized rounding of the SDP above. First, pick a random unit vector u. Then, we say i is in S-side of the cut if $v_i \cdot u \geq 0$ and on the $V \setminus S$ side otherwise. In other words, $S = \{i | v_i \cdot u \geq 0\}$.

Note to do this rounding we need a spherically symmetric distribution, so we can use the Guassian with density $\alpha e^{-u_i^2}$ which is proportional to $\prod_i e^{-u_i^2} = e^{-}||u_i||_2^2$.

Now, we need to relate the solution to the value of the cut. By definition, $E[\delta(S)] = \sum_e c_e Pr[e \in \delta(S)]$. We need to determine the probability of some edge e being cut. By construction, $e = (i, j)$ crosses the cut if $v_i \cdot u \geq 0$ and $v_j \cdot u < 0$. In other words, the hyperplane defined by u must pass through the side of the triangle formed by connecting the two heads of the vectors v_i, v_j. If the angle between v_i and v_j is θ, then it must be that out of the total 2π directions that u could have with respect to the plane, u's direction or its negative direction is one of the θ possibilities lying between v_i and v_j. Hence, over the choice of u, the probability that e is cut is precisely $\frac{2\theta}{2\pi} = \frac{\theta}{\pi}$. Since the constraints ensure v_i and v_j are unit vectors, we know that $v_i \cdot v_j = cos(\theta)$, so $\frac{1}{2}(1 - v_i \cdot v_j) = \frac{1-cos(\theta)}{2}$. Hence, $\frac{\theta}{\pi} = \frac{\arccos(v_i \cdot v_j)}{\pi}$. Our approximation factor then depends on

$$\alpha = \min_{x \in [-1,1]} \frac{\arccos(v_i \cdot v_j)}{\pi} \frac{2}{1-x}$$

where $x = v_i \cdot v_j$. Thus, the cost of our rounding is exactly,

$$\begin{aligned}
\sum_e c_e Pr[e \in \delta(S)] &= \sum_e c_e \frac{\arccos(v_i \cdot v_j)}{\pi} \\
&\geq \frac{1}{2} \sum_e c_e \alpha (1 - v_i \cdot v_j) \\
&= \alpha \text{ value of the SDP} \\
&\geq \alpha OPT
\end{aligned}$$

Hence, this rounding gives us an α-approximation for MaxCut where $\alpha \approx .878$. This is the best know approximation factor and it is known that α is also the integrality gap of this program, so no better rounding based on this program can be found. In fact, under the the stronger assumption of the unique games conjecture, no better approximation is possible whatsoever.

13.5 Sparsest Cut

In the BalancedCut Problem the goal is to find a partition $(S, V \setminus S)$ with $|S|, |V \setminus S| \in [\frac{1}{3}, \frac{2}{2}]n$, that minimizes the cost of the cut. This problem has natural uses in divide and conquer algorithms since we want to recurse on instances that are a constant factor of the original instance in order to reduce the amount of work sufficiently. Related is a relaxation called the SparsestCut Problem where we wish to find a partition that minimizes $\frac{c(\delta(S))}{|S||V \setminus S|}$. Note, the denominator is maximized when $|S| = |V \setminus S|$, so this objective function is biased towards cuts that are fairly balanced. In

the Non-UniformSparsestCut Problem, we also have demands $d_{u,v}$ and we wish to minimize $\frac{c(\delta(S))}{d(S,V\setminus S)}$. The optimal solutions to these problems will always be cuts into 2 pieces, making more fine grained cuts doesn't help. Next time, we show how to use tree embeddings to approximate SparsestCut.