CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 17: MAX-CUT: Approximation and Hardness

Nov. 29, 2005

Lecturer: Venkatesan Guruswami *Scribe: Vibhor Rastogi*

1 Overview

We begin by looking at algorithms for approximating the MAX-CUT problem. In particular, we will study the Goemans-Williamson's algorithm which provides the best known approximation result for MAX-CUT. In the end, we will see a 2 query long-code test which will be used in the next lecture to prove hardness results for MAX-CUT.

2 Problem Definition

Definition 2.1. *:* MAX-CUT *is the following graph problem: Given a graph $G = (V, E)$ and a weight function $w : E \mapsto R^+$ such that each edge (u, v) in the graph has a weight w_{uv}. Find a cut $S \cup \overline{S} = V$ that maximizes*

$$\sum_{u \in S, v \in \overline{S}, (u,v) \in E} w_{uv}$$

As an example, we note that the bipartite graph has maxcut exactly equal to the sum of the weights of all its edges. A particular case of the MAX-CUT problem occurs when all the edge weights are reduced to unity. In that case it becomes equivalent to the MAX-2COLORING problem. In the MAX-2COLORING problem, we try to find the maximum number of edges in a given graph G which can be colored by using only two colors.

3 Simple Approximation Algorithms

There is a simple $\frac{1}{2}$-approximate randomized algorithm for MAX-CUT which works by choosing a cut at random. This means that each edge (u, v) in G is cut with probability $\frac{1}{2}$. Thus the expected weight of the edges crossing the cut, $A(I)$, is

$$A(I) = \frac{\sum_{(u,v) \in E} w_{uv}}{2} \geq \frac{OPT(I)}{2}$$

The above algorithm can be easily derandomized to get a $\frac{1}{2}$-approximate deterministic polyonimal time algorithm for MAX-CUT.

For the unweighted version of MAX-CUT i.e. when all the edges have unit weights, there is a simple $\frac{1}{2}$-approximate algorithm. The algorithm follows a method of iteratively updating the cut.

1. Initialize the cut $S \cup \overline{S}$ arbitrarily.
2. For any vertex v which has less than $\frac{1}{2}$ of its edges crossing the cut, we move the vertex to the other side of the cut.
3. If no such vertex exists then stop
   else goto step 2

It is easy to see that in each iteration, the edges crossing the cut increase by at least one. Since maxcut is atmost the number of edges in the graph, the algorithm would terminate in linear number of iterations. It is also easy to see that when algorithm terminates the cut would have size ateast half of the total number of edges. Thus the algorithm is $\frac{1}{2}$-approximate. However, the algorithm does not genrealize well to the weighted version of MAX-CUT as it may no longer run in polynomial time in the size of the inputs.

It can also be shown that a ratio of more than $\frac{1}{2}$ cannot be achieved if we use $\sum_{(u,v)\in E)} w_{uv}$ as the upper bound for the maxcut. Complete graphs are an example of graphs in which the maxcut is half the size of the upperbound. The ratio $\frac{1}{2}$ remained as the best known for quite a long time until Goemans-Williamson found an algorithm that obtained a much better approximation ratio.

4 Goemans-Williamson Algorithm

Goemans-Williamson algorithm provides the best known approximation result for MAX-CUT. Another improtant aspect of the algorithm lies in the first ever use of semi-definite programming for approximation. Since then semi-definite programming has been applied to obtain good approximation algorithms for many other important problems.

4.1 MAX-CUT as Quadratic Programming problem

Associate with each vertex v_i a variable x_i which takes value 1 or -1. Let $S \cup \overline{S}$ be any arbitrary cut. We define $x_i = 1\ \forall v_i \in S$ and $x_i = -1\ \forall v_i \in \overline{S}$. Note that $x_i x_j$ is -1 iff (v_i, v_j) belongs to the cut. Thus $\frac{(1-x_i x_j)}{2}$ is 1 if (v_i, v_j) belongs to the cut and 0 otherwise. Thus the following quadratic program is equivalent to MAX-CUT problem.

$$\begin{aligned} \text{MAX} \sum_{(v_i,v_j)\in E} w_{ij} \frac{(1 - x_i x_j)}{2} \\ \text{subject to} \quad x_i \in \{-1, 1\}\ \forall v_i \in V \end{aligned}$$

4.2 Relaxation to Semi-Definite Program

A semi-definite program is a quadratic program with the following properties

- Each variable is a vector u_i
- Constraints are defined on linear combination of pairwise dot products i.e. $u_i \cdot u_j$
- Objective function is also a linear combination of $u_i \cdot u_j$

For the quadratic program defined above, we form a semi-definite program by relaxing each variable x_i to a vector u_i of n dimensions.

$$\begin{aligned} \text{MAX} \sum_{(u_i,u_j)\in E} w_{ij}\frac{(1-u_i\cdot u_j)}{2} \\ \text{subject to} \quad u_i \cdot u_i = 1 \end{aligned}$$

The above SDP can be solved in polynomial time to any degree of accuracy. We represent the maximum obtained by the SDP as opt_{sdp} and use it as an upperbound for maxcut. This is beacuse $opt_{sdp} \geq opt_{\text{MAX-CUT}}$

4.3 Rounding and Analysis

The solution for the SDP would be a collection of vectors u_i in n-dimensions. By rounding, we mean a method of extracting a solution to MAX-CUT. We will look at such a method and analyze it to get a bound on the MAX-CUT solution.

Method: Choose a random hyperplane in n dimensions passing through the origin. It divides the n-dimensional space into two half-spaces. Each half-space corresponds to one particular side of the cut. Vectors in a particular half-space denote the vertices of the original graph which belong to that side of the cut. This uniquely defines the cut. More formally, if r represents the normal to the randomly chosen hyperplane, we define the cut $S \cup \overline{S}$ as

$$\begin{aligned} S &= \{v_i \mid r \cdot u_i \geq 0\} \\ \overline{S} &= \{v_i \mid r \cdot u_i < 0\} \end{aligned}$$

Analysis: Consider any two vectors u_i and u_j. We want to estimate the probability (v_i, v_j) being cut. Define r' as the projection of r on the 2-dimensioanl plane span(u_i, u_j). If r was chosen randomly using a uniform distribution, it can be proved that direction of r' is uniform on the circle in span(u_i, u_j). The probability that r' cuts the vectors u_i and u_j depends on the angle θ between the two vectors. It is easy to see that this probability is exactly equal to $\frac{2\theta}{2\pi} = \frac{cos^{-1}(u_i \cdot u_j)}{\pi}$. Thus the

expected value of the cut is

$$\begin{aligned} E &= \sum_{(v_i,v_j)\in E} w_{ij}\mathbf{Pr}\{(v_i,v_j)\ is\ cut\} \\ &= \sum_{(v_i,v_j)\in E} w_{ij}\frac{cos^{-1}(u_i\cdot u_j)}{\pi} \\ &\geq \sum_{(v_i,v_j)\in E} w_{ij}\alpha_{GW}\frac{(1-u_i\cdot u_j)}{2} \\ &= \alpha_{GW}\sum_{(v_i,v_j)\in E} w_{ij}\frac{(1-u_i\cdot u_j)}{2} \\ &= \alpha_{GW}opt_{sdp} \\ &\geq \alpha_{GW}opt_{\text{MAX-CUT}} \end{aligned}$$

where we define

$$\alpha_{GW} = \min_{-1\leq\rho\leq 1}\frac{\frac{cos^{-1}\rho}{\pi}}{\frac{(1-\rho)}{2}} = 0.87856$$

The minimum value of $\frac{2cos^{-1}\rho}{\pi(1-\rho)}$ occurs at the crictical value $\rho = \rho^* \approx -0.694$ (whic is roughly the cosine of 134 degrees). Thus we have seen a randomized algorithm for approximating MAX-CUT with a ratio of $\alpha_{GW} = 0.87856$.

5 Hardness of Approximating MAX-CUT

The best known hadrness result for MAX-CUT shows that there is no polynomial time algorithm which can approximate it within $(16/17+\epsilon)$ of the optimum unless $P = NP$.
However, under the assumption of Unique Games Conjecture it can be proved that the problem $\text{GAPMAX-CUT}_{\frac{1-\rho}{2}-\epsilon,\frac{cos^{-1}\rho}{\pi}+\epsilon}$ is NP hard for every $\rho\in(-1,0)$ and every ϵ. Thus picking $\rho=\rho^*$, we get that there is no $(\alpha_{GW}+\epsilon)$-approximate algorithm for MAX-CUT unless $P = NP$.
To prove this result it suffices to show that there is a 2-query PCP system with alphabet size two, completeness $c = \frac{1-\rho}{2}-\epsilon$ and soundness $s = \frac{cos^{-1}\rho}{\pi}+\epsilon$ in which the verifier makes only not-equal checks. This is because NP-hardness of $\text{GAPMAX-CUT}_{\frac{1-\rho}{2},\frac{cos^{-1}\rho}{\pi}+\epsilon}$ and the existence of such a PCP system are equivalent.
For constructing such a PCP system we shall first develop a 2-query long-code test that makes checks of the form $b_1 \neq b_2$ only.

5.1 2-query long-code test

Given a function $f : \{-1,1\}^n \mapsto \{-1,1\}$ we want to check whether f is a long code of some $a\in\{1,2,..,n\}$. Suppose we did the test by picking two strings x, y such that $x_i \neq y_i$ for every i,

i.e., $y = -x$, and check whether $f(x) \neq f(y)$. It is clear that the completeness for such a test is 1. However, along with long codes the following functions also pass the test with probability 1:

1. $f(x) = \chi_S(x)$ where $|S|$ is odd.
2. f is the Majority function, i.e., $f(x) = \text{sgn}(\sum_{i=1}^{n} x_i)$, (assume n is odd).

Thus the test is unable to disinguish between long codes and the above functions. A much better test albeit with completeness less than one uses a slightly different version of this test. The test picks a string $x \in \{-1, 1\}^n$ at random and checks $f(x) \neq f(x\mu)$ where μ is a string $\in \{-1, 1\}^n$. μ is chosen as follows

$$\begin{aligned} \mu_i &= -1 \quad \text{with probability } \tfrac{1-\rho}{2} \\ &= 1 \quad \text{with probability } \tfrac{1+\rho}{2} \end{aligned}$$

Thus the expected value of μ_i for each i is ρ.
Completeness: Suppose f is a long code function of some $a \in \{1, 2, \ldots, n\}$. This implies $f(x) = x_a$ and $f(x\mu) = x_a\mu_a$. Thus

$$\begin{aligned} \mathop{\mathbf{Pr}}_{x,\mu}[f(x) \neq f(x\mu)] &= \mathop{\mathbf{Pr}}_{x,\mu}[x_a \neq x_a\mu_a] \\ &= \mathop{\mathbf{Pr}}_{\mu}[\mu_a = 1] \\ &= \frac{1-\rho}{2} \end{aligned}$$

Soundness: Using the standard arithmetization,

$$\begin{aligned} \mathbf{Pr}[\text{Test accepts } f] &= \mathop{\mathbf{E}}_{x,\mu}[\frac{1 - f(x)f(x\mu)}{2}] \\ &= \frac{1}{2} - \frac{1}{2}\mathop{\mathbf{E}}_{x,\mu}[f(x)f(x\mu)] \\ &= \frac{1}{2} - \frac{1}{2}\text{Stab}_\rho(f) \end{aligned}$$

The quantity $\mathbf{E}_{x,\mu}[f(x)f(x\mu)]$ with μ distributed as above is called the stability of the function f at parameter ρ, and denoted $\text{Stab}_\rho(f)$. It is easy to show, and this was on your problem set, that $\text{Stab}_\rho(f) = \sum_S \hat{f}(S)^2\rho^{|S|}$. If $f = \chi_S$, then $\text{Stab}_\rho(f) = \rho^{|S|}$, which tends to 0 if $|S|$ is large when $0 > \rho > -1$. Therefore, the modified test is able to distinguish between long codes and linear functions of large support. So we have at least taken care of the first of the two bad functions mentioned above (for the test $f(x) \neq f(-x)$). The majority function remains to be contended with. In this case, one can show (and we will sketch this in the next lecture), that the test passes with probabilty tending to $\cos^{-1}(\rho)/\pi$ (as $n \to \infty$). By appealing to a result called the "Majority is Stablest theorem", we will also conclude that in fact the Majority is essentially the worst function for this test. Specifically, we will conclude that any function that passes the test with probability $\cos^{-1}(\rho)/\pi + \epsilon$ "looks like" (in a sense that will be made precise) one of a small number of long codes. Thus, the 2-query long code test with parameter ρ has completeness $(1-\rho)/2$ and soundness $\cos^{-1}(\rho)/\pi + \epsilon$, which matches the parameters we are ultimately after for our 2-query PCP.