Limits of Approximation Algorithms 11 Feb, 2010 (TIFR)

Lec. 3: MAXCUT and Introduction to Inapproximability

Lecturer: Prahladh Harsha *Scribe: Bodhayan Roy*

In the first part of today's lecture, we will see the application semi-definite programming (SDP) towards approximation algorithms. More specifically, we will look at the MAXCUT problem and then design LP-based and SDP-based approximation algorithms for the same. In the second part of today's lecture, we will begin the discussion of inapproximability problems, introduce GAP problems and show how GAP problems capture the hardness of approximation. We will conclude by stating the PCP theorem in terms of NP-hardness of GAP problems.

The references for this lecture include Lecture 2 of the DIMACS tutorial on Limits of a pproximation [HC09], Lectures 7 and 8 of Sudan's course on inapproximability at MIT [Sud99], and Lecture 1 from a course on PCPs theorem at the University of Chicago [Har07]. In fact, parts of these scribe notes are abstracted from the scribe notes for Lecture 2 of the DIMACS tutorial (scribed by Darakhshan Mir) and Lecture 1 from the PCP course at the Univ. of Chicago (scribed by Josh Grochow).

3.1 MAXCUT

The MAXCUT problem is defined as follows:

Definition 3.1.1 (MAXCUT)**.**

Input: *An undirected graph* $G = (V, E)$

Output: *A cut* $(S, \overline{S})$*, where* $S \subseteq V$

Objective: *Maximize the number of edges across the cut. More formally, maximize* $|E(S, \overline{S})|$ *where*

$$E(S, \overline{S}) = \{(u, v) \mid u \in S, v \in \overline{S}\}$$

We observed in last lecture a simple greedy 2-approximation algorithm for MAXCUT.

Greedy algorithm: Start with any partition of V into S and $\overline{S}$. If a vertex contributes more edges to the cut when moved from S to $\overline{S}$ or vice-versa, then move it. Each such step increases the cut by at least one edge and since the cut size is at most $(|E|)$, this process terminates in E steps. Furthermore, for each vertex, more than half of the edges incident to it are in the cut. Hence, at least half of all the edges in the graph are in the cut. Thus, we get a 2-approximation algorithm.

We can also emulate the randomized rounding technique from last time (applied to MAXSAT) to get an alternate 2-approximation algorithm.

Randomized rounding: For each $u \in V$, flip a coin, if heads (H), $u \in S$, else if tails (T), $u \notin S$. It is easy to see from below that each edge is cut with probability 1/2.

$$\Pr[(u,v) \in E(S,\overline{S})] = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{1}{2}$$

Thus, the expected size of the cut $\mathbb{E}\left[|E(S,\overline{S})|\right] = \sum_{(u,v)\in E} \Pr[(u,v) \in E(S,\overline{(S)})] = |E|/2$. This gives us a 2-approximation algorithm. Note that this is a randomized algorithm and the above analysis only shows that the expected size of the cut is $|E|/2$. But, we could repeat this experiment several times and choose the largest cut. By Markov's inequality, this cut will be close to expected value with high probability.

3.1.1 LP-based approximation for MAXCUT

We will now design a LP-based approximation algorithm for MAXCUT. Recall from last time, that we gave a LP-based algorithm from MAXSAT that achieved $\frac{1}{1-\frac{1}{e}}$ approximation. Combining this with the vanilla randomized rounding algorithm, we obtaine $\frac{4}{3}$-approximation for MAXCUT. Let us see, if we can achieve a similar improvement for MAXCUT. For this, we first design an integer program for MAXCUT and then relax it to get a LP.

Integer Program Version Define variables $x_u, u \in V$ and $e_{uv}, (u,v) \in E$ as follows which are supposed to imply the following.

$$e_{uv} \longleftarrow \begin{cases} 1 & \text{if } (u,v) \text{ is in cut} \\ 2 & \text{otherwise} \end{cases}.$$

$$x_u \longleftarrow \begin{cases} 1 & \text{if } u \in S \\ 0 & \text{otherwise} \end{cases}$$

MAXCUT can now be phrased as the following integer program.

$$\begin{aligned} \textbf{Maximize} \quad & \sum_{(u,v)\in E} e_{uv} \\ \textbf{subject to} \quad & e_{uv} \leq \begin{cases} x_u + x_v \\ 2 - (x_u + x_v) \end{cases}, \qquad \forall (u,v) \in E \\ & x_u \in \{0,1\}, \qquad \forall u \in V \\ & e_{uv} \in \{0,1\}, \qquad \forall (u,v) \in E \end{aligned}$$

Notice that $e_{uv} \neq 0 \iff x_u \neq x_v$.

LP relaxation for MAXCUT: We now relax $e_{uv} \in \{0,1\}$ to $0 \leq e_{uv} \leq 1$ and $x_{uv} \in \{0,1\}$ to $0 \leq x_{uv} \leq 1$ to obtain the following LP relation.

$$\begin{aligned}
\textbf{Maximize} \quad & \sum_{(u,v)\in E} e_{uv} \\
\textbf{subject to} \quad & e_{uv} \leq \begin{cases} x_u + x_v \\ 2 - (x_u + x_v) \end{cases}, \qquad \forall (u,v) \in E \\
& x_u \in [0,1], \qquad \forall u \in V \\
& e_{uv} \in [0,1], \qquad \forall (u,v) \in E
\end{aligned}$$

We can now solve the above LP using an LP-solver. Every solution to the Integer Program is also a solution to the Linear Program. So the objective function will only rise. If $\mathrm{OPT}_{\mathrm{LP}}$ is the optimal solution to the LP, then:

$$\mathrm{OPT}_{\mathrm{LP}} \geq \mathrm{MAXCUT}(G)$$

Rounding the LP solution: We now round the LP-solution to obtain an integral solution as follows: for each $u \in V$, flip a coin with Heads with proability x_u and Tails with $1 - x_u$. If Heads then set $u \in S$, and if Tails set $u \in \overline{S}$. The expected number of edges in such a cut, $\mathbb{E}[|E(S, \bar{S})|]$ can be then calculated as follows:

$$\begin{aligned}
\mathbb{E}[|E(S,\bar{S})|] &= \sum_{(u,v)\in E} Pr[(u,v) \text{ is in the cut}] \\
&= \sum_{(u,v)\in E} x_u(1 - x_v) + x_v(1 - x_u)
\end{aligned}$$

On the other hand, the LP-optimal solution is given by

$$\mathrm{OPT}_{\mathrm{LP}} = \sum_{(u,v)\in E} e_{uv} = \sum_{(u,v)\in E} \min\{x_u + x_v, 2 - (x_u + x_v)\}$$

But for all $x_u, x_v \in [0,1]$, we have

$$x_u(1 - x_v) + x_v(1 - x_u) \geq 2 \min\{(x_u + x_v), 2 - (x_u + x_v)\}.$$

Thus

$$\mathbb{E}\left[|E(S,\overline{S})|\right] \geq \frac{1}{2}\mathrm{OPT}_{\mathrm{LP}} \geq \frac{1}{2}\mathrm{MAXCUT}(G).$$

We thus, have a 1/2-approximation algorithm for MAXCUT using randomized rounding of the LP-relaxation of the problem. Actually, it is to be noted that the LP-relaxation is pretty stupid, the optimal to the LP is the trivial solution $x_i = 1/2$ for all i, which in turn leads to $OPT_{LP} = |E|$. But we do mention this example as it naturally leads to the following more powerful SDP relaxation.

3.1.2 Semi-definite programming (SDP)

Recall from last lecture the definition of LP.

$$\begin{aligned} \textbf{Maximize} \quad & \sum_i c_i x_i \\ \textbf{subject to} \quad & \sum_i a_i^{(k)} x_i \leq b^{(k)}, \qquad k = 1, \ldots, m \\ & x_1, \ldots, x_n \in \mathbb{R} \end{aligned}$$

It is known that LPs can be solved in time polynomial in n, m and l, where $l = \#$ bits required to represent the elements $c_i, a_i^{(k)}, b^{(k)}$.

Semi-definite programs are a special case of LPs in which the vector of variables x, now thought of as a matrix is further constrained to be a positive semi-definite matrix. Instead of defining a positive semi-definite matrix, we instead adopt the following (equivalent) alternative definition of SDPs in terms of vector dot-products. An SDP is of the following form.

$$\begin{aligned} \textbf{Maximize} \quad & \sum_{ij} c_{i,j} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \\ \textbf{subject to} \quad & \sum_{i,j} a_{ij}^{(k)} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \leq b^{(k)}, \qquad k = 1, \ldots, m \\ & \mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^n \end{aligned}$$

Thus, SDPs are LPs in which the variables are constrained to arise as the dot products of vectors. Note that SDPs can in principle have irrational solutions. Hence, one can expect to solve them optimally. However, it is known how to obtain arbitrarily good approximations of the SDPs. More precisely, It is known that a SDPs can be solved within accuracy ε in time polynomial in n, m, l and $1/\varepsilon$ where $l = \#$ bits required to represent the elements of $a_{ij}^{(k)}, b^{(k)}, c_{ij}$. As in the case of LPs, we will use these SDP-solvers as black-boxes while designing approximation algorithms.

3.1.3 SDP-based approximation for MAXCUT

We will now sketch a 0.878-approximation to MAXCUT due to Goemans and Williamson [GW95]. The main idea is to relax the integer problem defined above using vector valued variables. For this we first introduce semi-definite programming.

$$\begin{aligned} \textbf{Maximize} \quad & \sum_{(u,v) \in E} \frac{(1 - \langle \mathbf{x}_u, \mathbf{x}_v \rangle)}{2} \\ \textbf{subject to} \quad & \langle \mathbf{x}_u, \mathbf{x}_u \rangle = 1, \ \forall i \end{aligned}$$

Denote the optimal to the above SDP by $\mathrm{OPT}_{\mathrm{SDP}}$. We first observe that the SDP is in fact a relaxation of the integral problem. Let $\mathbf{x}_0$ be any vector of unit length, i.e., $\langle \mathbf{x}_0, \mathbf{x}_0 \rangle = 1$. Consider the optimal cut S that achieves MAXCUT. Now define,

$$\mathbf{x}_u = \begin{cases} \mathbf{x}_0 & \text{if} \quad i \in S \\ -\mathbf{x}_0 & \text{if} \quad i \notin S \end{cases}, \forall i.$$

Consider the quantity $\frac{(1-\langle \mathbf{x}_u, \mathbf{x}_v \rangle)}{2}$. This is 0 if the vectors $\mathbf{x}_u$ and $\mathbf{x}_v$ lie on the same side, and equals 1 if they lie on opposite sides. Thus, $\text{OPT}_{\text{SDP}} \geq \text{MAXCUT}$.

How do we round the SDP solution to obtain an integral solution. The novel rounding due to Goemans and Williamson is as follows: The SDP solution produces $n = |V|$ vectors $\mathbf{x}_u, u \in V$. Now pick a random hyperplane passing through the origin of the sphere and partition vectors according to which side tof the hyperplane they lie. Let $(S, \bar{S})$ be the cut obtained by the above rounding scheme. It is easy to see that

$$\begin{aligned} \mathbb{E}[|E(S,\bar{S})|] &= \sum_{(u,v)\in E} \Pr[(u,v) \in \text{cut}] \\ &= \sum_{(u,v)\in E} \Pr[\mathbf{x}_u, \mathbf{x}_v \text{ lie on opposite sides of the hyperplane}] \end{aligned}$$

Let θ_{uv} be the angle between vectors $\mathbf{x}_u$ and $\mathbf{x}_v$. Then the probability that they are cut is proportional to θ_{uv}, in fact exactly θ_{uv}/π. Thus,

$$\mathbb{E}[|E(S,\bar{S})|] = \sum_{(uv)\in E} \frac{\theta_{uv}}{\pi}$$

Let us know express OPT_{SDP} in terms of the θ_{uv}'s. Since $\theta_{uv} = \cos^{-1}(\langle \mathbf{x}_u, \mathbf{x}_v \rangle)$, we have

$$\text{OPT}_{\text{SDP}} = \sum_{(u,v)\in E} \frac{(1-\cos\theta_{uv})}{2}$$

By a "miracle of nature"(Mathematica?) Goemans and Williamson observed that

$$\frac{\theta}{\pi} \geq (0.878\ldots) \times \frac{1-\cos\theta}{2}, \ \forall \theta \in [0,\pi]$$

Hence,

$$\frac{\mathbb{E}[|E(S,\bar{S})|]}{\text{OPT}_{\text{SDP}}} \geq 0.87856\ldots.$$

Theorem 3.1.2. *$\forall \varepsilon > 0$, MAXCUT is $\left(\frac{1}{0.87856\ldots} + \varepsilon\right)$-approximable.*

3.2 Limits of Approximation – Introduction

We saw in the last few lectures, that combinatorial optimization problems, even though of similar hardness with respect to finding the optimal solution (in the sense that they all were all NP-complete) displayed an entire spectrum of hardness with respect to finding approximate solutions. For instance, we observed the following behavior.

FPTAS: The class of problems that were $(1+\varepsilon)$-approximable in time $\text{poly}(n, 1/\varepsilon)$ for every $\varepsilon > 0$, eg: KNAPSACK.

PTAS: The class of problems that were $(1+\varepsilon)$-approximable in time $\text{poly}_{\varepsilon}(n)$ for every $\varepsilon > 0$, eg: MIN-MAKESPAN is $(1+\varepsilon)$-approximable in time poly$n^{\frac{1}{\varepsilon}}$.

APX: The class of problems that allowed for constant factor approximation, eg: VERTEX-COVER, MAXSAT, MAXCUT.

log-APX: The class of problems that allowed for logarithmic factor approximation, eg: SET COVER.

poly-APX: And finally, the set of problems that allowed for only polynomial factor approximation, eg: COLOURING, CLIQUE.

Clearly, FPTAS $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ log-APX $\subseteq$ poly-APX. Why do problems display such a rich variety when it comes to how well they can be approximated? Does the fact that there exists a good approximation for one problem imply a good approximation for another. Like optimization problems, do approximation problems have their own hierarchy of hardness? The complexity of approximation was not well understood. In fact, even in the algorithmic world, approximation algorithms was not a major topic till the 80's. For instance, the 1985 book *Combinatorial Optimization: Annotated Bibliographies* did not have a chapter on approximation problems.

The situation changed dramatically with the paper of Papadimitriou and Yannakakis, "Optimization, Approximation and Complexity Classes" [PY91] in 1988 which initated the study of approximation problems from a complexity standpoint. They extended the notion of reductions to define approximation preserving reductions and defined a new class of approximation problems called MAX-SNP for which MAXSAT was the typical complete problem in the sense that if it had a PTAS, then the entire class of MAX-SNP had PTAS. Several other constant-factor approximable problems in APX were also shown to be MAX-SNP complete. However, even then, the assumption "hard to approximate" was considered to be a much stronger hypothesis than "hard to compute exactly" (ie., NP $\neq$ P). The seminal paper of Feige, Goldwasser, Lovász, Safra and Szegedy [FGL+96] in 1992 which established the dramatic connection between the ongoing work in probabilistic proof systems and hardness of approximation showed that for some problems these two assumptions might in fact be equivalent! This led to the birth of the field – limits of approximability.

3.2.1 GAP problems

How does one study the hardness of approximation problems? For this, it is instructive to recall how the hardness of optimization problems was studied in the theory of NP-completeness. For instance, suppose we wished to study the hardness of problems such as:

- $MAXCLIQUE$: Given a graph G, output the vertices in its largest clique.
- 3CNF satisfiability: Given a 3CNF Boolean formula, output a satisfying assignment if one exists. A related problem is that of $MAX3SAT$: Given a 3CNF Boolean formula, output an assignment which satisfies the maximum number of clauses.
- 3-COLOR: Given a graph G, color the vertices with 3 colors such that no edge is monochromatic if such a coloring exists.

- SET COVERING PROBLEM: Given a collection of sets $S_1, S_2, \ldots, S_m$ that cover a universe $U = \{1, 2, \ldots, n\}$, find the smallest sub-collection of sets $S_{i_1}, S_{i_2}, \ldots$ that also cover the universe
- ...

Rather than studying these problems directly, it is often convenient to study polynomially equivalent *decision problems*, i.e. computational problems whose output is always a simple "yes" or "no." For example, the decision problem equivalents of the above problems are:

- $CLIQUE = \{\langle G, k\rangle | \text{graph } G \text{ has a clique of size } \geq k\}$
- $3SAT = \{\varphi | \varphi \text{ is satisfiable}\}$
- SET-COVER $= \{\langle U; \{S_1, \ldots, S_m\}, k\rangle | \exists 1 \leq i_1 \leq i_2 \leq \ldots, i_k \leq m, \text{ such that } \bigcup_{j=1}^{m} S_{i_j} = U\}$
- ...

As in the case of computational problems, it would be nice if we could capture the hardness of the approximation problems via decision problems. The analogue of decision problems for approximation algorithms are known as *gap problems*. Gap problems are promise-problems.

Definition 3.2.1 (promise problems)**.** *A promise problem $\Pi \subseteq \Sigma^*$ is specified by a pair of sets $(\mathsf{YES}, \mathsf{NO})$ such that $\mathsf{YES}, \mathsf{NO} \subseteq \Sigma^*$ and $\mathsf{YES} \cap \mathsf{NO} = \emptyset$.*

Whereas a decision problem specifies a set of "yes" instances – and thus implicitly specifies that all other instances are "no" instances – a promise problem explicitly specifies both the "yes" instances YES and the "no" instances NO. Obviously we require $\mathsf{YES} \cap \mathsf{NO} = \emptyset$, but – unlike in decision problems – we do not require that $\mathsf{YES} \cup \mathsf{NO}$ covers all instances of the problem. The instances neither in YES nor NO are called *don't-care instances*. An algorithm is said to solve a promise problem $(\mathsf{YES}, \mathsf{NO})$ if it outputs "yes" on all $x \in \mathsf{YES}$ and "no" on all $x \in \mathsf{NO}$, and we don't care what the algorithm says on other instances x. Thus, if an algorithm solves a gap problem and outputs "yes" on input x, all we can conclude in general is that $x \notin \mathsf{NO}$, since x might be a "don't care" instance.

Gap problems are best described by example. For instance, the gap problem corresponding to the $1/\alpha$-approximating MAX3SAT, called gap$_\alpha$-MAX3SAT (for $\alpha \leq 1$) is defined as follows. gap$_\alpha$-MAX3SAT is a promise problem whose $(\mathsf{YES}, \mathsf{NO})$ are as follows:

$$\begin{aligned} \mathsf{YES} &= \{\langle \varphi, k\rangle | \text{there is an assignment satisfying } \geq k \text{ clauses of } \varphi\} \\ \mathsf{NO} &= \{\langle \varphi, k\rangle | \text{every assignment satisfies } \leq \alpha k \text{ clauses of } \varphi\} \end{aligned}$$

where φ is a 3CNF formula and k any positive integer.

As promised, approximation problems are polynomially equivalent to gap problems. We show this in the case of MAX3SAT below.

Proposition 3.2.2. *For any $0 < \alpha < 1$, $1/\alpha$-approximating MAX3SAT is polynomially equivalent to solving gap$_\alpha$-MAX3SAT.*

Proof. ($\Rightarrow$) Suppose there is an α-approximation algorithm A to MAX3SAT. Then, consider the following algorithm B for gap$_\alpha$-MAX3SAT.

B : "On input $\langle \varphi, k \rangle$
1. Run A on φ and let $k' = A(\varphi)$.
2. Accept iff $k' \geq \alpha k$. "

B solves gap$_\alpha$-MAX3SAT: For if $k' \geq \alpha k$, then there must be some assignment satisfying at least αk clauses, so $\varphi \notin \mathsf{NO}$ and the algorithm outputs "yes." Conversely, if $\alpha k > k'$, then since $k' \geq \alpha OPT(\varphi)$ it is the case that $k > OPT(\varphi)$, so there is no assignment satisfying at least k clauses. Thus $\varphi \notin \mathsf{YES}$ and the algorithm outputs "no."

($\Leftarrow$) Suppose instead there is an algorithm B that solves gap$_\alpha$-MAX3SAT. Then

A : "On input φ
1. Let m be the number of clauses of φ.
2. Run B on $\langle \varphi, 1 \rangle, \langle \varphi, 2 \rangle, \langle \varphi, 3 \rangle, \ldots, \langle \varphi, m \rangle$.
3. Let the largest k such that B accepted $\langle \varphi, k \rangle$
4. Output αk "

is an $1/\alpha$-approximation to MAX3SAT. For if B rejects $\langle \varphi, k+1 \rangle$, we know it is *not* a YES instance, so φ cannot have any assignment satisfying strictly more than k clauses, i.e. $k \geq OPT(\varphi)$, or, multiplying both sides by α, $\alpha k \geq \alpha OPT(\varphi)$. But since B accepted $\langle \varphi, k \rangle$, there must be some assignment to φ satisfying at least αk clauses, i.e. $OPT(x) \geq \alpha k$. Thus $OPT(x) \geq \alpha k \geq \alpha OPT(x)$. □

Thus, to show that approximating MAX3SAT is hard, it suffices (and is necessary) to show that gap$_\alpha$-MAX3SAT is hard. And this is precisely what the PCP Theorem demonstrates.

3.2.2 The PCP Theorem

One formulation of the PCP Theorem states that gap$_\alpha$-MAX3SAT is NP-hard for some $\alpha \in (0, 1)$.

Theorem 3.2.3 (PCP Theorem [AS98, ALM+98])**.** *There exists $0 < \alpha < 1$ such that 3SAT is polynomial time reducible to gap$_\alpha$-MAX3SAT, i.e. there is some polynomial time reduction $R : \{3CNF\} \to \{3CNF\} \times \mathbb{N}$, such that*

$$\begin{aligned} \psi \in 3SAT &\Rightarrow R(\psi) = \langle \varphi, k \rangle \in \mathsf{YES} \\ \psi \notin 3SAT &\Rightarrow R(\psi) = \langle \varphi, k \rangle \in \mathsf{NO}. \end{aligned}$$

Or equivalently, gap$_\alpha$-MAX3SAT is NP-hard.

An immediate corollary of the above theorem is the following hardness of approximation fo MAX3SAT.

Corollary 3.2.4. *There exists $\beta > 1$ such that β-approximating MAX3SAT is NP-hard.*

References

[ALM+98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. *Proof verification and the hardness of approximation problems.* J. ACM, 45(3):501–555, May 1998. (Preliminary Version in *33rd FOCS*, 1992). eccc:TR98-008, doi:10.1145/278298.278306.

[AS98] Sanjeev Arora and Shmuel Safra. *Probabilistic checking of proofs: A new characterization of NP.* J. ACM, 45(1):70–122, January 1998. (Preliminary Version in *33rd FOCS*, 1992). doi:10.1145/273865.273901.

[FGL+96] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. *Interactive proofs and the hardness of approximating cliques.* J. ACM, 43(2):268–292, March 1996. (Preliminary version in *32nd FOCS*, 1991). doi:10.1145/226643.226652.

[GW95] Michel X. Goemans and David P. Williamson. *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming.* J. ACM, 42(6):1115–1145, 1995. (Preliminary version in *26th STOC*, 1994). doi:10.1145/227683.227684.

[Har07] Prahladh Harsha. *CMSC 39600: PCPs, codes and inapproximability*, 2007. A course on PCPs at the University of Chicago (Autumn 2007).

[HC09] Prahladh Harsha and Moses Charikar. *Limits of approximation algorithms: PCPs and unique games*, 2009. (DIMACS Tutorial, July 20-21, 2009). arXiv:1002.3864.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis. *Optimization, approximation, and complexity classes.* J. Computer and System Sciences, 43(3):425–440, December 1991. (Preliminary Version in 20*th STOC*, 1988). doi:10.1016/0022-0000(91)90023-X.

[Sud99] Madhu Sudan. *6.893: Approximability of optimization problems*, 1999. (A course on Approximability of Optimization Problems at MIT, Fall 1999).