Geometric Complexity and Applications CIS 6930 September 9-26, 2008

Lecture on PCP theorem and approximability

Lecturer: Venkatakrishnan Ramaswamy Scribe: Venkatakrishnan Ramaswamy

1 Introduction

In the first part of these notes, we study *probabilistically checkable proof systems (PCPs)* and look at the celebrated PCP theorem which provides a probabilistic characterization of the class NP. We use the PCP theorem to prove some hardness of approximation results.

In the second part, we describe a type of reduction called *L-reductions*, which unlike the many-one polynomial-time reductions used in the theory of NP-completeness, preserve approximation ratios. We use these reductions in developing a complexity theory of approximability and show the existence of a large class of problems that do not have a PTAS, unless P=NP.

The material in the first part and the second part follows respectively the material in [1] and [2]. The reader will frequently be referred to these books for details that are not central to the exposition here.

2 The PCP Theorem

The PCP Theorem provides a powerful probabilistic characterization of the class NP. In this section, we will discuss probabilistic proof verifiers, state the PCP theorem, prove the easy direction of the inclusion, and present some intuition. We then use the PCP theorem to prove a hardness of approximation result for MAX k-FUNCTION SAT. This result will be used in the next section to show that no PTAS exist for a large class of problems, unless P=NP.

Recall that NP is usually defined to be the class of decision problems, whose "Yes" instances have short (polynomial in the length of the input) witnesses that can be verified in polynomial time. Informally, the PCP theorem states that for these problems, in fact, there exist witnesses which can be verified with high probability by only looking at a constant number of randomly chosen bits. The witness string is often called a proof in the literature. We will use these terms synonymously.

A *probabilistically checkable proof system* is described by two parameters namely the number of random bits it needs, and the number of bits of the proof it queries. The PCP theorem uses $O(\log n)$ number of random bits and a constant number of query bits.

The *verifier* for the PCP is a polynomial-time Turing machine, which apart from its input tape has a tape that provides random bits and another tape that has the proof on it. The proof tape is random access, i.e. any bit of the proof

can be read in constant time. Clearly, which bits of the proof are queried is then a function of the input and the random bits. The turing machine either halts and accepts the proof or halts and rejects the proof.

Definition 1 *A language $L \in PCP(r(n), q(n))$ if there is a verifier V and constants c and k, such that on input x, V obtains a random string of length $cr(n)$ and queries $kq(n)$ bits of the proof. Further,*

- *If $x \in L$, then there is a proof y that makes V accept with probability 1,*
- *If $x \notin L$, then for every proof y, V accepts with probability $< \frac{1}{2}$,*
  where the probabilities are over the random string.

The probability of accepting in case $x \notin L$ is called the *error probability.*

Intuitively, a PCP verifier is analogous to a lazy but competent TA who is grading proofs written by students. Since the TA is lazy, she only checks a small part of each student's proof and based on the correctness of that small part decides if the proof is right or wrong. Since the TA is competent, if the entire proof is right she will never fail to be convinced by any part of it. However, if the proof is incorrect, it might be the case that she happened to look at a correct portion of the proof and erroneously concluded that the entire proof is right. The error probability quantifies the chance of this happening. This is the intuitive reason for the one-sided error in the definition above.

It is immediate from the definition that NP=PCP$(0, poly(n))$. This is because a deterministic verifier requires zero random bits and queries atmost a polynomial number of bits of the proof and has error probability zero (which is smaller than 1/2). The PCP Theorem offers another characterization of NP.

Theorem 2 (PCP Theorem) *NP=PCP*$(\log n, 1)$.

The inclusion PCP$(\log n, 1) \subseteq$ NP is easy to prove. Consider a language $L \in$ PCP$(\log n, 1)$. It thus has a PCP verifier. The claim is that using this PCP verifier we can construct a deterministic verifier. Both these verifiers receive the input instance and a proof, and seek to verify the proof for the input instance. Additionally the PCP verifier uses $\log n$ random bits and gives answers that are true in probability over this random string. What we do is to simply run the PCP verifier over every possible "random" bit string and accept if and only if the PCP verifier accepts for every random string (while keeping the input instance and the proof the same). Since the random string is of length $\log n$ and the PCP verifier runs in polynomial time, the resulting deterministic verifier also runs in polynomial time.

The other inclusion NP $\subseteq$ PCP$(\log n, 1)$ is much harder to prove and will not be discussed in these notes. The interested reader is referred to [3] for details.

Next, we describe a weaker probabilistic verifier (than the PCP verifier) for 3SAT, which should give the reader a sense of how strong the PCP Theorem is. Consider a 3-CNF SAT boolean formula with m clauses. 3-SAT is in the class NP. Suppose we are given a certificate, which is an assignment to the variables and $O(\log n)$ bits. The weak verifier does the following: It randomly select one of the clauses. Since, the number of clauses is atmost the number of variables, $\log n$ random bits are sufficient to make this choice. It now queries the truth values of the variables present in the clause chosen, from the proof (the query size is therefore atmost 3 bits). It substitutes the truth values from

the proof onto the clause and checks if the clause evaluates to true. If this is so, it halts and accepts, else it halts and rejects. If the formula is satisfiable and the certificate is right (there is atleast one such certificate), then this verifier will accept with probability 1, since an assignment that satisfies the formula must make each clause evaluate to true. However, if the formula is unsatisfiable, then each certificate makes atleast one clause evaluate to false. We choose one such clause with probability atleast $1/m$. Therefore, our error probability for unsatisfiable instances for this weaker verifier is atmost $1-\frac{1}{m}$. The breakthrough with the PCP theorem is that it manages to get this error probability down to atmost $1/2$, while still querying a constant number of bits from the proof and using $O(\log n)$ random bits.

Next, we describe the MAX k-FUNCTION SAT problem and prove a hardness of approximation result for it using the PCP Theorem.

Problem 1 (MAX k-FUNCTION SAT) *Given n Boolean variables $x_1, \ldots, x_n$ and m functions $f_1, \ldots f_m$ each of which is a function of k of the boolean variables, find a truth assignment to $x_1, \ldots, x_n$ that maximizes the number of functions satisfied. Here k is a fixed constant (not part of input).*

Lemma 3 *There exists a constant k for which there is a polynomial-time reduction from SAT to MAX k-FUNCTION SAT that transforms a boolean formula ϕ to an instance I of MAX k-FUNCTION SAT such that,*

- *If ϕ is satisfiable, $OPT(I) = m$ and*
- *If ϕ is not satisfiable, then $OPT(I) < \frac{1}{2}m$*

Sketch of Proof: Since SAT$\in$ NP, it has a PCP($\log n$, 1) verifier, P. Since the random bit string used by P has length atmost $c \log n$, we can go over all possible random bit strings, of which there are atmost n^c. On being presented a random bit-string and ϕ, P queries atmost q bits of the proof. The instance I of MAX-k-FUNCTION SAT is constructed as follows: It has number of variables equal to the length of the proof (which can be atmost qn^c). Also, we have $k = q$. The idea is that for every "random" bit string, since ϕ is fixed, whether V accepts or rejects is a boolean function of just the bits of the proof queried. We construct this function. Since the number of bits q is a constant, we can run through all the 2^q bit-strings to construct the function in constant time. We thus have a function corresponding to every "random" bit-string. If ϕ is satisfiable, there exists a proof that makes the verifier accept for every random bit-string. Therefore there is an assignment to the variables of I, that makes each of the functions evaluate to true. If ϕ is not satisfiable, every proof leads to the verifier accepting on atmost half of the bit strings. That is, for every assignment to variables of I, less than half the functions evaluate to true. ■

3 Approximation and Complexity

In this section, we first describe a type of reduction called *L-reductions*, which unlike the many-one polynomial-time reductions used in the theory of NP-completeness, preserve approximation ratios. We then define a complexity class for optimization problems called MAXSNP, which is motivated by a structural

characterization of NP from finite model theory (Fagin's theorem). We list several problems that are known to be MAXSNP-complete (under L-reductions). We then show that every problem in this class has approximation threshold strictly greater than zero. Finally, we show that none of the MAXSNP-complete problems have a PTAS, unless P=NP, which is the strongest result that we could expect to hold.

3.1 L-Reductions

We observe that the many-one polynomial time reductions do not preserve approximation ratios of approximation algorithms. We thus construct another type of polynomial-time reduction for optimization problems which preserve approximation ratios of approximation algorithms across the reduction and which are transitive, i.e. If problem A reduced to problem B and problem B reduced to Problem C, then there exists a reduction from A to C with the same properties.

Definition 4 (L-Reduction) *An L-Reduction from an optimization problem A to B is a pair of functions (R, S) both computable in logarithmic space, such that if x is an instance of A, then $R(x)$ is an instance of B and if y is an answer (feasible solution) of $R(x)$, then $S(y)$ is an answer of x. Furthermore,*

1. *If x is an instance of A with optimum cost OPT(x), then R(x) is an instance of B with optimum cost satisfying*

   $OPT(R(x)) \leq \alpha \cdot OPT(x)$,
   where α is a positive constant

2. *If s is any feasible solution of $R(x)$, then $S(s)$ is a feasible solution of x such that,*

   $|OPT(x) - c(S(s))| \leq \beta \cdot |OPT(R(x)) - c(s)|$,
   where β is another positive constant particular to the reduction. In both cases, $c(\cdot)$ is used to denote the cost of of feasible solutions.

Observe that from the second property, we have that if s is the optimum solution of $R(x)$, then $S(s)$ is the optimum solution of x.

The following two propositions are an easy consequence of the definition.

Proposition 5 *If (R,S) is an L-reduction from problem A to problem B, and (R', S') is an L-reduction from B to C, then their composition $(R \cdot R', S' \cdot S)$ is an L-reduction from A to C.*

Proposition 6 *If there is an L-reduction (R,S) from A to B with constants α and β, and there is a polynomial-time ϵ-approximation algorithm for B, then there is a polynomial-time $\frac{\alpha\beta\epsilon}{1-\epsilon}$-approximation algorithm for A.*

The last proposition gives rise to the following corollary:

Corollary 7 *If there is an L-reduction from A to B and there is a PTAS for B, then there is a PTAS for A.*

The proofs are left as an exercise (and are also available in [2]).

3.2 The Class MAXSNP

Fagin's Theorem from finite-model theory [2] provides a structural characterization of the class NP. It states that all graph-theoretic properties in NP can be expressed in existential second-order logic. Strict NP or SNP is a subset of NP, which consists of properties expressible as

$$\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi(S, G, x_1, \ldots, x_k),$$

where ϕ is a quantifier-free First-Order expression involving the variables x_i and the structures G (the input) and S. SNP contains decision problems. however, we are interested in defining a class of optimization problems. We can achieve this by slightly modifying the definition. SNP consists of structures G, for which $\exists S \forall x_1 \forall x_2 \ldots \forall x_k \phi$ is true. Now, we can relax this requirement a little bit. Instead of requiring ϕ to hold for all k-tuples, we only ask for the S that makes it hold for the largest possible number of tuples. This immediately gives us an optimization problem.

We now define the class MAXSNP_0 of optimization problems: Problem A in this class is defined in terms of the expression,

$$\max_S |\{(x_1, \ldots, x_k) \in V^k : \phi(G_1, \ldots, G_m, S, x_1, \ldots x_k)\}|$$

The input to a problem A is a set of relations $G_1, \ldots, G_m$ over a finite universe V. We seek a relation $S \subseteq V^r$, such that the number of k-tuples $(x_1, \ldots, x_k)$ for which ϕ holds is as large as possible.

Finally, we define the class MAXSNP to be the class of all optimization problems that are L-reducible to a problem in MAXSNP_0.

Example 1 *The problem MAX-CUT is in MAXSNP_0. and therefore in MAXSNP. This is because MAX-CUT can be written as follows:*

$$\max_{S \subseteq V} |\{(x, y) : ((G(x, y) \vee G(y, x)) \wedge S(x) \wedge \neg S(y))\}|.$$

Here, the graph is represented as a directed graph. The problem asks for the subset S of nodes that maximize the number of edges that enter S or leave S, that is the maximum cut in the underlying directed graph.

The following theorem shows that every problem in MAXSNP has nonzero approximation threshold.

Theorem 8 *Let A be a problem in MAXSNP_0. Suppose that A has the form $\max_s |\{(x_1, x_2, \ldots, x_k) : \phi\}|$. Then A has a $(1-2^{-k_\phi})$-approximation algorithm, where k_ϕ is the number of atomic expressions in ϕ that involve S.*

Sketch of Proof: The proof consists of two parts. In the first part, we use the structural characterization of each problem in MAXSNP_0 to come up with a corresponding MAX-k-FUNCTION SAT instance. In the second part, we provide an approximation algorithm for MAX-k-FUNCTION SAT.

Consider an instance of A. We wish to find that S that maximizes the number of tuples that evaluate to true. For the MAX-k-FUNCTION SAT instance, we basically construct a formula for every possible tuple. (Since k is a constant, there are atmost polynomially many formulae). For each tuple, given that G

and the tuple are fixed, the only variables left in ϕ are of the form $S(x_i)$, where $S(\cdot)$ is the indicator function for S. Let k_ϕ be the number of times that $S(\cdot)$ appears in ϕ. We thus have a MAX-k-FUNCTION SAT instance where $k = k_\phi$. The optimum assignment to this instance is clearly the indicator function for the optimum S of the instance of A in question.

Now given MAX-k-FUNCTION SAT instance, here is a $(1 - 2^{-k})$ approximation algorithm: Partition the variables into sets of size k. For each set in the partition, consider the functions that can be evaluated by an assignment to the corresponding k variables. By the pigeonhole principle, there is an assignment to these k variables that makes atleast 2^{-k} of these formulae evaluate to true. Find such as assignment by going over the 2^k possible assignments to these variables (This can be done in constant time, since k is a constant). Since at each step, we make atleast 2^{-k} of the formulae evaluate to true, the algorithm has an approximation ratio of 2^{-k} and is therefore a $(1 - 2^{-k_\phi})$-approximation algorithm.

The result follows. ■

3.2.1 MAXSNP-Completeness

Definition 9 *A problem in MAXSNP is MAXSNP-complete if all problems in MAXSNP L-reduce to it.*

Theorem 10 *MAX3SAT is MAXSNP-complete.*

Proof Idea: The proof uses the construction in Theorem 8 to first obtain an instance of MAX-k-FUNCTION SAT, for some k. This is then L-reduced to a 3-CNF SAT instance. [2] has the complete proof.

[2] proves that several important problems are MAXSNP-complete. These include MAX-CUT, 4-DEGREE INDEPENDENT SET and 4-DEGREE NODE COVER among others.

Next, we prove that no MAXSNP-complete problem has a PTAS, unless P=NP. This uses the inapproximability result for MAX-k-FUNCTION SAT, which was proved in Section 1 using the PCP Theorem.

Theorem 11 *No MAXSNP-complete problem has a PTAS, unless P=NP.*

Proof: We show this result by showing the existence of a problem in MAXSNP that does not have a PTAS. This immediately implies the required result, because if a MAXSNP-complete problem has a PTAS, then every problem in MAXSNP has a PTAS.

Recall that Lemma 3, proved that there exists a constant k, for which MAX-k-FUNCTION SAT has a no 1/2-factor approximation algorithm (and hence no PTAS). Now, for every constant k, we can construct a problem in MAXSNP (using the structural characterization of MAXSNP_0.), which has number of atomic expressions equal to k. By the construction in Theorem 8, we can reduce this problem to MAX-k-FUNCTION SAT. Lemma 3 applies for atleast one such problem. This proves the result.

■

References

[1] V.V. Vazirani, *Approximation Algorithms*, Springer, 2001.

[2] C.H. Papadimitriou, *Computational Complexity*, Addison Wesley, 1993.

[3] S. Arora, B. Barak, *Complexity Theory: A Modern Approach*, Ca mbridge University Press, expected in 2009. (A draft is available at `http://www.cs.princeton.edu/theory/complexity/` .