6.893 Approximability of Optimization Problems	October 4, 1999
Lecture 8	
Lecturer: Madhu Sudan	*Scribe: Leonid Reyzin*

Hardness of Approximation

Types of Approximation Problems

In the exact optimization world, there are three common ways to look at problems: decision (e.g., does a given set of clauses have an assignment satisfying at least k clauses or not), computation (e.g., what's the maximal number of clauses that can be satisfied simultaneously in this set), and search (e.g., find an assignment that satisfies the maximal number of clauses). It is quite clear that decision and computation are equivalent (to reduce computation to decision, simply run the decision for $k = 1, \ldots, m$, where m is the number of clauses), and search is at least as difficult.

There is a clear parallel in the approximation world. Similar to decision problems are "verification gap problems." For example, define GapSAT_g as follows:

$$\mathrm{YES} = \{(c_1, \ldots, c_m, k) | \exists \text{ assignment satisfying at least } k \text{ clauses } \}$$

and

$$\mathrm{NO} = \{(c_1, \ldots, c_m, k) | \forall \text{ assignments, fewer than } k/g \text{ clauses are satisfied}\}$$

(we care that a machine answer YES and NO instances correctly, but don't care what it does on instances that are neither YES nor NO). Similar to computation problems are approximate computation problems: given $(c_1, \ldots, c_m)$, g-approximate CompSAT problem is to output k' s.t. no fewer than k' and no more than gk' clauses can be simultaneously satisfied. Finally, g-approximate MAX-SAT problem is to find a satisfying assignment that satisfies at least opt/g clauses, where opt is the maximal number of clauses that can be simultaneously satisfied.

Similar relations among these problem hold in the approximation world.

A search problem is at least as difficult as the computation problem, because the computation problem can simply output the number of clauses satisfied by the output of the search problem.

Computation and verification problems are equivalent. Indeed, to reduce decision on $(c_1, \ldots, c_m, k)$ to computation, simply get k' from the computation machine and output YES if $k' \geq k/g$ and NO otherwise. We need to verify that we output "YES" for YES instances and "NO" for NO instances. Indeed, if an instance is a YES instance, then the number of clauses that can be simultaneously satisfied $opt \geq k$, so the computation machine will output k' s.t. $gk' \geq opt$, so $k' \geq opt/g \geq k/g$ and we will output "YES." If an instance is a NO instance, then $opt < k/g$, so $k' \leq opt < k/g$ and we will output NO.

To reduce computation on $(c_1, c_2, \ldots, c_m)$ to decision, run GapSAT_g on

$$\begin{array}{rr} (c_1, c_2, \ldots, c_m, & 1), \\ (c_1, c_2, \ldots, c_m, & 2), \\ \ldots, & \\ (c_1, c_2, \ldots, c_m, & m-1), \end{array}$$

and output $k' = k/g$, where k is the largest value such that the answer on $(c_1, c_2, \ldots, c_m, k)$ was "YES." Then we know that $(c_1, c_2, \ldots, c_m, k)$ is not a "NO" instance, so there exists an assignment for which at least $k' = k/g$ clauses are satisfied, and we also know that $(c_1, c_2, \ldots, c_m, k+1)$ is not a "YES" instance, so for no assignment are there $k + 1 > gk'$ clauses satisfied.

So, from complexity-theory point of view, when we want to prove non-approximability results, it makes sense to work with the "weakest" problem—the approximate decision problem.

Hardness of Approximation and Proofs

How would we go about proving that, say, $\mathrm{GapSAT}_{1.01}$ is NP-hard for $k = m$ (the number of clauses)? A natural way would be to reduce an NP-hard problem to it, with YES instances going to YES instances, NO instances going to NO instances, and NP-witnesses going to satisfying assignments. Note, however, that now an interesting thing happens: if we have a NO instance of the NP-hard problem, then no assignment will satisfy more than $1/1.01$ of the clauses of the corresponding formula. Thus, if no satisfying assignment exists, we will be able to reject a given NP witness with arbitrarily close to 1 constant probability by simply checking a constant number of clauses (and thus reading only a constant number of variable values given to us by the NP witness).

Now think of the original NP statement as a theorem, and witness as its proof. If such a reduction exists, then we can very easily verify proofs of theorems: if a theorem is true and a proof exists, then there will always be a satisfying assignment of the corresponding $\mathrm{GapSAT}_{1.01}$ formula. If, on the other hand, a theorem is false and someone tries to give us a proof, we can reduce the theorem to a boolean formula and the proof to an assignment and verify with high probability that the assignment is not true by simply checking a constant number of places in the assignment.

Although no one quite put it this way at first, this strange way of verifying theorems was necessary before we could prove non-approximability results. When such ways of verifying theorems were found in the early 1990's, this connection became very clear. We will now talk some more about theorem verification.

Proofs

IP

Suppose we have a language L and a string x, and an agent called "prover" wants to convince an agent called "verifier" that $x \in L$. We will assume that the prover is omnipotent and that the verifier is limited to polynomial-time computations. If $L \in \mathrm{P}$, then the verifer can check whether $x \in L$ without the prover's help. If $L \in \mathrm{NP}$, then the prover can present the NP witness to the verifier. The idea of "interactive proofs" (IP), due to Goldwasser, Micali and Rackoff [GMR89] and, in a slightly different form, to Babai [Bab85], is that if we allow the prover and verifier to interact, we may be able to do better than just NP languages.

To formalize the notion of IP, consider two machines P (prover) and V (verifier) that are both given, as input, a string x. V is a probabilistic polynomial-time machine, and P can compute arbitrary functions. In the i-th round of interaction, V computes a question q_i based on the input x, its private random bits r, and the answers $a_1, \ldots, a_{i-1}$ from the previous rounds, and sends q_i to P. P sends the answer a_i back to V. After a certain number n of rounds, V computes a boolean verdict ("accept"/"reject") based on $(x, r, a_1, \ldots, a_n)$. $L \in \mathrm{IP}$ if there exists a probabilistic polynomial-time machine V such that the following two conditions are satisfied:

- Completeness with probability c: If $x \in L$, then there exists P such that V accepts with probability at least c.
- Soundness with probability s: If $x \notin L$, then for all P, V accepts with probability at most s.

The default value for is c is 1 and for s is $1/2$. We will often omit the parameters c and s when $c = 1$ and $s = 1/2$.

Note that if V is not allowed to be probabilistic, then P, being more powerful than V, can simply compute all the question V will ask based simply on x, and send all the answers at once. So, if P is deterministic, we get just NP. This is why it is crucial that P be allowed to use randomness. Note that in this definition, V does not see P's random coins.

It became clear that IP contains problems not believed to be in NP when Goldreich, Micali and Wigderson [GMW86] showed an interactive proof for graph non-isomorphism.

One can show that IP $\subseteq$ PSPACE by simply contsructing a PSPACE machine that traverses the tree of all possible interactions. The other direction is much harder. Lund, Fortnow, Karloff and Nisan showed that PH $\subseteq$ IP [LFKN92] and Shamir, building on their ideas, showed that PSPACE $\subseteq$ IP [Sha92], thus proving that IP = PSPACE.

MIP

What happens if we let the verifier interact with more than one prover in the definition of IP? The provers will still be all-powerful, but won't be able to interact with each other. This clearly limits the cheating opportunities for the prover: now if the prover wants to cheat the verifier, it has to somehow make sure that the verifier cannot detect this cheating by interacting with other provers.

The class of multi-prover interactive proofs (MIP) was defined by Ben-or, Goldwasser, Kilian and Wigderson [BGKW88]. The defintion is essentially the same as IP, except that now instead of considering one prover, we consider sets of provers.

Clearly, IP $\subseteq$ MIP,simply because the verifier can choose just one prover to interact with. It also seems that by limiting the cheating abilities of the prover, we should be able to construct verifiers for more statements and thus get more than just IP. Indeed, Babai, Fortnow and Lund [BFL91] showed that MIP = NEXPTIME.

OIP

So far, we allowed the prover to adaptively select responses to questions based on the previous questions asked. What if the prover has to fix its responses to all possible questions as soon as x is given? Then the prover becomes just a look-up table that the verifier queries. That is, the prover is now just an oracle.

Note that the prover now has to commit to answers before seeing the questions, and thus, again, has less room for cheating. As a toy example, suppose the verifier wants to check if a boolean formula $x = y, y = z, z \neq x$ is satisfiable by checking the values of only two of its variables (and thus only one equality or inequality). If the interactive prover's strategy is as follows:

question 1	answer 1	question 2	answer 2
x	T	y	T
		z	F
y	T	x	T
		z	T
z	T	x	F
		y	T

then the verifier will never detect cheating. If, however, the prover is required to write down the values of x, y and z in advance, and the verifier gets to choose two of those values, it will detect cheating with probability at least 1/3.

It turns out that this way of limiting the cheating power of the prover is equivalent to limiting the cheating power of the prover by introducting additional provers. Fortnow, Rompel and Sipser [FRS88] defined the class of Oracle Interactive Proofs and showed that OIP = MIP (note that it's pretty clear that MIP $\subseteq$ OIP because every set of provers can be converted to a lookup table; the other direction is the difficult one).

So, the oracle prover is just an exponential-size lookup table. Note that if the verifier is deterministic exponential time, then getting an exponential lookup table means that you can check languages in NEXPTIME. If the verifier is probabilistic polynomial time, and can simply query the table, then we get OIP which, by [FRS88] and [BFL91], is also NEXPTIME!

PCP

Now that we have oracle interactive proofs, we will give them another name, "probabilistically checkable proofs," and parameterize them more precisely. We will define (following Arora and Safra [AS92]) $\mathrm{PCP}_{c,s}(r, q)$ as a class of languages whose membership can verified using an oracle ineractive proof with completeness probability c, soundess probability s, where the verifier uses at most $r(|x|)$ random bits and looks up at most $q(|x|)$ bits in the table. If $c = 1$ and $s = 1/2$ (the default values), we may omit them in our notation.

Note that, by simply parsing definitions, we get that $\mathrm{PCP}(0, \mathrm{poly}) = \mathrm{NP}$, $\mathrm{PCP}(\mathrm{poly}, 0) = \mathrm{RP}$ and $\mathrm{PCP}(\mathrm{poly}, \mathrm{poly}) = \mathrm{OIP}$. Thus, by [FRS88] and [BFL91], $\mathrm{PCP}(\mathrm{poly}, \mathrm{poly}) = \mathrm{NEXPTIME}$.

What if try to scale this down from NEXPTIME to NP, to get proofs for NP languages as we wanted in the first part of the lecure? By "scaling down" the work of [BFL91], Babai, Fortnow, Levin and Szegedy [BFLS91] and Feige, Goldwasser, Lovász, Safra and Szegedy [FGL+91] showed that $\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{polylog}\, n, \mathrm{polylog}\, n)$. Subsequent work by Arora and Safra [AS92] demonstrated that $\mathrm{NP} = \mathrm{PCP}(\log n, \sqrt{\log n})$, and work by Arora, Lund, Motwani, Sudan and Szegedy [ALM+92] reduced the number of queries to a constant k (reputed to be about 10^6, although never precisely calculated) to give $\mathrm{NP} = \mathrm{PCP}(\log n, k)$.

There has been some further work to reduce the number of query bits, which we do not cite here. The current state of the art is the work of Håstad [Hås97], who reduced the number of query bits to 3, showing that $\forall \varepsilon > 0$ $\mathrm{NP} = \mathrm{PCP}_{1-\varepsilon,1/2}(\log n, 3)$, and the work of Guruswami, Lewin, Sudan and Trevisan [GLST98], who got perfect completeness by showing that $\forall \varepsilon > 0$ $\mathrm{NP} = \mathrm{PCP}_{1,1/2+\varepsilon}(\log n, 3)$.

That is, a proof of any statement in NP can be written down in such a way that it can be verified by looking at exactly three bits of the proof.

As before, the difficult direction in all these proofs is that $\mathrm{NP} \subseteq \mathrm{PCP}(r, q)$, not that $\mathrm{PCP}(r, q) \subseteq \mathrm{NP}$. In fact, we will prove the easy direction right now.

Claim 1 *$PCP(r, q) \subseteq NTIME(2^{r+q})$.*

Proof Once x is fixed, the locations queried by the verifier are a function of the random bits and the answers to previous queries. Thus, the total number of possible locations queried by the verifier is $2^r + 2^{r+1} + \ldots + 2^{r+q-1} < 2^{r+q}$. So, we can non-deterministically guess the values at all the locations and then go through all the possible 2^r random strings to count the probability of verifier's acceptance. ■

References

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd IEEE Symposium on Foundation of Computer Science*, pages 14–23, 1992.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *Proceedings of the 33rd IEEE Symposium on Foundation of Computer Science*, pages 2–13, 1992.

[Bab85] László Babai. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[BGKW88] L. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability. In *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pages 113–131, 1988.

[FGL+91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd IEEE Symposium on Foundation of Computer Science*, pages 2–12, 1991.

[FRS88] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. In *Proceedings of the 3rd IEEE Symposium on Structure in Complexity Theory*, pages 156–161, 1988.

[GLST98] Venkatesan Guruswami, Daniel Lewin, Madhu Sudan, and Luca Trevisan. A tight characterization of NP with 3 query PCPs. In *39th Annual Symposium on Foundations of Computer Science*, pages 8–17, Los Alamitos, California, November 1998. IEEE.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 174–187, Toronto, Ontario, Canada, 27–29 October 1986. IEEE.

[Hås97] J. Håstad. Some optimal inapproximability results. Technical Report TR97-37, ECCC (*Electronic Colloquium on Computational Complexity*, `http://www.eccc.uni-trier.de/eccc`), 1997. Preliminary appears in *Proceedings of the 29th ACM Symposium on the Theory of Computing*, 1997.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proofs systems. *Journal of the ACM*, 39(4):859–868, 1992.

[Sha92] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.