1 Tight Inapproximability Results

Lecture 8: June 11, 2003

We will give a high level overview of a technique for obtaining tight hardness of approximation results from the PCP theorem. Our initial goal in the course was to show that it is NP-hard to distinguish between satisfiable 3CNF formulas and those that are $(7/8+\epsilon)$ satisfiable. We will sketch the proof of a somewhat weaker theorem.

Theorem 1 *For every $\epsilon > 0$, it is NP-hard to distinguish between 3CNF formulas that are at least $(1-\epsilon)$-satisfiable and 3CNF formulas that are $(7/8+\epsilon)$-satisfiable.*

This will be a simple consequence of the following theorem.

Theorem 2 *For every $\epsilon > 0$, it is NP-hard to distinguish between 3LIN formulas that are at least $(1-\epsilon)$-satisfiable and 3LIN formulas that are $(1/2+\epsilon)$-satisfiable.*

Here 3LIN (or 3XOR) is a formula whose clauses are the *exclusive or* of three literals. Let us note that testing whether a 3LIN formula is satisfiable can be done in polynomial time using guassian elimination, and hence the $(1-\epsilon)$ term in Theorem 2 cannot be replaced by 1 (unless P=NP). Let us also note that the following simple greedy algorithm always satisfies at least half the clauses. Moreover, the algorithm satisfies strictly more than half the clauses if this is possible.

Iteratively, perform one of the following operations in order of decreasing priority.

1. Remove from the formula pairs of clauses where one clause is the complement of the other. Every assignment satisfies exactly half these clauses.

2. Satisfy a unit clause and simplify the formula.

3. Pick a variable from the shortest clause, assign the value 0 to it, and simplify the formula.

Hence Theorem 2 is essentially best possible. The only room for improvement is in determining the rate of decrease of ϵ as a function of n.

To see that Theorem 2 implies Theorem 1, replace every 3LIN clause by the four 3SAT clauses that are satisfied when the 3LIN clause is.

1.1 The overall approach

We shall construct a PCP that proves Theorem 2. This PCP will be composed of an outer PCP and an inner PCP. The outer PCP will have as its starting point the PCP theorem. Using a certain *parallel repetition* procedure, we will obtain an outer PCP with only two probes and arbitrarily small error. (We shall use here the parallel repetition theorem of [7], though in principle the analysis in [3] also suffices.) Moreover, this outer PCP will have a certain property, that is called the *projection* property, that will turn out useful in the construction of the inner PCP.

The inner PCP will be based on encoding every answer of the outer PCP using the so called *long code*, introduced in [1]. This code is extremely redundent, as it codes every n bit string by a string of length 2^{2^n}. As we have seen in Lecture 7, having highly redundent codes helps reduce the number of query bits, and the long code is the natural outcome of taking this approach to its extreme. The kind of test that we will perform on the long code is somewhat similar to the lineraity test that we performed on Hadamard codes, namely, that the exclusive or of two bits it equal to a third bit. This gives a linear equation (a 3LIN clause) where the variables are the answer bits, and naturally results in hardness of approximation results for 3LIN.

The particular test that is performed on the long code is somewhat tricky, and perhaps counter-intuitive. To understand why such a test is performed, one needs to understand its analysis. This analysis is based on the discrete Fourier transform, and realizing what may go wrong in the analysis helps in guiding us toward the test. Both the test and its analysis are presented in [4]. We note that Hastad also presents and analyses a different test that is used to remove the ϵ term from the $(1-\epsilon)$ in Theorem 1.

We would like to note here an important principle that helps in obtaining tight results. This is the use of *list decoding* rather than *unique decoding*. This principle comes up in the Fourier analysis.

1.2 A two prover proof system

The construction of the outer PCP is quite simple, given that we already have the PCP theorem. Recall that the following is an equivalent form of the PCP theorem: for some universal $\epsilon > 0$, given a 3CNF formula ϕ with n variables and m clauses, it is NP-hard to distinguish between the case that the formula is satisfiable and the case that it is not even $(1-\epsilon)$-satisfiable.

Given such a formula ϕ, we construct a two probe PCP, that we shall call a *one-round two-prover* proof system. Prover 1 is a table P_1 containing for every clause a satisfying assignment (one of 7 possibilities). Prover 2 is a table P_2 containing for every variable an assignment (one of 2 possibilities). The verifier picks a clause C_i at random, and in this clause, a variable x_j

at random. The verifier probes P_1 at C_i, and P_2 at x_j. Note that $P_1(C_i)$ implicitly contains an assignment to x_j. The verifier rejects if this assignment differs from $P_2(x_j)$. If ϕ is satisfiable, then one can prepare P_1 and P_2 such that the verifier never rejects.

Proposition 3 *If ϕ is at most $(1-\epsilon)$-satisfiable, then the verifier rejects with probability at least $\epsilon/3$.*

Proof: Table P_2 is an assignment for the variables of ϕ. At least ϵm clauses are not satisfied by it. For every such clause, P_1 needs to contradict P_2 on at least one variable. $\square$

1.3 Parallel repetition

The PCP constructed in Section 1.2 is too weak for our purpose, because the rejection probability is too small when ϕ is not satisfiable. We would like to make it close to 1, or equivalently, make the *error* (the probability of accepting an unsatisfiable formula) close to 0.

A natural approach to do so is via repeated sampling. Repeating the two probe procedure k independent times the error reduces to $(1-\epsilon/3)^k$. However, the number of probes increases to $2k$, whereas we would like it to remain two.

The approach that we use is called *parallel repetition*. For it, we need to change the tables P_1 and P_2. Their sizes will now be m^k and n^k respectively, rather than m and n. Each entry in P_1 will now be indexed by a sequence of k clauses, and will contain a sequence of k answers, where each answer is an assignment that satisfies the respective clause. Each entry in P_2 will be indexed by a sequence of k variables, and will contain a sequence of assignments to the k variables. One may further require that every entry is internally consistent (e.g., if a variable is repeated in a sequence, then its assignments are identical), and that permutation over the order of variables in a sequence results in a corresponding permutation in the answer bits, but we shall not use this here.

The verifier picks a sequence W of k clauses at random and a sequence U of k random variables, one from each clause of W. It probes P_1 at W and P_2 and U, and rejects if the answers are inconsistent in at least one coordinate. (Here we could have also checked cross coordinate consistency, but will not do so.)

This PCP has only two probes. But what is its error? The best we can hope for is an error of $(1+\epsilon/3)^k$, similar to the case of sequential repetition. But this need not be the case, as the following example illustrates.

Consider the following (nonproper) 3CNF formula:

$$(\bar{x_1}) \wedge (\bar{x_2}) \wedge (x_1 \vee x_2) \wedge (x_2 \vee x_1)$$

The formula is not satisfiable. For every table P_2, the two prover PCP of Section 1.2 accepts with probability at most 3/4. (Removing the last clause, that may seem redundent, increases the probability of acceptance to 5/6.)

Consider now repetition of this two prover protocol, with $k = 2$. For sequential repetition, the error decreases to $(3/4)^2 = 9/16$. For parallel repetition, the error is higher. Consider for example the following strategy for P_2: if both queries composing the probe are for the same variable, return 0 on both of them, else return 1 on the query to x_1 and 0 to the query to x_2. Once the strategy for P_2 is fixed, finding an optimal strategy for P_1 is straightforward. For each probe (query pair), choose the answer that maximizes acceptence probability, given the known strategy of P_2. A simple case analysis shows that the acceptance probability is 11/16. (We note that in this example the strategy for the two provers has what we called internal consistency and permutation consistency, and the error probability does not change if the verifier also checks cross consistency.)

Does parallel repetition reduce the error? This question was open for some time and attracted a lot of research (both in trying to analyse what happens by parallel repetition, and in finding ways other than parallel repetition to reduce the error in two prover protocols). The following parallel repetition theorem by Raz [7] finally gave a satisfactory answer to this question.

Theorem 4 *For every two prover proof system with error at most* $0 < \delta < 1$, *there is some* $\delta \leq \delta' < 1$ *(*δ' *depends only on* δ*), such that for every* k*, the* k*-fold parallel repetition has error at most* $(\delta')^{k/a}$*, where* a *is the answer length of the original two prover proof system.*

We shall not prove Theorem 4 in the course. For more details on parallel repetition of two prover proof systems the reader is referred to the survey [2].

In the two prover proof system of Section 1.2, the error is some universal constant smaller than 1, and the answer length is constant (3 bits for P_1, 1 bit for P_2, and in fact, in Theorem 4 we are allowed to take the smaller of the two). Hence Theorem 4 implies that if we take the number of repetitions k to be a large enough constant (independent of n), then the error can be made to be an arbitrarily small positive constant.

This completes our description of the outer PCP.

1.4 The long code

We now present the inner verifier. Basically the task of the inner verifier is as follows. Given that the outer verifier chose a pair of probes, the inner verifier needs to figure out by querying only 3 bits whether the outer verifier would have accepted. An inner prover will need to encode the answers of the outer prover in a way that facilitates the task of the inner verifier.

Which encoding should we choose? To simplify the task of the inner verifier, the inner prover is prepared to answer every yes/no question that the inner verifier might ask. That is, for a string X (the answer of the outer prover), the inner prover prepares answers to all yes/no questions about X. What is the most general such question? It is of the form: "is X one of the following list of values?". If X contains k bits, there are 2^{2^k} possible yes/no questions, and their answers form the *long code* of X. An equivalent view of the long code is as giving the values of all possible Boolean functions on X. Yet another equivalent view is that of having a matrix of 2^{2^k} rows listing all possible words of length 2^k, and taking in this matrix the column indexed by X.

It is easy to verify that the normalized Hamming distance of every two words in the long code is $1/2$.

1.5 The projection property

The only test that the outer verifier does on the answer of P_1 is that it is consistent with the answer of P_2. (Note that by construction, the answer of P_1 necessarily satisfies the respective clauses.) Moreover, the consistency test is of a very simple nature. Given the probes q_1, q_2 to P_1, P_2 and the answer a_1 of P_1, there is a *unique* answer of P_2 that would cause acceptance. This answer is determined by projecting the answer of P_1 (that gives values to all variables in the respective clauses) to the variables represented in the probe to P_2. Hence we say that the outer PCP has the *projection property*. This simplifies the design on the inner PCP, and improves its efficiency. (In fact, the uniqueness property is all that we really use.)

Let X denote the answer of P_1 (the assignment to the variables in W) and Y the answer of P_2 (the assignment to U). Let us design a protocol for the inner verifier. A guiding principle is that the verifier should try to ask random "high entropy" questions (for which a "yes" answer seems as likely as a "no" answer), as otherwise he is wasting his precious queries.

Attempt 1. For X, ask a random yes/no question. Namely, select a random set S_1 of satisfying assignments to W, and ask whether $X \in S_1$. But then, what should one ask about Y? No answer for X restricts the possible values of Y, because with high probability both the projection of S_1 and of its complement cover all possible assignments for U.

Attempt 2. For Y, ask a random yes/no question. Namely, select a random set S_2 of possible assignments, and ask whether $Y \in S_2$. Here, a "yes" answer does limit the possible values of X. There is a maximal set S_1 for X whose projection gives exactly S_2. We can now ask whether $X \in S_1$ as a consistency test. Observe that S_1 contains roughly half the possible assignments to X, but these answers are highly correlated. Hence the structure of S_1 reveals S_2, which in our particular protocol means that the outer prover P_1 knows what P_2 is asked. Under this condition, the outer provers need never fail. So we need to modify attempt 2.

Let us remark here that if we had a protocol in which the uniqueness property holds in both ways, the so called *two-way uniqueness* property, then the question S_1 would still reveal S_2, but would not allow the outer prover P_1 to infer the probe to the other outer prover P_2. (There are some technical details that need to be added here, but are ignored at this point.) This can make the additional query that we shall introduce in attempt 3 unnecessary, and give rise to efficient two query PCPs, and some new hardness of approximation results. This hypothetical line of research is investigated in [5, 6].

Attempt 3. Using the idea of self-correcting probes of Lecture 7, we replace the query to S_1 by two queries: one to a random set R_1, and the other to $R_1 \oplus S_1$. These are two random probes, and the exclusive or of the answers gives the answer to S_1 (for a truthful prover). Hence we have three queries altogether, and the verifier needs to check that the exclusive or of all three answers is 0.

1.6 Folding

There are still several problems with attempt 3 above. Here is one that is easy to take care of. The provers might answer 0 to every query, and then the xor test always passes.

To get over this problem, we remove some of the redundancy of the long code in a way that ensures consistency with our interpretation of the long code. Note that the long code for X contains for every set R answers both to the question "is $X \in R$?", and "is $X \notin R$?" (or equivalently, "is $X \in \bar{R}$?"). Clearly, the answers to these questions must be complementary, in particular, forcing half the answers to be 1. We can implicitly force this property on the prover if of each pair of complementary sets $R, \bar{R}$, we ask only the question "is $X \in R$?", and complement the answer if the information needed was really "is $X \in \bar{R}$?". This technique is known as *folding* (though perhaps its time to find a better name for it).

Attempt 4. Same as attempt 3, but uses also folding. Hence we have three queries altogether, and the verifier needs to check that the exclusive or of all three answers is either 0 or 1, depending on whether the number of answers that need to be complemented due to folding is even or odd.

1.7 The Fourier basis

To develop further intuition regarding what may go wrong, it is useful to look at a simplified version of the inner PCP. Recall that two of the queries are addressed to X, and one query is addressed to Y. Recall also that Y is supposedly the projection of X. Assume for simplicity that Y is in fact identical to X, and then we may view all three queries as being addressed to X. Comparing with Lecture 7, we see that these three queries are simply a linearity test on the string that supposedly encodes X. Intuitively, we would like only valid codewords to pass the test. That is, if X is viewed as a binary word of length k, then we would like only the 2^k codewords of the long code to pass the test. But we have seen in Lecture 7 that all codewords of the Hadamard code pass the linearity test, and there are 2^{2^k} such codewords (because the table of answers contains 2^{2^k} bits).

It is useful to note that the 2^{2^k} codewords of the Hadamard code over 2^{2^k} bits are exactly the 2^{2^k} different words that one can get by choosing a subset of words in $\{0,1\}^k$, and taking the pointwise xor (or product, if we replace bit b by $b' = (-1)^b$) of their long codes. (One may verify that we do get a set of orthogonal vectors this way, and moreover, that the linearity test passes on these vectors.) Hence we now have a complete characterization of which tables always pass the lineraity test, and this characterization is given in terms of long codes.

Let us remark here that since we are using folding, there are only $2^{2^k}/2$ bits, and hence the space has a basis of size $2^{2^k}/2$. The basis vectors turn out to be those which are products of an odd number of long codes. A folded word is always orthogonal to a product of an even number of long code words. (Consider a set R and its complement $\bar{R}$. For a folded word, the values at these locations differ. For a product of an even number of long code words, the values agree.)

1.8 Adding a small error

Our goal here is to modify attempt 4 so that tables that are products of many long codes fail the (modified) test. The idea is to pick an additional small random subset $S \subset \{0,1\}^n$, of cardinality $\varepsilon 2^n$. The point is that $Pr[X \in S] = \varepsilon$, which is small. On the other hand, if we have more that $1/\varepsilon$ vectors $X_1, \ldots, X_k$, the probability that an odd number of them is in S (over the choice of random S) is roughly $1/2$. Hence quering a table at random S can distinguish between the case that the table is the long code of a single X and the case that the table is the exclusive or of long codes for many X_i.

We now reach the test used by Hastad.

Inner verifier test. The inner prover prepares two folded tables, table A for X and table B for Y. For Y, the inner verifier asks a random yes/no question. Namely, it select a random set S_2 of possible values, and ask whether $Y \in S_2$. There is a maximal set S_1 for X whose projection gives exactly S_2. The inner verifier chooses a random set R_1 and a small random set S (each possible value is in S with probability ε). Let $R_2 = S_1 \oplus R_1 \oplus S$.

The verifier asks whether $X \in R_1$ and whether $X \in R_2$. The verifier accepts if the exclusive or of all three answers is either 0 or 1, depending on whether the number of answers that need to be complemented due to folding is even or odd.

Note that by incorporating the xor with the small set S we lost the property of perfect completeness. Namely, now the inner verifier accepts satisfiable 3CNF formula ϕ with probability $1 - \varepsilon$, rather than 1. But this was unavoidable, due to the fact that deciding whether a 3LIN formula is satisfiable can be done in polynomial time.

1.9 List decoding

We sketch here how one proves the soundness property for the inner PCP. Namely, that if the original outer PCP has only a small probability δ of accepting, then the inner verifier has probability at most $1/2 + \epsilon$ of accepting, where ϵ can be made arbitrarily small by decreasing δ.

The proof uses Fourier analysis, and is more complicated than the proof given for the linearity test in Lecture 7. Let A and B be the tables used by the inner prover that supposedly correspond to the long codes of X and Y. One shows that if the inner test passes with probability greater than $1/2 + \epsilon$, then there are a small number of words from the Fourier basis that have nonnegligible correlation with A and with B. Moreover, each such word from the Fourier basis is the product of only a small number of long codes (this is a consequence of xoring with the small random set S).

Summarizing the above, with table A we can associated a small number of words $X_1, \ldots, X_k$ whose long codes participate in the words of the Fourier basis that have nonnegligible correlation with A. Similarly, with B we can associated a list $Y_1, \ldots, Y_k$. These lists form the *list decoding* of A and of B. The value of k depends on ϵ.

Another important property that can be shown is that the two lists are correlated. The words Y_i are the projections of the respective words X_i.

Having established the above properties using Fourier analysis, the proof can be completed. Assuming that the inner verifier accepts with probability above $1/2 + \epsilon$, we show a strategy for the outer provers that causes the outer verifier to accept with probability above δ. The stragety is as follows. To answer a probe about the value of X (or of Y), the outer prover picks at random one member from the list of possible decodings of the respective table A (or B, respecively). It is important to note that in giving such an answer, the outer prover sees only the question X but not the quetion Y (or vice versa). The probability that the answers to X and Y are consistent is at least $1/k$, which is greater that δ if the number of parallel repetitions was large enough.

References

[1] M. Bellare, O. Goldreich, M. Sudan. "Free bits, PCPs and nonapproximability – towards tight results". *SIAM Journal on Computing*, 27(3):804–915, 1998.

[2] U. Feige. "Error reduction by parallel repetition – the state of the art". *Technical report CS95-32 of the Weizmann Institute, 1995.*

[3] U. Feige, J. Kilian. "Two Prover Protocols – Low Error at Affordable Rates". *SIAM Journal on Computing*, 30(1), 324–346, 2000.

[4] J. Hastad. "Some optimal inapproximability results". *JACM* 48(4): 798–859 (2001).

[5] S. Khot. "On the power of unique 2-prover 1-round games". *Procedings of 34th STOC*, 767–775, 2002.

[6] S. Khot, O. Regev. "Vertex cover might be hard to approximate to within 2 - ϵ". *Computational Complexity Conference*, 2003.

[7] R. Raz. "A prallel repetition theorem". *SIAM Journal on Computing*, 27(3):763–803, 1998.