1 Outer Verifiers and Inner Verifiers

Lecture 5: May 14, 2003

We have shown that $NP \subset PCP(\log n, \text{polylog } n)$. (Recall that the first parameter is the number of random bits, and the second parameter is the number of bits that are queried, up to a multiplicative constant factor.) We wish to show that $NP \subset PCP(\log n, 1)$. The approach that we use is based on composition of PCPs. At a high level, the idea is simple. The $PCP(\log n, \text{polylog } n)$ system is a method by which a prover convinces a verifier of the validity of an NP statement. Given an NP statement of size n (e.g., a 3CNF formula with n variables that is claimed to be satisfiable), the prover writes down a PCP witness of polynomial size. The verifier, which we will call here an *outer verifier*, reads at random only a polylogarithmic number of bits, and then bases its decision of whether to accept or reject only on these bits. The input to this decision is thus a string of length polylog n. The idea is that instead of requiring the outer verifier to read these polylog n bits, the prover will prepare an "inner" PCP witness (in fact, one for each possible sequence of polylog n bits that the verifier may wish to read) proving that had the outer verifier read the bits, he would have accepted. Now an *inner verifier* inspects the inner PCP witness, and needs to read from it only polylog (polylog n) bits, which is polynomial in $\log \log n$. Hence we achive a drastic reduction in the number of bits read. Applying this idea recursively, we may hope to eventually show that $NP \subset PCP(\log n, 1)$.

There are many potential problems in implementing the idea presented above. We shall explain some of them, and show the basic idea of how they are handled.

The approach described here is based on [2, 1]. We tried to keep the same notation as in [1]. The writing is less formal than that of [1], and follows more closely the lecture given in class.

1.1 Running time of verification procedure

The basic idea calls for applying a scaled down version of the PCP verifier as the inner verifier. The problem is scaled down in the sense that the input to the problem is much shorter, only polylogarithmically many bits. However, this might not be enough. If the running time of the outer verifier on these query bits is too long (e.g., polynomial in the original n, rather than polynomial in the number of query bits), then not all parameters of the problem have been scaled down, and we may fail to construct an inner PCP. For this reason we showed in Lecture 3 that even though the decision of which bits to query takes polynomial time, the computations that the verifier perform on the queried bits take only polylogarithmic time.

We view the PCP outer verifier as working in three phases. In the first phase that may take polynomial time, it decides which bits to query, and writes down a circuit C (or a 3CNF formula) of size $c(n)$ that describes how to evaluate the answers to the query bits. In the second phase $q(n)$ bits are queried. In the third phase, the circuit is evaluated with the answer bits as its inputs, and the output of the circuit is either accept or reject.

To allow for composition, it is necessary not only that the number of query bits be small, but also that the circuit C is small, namely, $c(n) = O(\text{polylog } n)$.

1.2 Constructive proofs

In the PCPs that we constructed, for any particular choice of random bits of the verifier it is the case that there is a sequence of answer bits of the provers that causes V to accept. (The difficulty for the prover is to find one PCP witness that works for every random string r of the verifier. That is, the PCP witness simulatneously causes all respecive circuits C_r to accept, where each circuit C_r has as input some of the bits of the PCP witness.) Hence we do not want the statement proven to the inner verifier to be "there are inputs that would cause the circuit C_r to accept". We want a more constructive version saying: "the answer bits of the outer PCP witness would cause C_r to accept".

The approach that we take is as follows. For every r, the inner prover writes down a PCP witness π_r that circuit C_r has a satisfying assignment. As PCPs are constructive, then in fact π_r includes the encoding of such a satisfying assignment. Call the encoding scheme used E. (In Lecture 3 we saw that the satisfying assignment is encoded using degree-h multivariate polynomials.) The inner verifier, in addition to treating π_r as a PCP witness for the statement that C_r is satisfiable, will also verify that the encoded assignment is equal to the assignment written (for the respective answer bits) in the outer PCP witness.

1.3 Small number of probes

We now face the following problem. There is an assignment encoded in the inner PCP witness π_r, and the inner verifier needs to verify that it is identical to the assignment written in some prespecified locations in the outer PCP witness. The inner verifier does not have sufficient queries to read these locations. So how can he check equality?

To get over the above problem, an approach that may be thought of as "parallelization" is used. The outer PCP is modified so that for every random string r of the outer verifier, the $q(n)$ query bits are not spread out over the whole PCP witness. Instead, they are grouped into p blocks, where each block contains $a(n)$ answer bits, with $q(n) = a(n)p$. In fact, we may think of the outer PCP witness as composed of characters rather than bits, where each character is encoded by a bit-string of length $a(n)$. The outer verifier probes p characters, where p is some constant.

To allow for the consistency check between the outer PCP witness and the inner PCP witnesses, each character is encoded using the same encoding scheme E that is used for the inner PCPs. Using the fact that E is a code with large hamming distance, one can probabilistically check that two code words are the same by querying only a small number of bits from each code word. Hence the consistency check can in principle be performed using only $O(p) = O(1)$ queries, and with high confidence show that the assignment implicit in the inner PCP π_r is the same assignment one would get from the answer bits of the original outer PCP.

1.4 What should the inner verifer verify

After running the first stages of the outer verifier, one has a list of p locations to probe, and circuit C_r to check. The outer prover claims that in these locations there is an encoding of a satisfying assignment for the circuit C_r. The inner verifier first checks that in every one of these locations, the string written is close to a valid codeword. (This is similar in nature to the degree-h test that we have already seen.) This requires reading a small number of bits from each probe location. Thereafter, the inner verifier checks consistency of the inner PCP witness π_r with the respective codewords of the outer PCP. Finally, the inner verifier checks whether π_r is a convincing PCP witness for the circuit C_r.

1.5 The inner verifier is an outer verifier

To allow further shrinking the answer sizes, the inner verifier serves as an outer verifier for an even more inner verifier. Hence it too will need to construct a circuit $C'_{r'}$ of size polynomial in the number of its own query bits. Likewise, it too should probe π_r only in a constant number of locations (where each location contains a character from some large alphabet, represented as a string of polylog $(a(n))$ bits).

1.6 Ending the recursion

The number of bits required in order to describe a single character in a probe location decreases by a polylogarithmic factor in each application of the inner verifier. To get $a(n)$ down to $O(1)$, one needs $\log^* n$ compositions of inner verifiers as described above. This is problematic, because the number of probe locations p seems to increase in every iteration, and hence when $a(n)$ becomes constant, p is no longer a constant. Hence $q(n) = a(n)p$ is never constant.

The idea around this problem is to have two types of inner verifiers. One is as described above. The other will be applied only when the input size becomes so small that one can afford a PCP witness of length exponential in the size of the input statement. Then, a PCP with different tradeoffs between the various parameters is constructed, namely, $PCP(r = O(n^2), q = O(1))$.

1.7 The third inner verifier

(This part may seem cryptic at the moment, and can be skipped at this point in the course.)

The approach that we described above uses one outer verifier and two types of inner verifiers. To get even more efficient PCPs (e.g., efficient enough to prove that max-3SAT is had to approximate within a ratio better than 7/8) we shall need a third inner verifier. Its outer verifier, rather than being a PCP, will be a simple two prover proof system for max-3SAT (based on hardness of approximating max-3SAT within a ratio better than $1 - \epsilon$). The error of this proof system will be reduced using parallel repetition (that keeps the number of probes equal to 2). Then an inner verifier will be build on top of it. It is possible to show that for some hardness of approximation results (e.g., $n^{1-\epsilon}$ for clique), there is a fundamental limitation with our approach based on the unique decoding feature of error correcting codes. To get around this limitation we need to give up unique decoding, and go to analysis based on *list decoding*. Analysis based on list decoding often requires the error of the outer PCP to be very small.

It appears that we might not have time in the course to describe this third inner verifier. Details appear in [3, 4].

1.8 Formal definition for inner verifier

To allow for composition of inner verifier and outer verifiers, we make explicit the parameters of an inner verifier. First, there is some encoding scheme E. There is a decoding scheme E^{-1} that is applicable for every valid code word, and also to some noncode words (as E is an error correcting code). Suppose that the outer verifier probes k locations, each including a character represented by m bits. The outer verifier evaluates the answers using a circuit C of size n. Then the inner verifier expects to see k code words over some alphabet with at most $2^{a(n)}$ characters, and an additional inner PCP π. The parameters of the inner verifier are denoted by $(k, r(n), p(n), c(n), a(n), e)$.

k – number of code words that the inner verifier expects as input.

$r(n)$ – number of random bits used by inner verifier.

p – number of locations that are probed by inner verifier.

$c(n)$ – size of circuit C' generated by the inner verifier.

$a(n)$ – number of bits needed to represent the character in a single probe location.

e – error.

Using his random bits, the inner verifier decides on p probe locations and on a circuit C'. The inner verifier accepts if the answers to the p probes satisfy circuit C'.

To have a PCP system $RPCP(k, r(n), p, c(n), a(n), e)$ we need:

Completeness: If $x_1, \ldots, x_k$ is a satisfying assignment for C, then there is an inner PCP witness π such that the inner verifier accepts $E(x_1), \ldots, E(x_k)$, π.

Soundness: If the inner verifier has probability higher than e of accepting $X_1, \ldots, X_k, \pi$, then $E^{-1}(X_1), \ldots, E^{-1}(X_k)$ is a satisfying assignment for C.

(**Remark:** the soundness ondition is based on the unique decoding principle that was extended to list decoding in later work of Hastad [3, 4].)

1.9 Composition theorem

Say that we have two PCP systems. One is

$$RPCP(k, r_1(n), p_1, c_1(n), a_1(n), e_1),$$

the other is

$$RPCP(p_1, r_2(n), p_2, c_2(n), a_2(n), e_2).$$

Let us denote $c_2(n)$ by n'. Then they can be composed to give an

$$RPCP(k, r_1(n) + r_2(n'), p_2, c_2(n'), a_2(n'), e_1 + e_2 - e_1 e_2).$$

The new PCP works as follows. Given an input x of size n, we use the first PCP as an outer verifier. That is, the outer prover writes down a proof in which each location contains a character that is represented by a sequence of $a_1(n)$ bits. The outer verifier uses $r_1(n)$ random bits (denote this random string by r) to generate a circuit C_r of size $n' = c_1(n)$, and to compute p_1 locations to probe. Thereafter, the inner prover encodes the contents of each location of the outer PCP using encoding scheme E. In addition, for each possible random string r of the outer verifier, the inner prover writes down a proof π_r. The inner verifier has as input a circuit C_r of size n', and p_1 codewords. It uses $r_2(n')$ random bits (denote this random string by r') to generate a circuit $C'_{r'}$ of size $c_2(n')$, and $p_2(n')$ locations to probe. It runs the second RPCP on the k codewords indexed by the first PCP, and on π_r.

It is clear from the description that we get the comlexity parameters of the new PCP. In addition, it is clear that if $x \in L$ the combination of inner and outer prover will cause the inner verifier to always accept. Conversely, assume that $x \notin L$. Construct an outer prover by decoding all code words created by the inner prover (whenever unique decoding is possible), and discarding all π_r. Then on this the outer verifier accepts with probability at most e_1. When the outer verifier does not accept, the inner verifier may still be unlucky and accept, but by the soundness of the inner verifier, this has probability at most e_2. Hence the probability of acceptance is at most $e_1 + e_2 - e_1 e_2$.

1.10 Error reduction

The composition slightly increases the error. This is not a problem, as error can always be reduced to its kth power by k repetitions, with a multiplicative increase in (some of) the complexity parameters by a factor of k. We shall ignore this issue.

1.11 The PCPs that we will use

Theorem 1 *For every integer $k \geq 1$, there are constants c_1, c_2, c_3, positive integer p (that will be larger than k) and $e < 1$ such that there is a*

$$(k, r = c_1 \log n, p, c = O(\log^{c_2} n), a = O(\log^{c_3} n), e)$$

inner verifier.

Note that we were quite close to designing the above in Lecture 3. The main problem is that p is not a constant there, and that k should be though of as equal to 1, rather than an arbitrary constant.

To finish off and end with a constant number of query bits we use:

Theorem 2 *For every integer $k \geq 1$, there is a constant c_1, an integer $p \geq 1$ and $e < 1$ such that there is a*

$$(k, r = c_1 n^2, p, c = 2^p, a = 1, e)$$

inner verifier.

Applying the composition theorem to Theorems 1 and 2 will show that $NP \subset PCP(\log n, 1)$. First, by modifying the PCP constructed in Lecture 3 along the lines of Theorem 1 we obtain:

$$NP \subset RPCP(k = 1, r_1 = O(\log n), p_1, c_1 = O(\text{polylog } n), a_1 = O(\text{polylog } n), e)$$

Then, applying Theorem 1 once more we obtain:

$$NP \subset RPCP(k = 1, r_2 = O(\log n), p_2, c_2 = (\log \log n)^{O(1)}, a_2 = (\log \log n)^{O(1)}, e)$$

Finally, applying Theorem 2 we obtain:

$$NP \subset RPCP(k = 1, r_3 = O(\log n), p_3, c_3 = 2^{p_3}, a_3 = 1, e)$$

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy. "Proof verification and hardness of approximation problems". *JACM 45(3): 501–555 (1998).*

[2] S. Arora, S. Safra. "Probabilistic checking of proofs: a new characterization of NP". *JACM 45(1): 70–122 (1998).*

[3] J. Hastad. "Clique is hard to approximate within $n^{1-\epsilon}$". *Acta Mathematica* Vol. 182, 1999, 105–142.

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