IAS/PCMI Summer Session 2000
Clay Mathematics Undergraduate Program
Basic Course on Computational Complexity

Lecture 15: A Brief Look at PCP

David Mix Barrington and Alexis Maciel
August 4, 2000

1. Overview

We come now to the strangest of our models of "efficiently provable languages", that of *probabilistically checkable proofs*. By a theorem of Arora and Safra, if A is any language in NP, Alice can prove to Bob that $x \in A$ by giving him a proof and having him look at *only a constant number of randomly chosen bits of it*. We won't be able to prove this theorem here. Instead we will show some of its implications for optimization problems and give an overview of one example of a probabilistically checkable proof (though this won't meet the conditions of the Arora-Safra theorem).

- We begin with a look at *optimization problems* such as MAX-3SAT, where the input is a 3CNF boolean formula and the output is the maximum number of clauses that can be satisfied by any one assignment. It is NP-hard to answer this question exactly, and we investigate the difficulty of *approximating* the correct answer to within some multiplicative factor.

- We define the notion of a probabilistically checkable proof, and the class PCP$(r(n), q(n))$ of languages for which such proofs exist. The parameter $r(n)$ refers to the number of random bits Bob can use, and $q(n)$ refers to the number of bits of the proof he is allowed to see. The PCP *theorem* says that NP $=$ PCP$(\log n, 1)$. We show here that PCP$(\log n, 1) \subseteq$ NP.

- We show that the PCP theorem is *equivalent* to a statement about optimization problems, that there exists a constant $\epsilon > 0$ such that MAX-3SAT cannot be approximated within a multiplicative error factor of ϵ unless P $=$ NP.

- We outline an example of a probabilistically checkable proof for SAT, though this uses $O(n^3)$ random bits and thus only places SAT in the class PCP$(n^3, 1)$.

We begin to explain how Bob's $O(1)$ queries for bits of the proof can give him any confidence that the proof is correct, by introducing the notion of a *linearity test*.

2. Optimization Problems

NP-completeness, like all our other complexity notions, is defined in terms of decision problems, where the input is a string and the output is a boolean telling whether the string is in some language. Much of the interest in NP-completeness, however, is due to the fact that (unless P = NP) it allows us to prove that certain *optimization problems* are not solvable in polynomial time. In general, we can think of an optimization problem as an input defining a mapping from strings to integers, where the output is to determinine the maximum (or minimum) value of the function on all strings of a certain length.

For example, in the MAX-3SAT problem the input is a 3CNF boolean formula and the output is the maximum number of clauses that can be satisfied by any assignment. Since we know that (unless P = NP) it is hard to tell whether all the clauses can be satisfied, there can be no poly-time algorithm to get the exact answer for MAX-3SAT.

Similarly, the Traveling Salesperson Problem or TSP inputs a weighted graph (where the edges are labeled by numbers) graph and asks the minimum total weight of any path the visits all the nodes. The language of pairs $\langle G, k\rangle$, such that G has a path of weight k or less, is proved to be NP-complete in Sipser, for example. Again it follows that the optimization problem has no exact poly-time solution unless P = NP.

A natural question, if we believe that there is no exact solution to these problems, is how well we can *approximate* the solution with a poly-time algorithm. Can we, for example, come within a factor of ϵ of the correct solution (that is, produce a solution for a maximization problem that we know is between $(1-\epsilon)s$ and s, where s is the optimal solution)? Here the various NP-complete problems exhibit different sorts of behavior:

- TSP cannot be approximated within *any* multiplicative factor unless P = NP. Consider the NP-complete *Hamilton path problem* (whether there exists a path visiting each node exactly once) and translate it into a TSP problem as follows. Given a graph G, create H with the same node set. For each edge of G, make an edge of H with weight 1. For each non-edge of G, make an edge with weight Z, where Z is some huge number. There is a Hamilton path in G iff there is a path through all the nodes of H with total weight less than $Z + (n-2)$, where

n is the number of nodes. Any approximation of TSP to a factor much less than Z/n would tell us whether the Hamilton path exists and thus solve an NP-complete problem.

- Other problems, such as SUBSET SUM, have poly-time approximate solutions for any ϵ. We showed in an exercise that SUBSET SUM can be solved exactly if the numbers are small enough to represent in unary. By approximating the actual numbers to the desired degree of accuracy and then finding the exact solution to the approximate problem, we can achieve an approximation to within a factor of ϵ for any constant.

- MAX-3SAT is among a class of problems that resisted classification into either of these categories for many years. We will show below that it has a poly-time approximation algorithm that finds a setting that satisfies at least 7/8 (for "MAX-EXACTLY-3SAT", where each clause contains three distinct literals) of the number of clauses that can possibly be satisfied. But now, as a result of the PCP theorem, it is known that there is a constant $\epsilon > 0$ such that (unless P = NP) no poly-time algorithm can find a setting to satisfy more than a $1 - \epsilon$ fraction of the best possible number of satisfied clauses.

The 7/8 approximation algorithm is a greedy one. Let ϕ be a 3CNF formula with c clauses, n variables, and three distinct literals per clause. Note that each clause is satisfied by exactly 7/8 of the possible settings. Since expectation of different random variables always add, the expected number of clauses satisfied by a random setting of the variables is exactly $7c/8$. Our greedy algorithm will find a setting that does at least this well.

Consider the formulas ϕ_0 and ϕ_1 obtained from ϕ by setting variable x_1 to 0 and 1 respectively. We can calculate the expected number of clauses of each of these formulas satisfied by a random setting of $x_2, \ldots, x_n$. (We simply add together the probabilities of satisfaction for each clause.) Since the average of these is $7c/8$, at least one of them is no less than $7c/8$, and we choose that assignment of x_1 for our setting. In the same way, we asssign the rest of the variables one by one in such a way as to maximize the expected number of clauses satisfied. Since this number starts as $7c/8$, never goes down, and finishes as the number of clauses that *are* sastisfied by our greedy setting, this setting must satisfy at least $7c/8$ clauses. Since the optimal setting can do no better than c clauses, we have an approximation algorithm with error factor at most 1/8.

3. The Definition of Probabilistically Checkable Proofs

We now formally define probabilistically checkable proofs and the classes PCP$(r(n), q(n))$. A language A is in PCP$(r(n), q(n))$ iff there exists a proof system for A as follows. A clever Alice provides a string, called a proof, which may be of any length but is *invisible to Bob* except as follows. The polynomial-time Bob flips $O(r)$ random bits, and based on those bits and on the input string x, chooses $O(q)$ particular bits of the proof string. On the basis of those bits and x, he decides whether to accept. If $x \in A$, there must exist a proof string such that Bob accepts with probability 1. (Note that we are in the "one-sided error" case here.) If $x \notin A$, no proof may lead Bob to accept with probability greater than 1/2. Note that this 1/2 may be replaced by any constant, but it is not clear that it can become smaller than constant because repeating the proof more than $O(1)$ times costs Bob in terms of random bits and proof bits.

The PCP Theorem states that PCP$(\log n, 1) =$ NP. That is, Bob can flip $O(\log n)$ bits, in effect choosing $O(1)$ random numbers that are polynmomial in n. On the basis of these, he chooses $O(1)$ bits to query and decides whether $x \in A$ after looking at those bits. The hard half of the PCP Theorem is to prove that NP-complete problems have these strange proofs. Here we prove the easy half:

Lemma 1 PCP$(\log n, 1) \subseteq$ NP.

Proof Suppose that Bob flips $c \log n$ bits and consider the n^c possible results of these flips. In each of them, Bob will query $O(1)$ bits of Alice's proof, so he asks about $O(n^c)$ of them in all. But then consider an ordinary proof system, where Alice provides all these bits as her proof and Bob may look at all of them. A deterministic poly-time Bob may simulate the randomized Bob in each of the n^c cases, using the provided bits as needed, and determine the probability that the randomized Bob will accept the original proof. This probability determines whether $x \in A$. $\square$

4. The PCP Theorem Stated in Terms of MAX-3SAT

We now demonstrate the relationship between probabilistically checkable proofs and optimization problems, by showing:

Theorem 2 NP $\subseteq$ PCP$(\log n, 1)$ *iff there exists a constant* $\epsilon > 0$ *such that MAX-3SAT is* NP*-hard to approximate within* ϵ.

Proof Of course this theorem is true because both of the statements in the "iff" are true, but the relationship between the two hypotheticals is still instructive.

We first state an equivalent form of the hypothesis about MAX-3SAT that will be useful in our proof. (You will prove the equivalence in the exercises.) A *gap-producing reduction* from SAT to MAX-3SAT with parameter $r < 1$ is a function that takes satisfiable formulas to satisfiable formulas, and unsatisfiable formulas to formulas where at most an r fraction of the clauses can be satisfied. You are asked to prove that if P $=$ NP, MAX-3SAT has no poly-time approximation within ϵ for some $\epsilon > 0$ iff there is a gap-producing reduction from SAT to MAX-3SAT with parameter r for some $r < 1$.

Given this fact, we first assume that the gap-producing reduction exists and use it to construct a probabilistically checkable proof system for SAT with $O(\log n)$ random bits and $O(1)$ proof bits. Let ϕ be the input formula and ψ the formula given by the reduction. (Note that Bob can compute ψ himself from the input.) Our proof consists of an assignment for the variables of ψ. Bob will use his random bits to select k random clauses of ψ, and he will accept iff all k clauses are satisfied by the proof assignment. Thus he looks at no more than $3k$ bits of the proof, and uses at most $kc \log n$ random bits if the length of ψ is at most n^c. We have only to show that some constant k satisfies the other conditions of the problem.

This proof system is sound because if ϕ is satisfiable, so is ψ. Alice can and will use a satisfying assignment to ψ as her proof. In this case all Bob's random clauses will be satisfied and he will accept with probability 1. If ϕ is not satisfiable, the gap-producing reduction ensures that the probability that a random clause of ψ is satisfied, by whatever assigment Alice chooses to send, is at most r. We need merely choose k so that $r^k < 1/2$ (possible since $r < 1$ is a constant). Then the probability that Bob mistakenly accepts because he chooses a satisfied clause k times in a row is less than $1/2$, meeting the conditions for a sound probabilistically checkable proof.

Now assume that SAT is in PCP$(\log n, 1)$, and we will create a gap-producing reduction from SAT to MAX-3SAT. Given a formula ϕ, there is a proof string y such that Bob can flip $c \log n$ bits, look at k bits of y, and decide whether to accept. Consider the following k-CNF formula ψ. For each of the n^c random choices $z_1, \ldots, z_{n^c}$ of Bob, let S_i be the set of k proof bits that Bob queries in that case. The formula ψ will have a clause *ruling out* each setting of the bits in S_i that would cause Bob to reject, based on his poly-time algorithm that looks only at ϕ and S_i. Let d_i be the number of clauses in ψ arising from S_i. Thus ψ has a total of $d = \sum_i d_i \leq 2^k n^c$

clauses. If ϕ is satisfiable, ψ will be satisfied by the correct proof, meaning that all d clauses can be satisfied at once. If ϕ is not satisfiable, then in any setting at least half the possible random choices lead to at least one unsatisfied clause. Therefore only at most $d - n^c/2$ clauses of ψ are satisfied by any assignment. The ratio $(d - n^2/2)/d$ is bounded above by a constant, $1 - (1/2^{k+1})$.

We would be done, except that we have made a gap-producing reduction to MAX-kSAT rather than MAX-3SAT. In an exercise you are asked to fix this problem, by a technique similar to that used to show the NP-completeness of 3SAT given that of CNF-SAT. $\Box$

5. An Example of a Probabilistically Checkable Proof

The concept of probabilistically checkable proofs is a strange and unintuitive one, and the proof of the PCP theorem itself is rather difficult. Why should such strange proof systems exist at all? Here we sketch an example of a proof system for SAT based on work of Blum-Luby-Rubinfeld, where Bob in fact looks at only $O(1)$ bits of the proof, accepts a valid proof with probability 1, and rejects an invalid proof with probability at least ϵ, a constant. The reason that this is not a proof system proving the PCP theorem is that this proof will be $2^{O(n^3)}$ bits long, and thus Bob must flip $O(n^3)$ random bits to be able to have potential access to all of it. This argument does show NP $\subseteq$ PCP$(n^3, 1)$. The validity of the system will be largely left to the exercises.

The input string will be a formula ϕ in 3CNF with exactly three distinct literals per clause. The proof will be a *redundant encoding* of a particular assignment a, a string of n bits. The information in a will be spread out among $2^n + 2^{n^2} + 2^{n^3}$ bits of proof, in such a way that $O(1)$ randomly chosen queries by Bob will have a non-negligible chance of detecting either (a) a violation of the procedure for encoding a as a proof, or (b) a validly encoded a that is not a satisfying assignment for ϕ.

The proof will consist of three functions taking n, n^2, and n^3 input bits respectively, where each bit is viewed as an element of $\mathbf{Z}_2$, and returning single bits. These are defined in terms of the bits $a_1, \ldots, a_n$ of the assignment i, and are stored in the form of "truth tables", simply lists of the 2^n, 2^{n^2}, and 2^{n^3} values of each function. Whatever proof Alice sends, Bob will interpret it as truth tables for these three functions, so we can assume without loss of generality that Alice sends the three functions. The intended interpretation of the three functions is as follows:

- The function f is the $\mathbf{Z}_2$ dot product of a with its input string x, that is, $f(x_1, \ldots, x_n) = \sum_i a_i x_i$, where the sum is taken in $\mathbf{Z}_2$.
- The function g takes as input an n by n matrix of bits $y_{11}, \ldots, y_{nn}$ and outputs the dot product of this matrix with the tensor product of a with itself. Specifically, $g(y_{11}, \ldots, y_{nn}) = \sum_i \sum_j a_i a_j y_{ij}$.
- The function h takes as input an n by n by n cube of bits and outputs its dot product with the tensor product of a, a, and a. Specifically, $h(z_{111}, \ldots, z_{nnn}) = \sum_i \sum_j \sum_k a_i a_j a_k z_{ijk}$.

Bob must verify six facts about these three functions to his satisfaction, which he will be able to do with $O(1)$ query bits per fact. Crucially, he will be satisfied if he has shown that *most* of the entries of these functions are consistent with a properly encoded a — naturally he cannot hope to tell the difference between a proper encoding and a proof that differs from a proper encoding in less than a constant fraction of the entries.

- f is a linear function of the variables x_i.
- g is a linear function of the variables y_{ij}.
- h is a linear function of the variables z_{ijk}.
- The n^2 coefficients of g are the tensor product of the vector of n coefficients of f with itself.
- The n^3 coefficients of h are the tensor product of those of g with those of f.
- The assignment coded by f, g, and h in fact satisfies ϕ.

The following *linearity test lemma* allows Bob to verify the first three conditions above. The proof is an exercise in the use of elementary tools, and is assigned below. For a complete proof, see the web version of Arora's thesis on his site at Princeton University.

Lemma 3 *(Linearity Test Lemma) Suppose F is a function of m bits. Let u and v be randomly chosen vectors of m bits each. If $F(u) + F(v) = F(u + v)$ in $\mathbf{Z}_2$ with probability $1 - \delta$ (over the choice of u and v), then there is a linear function F^* such that $F^*(x) = F(x)$ with probability at least $1 - 6\delta$ (over choice of x).*

We test the fourth equality as follows (the fifth is similar). We pick two n-bit vectors u and v at random, and set w to be the tensor product of u and v, i.e., $w_{ij} = u_i v_j$. However, we don't simply test whether $f(u)f(v) = g(w)$, as it should, because this would only test certain entries of g. Instead we also generate random vectors u' and v' of length n and w' of length n^2, and test whether:

$$[f(u+u') - f(u')][f(v+v') - f(v')] = g(w+w') - g(w')$$

Lemma 4 *(Tensor Product Test Lemma) Let $\epsilon > 0$. There exists $\delta > 0$ such that if f and g pass the linearity test and the test above with probability $1-\epsilon$, then for the linear functions f^* and g^* given by the Linearity Test Lemma, g^* is the tensor product of f^* with itself with probability at least $1-\delta$.*

Let us conclude by seeing how Bob verifies the sixth condition, and thus what this construction has to do with satisfiability. For each clause of ϕ, Bob constructs a polynomial c_i of degree 3 that is equal to 0 if that clause is satisfied and 1 otherwise. For example, if the clause were $x_1 \vee (\neg x_2) \vee x_3$, the polynomial would be $(1-x_1)(x_2)(1-x_3)$. If a setting a satisfies ϕ, the values $a_1, \ldots, a_n$ will make each polynomial c_1 equal zero. If a does not satisfy ϕ, at least one c_i will be equal to one.

As in Smolensky's Theorem (Basic Lecture 10) we take a *random linear combination* of the polynomials c_i. We choose a random vector r, whose length is m, the number of clauses of ϕ, and define the polynomial $\Pi_r = \sum_i r_i c_i$. Note that for any given non-satisfying a, $\Pi_r(a) = 1$ for exactly half the possible r. (This is because for the i with $c_i(a) = 1$, $r_i c_i(a)$ is equally likely to be 0 or 1 and this is added, modulo 2, to the rest of the sum.)

Once r is chosen, we can write $\Pi_r(a)$ as the sum of a constant, a pure linear, a pure quadratic, and a pure cubic polynomial in $a_1 \ldots, a_n$. But each of these can be obtained from one of our proof polynomials:

- $b_0 = B_0$
- $\sum_i b_i a_i = f(b_1, \ldots, b_n) = f(B_1)$
- $\sum_i \sum_j b_{ij} a_i a_j = g(b_{11}, \ldots, b_{nn}) = G(B_2)$
- $\sum_i \sum_j \sum_k b_{ijk} a_i a_j a_k = h(b_{111}, \ldots, h_{nnn}) = H(B_3)$

We compute $\Pi_r(a)$ as

$$B_0 + f(B_1 + E_1) - f(E_1) + g(B_2 + E_2) - g(E_2) + h(B_3 + E_3) - h(E_3)$$

where E_1, E_2, and E_3 are three final randomly chosen vectors. Bob rejects if the value of Π_r calculated from the proof is not 0.

It is easy to see that if a is a satisfying assignment and all of Alice's functions are as advertised, all the tests will be passed and Π_r will evaluate to 0. The hard part of the proof, of course, is verifying that if Alice provides a bad proof, Bob has at least an ϵ chance of rejecting it. The argument goes roughly as follows. If Alice's functions differ significantly from linear functions, there is a good chance Bob will reject because of the linearity tests. If they are close to linear functions that fail to satisfy the tensor product conditions, there is a good chance that Bob will reject because of the tensor product tests. But if we get this far, then there is probably an assignment a such that most of Alice's f, g, and h are consistent with the functions defined from a.

Now this a cannot satisfy ϕ if ϕ is unsatisfiable. So there is at least one clause that it fails to satisfy, meaning that $\Pi_r(a) = 1$ for half the choices of r. If Bob's calculation of $\Pi_r(a)$ uses only bits of f, g, and h that agree with the functions computed from a, it will be correct, and with probability 1/2 he will reject because of the sixth chance. We have merely to complete the details to show that whatever bad proof Alice submits, Bob has a probability of rejecting that is greater than some constant.

6. Exercises

1. Verify that SAT is in PCP$(r(n), q(n))$ iff NP $\subseteq$ PCP$(r(n), q(n))$.
2. Prove that if P $\neq$ NP, there exists $r < 1$ such that a gap-producing reduction exists from SAT to MAX-3SAT with parameter r iff there exists $\epsilon > 0$ such that it is impossible to approximate MAX-3SAT within a factor of ϵ in polynomial time.
3. Assume that there is a gap-producing reduction from SAT to MAX-kSAT with parameter $t < 1$. Show that there is a gap-producing reduction from SAT to MAX-3SAT (where each clause contains three distinct literals) with some parameter $r < 1$. (Hint: For each k-variable clause make a 3SAT formula, adding new variables, that is satisfiable iff the clause is satisfiable.)
4. Prove the Linearity Test Lemma.
5. Prove the Tensor Product Test Lemma.
6. Prove that if $c_i = 1$ for some i, then $\sum_i r_i c_i$ is one for exactly half the possible vectors $r_1, \ldots, r_m$, where the sum is taken modulo 2.