Probabilistically Checkable Proofs

Arindam Kulkarni

§1 Introduction

A probabilistically checkable proof (PCP) is a type of proof that can be checked using a randomized algorithm. The idea is that a proof is a string that can be read. We can check specific parts at random of this string, and probabilistically decide whether the proof is correct or not, through these bounded random parts. Thus, the algorithm is expected to verify/reject proofs at a decently high probability.

Although you cannot guarantee full accuracy, as we check random parts, you can make a good predictor, and questions come up such as: How many parts much we check on average? How accurate is it

There are different number of complexity classes based on how many bits of info we take. $PCP[r(n), q(n)]$ is a class that refers to problems with probabilistically checkable proofs that can be verified in polynomial time using at most $r(n)$ random bits through reading a max of $q(n)$ bits of the proof. Through the PCP theorem, we show that $PCP[O(logn), O(1)] = NP$.

Probabilistically checkable proofs have verifiers through the form of Turing Machines in NP. Through reading the input in a string and a string of random bits, the PCP will decide which of the random bits it wants to check. Through the process, the PCP either accepts the input x or rejects it if the proof is wrong – only on the knowledge of the $q(n)$ bits read.

§2 Definitions

A language (a decision problem) is said to have a probabilistically checkable proof if it goes as follows:

1. For all input strings x in the language that should be accepted, there exists a proof which the verifier accepts for all random strings
2. For all x not in the language, the verifier rejects all proofs over $\frac{1}{2}$ the time.

More formally:

Definition 2.1. A probabilistically checkable proof system for language L is a probabilistic polynomial-time verifier, denoted M, satisfying:

1. Completeness: For every $x \in L$ there exists an oracle π_x such that PCP[M^{π_x} where x accepts] $= 1$
2. Soundness: For every $x \notin L$ there exists an oracle π_x such that PCP[M^{π_x} where x does not accept] $\geq \frac{1}{2}$

where the probability is taken over M's internal coin tosses

Various complexity measures influence the probabilities of completeness and soundness. In this definition, we can think of a "coin toss" as one chance to potentially see a bit. It would thus count as a part of $r(n)$. The idea of adaptability, the idea of a decision tree in which bits are read systematically, is measured through the query complexity $q(n)$. Normally, unless specified we consider a verifier that can consider bits to check as it goes. If not, $q = 0$, leaving us with $r(n)$.

Theorem 2.2

The first 2 parts are beyond the scope of this paper to prove, but we will prove the 3rd statement. Note that *poly* is polynomial time.

1. If $NP \subseteq PCP(o(log), o(log))$ then $NP = P$
2. $PCP(poly, poly) = NEXP$
3. $PCP(0, poly) = NP$

Proof. Our $q = poly$, $r = 0$. Thus, we are not allowed to flip any counts. However, we have a polynomial time number of queries. As a result, we can simply read and confirm the proof in polynomial time, resulting in NP. □

This shows that languages in NP are captured through no randomness ($r = 0$) and a polynomial number of queries.

§3 PCP Theorem

We must then consider whether there could be a combination of these complexity measures, still capturing NP. The answer is yes. Through the PCP Theorem, we show that the NP complexity class has probabilistically checkable proofs that are with constant query complexity q and logarithmic randomness complexity r. In other words, we can bring the number of queries the verifier makes down to a constant while just using a logarithmic number of coin tosses.

The basis for the proof starts with the following theorem:

Theorem 3.1
$PCP(log, poly) \subseteq NP$

Proof. This is the class of languages having a PCP system where the verifier has a logarithmic number of coin tosses, and a polynomial number of queries.

Let L be a language in $PCP(log, poly)$. We can show how we can use the PCP system in order to have a nondeterministic turing machine M_t be able to decide L in logarithmic time, consequently showing L is in NP.

Let M be the probabilistic polynomial time machine in the above $PCP(log, poly)$ system for L. Due to the difficulty, we will only prove this claim for a non-adaptive M. We will have a set of $\{a_1, ..., a_m\}$ containing all possible outcomes of the coin tosses made by M (understand that $|a_i| = O(log(|x|))$ and thus $m = 2^{O(log(|x|))} = poly(|x|))$ as $2^{log} \in poly$. With coin sequence a_i, we will call $({q_1}^i, ..q^i_{n_i})$ the queries that M makes. We must consider that n_i could differ by i, and is polynomial in $|x|$. Since we are assuming this to be non-adaptive, we have the queries dependent on our input x and the coin sequence a_i. They do not depend on q, or our previous queries.

Through our definition of completeness, we know that for every $x \in L$, there exists an oracle π_x for which the PCP system accepts with a probability of 1 if given access to π_x.

We then create a witness w, given $x \in L$ and an oracle pi. Now we must consider all possible executions of M given our input x. It is important to note this depends on our coin tosses, a_i. Take the substring of π containing all the bits $\pi^i_{q_j}$ examined by M during these executions. For each substring, we will thus have an entry as $(index, \pi_index)$, or effectively, (query, answer). Thus, our resulting encoded string is ${W_x}^{\pi}$.

Now, we describe the turing machine that decides L in polynomial time. Given input x, our turing machine $M\prime$ first guesses the part of the oracle needed by M. Let us call this guess w. $M\prime$ then makes a simulation of M's execution on our input for each a_i. Every query made by M will be answered by $M\prime$ by the answers which are held in w. In addition, if the answer is not in w, that means $M\prime$ guessed wrong, and it will reject accordingly. This process goes on, and $M\prime$ will accept if and only if M accepts x for all possible a_i.

$M\prime$ creates a simulation of M m times, as this is the number of possibilities in our randomness complexity set a_i (which is polynomial in —x—). As M is a polynomial time machine, $M\prime$ is also a polynomial time since we are effectively multiplying two polynomials.

Now, all that remains to be shown is that $L(M\prime) = L$

1. For all $x \in L$, we will show that there exists w such that $M\prime$ started with x accepts. By the completeness condition which follows from the PCP system for L, there exists an oracle π such that $\Pr[M^{\pi} \text{ accepts}] = 1$. This shows that for all coin sequences a_i, we have that M accepts x. By definition, we have that that $M\prime$ accepts when guessing ${W_x}^{\pi}$
2. For all $x \notin L$, we show that for all w's it is true that $m\prime$ with x will always reject. By the soundness condition of the PCP system for L, all oracles π have that $\Pr[M^{\pi} \text{ accepts}] \leq \frac{1}{2}$. Therefore, for each π, there is at least one coin sequence a_i where the corresponding M fails to accept x when looking at π. Thus, we have that for at least one ${W_x}^{\pi}$ which $M\prime$ predicts, there will also be at least one coin sequence a_i for which $M\prime$ finds a rejecting state for M. This concludes M rejects x.

□

This proof effectively states that given a PCP proof with logarithmic randomness, we utilize the fact that m coinflips means 2^m possibilities, meaning $2^{mlog} = poly$. This is called "packing" it into polynomial size and transforming it into an NP-witness for x. Everything the verifier uses through the coin sequences is bounded by a polynomial.

We then connect this to the nondeterministic turing machine. This will not be shown as it is too complex, but if one is able to prove $NP \subseteq PCP(log, poly)$ we have that $NP = PCP(log, poly)$.

The goal of the PCP theorem is to replace $poly$ with $O(1)$. It is too complex to prove, but we will show the proof to wrap up the paper.

Theorem 3.2
$NP \subseteq PCP(log, O(1))$

This is one of the most complex proofs in theory of computation. In essence, it makes the bound we found in the previous theorem stronger.

§4 References

1. https://www.cs.jhu.edu/ scheideler/courses/600.471_S05/lecture_8.pdf
2. https://en.wikipedia.org/wiki/PCP_theorem
3. https://www.sciencedirect.com/science/article/pii/0012365X9400112V