Probabilistically Checkable Proofs

Doratha Drake

A Probabilistically Checkable Proof system can be thought of as a generalization of $\mathcal{NP}$ and co-$\mathcal{RP}$ because it combines the non-determinism of $\mathcal{NP}$ with the randomness of co-$\mathcal{RP}$. It is a model of computation which consists of a verifier (a polynomial time Turing Machine) which has access to an input x, a string of random bits τ, and a certificate (a proof) π. Unlike NP, the verifier doesn't access the proof directly but requests the contents of specific bits of the proof from an oracle. If the length of the proof is $|\pi|$, then the verifier only needs $\log|\pi|$ many bits to access any of the bits of the proof. This allows a probabilistically checkable proof to be somewhat longer than a proof for $\mathcal{NP}$. In addition to these fixed elements, a PCP system has two parameters $r(n)$ and $q(n)$ which respectively restrict the number of random bits that the verifier is allowed to read and the number of bits of the proof that the verifier is allowed to querie from the oracle. A language L belongs to $\mathcal{PCP}(r(n), q(n))$ if it can be recognized by verifier that reads $\mathcal{O}(r(n))$ random bits and makes $\mathcal{O}(q(n))$ queries to the proof through the oracle. Letting the result of a verifier V's computation be denoted by $\mathrm{V}(x, \tau, \pi)$, recognition is defined as follows:

(i) If $x \in$ L then there exists a proof π_x such that

$$Prob_\tau[V(x, \tau, \pi_x) = \text{ACCEPT}] \;=\; 1$$

(ii) If $x \notin$ L then for every proof π

$$Prob_\tau[V(x, \tau, \pi) = \text{ACCEPT}] \;\leq\; \tfrac{1}{2}.$$

The probability is taken with respect to a uniform distribution over all τ.

It is clear from the above definition that

$$\mathcal{PCP}(0, poly) = \mathcal{NP}.$$

and

$$\mathcal{PCP}(poly, 0) = \text{co-}\mathcal{RP}.$$

Further examples of how the parameters $r(n)$ and $q(n)$ affect the expressiveness of the verifier are

$$\mathcal{PCP}(0, 0) = \mathcal{P}$$

and

$$\mathcal{PCP}(poly, poly) = \mathcal{NEXP}.$$

The most interesting problem was to find out what are the smallest parameters for which $\mathcal{PCP}$ still captures $\mathcal{NP}$. The following results to this end were found

$\mathcal{NP} \subseteq \mathcal{PCP}(\text{poly log } n, \text{poly log } n)$

(Babai, Fortnow, Levin, and Szegedy, 1991),

$\mathcal{NP} \subseteq \mathcal{PCP}(\log n \cdot \text{loglog } n, \log n \cdot \text{ loglog } n)$

(Feige, Goldwasser, Lovasz, Safra, and Szegedy, 1991).

At this point it was assumed that $\mathcal{NP} = \mathcal{PCP}(\log n, \log n)$, but then the following result showed that even these parameters were too large.

$\mathcal{NP} \subseteq \mathcal{PCP}(\log n, \text{poly loglog } n)$

(Arora, Safra, 1992).

The question was finally decided a few weeks later by the following result, also known as the **PCP-Theorem**.

$\mathcal{NP} = \mathcal{PCP}(\log n, 1)$

(Arora, Lund, Motwani, Sudan, Szegedy, 1992)

showing that a verifier that is allowed access to $\mathcal{O}(\log n)$ random bits can decide a language in $\mathcal{NP}$ by accessing only a constant number of bits from the proof via the oracle.

Not only is this result interesting from the standpoint of complexity theory but it has also had a large impact on the theory of approximation algorithms. The PCP-Theorem made it possible to establish lower bounds on some approximation problems which experience had shown to be hard to approximate but for which there were no known reasons why they were hard to approximate. In the first part of this paper a weaker version of the proof that $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$ will be shown, namely that $\mathcal{NP} \supset \mathcal{PCP}(\log n, 1)$ and that $\mathcal{NP} \subset \mathcal{PCP}(n^3, 1)$. In the second part of the paper a result of the application of the PCP-Theorem to approximation algorithms will be shown, that if $\mathcal{P} \neq \mathcal{NP}$ then there exists no PTAS for Max3Sat or for any other $\mathcal{APX}$-complete problem.

Part 1

To show that $\mathcal{NP} \supset \mathcal{PCP}(\log n, 1)$ it is enough to show that there exists some proof π_x such that a non-deterministic Turing Machine M accepts x if and only if there is a $(\log n, 1)$-verifier V that accepts x. Let M then simulate V as follows. Since M has no access to a random strings, M computes all possible strings τ of length $\mathcal{O}(\log n)$ and then makes a separate simulation of V for each τ. There are $2^{\mathcal{O}(\log n)}$ such strings. Since each simulation queries at most $c = \mathcal{O}(1)$ bits from the proof, the maximum number of different bits queried from the proof over all strings τ is $c \cdot 2^{\log n}$. The proof guessed for M is the answers to these $\mathcal{O}(n)$ queries. M accepts input x when every of the $2^{\mathcal{O}\log n}$ simulations of V leads to an accepting computation by V.

To show that $\mathcal{NP} \subset \mathcal{PCP}(n^3, 1)$ it is sufficient to show that 3SAT $\in \mathcal{PCP}(n^3, 1)$.

A 3SAT instance is a Boolean formula in conjunctive normal form over a set of n variables such that each clause contains exactly three literals. The problem is to decide if there exists some truth assignment for the n variables such that the boolean formula is true for this assignment. The usual non-deterministic proof that the boolean formula is satisfiable is a satisfying assignment for the formula which the proof checker reads and verifies, or randomly reads a certain proportion of and verifies with a corresponding amount of certainty. In either case the verifier must look at a number of bits of the proof that is proportional to n. However, in this case the verifier is only allowed to access a constant number of bits, independent of the length of the input. To do this the verifier doesn't access the truth assignment directly, but asks instead that the proof be a function that is based on the truth assignment for the n variables. The verifier asks for a function that assigns to every boolean function over these n variables $x_1, ..., x_n$ its value under this particular assignment. In the following, let this assignment of the n variables be represented by the vector $a \in \mathbb{F}_2^n$ with $a = (a_1, ..., a_n)$. If the verifier has access to enough random bits it can assure itself that it can trust the function by just querying a constant number of random bits of the function. Then when it is satisfied that the function can be trusted, it sends the boolean formula in which it is really interested, hidden behind a certain amount of randomness so that it can be certain that the function can't recognize it, in order to find out what its value is under this assignment.

3SAT $\in \mathcal{PCP}(n^3, 1)$

In the following let a 3SAT formula be given with n variables and m clauses. Let $m = n$ by adding dummy variables if necessary. One wants to associate a vector with the given 3SAT formula in such a way that the vector is the zero vector iff the current truth assignment for the n variables is a satisfying truth

assignment. One can do this by letting each component of the vector be the arithmetization of the negation of a different clause of the boolean formula, then it is clear that all of the components of the vector are zero if and only if all of the clauses of the boolean formula are satisfied.

As a first step, the formula will be arithmetized, so that instead of a boolean formula over the values True and False one has an arithmetic formula over the values 0 and 1. The arithmetization rules are as follows:

$\mathrm{Ar}(x_i) := x_i$

$\mathrm{Ar}(\bar{\alpha}) := 1 - \mathrm{Ar}(\alpha)$

$\mathrm{Ar}(\alpha \wedge \beta) := \mathrm{Ar}(\alpha) \cdot \mathrm{Ar}(\beta)$

$\mathrm{Ar}(\alpha \vee \beta) := 1 - (1 - \mathrm{Ar}(\alpha)) \cdot (1 - \mathrm{Ar}(\beta))$

Then the arithmetized formula is 1 under an assignment iff the Boolean formula is true under the assignment and 0 iff the Boolean formula is false under the corresponding assignment.

example

$$\begin{aligned}(\overline{x}_1 \vee x_2 \vee x_3) &= 1 - (1 - (1 - (1 - x_1))(1 - x_2))(1 - x_3) \\ &= x_1 + x_3 - x_1x_2 - x_1x_3 + x_1x_2x_3\end{aligned}$$

Let $\hat{\mathcal{C}}_i(x)$ be the arithmetization of the negation of the ith clause of the Boolean formula. Let $\mathcal{C}(x) = (\hat{\mathcal{C}}_1, ..., \hat{\mathcal{C}}_n)$ represent the arithmetization of the negation of each of the clauses in the Boolean formula. Then it is clear that $a \in \mathbb{F}_2^n$ is a satisfying assignment for the Boolean formula if and only if $\mathcal{C}(a) \in \mathbb{F}_2^n$ is identical to the zero vector.

The reason why one wants to represent a satisfying assignment for the Boolean function, call it F, by the zero-vector is that eventually the verifier will want to send the arithmetization of F to the oracle to find out if the assignment that the oracle is claiming is a satisfying for F is really that. But the verifier has to add a certain amount of randomness to the Boolean formula to prevent the proof from recognizing it and returning a false positive. Therefore, the verifier first takes the scalar product of $\mathcal{C}(x)$ with a random vector $r \in \mathbb{F}_2^n$ causing certain clauses of the formula to be deleted. It is clear that the verifier can still determine if a is a satisfying assignment for F even after taking the scalar product of it with a

random vector r. Let $r \in \mathbb{F}_2^n$. If $\mathcal{C}(a)$ is the zero-vector and r is a random vector, then $\mathcal{C}(a)^T r$ is always zero. If $\mathcal{C}(a)$ is not the zero-vector then the probability that $\mathcal{C}(a)^T r$ is not zero is $\frac{1}{2}$. The verifier can therefore ask the oracle for the value of $\mathcal{C}(a)^T r_i$ for k different random vectors r_i, $(i = 1, ..., k)$ forcing the probability arbitrarily close to 1 that if $\mathcal{C}(a)$ is not the zero-vector the function will compute a 1 causing the verifier to reject. Before the verifier can carry out this test it must first assure itself that the function really computes what it is supposed to compute.

The construction of the function that the verifier expects is based on the fact that the scalar product of the arithmetization of some Boolean function $(\hat{\mathcal{C}}_1(x), ..., \hat{\mathcal{C}}_n(x))$ with some random vector $r \in \mathbb{F}_2^n$ can be written as

$$\sum_{i=1}^{n} r_i \cdot \hat{\mathcal{C}}_i(x) = c(r) \; + \sum_{(i) \in S_1(r)}^{n} x_i \; + \sum_{(i,j) \in S_2(r)}^{n} x_i x_j \; + \sum_{(i,j,k) \in S_3(r)}^{n} x_i x_j x_k,$$

which is just a regrouping of the summands into four groups - the constants, the single variables, the double variables, and the triple variables. One can confirm that in the arithmization of the clauses there are no summands that contain more that three variables. The sets S_1, S_2, and S_3 tell which of the doubles, triples, and singles are present in the formula. The verifier expects that the proof is three functions, one for each possible S_1, S_2 and S_3, which each assign the value of each set under the truth assignment a. There is no function necessary for the constant because this is independant of a and can be calculated by the verifier. The three functions ideally look like the following linear functions.

$$\begin{aligned}
A : \mathbb{F}_2^n \to \mathbb{F}_2, \qquad & A(x) := \sum_{i=1}^{n} a_i x_i \\
B : \mathbb{F}_2^{n^2} \to \mathbb{F}_2, \qquad & B(x) := \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j x_{i,j} \\
C : \mathbb{F}_2^{n^3} \to \mathbb{F}_2, \qquad & C(x) := \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} a_i a_j a_k x_{i,j,k}
\end{aligned}$$

The functions that the verifier really gets are $\tilde{A}, \tilde{B}$ and $\tilde{C}$. The verifier must first test these three functions to be certain that they are close enough to A, B and C to be used as A, B and C. Integral to the idea that they are close enough is made more concrete by the following definition of δ-closeness.

$\underline{\delta - close}$
Two functions $f, g : F \to G$, where F and G are finite fields, are called $\delta - close$

iff the number of vectors $x \in F$ for which $f(x) = g(x)$ is at least $(1-\delta)|F|$, in other words, if $\text{Prob}_{x \in_R F}[f(x) = g(x)] \geq 1 - \delta$.

The object is to show that $\tilde{A}, \tilde{B}$ and $\tilde{C}$ are δ-close to A, B, and C. To do this one must do the following two things:

1. show that $\tilde{A}, \tilde{B}$ and $\tilde{C}$ are δ-close to linear functions and
2. show that $\tilde{A}, \tilde{B}$ and $\tilde{C}$ are δ-close to linear functions with respect to the same vector a.

To show that the three functions are linear one needs the following lemma.

Lemma Let $\delta < \frac{1}{3}$ be a constant and let $\tilde{g} : \mathbb{F}_2^n \to \mathbb{F}_2$ be a function such that

$$\text{Prob}_{x,y}[\tilde{g}(x) \neq \tilde{g}(x+y)] \leq \tfrac{\delta}{2}.$$

Then there exists $h \in \mathbb{F}_2^n$ so that the functions $g(x) = h^T x$ and $\tilde{g}$ are $\delta - close$. In other words, there exists a linear function g so that g and $\tilde{g}$ are $\delta - close$. One must then show the existence of g.

The property $\text{Prob}_{x,y}[\tilde{g}(x) \neq \tilde{g}(x+y)] \leq \frac{\delta}{2}$ can be tested with a constant number of queries to the proof. One can also construct the following function g by using this property.

Let g be constructed as follows.

$$g(x) := \text{majority}_y\{\tilde{g}(x+y) - \tilde{g}(y)\}$$

The function g so constructed is linear and $\tilde{g}$ is δ-close to g. This provides the following test to see if $\tilde{A}, \tilde{B}$ and $\tilde{C}$ are each δ-close to some linear function.

Linearity Test

Pick $x, x' \in_R \mathbb{F}_2^n$,

verify that $\tilde{A}(x) + \tilde{A}(x') = \tilde{A}(x + x')$.

Pick $y, y' \in_R \mathbb{F}_2^{n^2}$,

verify that $\tilde{B}(y) + \tilde{B}(y') = \tilde{B}(y + y')$.

Pick $z, z' \in_R \mathbb{F}_2^{n^3}$,

verify that $\tilde{C}(z) + \tilde{C}(z') = \tilde{C}(z + z')$.

If $\tilde{A}, \tilde{B}$ and $\tilde{C}$ pass the linearity test then it still remains to show that they are all linear with respect to the same vector a. In other words, it remains to show for the three vectors in the three linear functions $a = (a_i), b = (b_{i,j})$, and $c = (c_{i,j,k})$, that $b_{i,j} = a_i a_j$ (writen $a \circ a$) and $c_{i,j,k} = a_i a_j a_k$ (writen $a \circ a \circ a$).

This suggests the following simple test. Taking $x, x' \in \mathbb{F}_2^n$ and setting $x \circ x' = x_i x'_j = y \in \mathbb{F}_2^{n^2}$ one checks to see if $A(x) \cdot A(x') = B(x \circ x')$, because

$$A(x) \cdot A(x') = \sum_{i=1}^{n} a_i x_i \cdot \sum_{j=1}^{n} a_j x_j = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_i x_i x_j = \sum_{i=1}^{n} \sum_{j=1}^{n} b_{i,j} x_i x_j = B(x \circ x`)$$

is true for all $x, x' \in \mathbb{F}_2^n$ if and only if $b = a \circ a$. An analogous test suggests itself for the function C. However, $x \circ x'$ can be at most 2^{2n} vectors, but $\mathbb{F}_2^{n^2}$ contains 2^{n^2} vectors so $x \circ x'$ is not a random element of $\mathbb{F}_2^{2^n}$ and one can not use the fact that the functions $\tilde{A}, \tilde{B}$ and $\tilde{C}$ are δ-close to linear functions. Therefore there are the following self-correcting-codes.

Self-Correcting Functions

$SC - \tilde{A}(x)$: Pick $r \in_R \mathbb{F}_2^n$, return $\tilde{A}(r + x) - \tilde{A}(r)$.

$SC - \tilde{B}(y)$: Pick $r \in_R \mathbb{F}_2^{n^2}$, return $\tilde{B}(r + y) - \tilde{B}(r)$.

$SC - \tilde{C}(z)$: Pick $r \in_R \mathbb{F}_2^{n^3}$, return $\tilde{C}(r + z) - \tilde{C}(r)$.

The vectors returned by the self-correcting functions are random elements of $\mathbb{F}_2^n$, $\mathbb{F}_2^{n^2}$, and $\mathbb{F}_2^{n^3}$ respectively and so one can use the δ-closeness of $\tilde{A}, \tilde{B}$ and $\tilde{C}$ to linear functions to design the following consistency test.

Consistency Test

Pick $x, x' \in_R \mathbb{F}_2^n$,
Verify that $SC - \tilde{A}(x) \cdot SC - \tilde{A}(x') = SC - \tilde{B}(x \circ x')$.
Pick $x \in_R \mathbb{F}_2^n$, $y \in_R \mathbb{F}_2^{n^2}$
Verify that $SC - \tilde{A}(x) \cdot SC - \tilde{B}(y) = SC - \tilde{C}(x \circ y)$.

If it is not the case that $b = a \circ a$ and $c = a \circ a \circ a$, then the consistency test will fail with a constant probability. Repeating this test a constant number of times

can push the probablity arbitrarily close to 1.

After passing the linearity test and the consistency test the verifier only needs to check to see what its Boolean formula is under the assignment a with the following satisfiability test. Taking the scalar product of a vector $r \in_R \mathbb{F}_2^n$ with its Boolean formula F in vector form as described above, it computes the sets S_1, S_2 and S_3 and verifies that $\tilde{A}(S_1) + \tilde{B}(S_2) + \tilde{C}(S_3) + c = 0$.

If a is not a satisfying assignment for F then the satisfiability test will fail with a constant probability, so repeating the test a constant number of times can push the probabilty of failure arbitrarily close to 1 causing the verifier to reject. Therefore, repeating the linearity test, consistency test and satisfiabilty test a constant number of times forms a $(n^3, 1)$-restricted verifier for $3SAT$ and it follows that $\mathcal{NP} \subseteq \mathcal{PCP}(n^3, 1)$.

Part 2

For Max3Sat a Boolean formula in conjunctive normal form is given and the problem is to find an assignment for the n variables that fulfills the most clauses. In the following it will be shown that there is no PTAS for Max3Sat if $\mathcal{P} \neq \mathcal{NP}$. As a reminder, if a problem has a PTAS then for every ϵ there exists a polynomial ϵ-approximation algorithm for the problem.

Let L be a language in $\mathcal{NP}$ and V the verifier for L. Then one constructs a 3SAT instance from input x as follows. For each τ one simulates all possible c queries that the verifier makes to the proof (all possible proofs) and one notes for which proofs the verifier accepts input x. Then for each τ one constructs a Boolean formula F_τ such that if the verifier accepts for at least one proof then the formula F_τ is true. Since the verifier makes c queries there are 2^c clauses in the formula which each contain c literals. By splitting the clauses at most c times one obtains a 3SAT formula with at most $c \cdot 2^c$ clauses. One repeats this for every τ and makes S_x the conjunction of all the F_τ. If $x \in L$ then there exists some proof such that the verifier V accepts for all random strings τ and S_x is satisfiable. If $x \notin L$ then for every proof V must reject for at least half of all τ. Therefore for $\frac{1}{2}$ of all F_τ at least $\frac{1}{c \cdot 2^c}$ clauses are not satisfied. This means that $\frac{1}{2 \cdot c \cdot 2^c}$ of the clauses in S_x aren't satisfied. If Max3Sat had a PTAS then one could set $\epsilon < \frac{1}{2 \cdot c \cdot 2^c}$ and use the *epsilon*-approximation algorithm to decide between the two cases.

References

[1] Stefan Hougardy. *Proof Checking and Non-Approximability,* Lecture Notes in Computer Science Vol. 1367, Springer, 1998.

[2] Stefan Hougardy. *Skript zur Vorlessung Graphen und Algorithm II,* Humboldt Universität zu Berlin, 2001.

[3] S. Hougardy, H. Prömel, A. Steger. *Probabilistically Checkable Proofs and their Consequences for Approximation Algorithms,* Discrete Mathematics, 136, 1994.

[4] D. S. Johnson. *The Tale of the Second Prover,* J. of Algorithms 13, pp. 502-524, 1992.

[5] H. Prömel, A. Steger. *The Steiner Tree Problem,* Viewweg, 2002.