Introduction to the PCP Theorem

Yijia Chen

Definition 1. A verifier V is a (probabilistic) polynomial time Turing machine with access to an input $x \in \Sigma^*$ and a string $\tau \in \Sigma^*$ of random (binary) bits. Furthermore the verifier has access to a given proof $\pi \in \Sigma^*$ via an oracle, which takes as input the position of the proof the verifier wants to query and outputs the corresponding bit of the proof π.

Depending on the input x, the random string τ and the proof π the verifier V will either accept or reject the input x.

We require that the verifier is non-adaptive, i.e., it first reads the input x and the random bits τ, and then decides which positions in the proof τ it wants to query. Especially this means that the positions it queries do not depend on the answers the verifier got from previous queries.

We will denote the result of V's computation on x, τ and π as $V(x,\tau,\pi)$. As usual 1 means accepting, 0 for rejecting.

Remark 2. (1) V is polynomial time in $|x|$.

(2) The query positions depend on and only on x and τ (non-adaptive), so we might view the query positions as a function

$$p_V : \Sigma^* \times \Sigma^* \to \mathbb{N}^*,$$

such that $p_V(x,\tau)$ is computable in time polynomial in $|x|$. Sometimes to simplify presentation, we may assume p_V is given explicitly with the verifier V.

(3) The run of $V(x,\tau,\pi)$ that does not involve oracle might also depend on the random string τ. This is important for the proof of Theorem 8.

(4) In general we can divide the computation of $V(x,\tau,\pi)$ into 3 parts:

(a) Compute from x and τ a sequence of positions $\bar{p} = p_1 p_2 \dots p_\ell = p_V(x,\tau)$.

(b) Read the letters on the positions $\bar{p}$ of π to form a string

$$\pi \restriction \bar{p} := \pi(p_1)\pi(p_2)\dots\pi(p_\ell),$$

where $\pi(p)$ denotes the letter in the p-th position of π.

(c) Compute an answer from x, τ, and $\pi \restriction \bar{p}$.

Definition 3. Let $r, q : \mathbb{N} \to \mathbb{N}$ be two functions[1]. An $(r(n), q(n))$-restricted verifier is a verifier that for inputs of length n uses at most $O(r(n))$ random bits and queries at most $O(q(n))$ bits from the proof.

Remark 4. (1) In other words, if V is $(r(n), q(n))$-restricted, then for any input $x \in \Sigma^*$, we have the corresponding τ with $|\tau| = O(r(n))$ and $|p_V(x,\tau)| = O(q(n))$.

(2) Note there is no restriction on the length of the proof π.

[1] Throughout the notes, we only consider monotone functions.

Definition 5. Let $r, q : \mathbb{N} \to \mathbb{N}$ be two functions. The class $\underline{\mathrm{PCP}(r(n), q(n))}$ consists of all languages L where there exists an $(r(n), q(n))$-restricted verifier V such that for all x

$$\begin{aligned} x \in L &\iff \exists \pi \ \Pr_{\tau}[V(x, \tau, \pi) = 1] = 1, \\ x \notin L &\iff \forall \pi \ \Pr_{\tau}[V(x, \tau, \pi) = 1] < \frac{1}{2}. \end{aligned}$$

Here in $\Pr_\tau[\ldots]$ the probability is taken over all random strings τ of length $O(r(n))$.

Remark 6. The number $1/2$ in Definition 5 is arbitrary, as we can always decrease the probability of falsely accepting by repeating the verifier constantly many times. This would not change the class, since in Definition 3, we can choose any hidden constant in $O(r(n))$ and $O(q(n))$.

Definition 7. Let R, Q be two classes of functions over $\mathbb{N}$.

$$\mathrm{PCP}(R, Q) := \bigcup_{r \in R, q \in Q} \mathrm{PCP}(r(n), q(n)).$$

Let poly denote the class of all polynomials over $\mathbb{N}$, i.e. $\mathbb{N}[x]$.

Theorem 8.

$$\begin{aligned} \mathrm{PCP}(\mathrm{poly}, 0) &= \mathrm{coRP} \\ \mathrm{PCP}(0, \mathrm{poly}) &= \mathrm{NP} \end{aligned}$$

Proof: Recall the complement $\bar{L}$ of a language L is in RP, if there is a polynomial time probabilistic Turing machine M such that

$$\begin{aligned} x \notin L &\iff x \in \bar{L} \iff \Pr_{\tau}[M(x, \tau) = 1] \geq \tfrac{1}{2}, \\ x \in L &\iff x \notin \bar{L} \iff \Pr_{\tau}[M(x, \tau) = 1] = 0, \end{aligned}$$

where τ is a random string of length polynomial in $|x|$, i.e.

$$\begin{aligned} x \notin L &\iff \Pr_{\tau}[M(x, \tau) = 0] \leq \frac{1}{2}, \\ x \in L &\iff \Pr_{\tau}[M(x, \tau) = 0] = 1. \end{aligned}$$

Thus the machine $\bar{M}$ which is the same as M except exchanging accepting and rejecting states witnesses $L \in \mathrm{PCP}(\mathrm{poly}, 0)$. Therefore $\mathrm{coRP} \subseteq \mathrm{PCP}(\mathrm{poly}, 0)$. The other direction $\mathrm{PCP}(\mathrm{poly}, 0) \subseteq \mathrm{coRP}$ can be proved similarly.

It is easy to show $\mathrm{NP} \subseteq \mathrm{PCP}(0, \mathrm{poly})$. Now given a language $L \in \mathrm{PCP}(0, \mathrm{poly})$, since no random bit is needed, there is a polynomial time deterministic machine V such that

$$\begin{aligned} x \in L &\iff \exists \pi \ V(x, \pi) = 1, &(1) \\ x \notin L &\iff \forall \pi \ V(x, \pi) = 0. &(2) \end{aligned}$$

Since $L \in \mathrm{PCP}(0, \mathrm{poly})$, we have $|p_V(x)| = q(|x|)$ for some polynomial q. Now let M be the following nondeterministic Turing machine: Let $x \in \Sigma^*$ be a given instance.

$\mathbb{M}(x)$

1. $\ell \leftarrow q(|x|)$.
2. Guess a string $\bar{a} = a_1 \dots a_\ell \in \Sigma^\ell$.
3. $\bar{p} = p_1 \dots p_\ell \leftarrow p_V(x)$,
4. Simulate $V(x, \pi)$ for an ''imaginary'' π by replacing each π_{p_i} by a_i for $1 \leq i \leq \ell$.

It is easy to see that M runs in polynomial time. If $x \in L$, then we have a π witnessing the membership according to (1). Thus M can guess $\bar{a} = \pi \restriction p_V(x)$, and accept x. On the other hand, if $x \notin L$, we see by (2) there is no way for M to guess an $\bar{a}$ to accept x. Thus M decides L.

□

Now the PCP theorem is the following.

Theorem 9. $\mathrm{NP} = \mathrm{PCP}(\log n, 1)$.

1. Max-3SAT and Gap-3SAT

Let $d \in \mathbb{N}$.

Max-dSAT	
Input:	An $\alpha \in d$CNF.
Solutions:	An assignment $\mathcal{V}$ for α.
Cost:	the number of clauses in α that $\mathcal{V}$ satisfies.
Goal:	max.

Definition 10. For every dCNF-formula α and an assignment $\mathcal{V}$ for α, we let

$$\mathrm{sat}(\alpha, \mathcal{V}) := \text{the number of clauses in } \alpha \text{ that } \mathcal{V} \text{ satisfies}$$

and

$$\mathrm{optsat}(\alpha) := \max_{\mathcal{V}} \mathrm{sat}(\alpha, \mathcal{V}).$$

Definition 11. Let $\mathbb{A}$ be an algorithm that on every dCNF-formula α always output an assignment for α which we denote by $\mathcal{V}_\alpha$. The performance ratio of $\mathbb{A}$ is

$$\max_{\alpha \in d\mathrm{CNF}} \frac{\mathrm{optsat}(\alpha)}{\mathrm{sat}(\alpha, \mathcal{V}_\alpha)}.$$

Theorem 12. For some constants $d \in \mathbb{N}$ and $r > 1$, there is no polynomial time approximation algorithm for Max-dSAT with performance ratio r, unless P = NP.

Proof: Let $L \subseteq \Sigma^*$ be an NP-complete language. By Theorem 9, there is a $(\log n, 1)$-restricted verifier V accepting L. More precisely for each instance $x \in \Sigma^*$, the verifier V uses $c \cdot \log |x|$ many random bits and makes d many oracle queries for two constants $c, d \in \mathbb{N}$.

$$\text{(R1)} \quad x \in L \iff \exists \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] = 1,$$

$$\text{(R2)} \quad x \notin L \iff \forall \pi \Pr_{\tau \in \{0,1\}^{c \cdot \log |x|}} [V(x, \tau, \pi) = 1] < \frac{1}{2}.$$

We define the following algorithm for later use.

$\mathbb{A}(x, \tau, a_1, a_2, \ldots, a_d)$

1. $\bar{p} = p_1 \ldots p_d \leftarrow p_V(x, \tau)$,
2. Simulate $V(x, \pi)$ for an ''imaginary'' π by replacing each π_{p_i} by a_i for $1 \leq i \leq d$.

where $\tau \in \{0,1\}^{c \cdot \log |x|}$, and $a_1, a_2, \ldots, a_d \in \{0,1\}$. It should be clear that $\mathbb{A}$ is polynomial time in $|x|$.

We reduce L to Max-dSAT. Given an instance $x \in \Sigma^*$, we will construct a dCNF formula α_x satisfying

$$x \in L \;\Rightarrow\; \alpha_x \text{ is satisfiable,} \tag{3}$$

$$x \notin L \;\Rightarrow\; \text{for any assignment } \mathcal{V}, \text{ at least } \underline{2^{-d-1}\text{-fraction}} \text{ of clauses are not satisfied.} \tag{4}$$

First let

$$P_x := \left\{ p \;\middle|\; \text{for some } \tau \in \{0,1\}^{c \cdot \log |x|} \; p \text{ appears in the sequence } p_V(x, \tau) \right\},$$

i.e. P_x is the set of positions that V makes oracle queries. It should be clear P_x is computable in time polynomial in $|x|$. In addition

$$|P_x| \leq d \cdot |x|^c.$$

For each $p \in P_x$, we introduce a propositional variable X_p. Now any proof π can be viewed as an <u>assignment</u> $\mathcal{V}_\pi$ with

$$\mathcal{V}_\pi(X_p) := \pi(p). \tag{5}$$

Then for every $\tau \in \{0,1\}^{c \cdot \log |x|}$, we define a dCNF formula $\alpha_{x,\tau}$ as follows: for every $a_1, \ldots, a_d \in \{0,1\}$, and $\bar{p} = p_1 \ldots p_d = p_V(x, \tau)$, if $\mathbb{A}(x, \tau, a_1, \ldots, a_d)$ <u>rejects</u>, then $\alpha_{x,\tau}$ contains the clause

$$T_1 \vee T_2 \vee \cdots \vee T_d$$

where

$$T_i := \begin{cases} X_{p_i} & \text{if } a_i = 0, \\ \neg X_{p_i} & \text{if } a_i = 1. \end{cases}$$

It follows that $\alpha_{x,\tau}$ has at most 2^d many clauses. More importantly: Let π be a proof and $\mathcal{V}_\pi$ the assignment defined as in (5)

(F1) if $V(x, \tau, \pi)$ accepts, then $\mathcal{V}_\pi$ satisfies $\alpha_{x,\tau}$,

(F2) if $V(x, \tau, \pi)$ rejects, then $\mathcal{V}_\pi$ makes at least <u>one</u> of the clauses in $\alpha_{x,\tau}$ <u>false</u>.

Now let

$$\alpha_x := \bigwedge_{\tau \in \{0,1\}^{c \cdot \log |x|}} \alpha_{x,\tau}$$

Clearly α_x has at most $|x|^c \cdot 2^d$ many clauses. We have to show (3) and (4):

Let $x \in L$. By (R1) we have a proof π such that $V(x, \tau, \pi)$ always accepts for any $\tau \in \{0,1\}^{c \cdot \log |x|}$. Thus $\mathcal{V}_\pi$ satisfies all $\alpha_{x,\tau}$ by (F1), and hence α_x, i.e., (3) holds.

Let $x \notin L$. For any assignment $\mathcal{V}$ of α_x, it is easy to see we can view it as some $\mathcal{V}_\pi$ for an appropriate proof π. The by (R2) we have for more

than half of all possible $\tau \in \{0,1\}^{c\cdot\log|x|}$, $V(x,\tau,\pi)$ rejects. For any τ with $V(x,\tau,\pi)$ rejecting, (F2) implies that at least one clause in $\alpha_{x,\tau}$ is false under $\mathcal{V}_\pi = \mathcal{V}$. Recall $\alpha_{x,\tau}$ contains at most 2^d many clauses. Thus in total we have more than

$$2^{-1}\cdot 2^{-d} = 2^{-d-1}$$

fraction of clauses in α_x are not satisfied under $\mathcal{V}$. (4) is proved.

Now let

$$r := \frac{1}{1-2^{-d-1}}$$

We claim Max-dSAT cannot have a polynomial time approximation algorithm with performance ratio r. Assume otherwise, let $\mathbb{B}$ be such an algorithm. We would use it to decide L: Given an instance $x \in \Sigma^*$. We compute the dCNF-formula α_x as described above. If $x \in L$, by (3), α_x is satisfiable. Assume α_x has m many clauses. Thus $\mathbb{B}$ would find an assignment that satisfies at least

$$\frac{m}{r} = m\cdot(1-2^{-d-1})$$

many clauses of α_x. On the other hand if $x \notin L$, by (4), less than

$$m\cdot(1-2^{-d-1})$$

many clauses of α_x can be satisfied by any assignment. In particular the assignment that $\mathbb{B}$ outputs on x cannot satisfy $m\cdot(1-2^{-d-1})$ many clauses. In conclusion in polynomial time we can decide whether $x \in L$ by looking at how many clauses the assignment output by $\mathbb{B}$ on x can satisfy. □

Theorem 12 can be reformulated as the hardness of the some gap problems. Again fix some $d \in \mathbb{N}$. Let $\alpha \in d$CNF, we define

$$\text{UNSAT}(\alpha) := \text{the smallest fraction of unsatisfied clauses of } \alpha \text{ under any assignment.}$$

Clearly α is satisfiable if and only if $\text{UNSAT}(\alpha) = 0$; and α is not satisfiable if and only if $\text{UNSAT}(\alpha) \geq 1/m$ where m is the number of clauses in α.

Let $\varepsilon > 0$. We define the following promise problem.

Gap-dSAT$_\varepsilon$

Instance: A propositional formula $\alpha \in d$CNF, such that either $\text{UNSAT}(\alpha) = 0$ or $\text{UNSAT}(\alpha) \geq \varepsilon$.

Problem: Decide whether α is satisfiable.

Theorem 13. There exist $d \in \mathbb{N}$ and $\varepsilon > 0$ such that Gap-dSAT$_\varepsilon$ is NP-hard.

Proof: Recall in the proof of Theorem 12, for some NP-hard language L, appropriate $d \in \mathbb{N}$ and $\varepsilon > 0$, we implicitly define an algorithm $\mathbb{A}$, such that for each instance $x \in \Sigma^*$, $\mathbb{A}$ produces a dCNF-formula α_x satisfying:

$$\begin{aligned} x \in L &\Rightarrow \alpha_x \text{ is satisfiable, i.e., } \text{UNSAT}(\alpha_x) = 0, \\ x \notin L &\Rightarrow \text{for any assignment } \mathcal{V}, \text{ at least } \varepsilon\text{-fraction of} \\ &\qquad \text{clauses are not satisfied, i.e., } \text{UNSAT}(\alpha_x) \geq \varepsilon. \end{aligned}$$

It is easy to see $\mathbb{A}$ gives a reduction from L to Gap-dSAT$_\varepsilon$. □

Theorem 14. There exists some $\varepsilon > 1$, such that Gap-3SAT$_\varepsilon$ is NP-hard.

Theorem 14 follows easily from Theorem 13 and the following lemma.

Lemma 15. Let $d > 3$ and $r \geq 1$, there is a polynomial time algorithm $\mathbb{A}$ such that, given a propositional formula α in dCNF, $\mathbb{A}$ produce a formula α^* in 3CNF such that

(i) α is satisfiable $\Rightarrow$ α^* is satisfiable.

(ii) For any assignment, at least ε-fraction of clauses in α is not satisfied $\Longrightarrow$ for any assignment, at least $\frac{\varepsilon}{d-2}$-fraction of clauses in α^* is not satisfied.

Proof: Let α be a propositional formula in dCNF, i.e., for an index set I,

$$\alpha = \bigwedge_{i \in I} \beta_i$$

where each

$$\beta_i = \lambda_{i,1} \vee \lambda_{i,2} \vee \ldots \vee \lambda_{i,t_i}$$

for some literals $\lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,t_i}$ and $t_i \leq d$. Now we use the standard reduction from dSAT to 3SAT. For every $i \in I$, with $t_i > 3$, let $Y_{i,1}, \ldots, Y_{i,t_i-3}$ be new variables. And let

$$\gamma_i := (\lambda_{i,1} \vee \lambda_{i,2} \vee Y_{i,1}) \wedge (\neg Y_{i,1} \vee \lambda_{i,3} \vee Y_{i,2}) \wedge \cdots \wedge (\neg Y_{i,t_i-3} \vee \lambda_{i,m-1} \vee \lambda_{i,t_i}).$$

Otherwise let $\gamma_i := \beta_i$. Then we take

$$\alpha^* := \bigwedge_{i \in I} \gamma_i$$

as the required 3CNF formula.

(i) is easy. We show (ii). Let $\mathcal{V}^*$ be an assignment for α^*. And let $\mathcal{V}$ be the restriction of $\mathcal{V}^*$ to the variables of α. Assume α contains m clauses. By assumption, under $\mathcal{V}$, at least

$$\lceil \varepsilon \cdot m \rceil$$

many clauses are <u>not</u> satisfied. Let β_i be an unsatisfied clause, it follows that at least one clause in γ_i is not satisfied under $\mathcal{V}^*$. Thus we have at least $\lceil \varepsilon \cdot m \rceil$ unsatisfied clauses in α^*. Note α^* contains at most

$$(d-2) \cdot m$$

many clauses. Thus the fraction of clauses that are not satisfied under $\mathcal{V}^*$ is at least

$$\frac{\lceil \varepsilon \cdot m \rceil}{(d-2) \cdot m} \leq \frac{\varepsilon}{d-2}.$$

□

Theorem 16. There is a constant $r > 1$ such that there is no polynomial time approximation algorithm for Max-3SAT with performance ratio r, unless P = NP.

Proof: By Theorem 14, there is some $\varepsilon >$ such that Gap-3SAT$_\varepsilon$ is NP-hard. Without loss of generality, we assume $\varepsilon < 1$. Let

$$r := \frac{1}{1-\varepsilon}.$$

Assume there is an algorithm $\mathbb{A}$ that approximates Max-3SAT with performance ratio r. We show Gap-3SAT$_\varepsilon$ can be decided in polynomial time by $\mathbb{A}$: Let $\alpha \in 3\text{CNF}$ with m many clauses.

(i) If $\text{UNSAT}(\alpha) = 0$, then $\mathbb{A}$ outputs an assignment that satisfies at least

$$\left\lceil \frac{m}{r} \right\rceil \geq (1-\varepsilon) \cdot m$$

many clauses.

(ii) If $\text{UNSAT}(\alpha) \geq \varepsilon$, then any assignment (including the one $\mathbb{A}$ outputs) can satisfy less than

$$m - \lceil \varepsilon \cdot m \rceil \leq m - \varepsilon \cdot m$$

many clauses.

□

While Theorem 14 follows from the PCP Theorem, interestingly the converse is also true, i.e.,

Theorem 17. If there exists some $1 \geq \varepsilon > 0$, such that Gap-3SAT$_\varepsilon$ is NP-hard, then $\text{NP} = \text{PCP}(\log n, 1)$.

Proof: Let $L \subseteq \Sigma^*$ be a language in NP. By assumption, we have a polynomial time algorithm $\mathbb{A}$ such that for any $x \in \Sigma^*$, $\mathbb{A}$ computes a 3CNF formula α_x with

$$\text{UNSAT}(\alpha_x) = 0 \quad \text{or} \quad \text{UNSAT}(\alpha_x) \geq \varepsilon.$$

Without loss of generality, we assume α_x contains at most $|x|^c$ many clauses, for a constant $c \in \mathbb{N}$.

Let d be a natural number satisfying

$$\left(1 - \frac{\varepsilon}{2}\right)^d \leq \frac{1}{2}^2. \tag{6}$$

We define the following verifier V with $|\tau| = c \cdot d \cdot \log |x|$:

$V(x, \tau, \pi)$

1. We view τ as a sequence of d many $c \cdot \log |x|$-bit binaries, which correspond to numbers $t_1, \ldots, t_d$, each between 0 and $|x|^c - 1$.
2. Simulate $\mathbb{A}$ on x to compute α_x.
3. $m \leftarrow$ the number of clauses in α_x.
4. For all $1 \leq i \leq d$, $p_i \leftarrow (t_i \mod m) + 1$.
5. Check whether π, as an assignment, satisfies the p_1-, ..., p_d-th clauses. If so, accept. Otherwise reject.

[2] It suffices to take $d := \lceil 2/\varepsilon \rceil$.

Clearly V is a $(\log n, 1)$-restricted verifier. Now we have to check V correctly decides L.

- If $x \in L$, then $\mathrm{UNSAT}(\alpha_x) = 0$. In other words, we have an assignment π satisfy α_x. Thus no matter what τ we have, $V(x, \tau, \pi)$ always accepts.
- If $x \notin L$, then $\mathrm{UNSAT}(\alpha_x) \geq \varepsilon$, i.e., no matter what assignment π we have, at least ε-fraction of clauses are not satisfied. We pick a number t with $0 \leq t < |x|^c$ in random, and let

  $$p := (t \mod m) + 1$$

  where m is the number of clauses in α_x. And let

  $$s := \left\lfloor \frac{|x|^c}{m} \right\rfloor.$$

  $s \geq 1$ by our assumption, and

  $$s \cdot m \leq |x|^c \leq (s+1) \cdot m.$$

  It follows that when t ranges from 0 to $|x|^c - 1$, for at least

  $$\lceil \varepsilon \cdot m \rceil \cdot s$$

  many corresponding ps, the p-th clause is *not* satisfied. Note

  $$\frac{\lceil \varepsilon \cdot m \rceil \cdot s}{|x|^c} \geq \frac{s \cdot m}{(s+1) \cdot m} \cdot \varepsilon \geq \frac{\varepsilon}{2} \qquad \text{by } s \geq 1.$$

  Thus the probability that the p-th clause is satisfied is less than

  $$1 - \frac{\varepsilon}{2}.$$

  It implies that the probability that V accepts is less than

  $$\left(1 - \frac{\varepsilon}{2}\right)^d,$$

  hence less than $1/2$ by (6).

□