CS 505 Spring 2025 — Homework 5 (Sample Solutions)

Alexander R. Block

Due Date: May 06, 2025, no later than 2:00pm Central Time.

1 PCP Theorem (25 Points)

1.1 Part 1 (5 Points)

Prove that for every two functions $r, q\colon \mathbb{N} \to \mathbb{N}$ and constants $s < 1$, if $\mathbf{PCP}_s(r, q)$ is identical to the class $\mathbf{PCP}(r, q)$ except with the soundness error replaced with s instead of $1/2$, then $\mathbf{PCP}_s(r, q) = \mathbf{PCP}(r, q)$.

Proof of Problem 1 Part 1. Note that for $s \leq 1/2$, clearly we have $\mathbf{PCP}_s(r, q) \subseteq \mathbf{PCP}(r, q)$. For $s \in (1/2, 1)$, we can show that $\mathbf{PCP}_s(r, q) \subseteq \mathbf{PCP}(r, q)$ by having the verifier do a constant number of repetitions in parallel and by taking the majority of the results; the constant depends on s and is chosen so that the resulting probability is upper bounded by $1/2$. Note also that in both cases, the PCPs have perfect completeness, and that the constant needed to amplify s down to less than $1/2$ only increases r and q by constants, so the classes remain the same. Similarly, for $\mathbf{PCP}(r, q) \subseteq \mathbf{PCP}_s(r, q)$, if $s \in (1/2, 1)$, we are done and containment already holds; for $s < 1/2$, we again use a Chernoff bound to amplify $1/2$ down to less than s. This amplification incurs only a constant overhead in both r and q. □

1.2 Part 2 (10 Points)

Prove that $\mathbf{PCP}(0, \log(n)) = \mathbf{P}$ and $\mathbf{PCP}(0, \text{poly}(n)) = \mathbf{NP}$.

Proof of Problem 1 Part 2. First, we prove that $\mathbf{PCP}(0, \log(n)) = \mathbf{P}$. To begin, recall the definition of $\mathbf{PCP}(r, q)$. A language L is in the class $\mathbf{PCP}(r, q)$ if there exists a probabilistic polynomial-time verifier algorithm V such that for any x:

- For any $x \in \{0,1\}^*$ and any proof $\pi \in \{0,1\}^*$, $V^\pi(x)$ uses $O(r(|x|))$ random bits and reads $O(q(|x|))$ bits of π.
- If $x \in L$, there exists π such that $\Pr[V^\pi(x) = 1] = 1$.
- If $x \notin L$, then for all π^*, it holds that $\Pr[V^{\pi^*}(x) = 1] \leq 1/2$.

First, assume that $L \in \mathbf{P}$. We claim that $L \in \mathbf{PCP}(0, 0)$. This is because a verifier for this PCP on input x simply runs the $\mathbf{P}$ algorithm for deciding if $x \in L$. This uses 0 random bits and queries 0 locations of (any) given proof π; moreover, 0 is $O(\log(|x|))$, so we are done since $\mathbf{PCP}(0, 0) \subseteq \mathbf{PCP}(0, \log(n))$. Now, consider $L \in \mathbf{PCP}(0, \log(n))$. This means that for any x of length n, the verifier of the PCP proof system samples 0 random bits and reads $O(\log(n))$ positions of the proof π. Moreover, it must read the same $O(\log(n))$ positions during any execution (depending on $|x|$) since the verifier samples no randomness. So if $x \in L$, there exists a valid proof π such that this deterministic verifier accepts. Moreover, if $x \notin L$, it is required by the definition of a PCP that the probability of accepting any proof is at most $1/2$. Since the verifier is deterministic, it must hold that the verifier rejects with probability 1 if $x \notin L$. Given this, we can construct a DTM which decides L. The algorithm iterates over all possible $O(\log(n))$ size proofs and runs the PCP verifier on each of these proofs. If there is at least one accepting verifier, output accept; otherwise output reject. Since $2^{O(\log(n))} = \text{poly}(n)$, we have that this new DTM runs in polynomial time.

Now, we show that $\mathbf{PCP}(0, \text{poly}(n)) = \mathbf{NP}$. The easy direction is $\mathbf{NP} \subseteq \mathbf{PCP}(0, \text{poly}(n))$. For any $L \in \mathbf{NP}$, using the verifier definition of $\mathbf{NP}$, if $x \in L$ there exists w such that $|w| = \text{poly}(|x|)$ and a deterministic verifier M_L such that $M_L(x, w) = 1$. Setting $V = M_L$ and $\pi = w$ gives us a valid PCP for L, and thus $L \in \mathbf{PCP}(0, \text{poly}(n))$ (it just reads the whole proof π). Note that the other direction $\mathbf{PCP}(0, \text{poly}(n)) \subseteq \mathbf{NP}$ is also easy. Take $L \in \mathbf{PCP}(0, \text{poly}(n))$. This means that L has a PCP verifier V such that V is deterministic, polynomial-time, and if $x \in L$ there exists π such that V reads $\text{poly}(n)$ bits of π and outputs accept with probability 1. Moreover, since V is deterministic, if $x \notin L$, then V rejects all proofs π^* with probability 1. Finally, V reads the same bits of any π (which depends only on $|x|$), so there is at least one $\text{poly}(n)$ length string that causes V to accept. This is exactly a verifier for $\mathbf{NP}$, so $L \in \mathbf{NP}$. $\square$

1.3 Part 3 (10 Points)

Let ϕ be any $3CNF$ on n variables and m clauses such that each clause of ϕ has exactly 3 distinct variables in each clause (i.e., you cannot repeat variables in each clause). Give a probabilistic polynomial-time algorithm which, on input any such ϕ above, outputs some assignment of ϕ which satisfies at least 7/8 of the clauses.

Hint: Show that the expected number of satisfied clauses from a random assignment is at least $(7/8) \cdot m$*, then use Markov's inequality to show that the probability of satisfying at least* $(7/8 - 1/(2m)) \cdot m$ *clauses is at least* $1/\text{poly}(m)$*.*

Proof of Problem 1 Part 3. First, we show that for such a ϕ as above with m clauses and n variables, a random assignment is expected to satisfy $7/8 \cdot m$ clauses. Let C_i be a random variable that is 1 if and only if the clause ϕ_i is satisfied; otherwise $C_i = 0$. Let $C = \sum_{i \in [m]} C_i$. Then, by linearity of expectation, we have

$$\mathbb{E}[C] = \sum_{i=1}^{m} \mathbb{E}[C_i].$$

Here, the expectation is taken over a uniformly chosen assignment $x \xleftarrow{\$} \{0,1\}^n$. Now, for each i, we have

$$\begin{aligned} \mathbb{E}[C_i] &= 0 \cdot \Pr[C_i = 0] + 1 \cdot \Pr[C_i = 1] \\ &= \Pr[C_i = 1]. \end{aligned}$$

Suppose that $\phi_i = (\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$. Then, $C_i = 1$ if and only if a random assignment satisfies at least one of ℓ_{i_j} for $j \in [3]$. This gives us

$$\begin{aligned} \Pr[C_i = 1] &= \sum_{j=1}^{3} \binom{3}{j} 2^{-j} \cdot 2^{-(3-j)} \\ &= 2^{-3} \cdot \sum_{j=1}^{3} \binom{3}{j} \\ &= \frac{1}{8} \cdot (2^3 - 1) = \frac{7}{8}. \end{aligned}$$

All together, this yields

$$\mathbb{E}[C] = \sum_{i=1}^{m} \mathbb{E}[C_i] = \sum_{i=1}^{m} \Pr[C_i = 1] = \frac{7}{8} \cdot m.$$

This hints at the following randomized algorithm for finding a satisfying assignment. The algorithm simply samples $x \xleftarrow{\$} \{0,1\}^n$ and counts the number of satisfied clauses of $\phi(x)$. If the number is at least $\frac{7}{8} \cdot m$, then output x; otherwise, try again.

We now argue that the expected number of iterations of this algorithm is $\text{poly}(m)$. First, let $D = \overline{C}$; i.e., D is the random variable denoting the number of clauses that are *unsatisfied* under a random assignment

x. Notice that since $\mathbb{E}[C] = (7/8)m$, it holds that $\mathbb{E}[D] = m - \mathbb{E}[C] = m/8$. Now, consider $\Pr[C \geq (7/8 - 1/(2m)) \cdot m]$. We have

$$\begin{aligned}\Pr[C \geq (7/8 - 1/(2m)) \cdot m] &= 1 - \Pr[C < (7/8 - 1/(2m)) \cdot m] \\ &= 1 - \Pr[D > m - (7/8 - 1/(2m)) \cdot m] \\ &= 1 - \Pr[D > (1/8 + 1/(2m)) \cdot m].\end{aligned}$$

By Markov's inequality, we have

$$\begin{aligned}\Pr[D > (1/8 + 1/(2m)) \cdot m] &\leq \frac{\mathbb{E}[D]}{(1/8 + (1/(2m))m} = \frac{m/8}{(1/8 + (1/(2m))m} \\ &= \frac{1}{1 + 4/m} = 1 - \frac{4}{m}.\end{aligned}$$

This implies

$$\begin{aligned}\Pr[C \geq (7/8 - 1/(2m)) \cdot m] &= 1 - \Pr[D > (1/8 + 1/(2m)) \cdot m] \\ &\geq 1 - \left(1 - \frac{4}{m}\right) \\ &= \frac{4}{m}.\end{aligned}$$

This tells us that with probability at least $4/m$, a random assignment of variables will satisfy at least $7m/8 - 1/2$ clauses (i.e., basically at least $7m/8$ clauses). Without loss of generality, let $p \geq 4/m$ be this probability Turning back to our algorithm, we use this to argue that the expected number of executions of the algorithm is at most $\text{poly}(m)$. Let X be the number of executions the algorithm takes before outputting an assignment which satisfies at least $7m/8$ clauses. Then, for some number T, we have

$$\Pr[X \leq T] = 1 - \Pr[X \geq T] \geq 1 - \frac{\mathbb{E}[X]}{T}.$$

Analyzing $\mathbb{E}[X]$, we see that

$$\begin{aligned}\mathbb{E}[X] &= \sum_{j \geq 1} j \cdot \Pr[C \geq (7/8 - 1/(2m)) \cdot m] \cdot \Pr[C < (7/8 - 1/(2m)) \cdot m]^{j-1} \\ &= \sum_{j \geq 1} j \cdot p \cdot (1-p)^{j-1} = \frac{1}{p} \leq \frac{m}{4}.\end{aligned}$$

Therefore,

$$\Pr[X \leq T] = 1 - \Pr[X \geq T] \geq 1 - \frac{\mathbb{E}[X]}{T} \geq 1 - \frac{m/4}{T} = 1 - \frac{m}{4T}.$$

So the probability that the algorithm terminates within T steps is at least $1 - m/(4T)$. Taking $T = m^{c+1}$ for some constant $c \geq 1$ tells us that with probability at least $1 - 1/(4m^c)$, the algorithm terminates in at most m^{c+1} steps. Since generating and checking a random assingment is polynomial-time, the overall algorithm runs in polynomial time (with high probability). □

2 Crypto and Complexity (25 points)

2.1 Part 1 (5 Points)

Show that if $\mathbf{P} = \mathbf{NP}$, then one-way functions do not exist.

Proof of Problem 2 Part 1. First, let us recall the definition of a one-way function. A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is a one-way function if f is polynomial-time computable and for all PPT algorithms A there exists a negligible function ε such that for all sufficiently large $n \in \mathbb{N}$,

$$\Pr_{x \xleftarrow{\$} \{0,1\}^n}[f(A(y)) = y \mid y = f(x)] \leq \varepsilon(n).$$

In particular, for any function f that is polynomial-time computable, consider the relation

$$R_f = \{(x, y) \mid f(x) = y\}.$$

Since f is polynomial-time computable, $|y| = \text{poly}(|x|)$. In particular, we can consider the language $L_f = \{y \colon \exists x \text{ s.t. } f(x) = y\}$. Clearly, for any polynomial-time computable f, $L_f \in \mathbf{NP}$. Since $\mathbf{P} = \mathbf{NP}$, given a y we can always decide if $y \in L_f$ in deterministic polynomial time. Now, the only assumption on f is that it is polynomial-time computable. This breaks the definition of a one-way function: there exists a DTM M which can invert any polynomial-time computable f. So one-way functions do not exist if $\mathbf{P} = \mathbf{NP}$. □

2.2 Part 2 (10 Points)

Prove that if f is a one-way function, then g defined as $g(x, y) = (f(x), y)$, where $|x| = |y|$, is also a one-way function.

Proof of Problem 2 Part 2. Two different methods come to mind to prove this result. First, assuming the definition of a one-way function, directly showing that g must satisfy the same definition. The proof I'll use is a reduction. We'll show that if g is not a one-way function, then f cannot be a one-way function.

Suppose that g is not a one-way function. This implies that there exists a PPT adversary A^*, a polynomial p, and infinitely many $n \in \mathbb{N}$ such that

$$\Pr_{a,b \xleftarrow{\$} \{0,1\}^n}[g(A(z_1, z_2)) = (z_1, z_2) \mid (z_1, z_2) = g(a, b)] \geq \frac{1}{p(2n)}.$$

We construct a new PPT adversary $\mathcal{A}$ to invert f. First, notice by definition of g, for any (a, b), we have $g(a, b) = (f(a), b)$. So the adversary A^* is actually already inverting f. In particular, $\mathcal{A}$, on input z for $z = f(x)$ for random $x \xleftarrow{\$} \{0,1\}^n$, simply samples a random y, runs $A^*(z, y)$, obtains (z_1, z_2), and outputs z_1. It has at least $1/p(2n)$ probability of being successful, so f is not a one-way function. □

2.3 Part 3 (10 Points)

Show that if one-way functions exist, then $\mathbf{distNP} \not\subseteq \mathbf{distP}$.

Proof of Problem 2 Part 3. First, we recall what it means for a pair (L, D) to be a *distributional problem.* (L, D) is a distributional problem if (1) $L \subseteq \{0,1\}^*$ is a language, and (2) $D = \{D_n\}_{n\in\mathbb{N}}$ is a family of distributions over $\{0,1\}^n$ for each n. In general, with a distributional problem, we are interested in $\Pr_{x \leftarrow D_n}[x \in L]$ for every n.

Given this, the class $\mathbf{distP}$ is the set of distributional problems (L, D) such that $L \in \mathbf{P}$, $D = \{D_n\}_{n\in\mathbb{N}}$ is a family of distributions, and there exists an algorithm A (i.e., a decider) and constants $C, \epsilon \geq 0$ such that for all $n \in \mathbb{N}$, it holds that

$$\mathop{\mathbb{E}}_{x \leftarrow D_n}\left[\frac{\text{time}_A(x)^\epsilon}{n}\right] \leq C.$$

Next, the definition of $\mathbf{distNP}$ is the set of all distributional problems (L, D) such that $L \in \mathbf{NP}$ and $D = \{D_n\}_{n\in\mathbb{N}}$ is a family of $\mathbf{P}$-computable distributions, where D is $\mathbf{P}$-computable if for every n, the cumulative probability

$$\mu_{D_n}(x) = \sum_{y \in \{0,1\}^n \,:\, y \leq x} \Pr_{z \leftarrow D_n}[y = z]$$

is computable in poly($|x|$) time.

Now, to show that **distNP** $\not\subseteq$ **distP**, it suffices to show that there exists a distributional problem (L, D) such that $(L, D) \in$ **distNP** but $(L, D) \notin$ **distP**. Suppose that one-way functions exist and let f be a one-way function. Then, recall the language L_f from the proof of Problem 2 Part 1; namely, $L_f = \{y \colon \exists x \text{ s.t. } y = f(x)\}$. Clearly $L_f \in$ **NP**; moreover, by Problem 2 Part 1, since one-way functions exist, we know that **NP** $\neq$ **P**, so $L_f \notin$ **P**.

Now take D to be any **P**-computable distribution. It holds that $(L_f, D) \in$ **distNP** but $(L_f, D) \notin$ **distP**, therefore **distNP** $\not\subseteq$ **distP**. □