CS 278: Computational Complexity Theory
Homework 3

Due: **November 20, 2025**

Fall 2025

Instructions:

- Collaboration is allowed but solutions must be written independently. List collaborators and any external resources you used.
- Write your solutions in LaTeX and submit a single PDF to the course Gradescope.
- **Deadline:** 11:59pm Pacific Time November 20.
- Late submissions lose **10%** per day (e.g., three days late $\rightarrow 0.9^3$ of your score).
- The maximum score of this homework is 200. There are 5 problems, and each problem is worth 40 points. If you get n points, your score for this homework is

$$a_3 = \frac{n}{100} \times 12.5$$

- Let a_1, a_2, a_3, a_4 be the scores for the 4 homeworks, your final grade of homework is $\min(a_1 + a_2 + a_3 + a_4, 50)$.
- In other words, you don't have to solve all the problems to get a perfect score on homeworks.

1 Problem 1: Downward self-reducibility

1.1 Determinant and Permanent

For an $n \times n$ matrix $A = (a_{ij})$, recall the *minor* M_{ij} is the $(n-1) \times (n-1)$ matrix obtained by deleting row i and column j.

Definition (Determinant). Given an $n \times n$ matrix $A = (a_{ij})$ over a field, the *determinant* of A, denoted $\det(A)$, is defined by

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma)\ a_{1,\sigma(1)} a_{2,\sigma(2)} \cdots a_{n,\sigma(n)},$$

where the sum is over all permutations σ of $\{1, 2, \ldots, n\}$, and $\operatorname{sgn}(\sigma) \in \{+1, -1\}$ denotes the sign of permutation σ.

The *sign* of a permutation $\sigma \in S_n$, denoted $\operatorname{sgn}(\sigma)$, is defined as $+1$ if σ is an even permutation (i.e., it can be written as the product of an even number of transpositions), and -1 if σ is odd (i.e., it can be written as the product of an odd number of transpositions). Formally,

$$\operatorname{sgn}(\sigma) = \begin{cases} +1 & \text{if } \sigma \text{ is even,} \\ -1 & \text{if } \sigma \text{ is odd.} \end{cases}$$

Equivalently, $\operatorname{sgn}(\sigma) = (-1)^{\#\text{ of inversions in }\sigma}$, where an inversion is a pair (i, j) with $1 \le i < j \le n$ and $\sigma(i) > \sigma(j)$.

Definition (Permanent). Given the same matrix $A = (a_{ij})$, the *permanent* of A, denoted $\operatorname{perm}(A)$, is defined by

$$\operatorname{perm}(A) = \sum_{\sigma \in S_n} a_{1,\sigma(1)} a_{2,\sigma(2)} \cdots a_{n,\sigma(n)},$$

where the sum is over all permutations σ of $\{1, 2, \ldots, n\}$. Unlike the determinant, no sign is included in the summation for the permanent.

For simplicity, we will assume that the field is over $\mathbb{F}_p$ for a fixed prime p (independent of n).

(a) (10 pts) Show that $\det(A)$ can be computed in polynomial time with the help of the oracle access to det on $(n-1) \times (n-1)$ matrices. Hint: use the Laplace expansion for the determinant.

(b) (10 pts) Prove the analogous statement for the permanent.

1.2 PSPACE-complete problem

Definition (Downward Self-Reducibility). A language (decision problem) L is said to be *downward self-reducible* if there exists a polynomial-time oracle Turing machine M such

that $M^L(x) = L(x)$ for every input x, and on any input x, every oracle query y made by M satisfies $|y| < |x|$ (that is, the queries are always to strictly smaller input lengths).

Intuitively, this means that we can decide membership in L for an input x efficiently, provided we have access to an oracle that solves L on smaller inputs.

(c) (10 pts) Define a NP-complete problem that is downward self-reducible.

(d) (10 pts) Define a PSPACE-complete problem that is downward self-reducible. Hint: think about the TQBF problem.

For (c) and (d) you can cite the textbook for NP or PSPACE-completeness, but you need to prove the downward self-reducibility yourself.

2 Problem 2: $\mathsf{AC}^0[2]$ lower bound via probabilistic polynomials over $\mathbb{F}_2$

A *probabilistic polynomial* over $\mathbb{F}_2$ for a Boolean function $F : \{0,1\}^n \to \{0,1\}$ with error ε is a distribution $\mathcal{D}$ over $\mathbb{F}_2[x_1, \ldots, x_n]$ such that

$$\Pr_{P \leftarrow \mathcal{D},\, x \leftarrow \{0,1\}^n} \big[\, P(x) = F(x) \,\big] \;\geq\; 1 - \varepsilon.$$

We say a probabilistic polynomial has degree d if all polynomials in the distribution have degree at most d.

(a) (10 pts) Show that for every m and $\varepsilon > 0$, the AND_m function has an ε-error probabilistic polynomial over $\mathbb{F}_2$ of degree $O(\log(m/\varepsilon))$.

(b) (10 pts) Deduce the same for OR_m. Observe that XOR is exactly linear over $\mathbb{F}_2$.

(c) (10 pts) Prove that any depth-d, size-$n^{O(1)}$ $\mathsf{AC}^0[2]$ circuit has a 1/8-error probabilistic polynomial over $\mathbb{F}_2$ of degree $(\log n)^{O(d)}$.

(d) (10 pts) Prove that MOD_3 is *not* in $\mathsf{AC}^0[2]$. Hint: prove that MOD_3 requires degree $\Omega(\sqrt{n})$ over $\mathbb{F}_2$ to be approximated with constant error.

Definition (MOD_3). The MOD_3 function on n bits, denoted $\mathsf{MOD}_3 : \{0,1\}^n \to \{0,1\}$, is defined by

$$\mathsf{MOD}_3(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } x_1 + x_2 + \cdots + x_n \equiv 1 \pmod 3 \\ 0 & \text{otherwise} \end{cases}$$

or more generally, $\mathsf{MOD}_3(x_1, \ldots, x_n) = 1$ if and only if the Hamming weight of $(x_1, \ldots, x_n)$ is congruent to 1 (mod 3).

3 Problem 3: Consequences of derandomization

In this problem we will explore some interesting consequences of derandomization.

Definition (prBPPand prP). The class prBPP (promise BPP) consists of all promise problems that can be decided by a probabilistic polynomial-time Turing machine with bounded error. That is, for a promise problem $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$, there exists a probabilistic polynomial-time algorithm A such that:

- For all $x \in \Pi_{\text{yes}}$, $\Pr[A(x) = 1] \geq 2/3$;
- For all $x \in \Pi_{\text{no}}$, $\Pr[A(x) = 0] \geq 2/3$;

where the probability is taken over the random coins of A. No guarantee is made for $x \notin \Pi_{\text{yes}} \cup \Pi_{\text{no}}$.

The class prP (promise P) consists of all promise problems that can be decided by a deterministic polynomial-time Turing machine, that is, there exists a polynomial-time algorithm A such that:

- For all $x \in \Pi_{\text{yes}}$, $A(x) = 1$;
- For all $x \in \Pi_{\text{no}}$, $A(x) = 0$.

As above, nothing is required for $x \notin \Pi_{\text{yes}} \cup \Pi_{\text{no}}$.

(a) (20 pts) Show that if prBPP = prP, then for every $k \in \mathbb{N}$, NP $\not\subset$ SIZE(n^k). You can use the fact that $\mathsf{prMA} \not\subset \text{SIZE}(n^k)$ for every $k \in \mathbb{N}$.

For completeness, we recall the definition of prMA.

Definition (prMA). The class prMA (promise Merlin-Arthur) consists of all promise problems $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ for which there is a polynomial-time randomized verifier $V(x, w)$ and a polynomial $p(\cdot)$ such that for every input x of length n:

- If $x \in \Pi_{\text{yes}}$, then there exists a witness $w \in \{0,1\}^{p(n)}$ such that $\Pr[V(x, w) = 1] \geq 2/3$,
- If $x \in \Pi_{\text{no}}$, then for *every* $w \in \{0,1\}^{p(n)}$, $\Pr[V(x, w) = 1] \leq 1/3$,

where the probability is taken over the random coins of V. No guarantee is made for $x \notin \Pi_{\text{yes}} \cup \Pi_{\text{no}}$.

(b) (10 pts) Show that, if "prBPP = prP implies P $\neq$ NP", then P $\neq$ NP. Hint: note that if P = NP, then prBPP = prP. You can use this fact in your proof.

(c) (10 pts) Prove that if prBPP = prP, then there is a polynomial-time algorithm A such that for every large enough $n \in \mathbb{N}$, $A(1^n)$ outputs an n-bit prime number (i.e., a prime number in the range of 2^{n-1} to $2^n - 1$).

- Hint 1: you need to do a search to decision.

- Hint 2: If prBPP = prP, then there is a polynomial-time algorithm A that takes a circuit C and 1^k as input, and outputs an estimate τ which satisfies

$$\left|\tau - \Pr_r[C(r) = 1]\right| \leq 1/k.$$

  You can use this fact for your prime construction algorithm.

Prime Number Theorem. The *prime number theorem* states that the number of prime numbers less than or equal to N, denoted by $\pi(N)$, satisfies

$$\pi(N) \sim \frac{N}{\ln N}$$

as $N \to \infty$. That is,

$$\lim_{N\to\infty} \frac{\pi(N)}{N/\ln N} = 1.$$

Equivalently, the probability that a random integer in $[1, N]$ is prime is approximately $1/\ln N$ for large N.

4 Problem 4: Instantiations of Nisan-Wigderson

In this problem, we will instantiate the Nisan-Wigderson PRG to get different pseudorandom objects.

In the first half, we will first instantiate the NW PRG to give a *k-wise independent generator* $G : \{0,1\}^{O(k^2 \log m)} \to \{0,1\}^m$.

The generator will be based on the following "hard predicate" against functions that look at most k bits (aka k-juntas). Let $\text{Parity}_\ell : \{0,1\}^\ell \to \{0,1\}$ be the Parity function on ℓ bits, i.e., $\text{Parity}_\ell(x_1, \ldots, x_\ell) = x_1 + x_2 + \ldots + x_\ell \mod 2$.

1. (10 points) Let $k < \ell$. Show that any k-junta $g : \{0,1\}^\ell \to \{0,1\}$ agrees with Parity_ℓ on exactly $1/2$ of the inputs.

2. (20 points) Recall that there exists an (ℓ, a, d) combinatorial design $S_1, \ldots, S_m \subseteq [d]$ such that $a = \log(m)$, $\ell = (k+1)\log(m)$ and $d = 100(k+1)^2 \log(m)$.

   Let $\text{NW}^f : \{0,1\}^d \to \{0,1\}^m$ be the Nisan-Wigderson construction with predicate $f = \text{Parity}_\ell$ and the above combinatorial design. We call a distribution $\mathcal{D}$ on $\{0,1\}^m$ "k-junta next-bit-unpredictable" if for any $i \in \{1, \ldots, m\}$ and any k-junta $g_i : \{0,1\}^{i-1} \to \{0,1\}$ it holds that
   $$\Pr_{x \sim \mathcal{D}}[g(x_1, \ldots, x_{i-1}) = x_i] = 1/2.$$

   Show that the output of NW^f is k-junta next-bit-unpredictable (or k-junta NBU).

3. (10 points) Use the connection between NBU and pseudorandomness, specialized to the case of k-juntas, to prove that NW^f is a k-wise independent generator.

Definition (k-wise independent generator). A function $G : \{0,1\}^s \to \{0,1\}^m$ is called a *k-wise independent generator* if the distribution on $\{0,1\}^m$ defined by $G(U_s)$ (where U_s is the uniform distribution on $\{0,1\}^s$) is k-wise independent. That is, for any choice of k distinct indices $i_1, \ldots, i_k \in \{1, \ldots, m\}$ and any $b_1, \ldots, b_k \in \{0,1\}$, we have

$$\Pr_{x \sim U_s}[G(x)_{i_1} = b_1, \ldots, G(x)_{i_k} = b_k] = 2^{-k}.ß$$

Equivalently, any k output bits of $G(U_s)$ are independent and uniformly distributed over $\{0,1\}^k$.

Definition ((ℓ, a, d) combinatorial design). A collection of sets $S_1, \ldots, S_m \subseteq [d]$ is called an (ℓ, a, d) *combinatorial design* if:

1. Each S_i has size exactly ℓ, i.e., $|S_i| = \ell$ for all $i \in \{1, \ldots, m\}$.

2. For any distinct $i \neq j$, we have $|S_i \cap S_j| \leq a$.

Here, d is the size of the universe, ℓ is the size of each set, and a bounds the intersection size between any two different sets in the collection.

5 Problem 5: Derandomization of MA and AM

In this problem, we will explore the derandomization of MA and AM, and show that they both collapse to NP under plausible assumptions.

5.1 Derandomization of MA.

MA (Merlin-Arthur) is the class of languages for which there exists a probabilistic polynomial-time verifier V such that:

- (Completeness) If $x \in L$, then there exists a "proof" w (also called a *witness*) such that $V(x, w; r) = 1$ with probability at least $2/3$ over the random coins r.
- (Soundness) If $x \notin L$, then for every "proof" w, $V(x, w; r) = 1$ with probability at most $1/3$ over the random coins r.

Intuitively, Merlin (the prover) sends a string w to Arthur (the verifier), who then tosses random coins and decides to accept or reject based on x, w, and the random coins.

Part (a): 20 pts. Show that if E requires $2^{\epsilon n}$-size circuits for some $\epsilon > 0$, then MA = NP.

Hint: You can use the fact that if E requires $2^{\epsilon n}$-size circuits for some $\epsilon > 0$, then there exists a pseudorandom generator (PRG) with $O(\log n)$-bit seed that fools $O(n)$-size circuits with error at most 0.1. That is, for every n, there is an efficiently computable PRG $G : \{0,1\}^{c \log n} \to \{0,1\}^n$ such that for every Boolean circuit C of size $O(n)$,

$$\left| \Pr_{x \sim U_n} [C(x) = 1] - \Pr_{s \sim U_{c \log n}} [C(G(s)) = 1] \right| \leq 0.1,$$

where U_k denotes the uniform distribution over $\{0,1\}^k$ and $c > 0$ is some constant.

5.2 Derandomization of AM.

AM (Arthur-Merlin) is the class of languages for which there exists a probabilistic polynomial-time verifier V such that:

- (Completeness) If $x \in L$, then with probability at least $2/3$ over Arthur's random coins r, there exists a string w such that $V(x; r, w) = 1$
- (Soundness) If $x \notin L$, then with probability at least $2/3$ over Arthur's random coins r, for every string w, $V(x; r, w) = 0$.

In the AM protocol (in its standard 2-message form), Arthur sends random coins r to Merlin, Merlin responds with a string w, and then Arthur computes $V(x; r, w)$.

Part (b): 20 pts. Show that if E cannot be $(1/2 + 2^{-\epsilon n})$-approximated by $2^{\epsilon n}$-size SAT-oracle circuits for some $\epsilon > 0$, then AM = NP.

Definition (SAT-oracle circuit). A *SAT-oracle circuit* is a Boolean circuit that, in addition to standard logic gates (such as AND, OR, NOT), may also include special gates that can compute the solution to instances of the SAT problem. That is, the circuit can make queries to an oracle that, given an (encoding of) Boolean formula φ as input, outputs whether φ is satisfiable. The size of a SAT-oracle circuit is defined as the total number of gates (including standard and oracle gates).

Definition ((1/2+δ)-approximation of a function). Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and C be a (possibly oracle) circuit. We say that C *(1/2+δ)-approximates* f if

$$\Pr_{x \sim U_n}[C(x) = f(x)] \geq \frac{1}{2} + \delta,$$

where the probability is over a uniformly random input $x \in \{0,1\}^n$ and $\delta > 0$ is a real (possibly depending on n).

Remark. In the context of the hardness assumption for this problem, the statement is that for some $\epsilon > 0$, no SAT-oracle circuit of size $2^{\epsilon n}$ can compute a function in E correctly on more than a $1/2 + 2^{-\epsilon n}$ fraction of the inputs.

Solution to Problem 1

Solution to Problem 2

Solution to Problem 3

Solution to Problem 4

Solution to Problem 5