CS 278: Computational Complexity Theory
Homework 3 Solutions

Due: **November 20, 2025**

Fall 2025

Problem 1: Downward self-reducibility

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.

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.

Solution to Problem 1: Downward self-reducibility

Part (a)

We want to show that $\det(A)$ can be computed in polynomial time given oracle access to det on $(n-1) \times (n-1)$ matrices. Using the Laplace expansion along the first row:

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(M_{1j})$$

where M_{1j} is the $(n-1) \times (n-1)$ minor obtained by removing the first row and j-th column. The algorithm is:

1. For $j = 1$ to n:
2. Construct M_{1j}.
3. Query the oracle to get $d_j = \det(M_{1j})$.
4. Compute $D = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} d_j$.
5. Return D.

This algorithm makes n oracle queries and performs $O(n)$ arithmetic operations (multiplications and additions) in the field. Constructing minors takes polynomial time. Thus, it is a polynomial-time reduction.

Part (b)

Similarly for the permanent, we use the expansion along the first row:

$$\text{perm}(A) = \sum_{j=1}^{n} a_{1j}\text{perm}(M_{1j})$$

The algorithm is identical to part (a), but without the $(-1)^{1+j}$ factor:

1. For $j = 1$ to n:
2. Construct M_{1j}.
3. Query the oracle to get $p_j = \text{perm}(M_{1j})$.
4. Compute $P = \sum_{j=1}^{n} a_{1j}p_j$.
5. Return P.

This is also a polynomial-time reduction.

Part (c)

We define SAT as the downward self-reducible NP-complete problem. Let SAT be the language of satisfiable Boolean formulas in CNF. Given a formula $\phi(x_1, \ldots, x_n)$ with n variables. $\phi \in \text{SAT}$ if and only if $\phi|_{x_1=0} \in \text{SAT}$ or $\phi|_{x_1=1} \in \text{SAT}$. Here $\phi|_{x_1=b}$ is the formula obtained by substituting $x_1 = b$, which has $n - 1$ variables (after simplification). The algorithm $M^{\text{SAT}}(\phi)$:

1. Construct $\phi_0 = \phi|_{x_1=0}$ and $\phi_1 = \phi|_{x_1=1}$.
2. Query oracle: $res_0 = \text{SAT}(\phi_0)$, $res_1 = \text{SAT}(\phi_1)$.
3. Return $res_0 \vee res_1$.

The queries are on instances with fewer variables (or shorter length). Thus SAT is downward self-reducible. SAT is well-known to be NP-complete.

Part (d)

We define TQBF (True Quantified Boolean Formulas) as the downward self-reducible PSPACE-complete problem. A TQBF instance is of the form $\Psi = Q_1x_1Q_2x_2 \ldots Q_nx_n\phi(x_1, \ldots, x_n)$, where $Q_i \in \{\exists, \forall\}$. Base case: if no quantifiers, evaluate ϕ (constant time/poly time). Recursive step: If $\Psi = \exists x_1\Psi'$, then Ψ is true iff $\Psi'|_{x_1=0}$ is true or $\Psi'|_{x_1=1}$ is true. If $\Psi = \forall x_1\Psi'$, then Ψ is true iff $\Psi'|_{x_1=0}$ is true and $\Psi'|_{x_1=1}$ is true. The subproblems $\Psi'|_{x_1=b}$ have $n - 1$ quantifiers. The algorithm $M^{\text{TQBF}}(\Psi)$:

1. Let Q_1 be the first quantifier.

2. Construct $\Psi_0 = \Psi|_{x_1=0}$ and $\Psi_1 = \Psi|_{x_1=1}$ (removing $Q_1 x_1$).
3. Query oracle: $v_0 = \text{TQBF}(\Psi_0)$, $v_1 = \text{TQBF}(\Psi_1)$.
4. If $Q_1 = \exists$, return $v_0 \vee v_1$.
5. If $Q_1 = \forall$, return $v_0 \wedge v_1$.

The queries are strictly smaller. TQBF is PSPACE-complete.

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).

Solution to Problem 2

Part (a)

Over $\mathbb{F}_2$, pick independent random linear forms $L_1, \ldots, L_k$ where $L_j(x) = \sum_{i=1}^{m} r_i^{(j)} x_i \pmod 2$ for uniform $r^{(j)} \in \{0,1\}^m$. Define

$$P_{\mathrm{OR}}(x) \;=\; 1 + \prod_{j=1}^{k} \big(1 + L_j(x)\big) \quad \text{in } \mathbb{F}_2[x_1, \ldots, x_m].$$

If $\mathrm{OR}_m(x) = 0$ then all $L_j(x) = 0$, so $P_{\mathrm{OR}}(x) = 0$. If $\mathrm{OR}_m(x) = 1$ then for each j, $\Pr[L_j(x) = 1] = 1/2$ independently, hence $\Pr[P_{\mathrm{OR}}(x) = 1] = 1 - 2^{-k}$.

Choosing $k = \lceil \log_2(1/\varepsilon) \rceil$ gives an ε-error probabilistic polynomial of degree $k = O(\log(1/\varepsilon))$. For AND_m, let $y_i = 1 + x_i$ (in $\mathbb{F}_2$) and set

$$P_{\mathrm{AND}}(x) \;=\; 1 + P_{\mathrm{OR}}(1 + x_1, \ldots, 1 + x_m),$$

which has the same degree. Since $O(\log(1/\varepsilon)) \subseteq O(\log(m/\varepsilon))$, this matches the stated bound.

Part (b)

From Part (a), $P_{\mathrm{OR}}(x) = 1 + \prod_{j=1}^{k} \big(1 + L_j(x)\big)$ (with $k = \Theta(\log(1/\varepsilon))$) is an ε-error probabilistic polynomial for OR_m, of degree $k = O(\log(1/\varepsilon))$.

Part (c)

Let C be a depth-d, size-$S = n^{O(1)}$ $\mathsf{AC}^0[2]$ circuit. Replace:

- Input gates by variables (degree 1, error 0);
- Unbounded-fan-in XOR by linear forms over $\mathbb{F}_2$ (degree 1, error 0);
- Each AND/OR gate by a probabilistic polynomial of degree $O(\log(S/\delta))$ with error at most δ (from Parts (a),(b)).

By a union bound, setting $\delta = 1/(8S)$ makes the overall error at most $1/8$. Along any input-to-output path, degrees multiply, so the total degree is $(O(\log(1/\delta)))^d = O((\log S)^d) = O((\log n)^d)$ since $S = n^{O(1)}$ and d is constant.

Part (d)

Refer to Theorem 14.4 of Arora and Barak's book for a proof of the hinted lower bound.

If MOD_3 were in $\mathsf{AC}^0[2]$, Part (c) would yield an $(\log n)^{O(1)}$-degree probabilistic polynomial over $\mathbb{F}_2$ with constant error, contradicting the lemma. Hence $\mathsf{MOD}_3 \notin \mathsf{AC}^0[2]$.

Problem 3: Consequences of derandomization

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

Definition (prBPP and 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.

Solution to Problem 3

Part (a)

Assume prBPP = prP. We want to show NP $\not\subset$ SIZE(n^k). We use the fact $\mathsf{prMA} \not\subset$ SIZE(n^k). We claim that under prBPP = prP, we have $\mathsf{prMA} = \mathsf{prNP}$.

Recall prMA definition: $\exists w, \Pr[V(x, w) = 1] \geq 2/3$ for $x \in \Pi_{\text{yes}}$ and $\Pr[V(x, w) = 1] \leq 1/3$ for $x \in \Pi_{\text{no}}$ for all $w \in \{0, 1\}^{p(n)}$.

The verification between $\Pr[V(x, w) = 1] \geq 2/3$ and $\Pr[V(x, w) = 1] \leq 1/3$ is a prBPP problem (given x, w). Since prBPP = prP, there is a deterministic poly-time algorithm $D(x, w)$ that decides if the probability is high or low.

So $x \in \Pi_{yes} \iff \exists w, D(x, w) = 1$. This puts prMA in prNP. So $\mathsf{prMA} = \mathsf{prNP}$. Since $\mathsf{prMA} \not\subset$ SIZE(n^k), we have $\mathsf{prNP} \not\subset$ SIZE(n^k). If NP $\subset$ SIZE(n^k), then $\mathsf{prNP} \subset$ SIZE(n^k) (since any promise problem in prNP can be reduced to SAT, and if SAT has small circuits, so does the promise problem). Thus NP $\not\subset$ SIZE(n^k).

Part (b)

Assume (prBPP = prP $\implies$ P $\neq$ NP). We want to show P $\neq$ NP.

Proof by contradiction. Assume P = NP. If P = NP, then P = NP = PH. Also prBPP $\subseteq \Sigma_2^P \cap \Pi_2^P$ = P. So prBPP = prP. Then the premise of the implication (prBPP = prP $\implies$ P $\neq$ NP) is satisfied. Therefore, the conclusion P $\neq$ NP must hold. But we assumed P = NP. Contradiction. Thus P $\neq$ NP.

Part (c)

Assume prBPP = prP. By the hint, there is a deterministic poly-time estimator E that, given a circuit C and 1^k, outputs τ with $|\tau - \Pr_r[C(r) = 1]| \le 1/k$. We give a deterministic poly-time algorithm that outputs an n-bit prime.

Algorithm (bit-fixing with approximate counting).

1. Target interval: $[2^{n-1}, 2^n - 1]$. Initialize prefix $u \leftarrow 1$.
2. For $i = 2$ to n (let $m = n - i + 1$ be the remaining length):
   (a) For each $b \in \{0,1\}$, define $C_{u,b} : \{0,1\}^m \to \{0,1\}$ by $C_{u,b}(v) = \text{PRIME}(\text{bin}(u \circ b \circ v))$, where PRIME $\in$ P.
   (b) Run E on $C_{u,b}$ with accuracy $k = n^3$ to obtain $\widehat{p}_b$.
3. Let $b^\star \in \{0,1\}$ maximize $\widehat{p}_b$ and set $u \leftarrow u \circ b^\star$.
4. Output the number with binary expansion u.

Correctness. Let S be the set of n-bit primes and $S(u)$ those with prefix u. Write $\delta(u) = |S(u)|/2^m$ for the (true) density among completions of u when m bits remain. By the prime number theorem, initially $\delta(\epsilon) \ge c/n$ for some absolute $c > 0$ and all large n. Also $\delta(u) = \frac{1}{2}(\delta(u0) + \delta(u1))$, so $\max_b \delta(ub) \ge \delta(u)$. With estimator error $1/n^3 \ll \delta(u)/10$, the chosen $b^\star$ satisfies $\delta(ub^\star) \ge \delta(u)/2$. Inductively after $n - 1$ extensions,

$$|S(u)| \;\ge\; \delta(\epsilon) \cdot 2^{n-1} \;\ge\; \frac{c}{n} \cdot 2^{n-1},$$

so when exactly one completion remains, it must be prime. Running time is polynomial since each call to E and PRIME is polynomial and there are $O(n)$ iterations.

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

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

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.

Solution to Problem 4

Part (a)

Let $g : \{0,1\}^\ell \to \{0,1\}$ be a k-junta with $k < \ell$, and let J be its dependency set, $|J| \le k$. Write $\mathrm{Parity}_\ell(x) = \big(\sum_{j\in J} x_j\big) + \big(\sum_{j\notin J} x_j\big) \bmod 2$. For any fixed x_J, $g(x)$ is constant while $\sum_{j\notin J} x_j \bmod 2$ is a uniform bit over random $x_{\bar{J}}$. Hence

$$\Pr_{x_{\bar{J}}}\big[\mathrm{Parity}_\ell(x) = g(x)\big] \;=\; \tfrac{1}{2}.$$

Averaging over x_J gives overall agreement $1/2$.

Part (b)

Let $\mathrm{NW}^f : \{0,1\}^d \to \{0,1\}^m$ with $f = \mathrm{Parity}_\ell$ and an (ℓ, a, d)-design $S_1, \ldots, S_m$. For fixed i and any k-junta $g : \{0,1\}^{i-1} \to \{0,1\}$ depending on $\{z_{j_1}, \ldots, z_{j_k}\}$ with $j_r < i$, the inputs to g depend only on x via $\bigcup_{r=1}^k S_{j_r}$. The design guarantees

$$|S_i \cap \bigcup_{r=1}^{k} S_{j_r}| \;\le\; \sum_{r=1}^{k} |S_i \cap S_{j_r}| \;\le\; ka.$$

With $\ell = (k+1)\log m$ and $a = \log m$, we have at least $\ell - ka = \log m > 0$ coordinates in S_i that are disjoint from $\bigcup_r S_{j_r}$. Let $U = S_i \setminus \bigcup_r S_{j_r}$. Then

$$z_i \;=\; \mathrm{Parity}(x|_{S_i}) \;=\; \mathrm{Parity}(x|_U) \;+\; \mathrm{Parity}\big(x|_{S_i \cap \bigcup_r S_{j_r}}\big),$$

where $\mathrm{Parity}(x|_U)$ is a uniform bit independent of $g(z_1, \ldots, z_{i-1})$. Conditioning on the inputs to g therefore leaves z_i uniform, so $\Pr[g(z_1, \ldots, z_{i-1}) = z_i] = 1/2$. Thus NW^f is k-junta NBU.

Part (c)

Fix distinct indices $i_1 < \cdots < i_k$. Any Boolean function of $(z_{i_1}, \ldots, z_{i_{k-1}})$ is a $(k-1)$-junta, so by NBU it cannot predict z_{i_k} better than $1/2$. Hence z_{i_k} is independent of $(z_{i_1}, \ldots, z_{i_{k-1}})$. Inducting on k shows every k outputs are mutually independent and uniform, i.e., NW^f is a k-wise independent generator.

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.

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.

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 5

Part (a)

Let $L \in \mathsf{MA}$ with verifier $V(x, w; r)$. Without loss of generality we can assume V runs in n^k time for some constant k.

Assume E requires $2^{\epsilon n}$-size circuits. Then there exists a PRG $G : \{0,1\}^{c \log n} \to \{0,1\}^n$ that ± 0.1-fools $O(n^k)$-size circuits.

For fixed (x, w), $C_{x,w}(r) := V(x, w; r)$ is a circuit of size $\mathrm{poly}(n)$. Define

$$A(x, w) \;=\; \frac{1}{2^{c \log n}} \sum_{s \in \{0,1\}^{c \log n}} V\big(x, w, G(s)\big).$$

Then $|A(x, w) - \Pr_r[V(x, w; r) = 1]| \leq 0.1$. If $x \in L$, some w has acceptance $\geq 2/3$, so $A(x, w) \geq 2/3 - 0.1 > 1/2$. If $x \notin L$, all w have acceptance $\leq 1/3$, so $A(x, w) \leq 1/3 + 0.1 < 1/2$. An NP verifier can guess w and deterministically compute $A(x, w)$ in polynomial time, accepting iff $A(x, w) > 1/2$. Hence $\mathsf{MA} \subseteq \mathsf{NP}$, and trivially $\mathsf{NP} \subseteq \mathsf{MA}$, so $\mathsf{MA} = \mathsf{NP}$.

Part (b)

Assume E cannot be $(1/2 + 2^{-\epsilon n})$-approximated by $2^{\epsilon n}$-size SAT-oracle circuits. Then there is a PRG G that fools such circuits. The crucial observation is go over the proof of NW PRG and realize that it does not care whether the circuits have oracle gates or not.

For $L \in \mathsf{AM}$ with verifier $V(x; r, w)$, define $f_x(r) = 1$ iff $\exists w\, V(x; r, w) = 1$. The map f_x is computable by a SAT-oracle circuit of size $\mathrm{poly}(n)$, so

$$\left| \Pr_r[f_x(r) = 1] - \frac{1}{2^{c \log n}} \sum_s f_x(G(s)) \right| \leq 0.1$$

for appropriate parameters. An NP verifier for L on input x guesses witnesses $\{w_s\}$ (one per seed s) and checks whether

$$\frac{1}{2^{c \log n}} \sum_s V\big(x; G(s), w_s\big) \;>\; 1/2.$$

If $x \in L$ then $\Pr_r[f_x(r) = 1] \geq 2/3$, so the average above exceeds $1/2$; if $x \notin L$ then $\Pr_r[f_x(r) = 1] \leq 1/3$, so it is below $1/2$. Thus $\mathsf{AM} \subseteq \mathsf{NP}$, and since $\mathsf{NP} \subseteq \mathsf{AM}$, we have $\mathsf{AM} = \mathsf{NP}$.