Problem Set 2

Pseudorandomness, Autumn 2023, University of Chicago
Instructor: William Hoza (williamhoza@uchicago.edu)

Submission. Solutions are due **Friday, October 27** at 5pm Central time. Submit your solutions through Canvas. You are encouraged to typeset your solutions in LaTeX. (Consider using Overleaf, a popular online LaTeX editor.) If you prefer, you may submit a photo of handwritten solutions instead.

The collaboration and resource policies below can also be found on the course webpage.

Collaboration. You are encouraged to collaborate with your classmates on problem sets, but you must adhere to the following rules.

- Before discussing a problem with a classmate, you must work on the problem on your own for at least 20 minutes.
- You must ultimately write your solution on your own. While writing your solution, you may not consult any notes that you took during a discussion with another classmate.
- In your write-up, you must list any classmates who contributed to your solution through discussion. The fact that A contributed to B's solution does not necessarily mean that B contributed to A's solution.

Permitted Resources for Full Credit. Beyond discussions with me and discussions with classmates as discussed above, you may use the course texts and any notes that might be posted in the "Course Timeline" section of the course webpage. If you wish to receive full credit on a problem, you may not use any other resources.

Permitted Resources for Partial Credit. If you wish, you may use outside resources (Wikipedia, ChatGPT, Stack Exchange, research papers, etc.) to solve a problem for partial credit. If you decide to go this route, you must make a note of which outside resources you used when you were working on each problem. You must disclose using a resource even if it was ultimately unhelpful for solving the problem. Furthermore, if you consult an outside resource while working on a problem, then you must not discuss that problem with your classmates.

Let G be a d-regular undirected graph on n vertices, and let M be the transition probability matrix of G. In class, we defined $\lambda(G)$ by the formula

$$\lambda(G) := \max_{\pi} \frac{\|\pi M - u\|_2}{\|\pi - u\|_2},$$

where the maximum is over all probability vectors $\pi \in \mathbb{R}^n$. We also discussed an equivalent characterization of $\lambda(G)$ in terms of PRGs. There is yet another characterization of $\lambda(G)$ in terms of the *eigenvalues* of M. You should read Section 2.4.3 in Vadhan's text to learn about this "eigenvalue characterization" of $\lambda(G)$. In this problem, you will use the eigenvalue characterization of $\lambda(G)$ to prove that $\lambda(G) \leq 1 - 1/\operatorname{poly}(n, d)$, a fact that we use use in class to analyze randomized and deterministic algorithms for undirected s-t connectivity.

Problem 1 (15 points)**.** Solve parts (1) through (5) of Problem 2.9 in Vadhan's text.

In class, we study standard-order ROBPs, which are the $k = 1$ case of the definition below.

Definition 1 (k-pass branching programs)**.** A *k-pass n-variable width-w branching program* is a directed graph in which the vertices are arranged in $kn+1$ layers, $V_0, \dots, V_{kn}$, with $|V_i| \leq w$ for every i. There is a designated "start vertex" $v_0 \in V_0$ and a designated set of "accepting vertices" $V_{\text{accept}} \subseteq V_{kn}$. For every $i \in [kn]$, every vertex $v \in V_{i-1}$ has two outgoing edges leading to V_i labeled zero and one.

The program computes a Boolean function $f \colon \{0,1\}^n \to \{0,1\}$ as follows. Given an input $x \in \{0,1\}^n$, we start at v_0. In step $i \in [kn]$, we traverse the outgoing edge labeled $x_{i \bmod n}$. After kn steps, we reach some vertex $v_{kn} \in V_{kn}$, and we accept ($f(x) = 1$) if and only if $v_{kn} \in V_{\text{accept}}$.

Example 1. Let $f \colon \{0,1\}^{n+\log n} \to \{0,1\}$ be the function $f(x,i) = x_i$. This function can be computed by a two-pass branching program of width n, whereas every one-pass branching program computing f has width at least 2^n.

Problem 2 (10 points)**.** Prove that there exists an explicit ε-PRG for k-pass n-variable width-w branching programs with seed length $O(k \cdot \log w \cdot \log n + \log^2 n + \log n \cdot \log(1/\varepsilon))$.

Hint: Mimic the INW generator, but use Theorem 3.1.2 in the HH text (on fooling communication protocols) to recycle the seed.

For this problem, let us suppose that we observe a sequence of values $x_1, x_2, \ldots, x_n \in \{0,1\}^t$ one at a time, where n is much larger than t. After making all of these observations, we would like to know the approximate number of *distinct* values that we observed, i.e., the cardinality of the set $S = \{x_1, \ldots, x_n\} \subseteq \{0,1\}^t$. The challenge is that we do not have enough space to store the entire set S, and we are not able to go back and re-read old observations x_i. (For example, maybe each time someone visits our website, we observe their IP address $x_i \in \{0,1\}^t$. We cannot afford to store all of the IP addresses of all of our website's visitors, but we would like to know approximately how many distinct IP addresses visited our website.) For simplicity, let us assume that either $|S| > 0.99 \cdot 2^{t/2}$ or else $|S| < 0.01 \cdot 2^{t/2}$; our goal is to determine which inequality holds. We can consider two models of randomized algorithms for this problem.

- In the *coin flip* model, the algorithm can toss a coin to decide what to do next. Notably, the algorithm cannot "re-read" old coin tosses: if it wants to know the outcome of a prior coin toss, then it needs to have written down that outcome at the time (which is costly in terms of space complexity).
- In the *random oracle* model, the algorithm has access to a sequence of "pre-flipped" random bits. The algorithm can look at the i-th random bit as many times as it wants.

Here is an algorithm for our problem in the random oracle model:

1. Sample a "hash function" $h\colon \{0,1\}^t \to \{0,1\}^{t/2-2}$ uniformly at random. (Assume t is even and $t > 4$.)
2. If we ever observe an x_i such that $h(x_i) = 0^{t/2-2}$ (the all-zeroes string), then halt and accept.
3. If we complete all of our observations and we never see any x_i such that $h(x_i) = 0^{t/2-2}$, then reject.

Since we are in the random oracle model, we do not need to actually write down the hash function h, and hence the algorithm above only uses $O(t)$ bits of space.

Problem 3 (10 points)**.**

(a) Show that the algorithm described above succeeds with probability at least 95%. That is, show that if $|S| > 0.99 \cdot 2^{t/2}$, then the algorithm accepts with probability at least 95%, and if $|S| < 0.01 \cdot 2^{t/2}$, then the algorithm accepts with probability at most 5%. You may use the following two useful inequalities:

$$1 - \varepsilon \le e^{\varepsilon} \qquad \text{for any } \varepsilon \in \mathbb{R}$$
$$(1-\varepsilon)^m \ge 1 - m\varepsilon \qquad \text{for any integer } m \ge 1 \text{ and any } \varepsilon \in (-1, \infty).$$

(b) Design an algorithm for this problem in the coin flip model that still succeeds with probability at least 95% and that uses only $O(t^2)$ bits of space. You may take for granted the fact that the space complexity of the INW generator is bounded by $O(s)$ where s is the seed length.

Caution: In general, if a space-efficient algorithm observes its random bits multiple times, then it is not necessarily safe to use the INW generator to replace its random bits with pseudorandom bits. You will need to provide a justification for why it is safe in this particular case.

Problem 4 (0 points)**.** Choose a topic for your project. If you decide to study a theorem on the list of possible project topics, then please tell me which theorem you choose. If you're interested in doing something else for your project, then you will need to get your plan approved; please briefly describe the proposed project here (and we might discuss it further in person or over email).