Instructor: Rocco A. Servedio

Computer Science 4236: Introduction to Computational Complexity
Problem Set #4 Spring 2023

Due 11:59pm Monday, April 24, 2023

Note: Even if you are using late days, **all assignments must be turned in no later than 11:59pm Friday, April 28** — solutions will be released soon thereafter (so that they are available before the final exam).

See the course Web page for instructions on how to submit homework.
Important: To make life easier for the TAs, **please start each problem on a new page.**

Problem 1 We say that a function $f : \Sigma^* \to \Sigma^*$ is *downward self-reducible* if there is a polynomial-time oracle algorithm A with the following property: given any input $x \in \Sigma^n$, the algorithm $A^{f_{n-1}}$ run on input x correctly computes $f(x)$. Here "f_{n-1}" denotes an oracle that computes f but only on inputs of length at most $n-1$.

(a) Show that the permanent function

$$PER(M) = \sum_{\pi \in S_n} M_{i\pi(1)} \cdots M_{n\pi(n)}$$

is downward self-reducible.
(b) Show that if f is downward self-reducible then f is computable in polynomial space.

Problem 2 A *monotone* Boolean formula has only ANDs ($\wedge$) and ORs ($\vee$) and contains no negations. For instance, $(x_1 \wedge x_2 \wedge x_3) \vee (x_2 \wedge (x_3 \vee x_4))$ is a monotone Boolean formula.

The function $\#MON$ is defined as follows: on input a monotone Boolean formula, it outputs the number of satisfying assignments for the formula.

Prove that $\#MON$ is $\#P$-complete. (Hint: Reduce from $\#3SAT$. Given a 3CNF formula $F(x_1, \ldots, x_n)$, construct two monotone formulas A and B (over a larger set of variables) such that the number of satisfying assignments to F equals the number of satisfying assignments to A minus the number of satisfying assignments to B.) Note that this reduction uses the power of oracle access more fully than the reductions we did in class, since the oracle is called more than once.

Problem 3 Show that $P^{\#P} = P^{PP}$.

Problem 4

(For this problem you should assume you can efficiently draw a uniform random integer from any set $\{1, \ldots, N\}$ in time $\text{poly}(\log N)$.)

(a) Suppose that A is a deterministic polynomial-time algorithm which, given a 3CNF formula φ as input, outputs the # of satisfying assignments for φ. Use A to give a randomized polynomial-time algorithm which, given a 3CNF formula φ as input, outputs a uniformly random satisfying assignment for φ (i.e. each assignment z that satisfies φ is equally likely to be output by your algorithm).

(b) Suppose now that A only *approximately* counts the number of satisfying assignments to a 3CNF. More precisely, suppose that A is a randomized algorithm that has has the following performance guarantee: given as input $\langle \varphi, \varepsilon \rangle$ where φ is a 3CNF formula over n variables and $0 < \varepsilon < 1$, algorithm A runs in time $\text{poly}(n, 1/\varepsilon)$ and with probability at least $1 - 1/5^n$ it outputs a value N such that $(1-\varepsilon)\#\varphi \leq N \leq (1+\varepsilon)\#\varphi$, where $\#\varphi$ denotes the number of satisfying assignments to φ.[1]

Use A to give a randomized algorithm A' that has the following performance guarantee: given as input $\langle \varphi, \varepsilon \rangle$ where φ is a 3CNF formula over n variables and $0 < \varepsilon < 1$, algorithm A' runs in $\text{poly}(n, 1/\varepsilon)$ time, outputs some satisfying assignment z for φ, and for each satisfying assignment z we have

$$\frac{(1-\varepsilon)}{\#\varphi} \leq \mathbf{Pr}[A' \text{ outputs } z] \leq \frac{(1+\varepsilon)}{\#\varphi}.$$

Problem 5 In this problem you will show how both the accuracy and the correctness probability of approximate counting algorithms can be improved.

(a) **Improving accuracy:** Suppose that A is a $\text{poly}(n)$-time algorithm that is an n-approximation algorithm for #3CNF: on input an n-variable 3CNF formula φ, the output $A(\varphi)$ satisfies

$$\#\varphi/n \leq A(\varphi) \leq n \cdot \#\varphi,$$

where $\#\varphi$ denotes the number of satisfying assignments of the n-variable 3CNF φ. Show that A can be used to obtain a fully polynomial-time approximation scheme for $\#3CNF$, i.e. an algorithm A' which, on input an n-variable 3CNF formula φ and an accuracy parameter $0 < \varepsilon < 1$, runs in $\text{poly}(n, 1/\varepsilon)$ time and outputs a value $A'(\varphi)$ that satisfies

$$(1-\varepsilon)\#\varphi \leq A'(\varphi) \leq (1+\varepsilon)\#\varphi.$$

(b) **Improving correctness probability:** Now suppose that A is a p.p.t. randomized algorithm that has the following performance: on input an n-variable 3CNF formula φ, with probability at least $2/3$ the output $A(\varphi)$ satisfies

$$\#\varphi/2 \leq A(\varphi) \leq 2 \cdot \#\varphi.$$

Show that A can be used to obtain an efficient algorithm A' with much lower failure probability: A' is also a p.p.t. algorithm and, on input φ, with probability at least $1 - 1/2^n$ the output $A'(\varphi)$ satisfies

$$\#\varphi/2 \leq A'(\varphi) \leq 2 \cdot \#\varphi.$$

[1] The failure probability $1/5^n$ may seem very low, but it is easy to achieve this using the same kind of amplification tricks we've seen in class, given an algorithm with failure probability at most (say) $1/3$.