CS 221: Computational Complexity **Prof. Salil Vadhan**

Problem Set 4

Assigned: Sat. Mar. 29, 2014 Due: Fri. Apr. 11, 2014 (5 PM sharp)

- You must *type* your solutions. LaTeX, Microsoft Word, and plain ascii are all acceptable. Submit your solutions *via email* to `cs221-hw@seas.harvard.edu`. If you use LaTeX, please submit both the compiled file (`.pdf`) and the source (`.tex`). Please name your files `PS4-yourlastname.*`.
- Strive for clarity and conciseness in your solutions, emphasizing the main ideas over low-level details. Do not despair if you cannot solve all the problems! Difficult problems are included to stimulate your thinking and for your enjoyment, not to overwork you. *'ed problems are extra credit.

Problem 1. (Cook reductions to promise problems) Note that for a promise problem Π, "running an algorithm with oracle Π" is not in general well-defined, because it is not specified what the oracle should return if the input violates the promise.[1] Thus, when we say that a problem Γ can be solved in polynomial time with oracle access to Π, we mean that there is a polynomial-time oracle algorithm A such that for *every* oracle $O : \{0,1\}^* \to \{0,1\}$ that solves Π (i.e. O is correct on $\Pi_Y \cup \Pi_N$), it holds that A^O solves Γ.

Let Π be the promise problem

$$\Pi_Y \stackrel{\text{def}}{=} \{(\varphi, \psi) : \varphi \in \text{SAT}, \psi \notin \text{SAT}\}$$
$$\Pi_N \stackrel{\text{def}}{=} \{(\varphi, \psi) : \varphi \notin \text{SAT}, \psi \in \text{SAT}\}$$

Show that $\Pi \in \mathbf{prNP} \cap \mathbf{prcoNP}$ but $\text{SAT} \in \mathbf{prP}^{\Pi}$. Deduce that $\mathbf{prNP} \subseteq \mathbf{prP}^{\mathbf{prNP} \cap \mathbf{prcoNP}}$. Note that an analogous inclusion seems unlikely for language classes, since $\mathbf{P}^{\mathbf{NP} \cap \mathbf{coNP}} = \mathbf{NP} \cap \mathbf{coNP}$.

Problem 2. (one-sided error vs. two-sided error)

1. Show that if $\mathbf{NP} \subseteq \mathbf{BPP}$, then $\mathbf{NP} = \mathbf{RP}$.
2. (*) Show that $\mathbf{prBPP} \subseteq \mathbf{prRP}^{\mathbf{prRP}}$, and thus $\mathbf{prRP} = \mathbf{prP}$ iff $\mathbf{prBPP} = \mathbf{prP}$. (Hint: look at the proof that $\mathbf{BPP} \subseteq \mathbf{PH}$.)

[1] A similar issue comes up with problems where there are multiple valid answers on a given input, such as search or approximation problems. Again, in such cases, we should require that the algorithm works correctly for every oracle that solves the problem.

Problem 3. (A hierarchy theorem for prBPTIME) Recall that in class we attempted to prove that for all time-constructible f, g such that $f(n) \log f(n) = o(g(n))$, we have $\mathbf{prBPTIME}(f(n)) \subsetneq \mathbf{prBPTIME}(g(n))$. Specifically, we defined a probabilistic TM M that on input x, runs the x'th probabilistic TM M_x on x for $g(|x|)$ steps and outputs the opposite. Then we considered the promise problem

$$\begin{aligned} \Pi_Y &= \{x : \Pr[M(x) = 1] \geq 2/3\} \\ \Pi_N &= \{x : \Pr[M(x) = 1] \leq 1/3\} \end{aligned}$$

and observed that $\Pi \in \mathbf{prBPTIME}(g(n))$. However, we ran into a difficulty in showing that $\Pi \notin \mathbf{prBPTIME}(f(n))$, i.e. every probabilistic time $f(n)$ TM N fails to decide Π on some input $x \in \Pi_Y \cup \Pi_N$. A natural choice is to take x so that $N = M_x$ (so that M does the opposite of N on input x). However, the problem was that x may not satisfy the promise for Π. Show how to fix this problem using the "lazy diagonalization" method from the proof of the nondeterministic time hierarchy theorem.

Problem 4. (#P-completeness)

1. A *matching* in a graph is a set S of edges such that every vertex touches *at most one* edge in S (as opposed to exactly one, as required in a perfect matching). Show that #MATCHINGS, the problem of counting all the matchings in a graph, is #**P**-complete. (Hint: reduce from #PERFECT MATCHINGS. Given a graph G, consider the graph G_k obtained by attaching k new edges to each vertex of G. G_k has $n + nk$ vertices, where n is the number of vertices in G. Show that the number of perfect matchings in G can be recovered from the number of matchings in each of $G_0, \ldots, G_n$.)

2. An *independent set* in a graph G is a set S of vertices such that no two elements of S are connected by an edge in G. Prove that #INDEPENDENT SETS, the problem of counting the number of independent sets in a graph, is #**P**-complete.

3. Prove that #MON2SAT, the problem of counting the number of satisfying assignments to a monotone 2-CNF formula, is #**P**-complete.