CS 535: Complexity Theory, Fall 2023

Homework 3

Due: 11:59PM, Tuesday, September 26, 2023.

Reminder. Homework must be typeset with LaTeX preferred. Make sure you understand the course collaboration and honesty policy before beginning this assignment. Collaboration is permitted, but you must write the solutions *by yourself without assistance*. You must also identify your collaborators. Assignments missing a collaboration statement will not be accepted. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Problem 1 (Decision vs. Optimization)**.** An **NP** minimization problem is specified by a polynomial-time computable *objective function* $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{N}$ and a polynomial p. Given an input $x \in \{0,1\}^*$, the problem is to find a $y \in \{0,1\}^{p(|x|)}$ that minimizes $f(x, y)$, i.e., find a string in $\underset{y \in \{0,1\}^{p(|x|)}}{\operatorname{argmin}} f(x, y)$.

(a) Given a collection of sets $S_1, \ldots, S_m \subseteq [n]$, a hitting set is a set T such that $T \cap S_i \neq \emptyset$ for every $i = 1, \ldots, m$. Describe how the task of finding a minimum-size hitting set can be stated as an **NP**-minimization problem. (2 points)

(b) Consider the decision problem $\mathsf{HS} = \{\langle n, k, S_1, \ldots, S_m \rangle \mid \exists$ a hitting set for $S_1, \ldots, S_m$ of size $\leq k\}$. Show that if $\mathsf{HS} \in \mathbf{P}$, then there is a poly-time algorithm for finding a minimum-size hitting set. (3 points)

(c) Show that $\mathbf{P} = \mathbf{NP}$ if and only if every **NP** minimization problem can be solved in polynomial time. (4 points)

Hint: We showed in class that $\mathbf{P} = \mathbf{NP}$ iff every **NP** search problem can be solved in poly-time. You can use this fact without proof.

Problem 2 (Sparse Languages)**.** A language $L \in \{0,1\}^*$ is *sparse* if there exists a polynomial $p(n)$ such that $|L \cap \{0,1\}^{\leq n}| \leq p(n)$ for every natural number n. That is, for every n, out of the $2^{n+1} - 1$ possible strings of length $\leq n$, only polynomially many (a tiny fraction) are in L. In this problem, you will prove and explore some consequences of *Fortune's Theorem*, which says that the existence of a **coNP**-complete sparse language implies $\mathbf{P} = \mathbf{NP}$.

So let's get started! Suppose there exists a **coNP**-complete language L such that there is a polynomial p such that $|L \cap \{0,1\}^{\leq n}| \leq p(n)$ for every n. Consider the **coNP**-complete language $\mathsf{TAUT} = \{\varphi \mid \varphi$ is a Boolean formula s.t. $\forall x\ \varphi(x) = 1\}$. Since L is **coNP**-complete, there is a poly-time reduction f from TAUT to L, and therefore some polynomial r such that $|f(\varphi)| \leq r(|\varphi|)$ for every formula φ.

(a) Briefly explain why, in order to prove Fortune's Theorem, it suffices to exhibit a poly-time algorithm for TAUT. (1 point)

To give such an algorithm, first consider (as a thought experiment) building the downward self-reduction tree for an input formula. That is, given a Boolean formula $\varphi(x_1, \dots, x_n)$, let $\varphi_0 = \varphi(0, x_2, \dots, x_n)$ and $\varphi_1 = \varphi(1, x_2, \dots, x_n)$. Then $\varphi \in \mathsf{TAUT} \iff \varphi_0 \in \mathsf{TAUT}$ and $\varphi_1 \in \mathsf{TAUT}$. We can recurse on the formulas φ_0 and φ_1, ultimately giving us a tree of depth n such that the original formula is a tautology iff all the leaves evaluate to 1, but this takes exponential time as there are 2^n leaves.

Instead, we're going to prune the tree as we explore it, ensuring that the number of formulas explored at each depth is bounded by a polynomial. We will take this polynomial to be (roughly) $t(n) := p(r(2n + 5))$, for reasons that will hopefully become clear. Suppose that at some level of the tree, we've materialized a collection of "active" formulas $\varphi_0, \dots, \varphi_k$ such that $\varphi \in \mathsf{TAUT} \iff \varphi_i \in \mathsf{TAUT}$ for all $i = 0, 1, \dots, k$. If $k \leq t(n)$, then we recurse on to the next level of the tree. Otherwise, if $k > t(n)$, we will prune the set of active formulas by one.

Here's how to do the pruning. For each $i = 1, \dots, k$ (note that we are not including $i = 0$), let $s_i = f((\varphi_0) \wedge (\varphi_i))$. That is, we apply the reduction f to the formula obtained by taking the logical AND of formulas φ_0 and φ_i.

(b) Explain why if every active formula $\varphi_0, \dots, \varphi_k$ has length at most n, then every string $s_1, \dots, s_k$ has length at most $r(2n + 5)$. (1 point)

Examining the collection of resulting strings $s_1, \dots, s_k$, there are two possible cases:

Case 1: All strings $s_1, \dots, s_k$ are distinct.

Case 2: There exists a pair of strings s_i, s_j where $s_i = s_j$ but $i \neq j$.

(c) Show that in Case 1, we can automatically conclude that φ is NOT a tautology, and therefore we can halt and reject. (2 points)

(d) Show that in Case 2, it is safe to prune, say, the formula φ_i. That is, the original formula φ is a tautology if and only if $\varphi_0, \dots, \varphi_{i-1}, \varphi_{i+1}, \dots, \varphi_k$ are all tautologies. (2 points)

Thus, if $k > t(n)$, our algorithm either halts or reduces the number of active formulas by 1.

(e) Briefly analyze the runtime of this algorithm to conclude that it indeed decides TAUT in polynomial time. (2 points)

(f) A natural question you might ask about TAUT (or SAT or whatever) is whether there is a poly-time algorithm that correctly solves it on *most* instances. More specifically, let's say that an algorithm A *almost solves* TAUT (with one-sided error) if:

- φ is not a tautology $\implies A(\varphi) = 0$. (I.e., A always produces the correct answer of 0 whenever φ is not a tautology.)

- There exists a polynomial $p(n)$ such that for every n, we have $A(\varphi) = 0$ for at most $p(n)$ tautologies φ of length at most n. (I.e., A produces the correct answer of 1 for all but polynomially many tautologies φ.)

Use Fortune's Theorem to show that if there is a poly-time algorithm that almost solves TAUT with one-sided error, then $\mathbf{P} = \mathbf{NP}$. (3 points)

(g) (*Bonus*) Show that the conclusion above holds even for algorithms A with two-sided error. That is, if there is a polynomial $p(n)$ and a poly-time algorithm A such that for every n,

- $A(\varphi) = 0$ for at most $p(n)$ tautologies φ of length n, and
- $A(\varphi) = 1$ for at most $p(n)$ non-tautologies φ of length n,

then $\mathbf{P} = \mathbf{NP}$.