Exercise VI, Computational Complexity 2024

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

Polynomial Hierarchy

1 Recall from Exercise III(5) that there are CNFs φ such that the smallest DNF equivalent to φ is exponentially larger than φ. Inspired by this, consider the following problem:

$$\text{DNF-Size} = \{\langle \varphi, 1^k \rangle : \varphi \text{ is a CNF and there is a DNF equivalent to } \varphi \text{ with } k \text{ terms}\}.$$

Show that DNF-Size $\in \Sigma_2\mathsf{P}$.

Solution: DNF-Size can be recasted as an $\exists\forall$ problem. Indeed $\langle \varphi, 1^k \rangle \in$ DNF-Size if and only if there exists a DNF formula φ' with k terms such that such that for any variable assignment x, $\varphi(x) = \varphi'(x)$. More formally, consider the Turing machine M which on input $(\varphi, 1^k, \varphi', x)$, checks that:

(a) φ is a CNF

(b) φ' is a DNF with at most k terms

(c) $\varphi(x) = \varphi'(x)$

The runtime of M is linear in its input and we have:

$$\langle \varphi, 1^k \rangle \in \text{DNF-Size} \iff \exists \varphi' \, \forall x \, M(\varphi, 1^k, \varphi', x) = 1$$

Observe that the search space for φ' is bounded by $k \cdot |\varphi|$ bits and the one for x is polynomial too. Thus, DNF-Size $\in \Sigma_2\mathsf{P}$.

2 Show that PH does not have a complete problem, unless PH collapses.

(*Hint: Assume some problem $L \in \mathsf{PH}$ is complete. Then it lies in some finite level $L \in \Sigma_i\mathsf{P}$. Proceed to show that $\Pi_i\mathsf{P} = \Sigma_i\mathsf{P}$ and apply a lemma from lecture.*)

Solution: Suppose that some L is PH-complete and fix an $i \in \mathbb{N}$ such that $\mathsf{L} \in \Sigma_i\mathsf{P}$ with verifier $\mathcal{M}$:

$$\forall x \in \{0,1\}^* : x \in L \iff \exists y_1 \, \forall y_2 \, \ldots \, Q_i y_i : \mathcal{M}(x, y_1, y_2, \ldots, y_i) = 1$$

We first show that $\Pi_i\mathsf{P} \subseteq \Sigma_i\mathsf{P}$. To do so, fix some $L' \in \Pi_i\mathsf{P}$ and let $f : \{0,1\}^* \to \{0,1\}^*$ be a poly-time computable reduction from L' to L (f exists because L is PH-complete). We have that:

$$\forall x \in \{0,1\}^* : x \in L' \iff f(x) \in L \iff \exists y_1 \, \forall y_2 \, \ldots \, Q_i y_i : \mathcal{M}(f(x), y_1, y_2, \ldots, y_i) = 1$$

This shows that $L' \in \Sigma_i\mathsf{P}$ with the verifier that applies f to its first input. On the other hand, note that for any $L \in \Sigma_i\mathsf{P}$, $\overline{L} \in \Pi_i\mathsf{P}$ and thus by the argument we just showed, $\overline{L} \in \Sigma_i\mathsf{P}$ and hence $L \in \Pi_i\mathsf{P}$, so that $\Sigma_i\mathsf{P} \subseteq \Pi_i\mathsf{P}$. Thus $\Sigma_i\mathsf{P} = \Pi_i\mathsf{P}$ and we conclude that the PH hierarchy collapses to level i using the argument seen in class.

3 Define the *difference polynomial-time* class DP as consisting of those languages L such that $L = L_1 \cap L_2$ for some $L_1 \in \mathsf{NP}$ and $L_2 \in \mathsf{coNP}$. (Do not confuse DP with $\mathsf{NP} \cap \mathsf{coNP}$!)

(a) Show that $\mathsf{DP} \subseteq \mathsf{P}^{\mathsf{NP}}$.

(b) Show that the following problem is in DP

$$\textsc{UniqueSat} = \{\langle\varphi\rangle : \varphi \text{ is a CNF and it has a unique satisfying assignment}\}.$$

(c) Show that the following problem is DP-complete:

$$\textsc{Sat-Unsat} = \{\langle\varphi, \varphi'\rangle : \varphi \text{ is a satisfiable CNF and } \varphi' \text{ is an unsatisfiable CNF}\}.$$

(d) True or false: If L is NP-complete and L' is coNP-complete, then $L \cap L'$ is DP-complete?

Solution:

(a) Fix some $L = L_1 \cap L_2 \in \mathsf{DP}$ with $L_1, \overline{L_2} \in \mathsf{NP}$. Note that for all $x \in \{0,1\}^*$, $x \in L \iff x \in L_1 \wedge x \notin \overline{L_2}$. This equivalence shows how to build a P^{NP} machine that decides L so that $L \in \mathsf{P}^{\mathsf{NP}}$.

(b) Note that $\textsc{UniqueSat} = L_1 \cap L_2$ where:

$$L_1 = \{\langle\varphi\rangle : \varphi \text{ is a sat CNF}\}$$
$$L_2 = \{\langle\varphi\rangle : \forall x^1, x^2 : x^1 \neq x^2 \implies \neg\varphi(x^1) \vee \neg\varphi(x^2)\}$$

L_1 is basically $\textsc{Sat}$ and thus $L_1 \in \mathsf{NP}$. On the other hand, $\overline{L_2} \in \mathsf{NP}$ because a certificate is simply two different inputs which satisfy φ. Therefore, $\textsc{UniqueSat} \in \mathsf{DP}$.

(c) $\textsc{Sat-Unsat} \in \mathsf{DP}$ as it is the intersection of $\textsc{Sat}$ and $\overline{\textsc{Sat}}$. Now, fix some $L \in \mathsf{DP}$. By assumption $L = L_1 \cap L_2$ with $L_1 \in \mathsf{NP}$ and $L_2 \in \mathsf{coNP}$. Since $\textsc{Sat}$ is NP-complete and $\overline{\textsc{Sat}}$ is coNP-complete, there exists poly-time functions f_1, f_2 such that for all $x \in \{0,1\}^*$:

$$\begin{aligned} x \in L &\iff x \in L_1 \wedge x \in L_2 \\ &\iff f_1(x) \in \textsc{Sat} \wedge f_2(x) \in \overline{\textsc{Sat}} \\ &\iff (f_1(x), f_2(x)) \in \textsc{Sat-Unsat} \end{aligned}$$

Thus, $\textsc{Sat-Unsat}$ is DP-complete

(d) False: Take $L_1 = \textsc{Sat}$ and $L_2 = \overline{\textsc{Sat}}$. Then $L_1 \cap L_2 = \emptyset$, which cannot be DP-complete (no $L \neq \emptyset$ reduces to $\emptyset$).

4 Prove that if $\mathsf{NP} \subseteq \mathsf{TIME}(n^{\log n})$ then $\mathsf{PH} \subseteq \bigcup_{k\in\mathbb{N}} \mathsf{TIME}(n^{\log^k n})$.

Solution: Assuming that $\mathsf{NP} \subseteq \mathsf{TIME}(n^{\log(n)})$, we show by induction on $i \geq 1$ that $\Sigma_i\mathsf{P} \subseteq \bigcup_{k\in\mathbb{N}} \mathsf{TIME}(n^{\log^k(n)})$. The base case corresponds to our hypothesis. Let us now suppose the claim is true for i and let us show it also holds for $i+1$. Fix any language $L \in \Sigma_{i+1}\mathsf{P}$. By definition there exists an efficient Turing machine M as well as a polynomial q such that for all $x \in \{0,1\}^n$:

$$x \in L \iff \exists u_1 \, \forall u_2 \cdots Q_{k+1} u_{i+1} \, M(u_1, u_2, \ldots, u_{i+1}, x) = 1$$

Where all $u_i \in \{0,1\}^{q(|x|)}$. Define now the language $L' \subseteq \{0,1\}^*$ with:

$$L' = \{(x, u_1) : \forall u_2 \cdots Q_{k+1} u_{i+1}\, M(u_1, u_2, \ldots, u_{i+1}, x) = 1\}$$

Observe that $L' \in \Pi_i\mathsf{P}$. Our induction hypothesis says that there exists some $k \in \mathbb{N}$ such that $\Sigma_i\mathsf{P} \subseteq \mathsf{TIME}(n^{\log^k(n)})$. Because deterministic time classes are closed under complement (by reversing the output), we have in turn that $\Pi_i\mathsf{P} \subseteq \mathsf{TIME}(n^{\log^k(n)})$. Thus, let D be a decider for L' that runs in time $t(n) := n^{\log^k(n)}$. Note that for any $x \in \{0,1\}^*$:

$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} : (x, u_1) \in L' \iff \exists u_1 \in \{0,1\}^{q(|x|)} : D(x, u_1)$$

Observe that if D was to run in polynomial time, we would be done because we would have $L \in \mathsf{NP}$. To circumvent the fact that D is non-polynomial, we use a padding argument. Let us define $L'' = \{(x, 1^{t(|x|)}) : x \in L\}$ and observe that for any $y \in \{0,1\}^*$:

$$\begin{aligned}
y \in L'' &\iff \exists x \in \{0,1\}^{\leq t^{-1}(|y|)} : (x, 1^{t(|x|)}) = y \text{ and } x \in L \\
&\iff \exists x \in \{0,1\}^{\leq t^{-1}(|y|)} : (x, 1^{t(|x|)}) = y \text{ and } \exists u_1 \in \{0,1\}^{q(|x|)} : D(x,u) = 1 \\
&\iff \exists x \in \{0,1\}^{\leq t^{-1}(|y|)}\, \exists u_1 \in \{0,1\}^{q(|x|)} : (x, 1^{t(|x|)}) = y \text{ and } D(x,u) = 1 \\
&\iff \exists z \in \{0,1\}^{\leq 2q(t^{-1}(|y|))} : Q(z, y)
\end{aligned}$$

Where z is the concatenation of x and u_1 and Q simply unzips $z = (x, u_1)$ and checks that $(x, 1^{t(|x|)}) = y$ and $D(x,u) = 1$. Note that $|z| \leq 2q(t^{-1}(|y|)) \in \text{poly}(|y|)$. The run-time of Q is dominated by computing D on z - which takes time $t(|z|) \leq t(2q(t^{-1}(|y|))) \in \text{poly}(|y|)$. We have thus shown that $L'' \in \mathsf{NP}$ and so using the initial assumption, there exists a decider P for L'' which runs in time $O(n^{\log(n)})$. Notice that the Turing machine that on input x, prepares the string $y = (x, 1^{t(|x|)})$ and then runs P on y decides correctly L. Furthermore, its run-time is dominated by the computation of P on y which has time $O(t(|x|)^{\log(t(|x|))}) \in O(|x|^{\log^{2k+1}(|x|)})$. This shows that $L \in \bigcup_{k\in\mathbb{N}} \mathsf{TIME}(n^{\log^k(n)})$.

5 (*) Denote by $\mathsf{P}^{\mathsf{NP}[\log n]}$ the class of problems soluble by a poly-time TM that makes at most $O(\log n)$ queries to a SAT oracle. Denote by $\mathsf{P}^{\|\mathsf{NP}}$ the class of problems soluble by a poly-time TM that queries a SAT oracle *in parallel*, that is, the TM first computes deterministically a list of $m = n^{O(1)}$ many SAT-instances, $\varphi_1, \ldots, \varphi_m$, then queries the oracle with all the φ_i at once, receives some string of answers $a \in \{0,1\}^m$, and then produces an output depending on a.

Show that

$$\mathsf{P}^{\mathsf{NP}[\log n]} = \mathsf{P}^{\|\mathsf{NP}}.$$

(*Hint: The inclusion $\mathsf{P}^{\mathsf{NP}[\log n]} \subseteq \mathsf{P}^{\|\mathsf{NP}}$ is the easier one to prove. For the harder inclusion, namely $\mathsf{P}^{\mathsf{NP}[\log n]} \supseteq \mathsf{P}^{\|\mathsf{NP}}$, consider the answer string $a \in \{0,1\}^m$ and first find its Hamming weight (i.e., number of $i \in [m]$ with $a_i = 1$). Then, knowing the Hamming weight, make one more clever NP-oracle query.*)

Solution: We first argue that $\mathsf{P}^{\mathsf{NP}[\log n]} \subseteq \mathsf{P}^{\|\mathsf{NP}}$. Indeed, fix some Turing machine witnessing that some language $L \in \mathsf{P}^{\mathsf{NP}[\log n]}$. We can see it as a decision tree of depth $O(\log(n))$ where each decision is made according to a oracle answer. This can be simulated in the second model by querying in advance the *complete* decision tree and then run it as usual. Note that the complete decision tree has $2^{O(\log(n))} = n^{O(1)}$ nodes and thus only a polynomial number of oracle queries

are necessary.

Fix some $\mathsf{P}^{\|\mathsf{NP}}$ machine A for language L and let us show how to simulate it with a $\mathsf{P}^{\mathsf{NP}[\log(n)]}$ machine B. On input $x \in \{0,1\}^*$, A prepares a list of queries $Q = (x_1, \ldots, x_m)$ with $m \in n^{O(1)}$, receives a list of answers $a = (a_1, \ldots, a_m)$ and decides whether $x \in L$ or not. It is not possible for B to learn the full vector a, but we will see that knowing its Hamming weight is sufficient. Let k be the Hamming weight of a, i.e. its number of 1-entries.

Note that for any $q \in \mathbb{N}$, the problem of deciding whether $k \geq q$ is in NP. Indeed, it amounts to finding q certificates for some q-subset of queries in Q. Thus, using binary search with queries of the form "$k \geq q$", B can pinpoint the value of k with $\log(m) = O(\log(n))$ oracle queries.

After figuring out k, we make one more NP oracle query to decide whether $x \in L$. Here is the oracle query:

> *Does there exist an $a' \in \{0,1\}^m$ of Hamming weight k such that for every $i \in [m]$ with $a'_i = 1$ the i-th query in Q has a positive answer, and moreover, A accepts x if the oracle answers are a'.*

A certificate for this query consists of a' and the k certificates for $a'_i = 1$; verifying it amounts to checking each certificate, and simulating A with oracle answers a'.

Thus, B performs $O(\log(n))$ oracle queries and decides L in poly-time.