Exercise IV, Computational Complexity 2025

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 :).

Diagonalisation and Oracles

1 Show that the problem

$$\textsc{ExpSim} = \{\langle M, x, 1^t\rangle : M \text{ is a TM, and } M(x) \text{ outputs 1 in } \leq 2^t \text{ steps}\}$$

is complete (under poly-time reductions) for the exponential-time class

$$\mathsf{EXP} = \bigcup_{k\in\mathbb{N}} \mathsf{TIME}(2^{n^k}).$$

Namely, show that $\textsc{ExpSim} \in \mathsf{EXP}$ and that for every $A \in \mathsf{EXP}$ we have $A \leq_p \textsc{ExpSim}$.

Solution: Let D be the Turing machine that given a tuple $(M, x, 1^t)$, runs M on x for at most 2^t steps using the universal Turing machine. If M accepts during the run, D accepts and else rejects (if M rejected or didn't finish in time). Note that D decides $\textsc{ExpSim}$ and runs in time exponential (the overhead incurred by the universal Turing machine is only polynomial) and hence $\textsc{ExpSim} \in \mathsf{EXP}$.

To show completeness, fix any $A \in \mathsf{EXP}$. By definition, there exists some Turing machine M that decides A in time $\leq 2^{n^k}$ for a fixed k. Consider the function $f : \{0,1\}^* \to \{0,1\}^*$ defined by $f(x) = \langle M, x, 1^{|x|^k}\rangle$. Note that f is poly-time computable and that for each $x \in \{0,1\}^n$:

$$x \in A \iff M(x) = 1 \iff M(x) \text{ outputs 1 in } \leq 2^{|x|^k} \text{ steps} \iff f(x) \in \textsc{ExpSim}$$

2 Show that $\mathsf{TIME}(n)$ is not closed under poly-time reductions. That is, exhibit languages A and B such that $A \leq_p B$, $B \in \mathsf{TIME}(n)$, but $A \notin \mathsf{TIME}(n)$.

(Hint: Use the Time Hierarchy theorem.)

Solution: Pick any $A \in \mathsf{TIME}(n^3) \setminus \mathsf{TIME}(n)$ (the existence of such a language is guaranteed by the time hierarchy theorem) and let $B = \{1\}$. Note that $A \leq_p B$, $B \in \mathsf{TIME}(n)$ but $A \notin \mathsf{TIME}(n^3)$

3 Show that the following problem is in $\mathsf{P}^{\mathrm{SAT}}$:

$$\textsc{MaxIS} = \{\langle G, k\rangle : G \text{ is a graph whose largest independent set is of size exactly } k\}$$

Do you think $\textsc{MaxIS}$ is in NP? What about coNP?

Solution: Let $\text{IS} = \{\langle G, k\rangle : G \text{ is a graph which contains an independent set of size } k\}$ be the independent set problem (recall it is NP-complete) and observe that:

$$\langle G, k\rangle \in \textsc{MaxIS} \iff \langle G, k\rangle \in \text{IS and } \langle G, k+1\rangle \notin \text{IS}$$

Since $\text{IS} \in \mathsf{NP}$ and SAT is complete for NP, there exists some function f that reduces IS to SAT. So it is enough to query the SAT oracle with $f(\langle G, k\rangle)$ and $f(\langle G, k+1\rangle)$ and use the above equivalence to decide whether $\langle G, k\rangle \in \textsc{MaxIS}$.

We show that it is unlikely that $\textsc{MaxIS} \in \mathsf{NP}$ (a similar argument shows that it is also unlikely $\textsc{MaxIS} \in \mathsf{coNP}$). Suppose that $\textsc{MaxIS} \in \mathsf{NP}$ and let (V, p) be a certificate system that witnesses this. Then we have:

$$\begin{aligned}\langle G, k\rangle \in \overline{\text{IS}} &\iff \exists q < k : \langle G, q\rangle \in \textsc{MaxIS} \\ &\iff \exists q < k\, \exists c \in \{0,1\}^{p(|\langle G,q\rangle|)} : V(\langle G, q\rangle, c) \text{ accepts}\end{aligned}$$

The above chain implies that there is an efficient certificate system for $\overline{\text{IS}}$ (namely q and c) and thus $\overline{\text{IS}} \in \mathsf{NP}$. Since IS is NP-complete, this further means that $\mathsf{NP} = \mathsf{coNP}$ (an unlikely collapse).

4 Show that the NP vs. coNP question relativises both ways. That is,

(a) There is an oracle A relative to which $\mathsf{NP}^A = \mathsf{coNP}^A$.
(b) There is an oracle B relative to which $\mathsf{NP}^B \neq \mathsf{coNP}^B$. Namely, consider the language

$$L_B = \{1^n : \forall x \in \{0,1\}^n, x \in B\}$$

and show that

- $L_B \in \mathsf{coNP}^B$ for every B.
- (*) $L_B \notin \mathsf{NP}^B$ for some B.

(Hint: Fix a poly-time nondeterministic oracle machine M^B. Can you define B for strings of large length n so that M^B reports the wrong answer for "$1^n \in L_B$"?)

Solution: A co-non-deterministic Turing machine is a non-deterministic Turing machine that accepts if and only if **all** the non-deterministic paths accepts. Using this definition, we can define coNP as the set of all languages decided by a polynomial-time co-non-deterministic Turing machine. We use this definition throughout the exercise.

(a) We pick $A = \textsc{ExpSim}$. Note that $\mathsf{NP}^A \subseteq \mathsf{EXP}$. Indeed, if $L \in \mathsf{NP}^A$, then there exists a non-deterministic polynomial-time Turing machine N with oracle A which decides L. Note that a polynomial number of non-deterministic choice can be simulated classically in exponential time and that queries to A can by definition be simulated in exponential time too. Thus, $L \in \mathsf{EXP}$. The same argument shows $\mathsf{coNP}^A \subseteq \mathsf{EXP}$ (instead of accepting if one non-deterministic path accepts, we accept if all paths accept).

On the other hand, $\textsc{ExpSim} \in \mathsf{P}^{\textsc{ExpSim}}$ and since ExpSim is complete for EXP (see problem 1), we have that $\mathsf{EXP} \subseteq \mathsf{P}^{\textsc{ExpSim}}$. Because $\mathsf{P} \subseteq \mathsf{NP}, \mathsf{P}$, we have $\mathsf{NP}^A = \mathsf{coNP}^A = \mathsf{EXP}$, as desired.

(b) We have $L_B = \{1^n : \forall x \in \{0,1\}^n, x \in B\} \in \mathsf{coNP}^B$ for every B since L_B can be decided by a co-non-deterministic machine that, on input 1^n, nondeterministically guesses $x \in \{0,1\}^n$ and queries B to check that $x \in B$; this machine accepts iff all computations accept.

The more challenging part is to construct B such that $L_B \notin \mathsf{NP}^B$. Let $\mathcal{N}$ be the set of all non-deterministic Turing machines which run in polynomial time while having oracle access. This set is countable and hence there exists a sequence $(N_i)_{i\in\mathbb{N}}$ of non-deterministic Turing machines that contains all of $\mathcal{N}$ and furthermore, each $N \in \mathcal{N}$ appears infinitely many times.

We will construct B dynamically, by specifying what elements of $\{0,1\}^n$ it contains for increasingly larger value of n. To do so, it might be easier to think of B as a *function* $B : \{0,1\}^* \to \{0,1,*\}$ where $B(x) = 0$ if $x \notin B$, $B(x) = 1$ if $x \in B$ and $B(x) = *$ if the membership is not yet fixed. We thus start with $B(x) := *$ for all $x \in \{0,1\}^*$ and fix membership values on the fly.

For increasing values of $i = 0, 1, 2, \ldots$, we set B in such a way that N_i fails to decide L_B. Suppose we are at step i and let n be the size of the smallest $x \in \{0,1\}^*$ such that $B(x) = *$. Run N_i on input 1^n with oracle B. Whenever a membership query is made with $B(x) = *$, set $B(x) := 1$. When the execution is over, if N_i made queries up to length k, set $B(x) := 1$ for all $x \in \{0,1\}^{\leq k}$ with $B(x) = *$. This way, all values of B up to length k are fixed.

If N_i rejected 1^n with oracle B, then nothing needs to be done, indeed, all strings $x \in \{0,1\}^n$ have $B(x) = 1$ so that $1^n \in L_B$ while $N_i^B(1^n) = 0$ thus ruling out N_i as a decider for L_B.

If N_i accepted 1^n with oracle B, it means that at least one non-deterministic path accepted 1^n. Let $\Gamma = \{x_1, \ldots, x_p\}$ be the set of membership queries made to B on this path and $\Gamma' \subseteq \Gamma$ the subset of queries with length exactly n. Since N_i runs in time polynomial, it means that $|\Gamma'| \leq n^{O(1)}$ and thus there exists $x^\star \in \{0,1\}^n \setminus \Gamma'$. Re-setting the value $B(x^\star) := 0$ still makes the path accept (as it never queried $x^\star$ in the first place) so that $N_i^B(1^n) = 1$ while $1^n \notin L_B$.

Remark: it is possible that for small values of n, there exists no such $x^\star$. Indeed, the run-time of N_i might be n^{100} which is greater than 2^n for small enough n. This is not a problem: N_i appears infinitely many times in the sequence and thus at some point it must hold that $n^{100} \leq 2^n$ causing an input to be wrongly decided by N_i. This hints that each $N \in \mathcal{N}$ fails to decide L_B on an infinite number of inputs.

5 (*) Call a language $A \subseteq \{0,1\}^*$ *sparse* if it contains only polynomially many strings of length n, that is, there is some $k \in \mathbb{N}$ such that $|A \cap \{0,1\}^n| \leq n^k$ for all n. Show that

$$L \in \mathsf{P/poly} \quad \text{iff} \quad L \in \mathsf{P}^A \text{ for some sparse } A.$$

Solution: Suppose first that $L \in \mathsf{P/poly}$. This implies that there exists a family of circuits $\{C_n\}_{n\in\mathbb{N}}$ with $|C_n| \leq n^k$ for some $k \in \mathbb{N}$ and:

$$C_{|x|}(x) = L(x) \qquad \text{for each } x \in \{0,1\}^*$$

For each $n \in \mathbb{N}$, let $(b_1^n, b_2^n, \ldots, b_{p_n}^n)$ be the binary encoding of C_n with $p_n \leq \text{poly}(n)$. The idea will be to "store" each C_n in A in such a way that a polynomial time Turing machine can reconstitute any C_n at will and then execute it locally. Since an oracle can only give yes/no

answer (instead of a full circuit in one go), we will need to query the circuit bit by bit. Formally, we define A as follows:

$$A = \{\langle 1^n, i, b_1^n, \ldots, b_i^n \rangle : n \in \mathbb{N} \text{ and } i \in [p_n]\}$$

Let us describe how to fetch the description of C_n by querying A. The answer to $\langle 1^n, 1, 1\rangle \in A$ and $\langle 1^n, 1, 0\rangle \in A$ determines whether the first bit b_1^n is zero or one. Then, the answer to $\langle 1^n, 2, b_1^n, 1\rangle \in A$ and $\langle 1^n, 2, b_1^n, 0\rangle \in A$ determines whether the second bit b_2^n is zero or one. This process can be carried on until both queries return false in which case we know the description is complete.

This argument shows that $L \in \mathsf{P}^A$. Indeed, on input $x \in \{0,\, 1\}^k$, one first fetch the circuit in poly-time and then execute it (again in poly-time) on x to get $L(x)$. It remains to show that A is sparse. To see why this is true, fix some string size q and let us bound the size of $A^q = \{x \in A : |x| = q\}$. Note that no description of a circuit $C_{>q}$ appears in A^q. So that at most q circuits are represented in A^q. Finally, a circuit $C_{\leq q}$ only incurs poly(q) strings to A, thus $|A^q| \leq q \cdot \text{poly(q)}$ and so A is sparse.

To prove the other direction fix some $L \in \mathsf{P}^A$ with A sparse. Let D be a poly-time Turing machine that decides L with oracle access to A. Note that for inputs of length n, D can only ask membership queries for strings of size poly(n). Hence when extracting the circuit C_n from D (like in the $\mathsf{P} \subseteq \mathsf{P/poly}$ proof) we only need to hard-code the strings of A up to size poly(n). This only incurs a polynomial blow-up to the size of C_n because:

$$|A^{\leq \text{poly}(n)}| \leq \sum_{i=1}^{\text{poly}(n)} i^k \leq \text{poly}(n)$$

Since we have shown the existence of a family of polynomial-sized circuits deciding L, we have $L \in \mathsf{P/poly}$.