Homework 2, Computational Complexity 2024

The deadline is 23:59 on Wednesday 6 November. Please submit your solutions on Moodle. Typing your solutions using LaTeX is strongly encouraged. The problems are meant to be worked on in groups of 2–3 students. Please submit only one writeup per team. You are strongly encouraged to solve these problems by yourself. If you must, you may use books or online resources to help solve homework problems, but you must credit all such sources in your writeup and you must never copy material verbatim.

1 Prove that $\mathsf{NP}^{\mathrm{SAT}} = \Sigma_2\mathsf{P}$.

Solution: We first prove the easier direction that $\Sigma_2\mathsf{P} \subseteq \mathsf{NP}^{\mathrm{SAT}}$. Fix any $L \in \Sigma_2\mathsf{P}$. By definition, there exist a polynomial time deterministic Turing machine $M(x, y, z)$ such that $x \in L \Leftrightarrow \exists y, \forall z, M(x, y, z) = 1$. Now we devise a oracle Turing machine $\hat{M}^{\mathrm{SAT}}$ as follows:

1. Given x, y, construct a formula $\phi_{x,y}$ such that $\phi_{x,y}(z) = \neg M(x, y, z)$ for all z.
2. Return $\neg\mathrm{SAT}(\phi_{x,y})$ by calling the oracle.

It is not hard to see the above process can be implemented in polynomial time. Moreover,

$$x \in L \Leftrightarrow \exists y, \forall z, M(x, y, z) = 1 \Leftrightarrow \exists y, \forall z, \phi_{x,y}(z) = 0 \Leftrightarrow \exists y, \hat{M}^{\mathrm{SAT}}(x, y) = 1.$$

Thus $L \in \mathsf{NP}^{\mathrm{SAT}}$, which implies that $L \in \Sigma_2\mathsf{P}$.

Now we turn our attention to the direction $\mathsf{NP}^{\mathrm{SAT}} \subseteq \Sigma_2\mathsf{P}$. Fix any $L \in \mathsf{NP}^{\mathrm{SAT}}$. By definition, L can be efficiently computed by a nondeterministic oracle Turing machine. Equivalently, there exists a polynomial-time deterministic oracle Turing machine M^{SAT} such that $x \in L$ iff. $M^{\mathrm{SAT}}(x, w) = 1$ for some w of polynomial length.

To get rid of the SAT oracle, we use the power of nondeterminisim to guess the query outcomes. Specifically, let $a = (a_1, \ldots, a_T) \in \{0, 1\}^T$, where $T = T(n)$ is the maximum running time of T on any n-bit string. We simulate M^{SAT} without calling the oracle by assuming that the i-th call returns a_i. The next step is to verify that $a_1, \ldots, a_T$ are indeed the query outcomes, in words, $\mathrm{SAT}(\phi_i) = a_i$ for all $i \in [T]$, where ϕ_i is the i-th instance fed into the oracle assuming that the previous outcomes are $a_1, \ldots, a_{i-1}$. To this end, we create two certificates $y_i, z_i \in \{0, 1\}^{n_i}$ for each $i \in [T]$, where n_i is the number of input bits to ϕ_i. Then we verify that $\exists y_i \in \{0, 1\}^{n_i}, \phi_i(y_i) = 1$ for all $a_i = 1$ and $\forall z_i \in \{0, 1\}^{n_i}, \phi_i(z_i) = 0$ for all $a_i = 0$.

Let $\hat{M}$ denote the above deterministic Turing machine which simulates M^{SAT} with treating a as query outcomes and then verifies that a is indeed the true outcomes. It is clear that $\hat{M}$ runs in polynomial-time. Moreover, it follows that

$$x \in L \Leftrightarrow \exists w, M^{\mathrm{SAT}}(x, w) = 1 \Leftrightarrow \exists w, a, y, \forall z, \hat{M}(x, w, a, y, z) = 1,$$

where $y = (y_1, \ldots, y_T), z = (z_1, \ldots, z_T)$. Thus $L \in \Sigma_2\mathsf{P}$. We conclude that $\mathsf{NP}^{\mathrm{SAT}} \subseteq \Sigma_2\mathsf{P}$.

2 Construct an oracle A such that $\mathsf{RP}^A \neq \mathsf{coRP}^A$. Namely, consider the class of oracles

$$\mathcal{A} \coloneqq \left\{ A \subseteq \{0,1\}^* : \forall n,\ |A \cap \{0,1\}^n|/2^n \in \{\tfrac{1}{2}, 0\} \right\}$$

and the associated language $L_A = \left\{1^n : |A \cap \{0,1\}^n|/2^n = \frac{1}{2}\right\}$. Show that

(i) $L_A \in \mathsf{RP}^A$ for every $A \in \mathcal{A}$.
(ii) $L_A \notin \mathsf{coRP}^A$ for some $A \in \mathcal{A}$.

Solution: To show $L_A \in \mathsf{RP}^A$, we devise the following simple algorithm B: Given input $x \in \{0,1\}^n$, if $x \neq 0^n$, simply reject. Otherwise, sample y uniformly from $\{0,1\}^n$ and return $A(y)$.

The running time of B is clearly linear in n. The correctness follows from the observation that

- If $1^n \in L_A$, then $\Pr[B(1^n) = 1] = \Pr_{x \sim \{0,1\}^n}[A(x) = 1] = 1/2$.
- If $1^n \notin L_A$, then $\Pr[B(1^n) = 0] = \Pr_{x \sim \{0,1\}^n}[A(x) = 0] = 1$.

For (ii), we use diagonalisation method. Let $\mathcal{M}$ denote the set of all polynomial-time probabilistic oracle Turing machine. Since $\mathcal{M}$ is countable, there exists a sequence $(M_i)_{i \in \mathbb{N}}$ that contains every machine in $\mathcal{M}$. We iteratively construct A to fool every M_i.

For the ease of notation, we will think of A as a function $A : \{0,1\}^* \to \{0,1,\star\}$, where $\star$ means "undetermined". In the beginning, we have that $A(x) = \star$ for all x. Then we enumerate $i = 1, 2, 3, \ldots$, each time we can find a large enough n_i such that

1. $T_{M_i}(n_i) < 2^{n_i - 1}$, where $T_{M_i}(n)$ is the maximum running time of M_i on 1^n.
2. $A(x) = \star$ for all $x \in \{0,1\}^{n_i}$.

Such n_i is guaranteed to exist since: (1) $T_{M_i}(n)$ is bounded by a polynomial of n; (2) The number of x for which $A(x) \neq \star$ is finite, which we will see later.

Now consider running $M_i^{A'}$ on 1^{n_i}, where $A'(x) = \begin{cases} 1 & A(x) = 1 \\ 0 & \text{otherwise} \end{cases}$. Let X_i denote the set of queries been made during the execution over all randomness seeds. We update $A(x) \leftarrow 0$ for all $x \in X_i$ such that $A(x) = \star$. Moreover, we update $A(x) \leftarrow 0$ for all $x \in \{0,1\}^{n_i}$. We consider the following two cases based on the output of $M_i^{A'}(1_{n_i})$:

- $\Pr[M_i^{A'}(1^n) = 1] = 1$. In which case, since $A(x) = A'(x)$ for all $x \in X_i$, we also have $\Pr[M_i^A(1^n) = 1] = 1$. On the other hand, $1^{n_i} \notin L_A$ since $A(x) = 0$ for all $x \in \{0,1\}^{n_i}$. Thus M_i^A does not compute L_A.

- $\Pr[M_i^{A'}(1^n) = 1] < 1$. For convenience, we explicitly spell out the randomness seed r to write $M_i^A(x)$ as $M_i^A(x, r)$. In this case, we can fix some r such that $M_i(A')(1^{n_i}) = 0$. Let X_i^r be the set of queries been made during the execution of $M_i^{A'}(x, r)$. By assumption, $|X_i^r| < 2^{n_i - 1}$. In particular, we can find $\hat{X} \subseteq \{0,1\}^{n_i}$ of size $|\hat{X}| = 2^{n_i - 1}$ such that $\hat{X} \cap X_i^r = \emptyset$. Then we update $A(x) \leftarrow 1$ for all $x \in \hat{X}$.

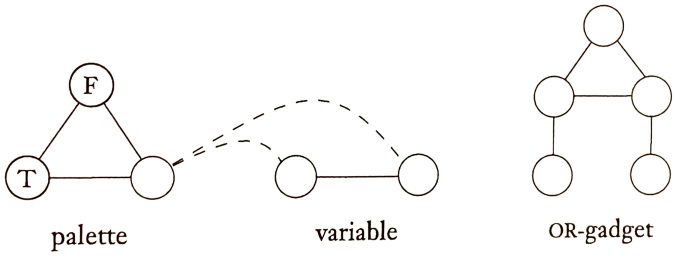
  Notice that $A(x) = A'(x)$ for all $x \in X_i^r$, we have that $M_i^A(1^{n_i}, r) = 0$. On the other hand, $1^{n_i} \in L_A$ since $A(x) = 1$ for exactly half of the elements $x \in \{0,1\}^n$. As a consequence, M_i^A does not compute L_A.

Since at each stage, we only update $A(x)$ for $x \in \{0,1\}^{n_i} \cup X_i$, which is a finite set, (2) holds as desired. Moreover, the modification to A in the i-th iteration does not change the value of $\Pr[M_j^A(1^{n_j}) = 1]$ for all $j < i$ due to our choice of n_i. We can finally conclude that the language $A \in \mathcal{A}$ we construct satisfies that $L_A \notin \mathsf{coRP}^A$.

3 A *3-colouring* of a graph $G = (V, E)$ is an assignment $c\colon V \to \{1, 2, 3\}$ such that no edge $\{u, v\} \in E$ is *monochromatic*, that is, $c(u) = c(v)$. Consider the following *3-colouring game* played by Alice and Bob on a graph G where $V = \{1, \ldots, n\}$. In each round of the game, we have a partial 3-colouring $c\colon V \to \{1, 2, 3, *\}$. Initially, $c(v) = *$ for all v. In round $i = 1, \ldots, n$, if i is odd (resp. even), then Alice (resp. Bob) chooses a colour $k \in \{1, 2, 3\}$ and we update the partial colouring by $c(i) \coloneqq k$. The player who first creates a monochromatic edge ($\{u, v\} \in E$ such that $c(u) = c(v) \neq *$) loses the game. (If after n rounds there is no monochromatic edge, Alice wins.) Prove that the following problem is PSPACE-complete

$$\text{3-ColourGame} \coloneqq \big\{\langle G\rangle : G \text{ is a graph such that Alice has a winning strategy for the 3-colouring game on } G\big\}.$$

(*Hint: Deciding whether a graph is 3-colourable is* NP*-complete. It might help you to first figure out how to prove this. For example, one can reduce from SAT using the following three subgraphs. Source: p. 325 in Sipser's textbook*)



Solution: We first give an algorithm for 3-ColourGame which runs in linear space: For each i and $c = (c_1, \ldots, c_{i-1}) \in [3]^{i-1}$ where c is a valid colouring on vertices $\{1, 2, \ldots, i-1\}$, we define $W(i, c) \in \{0, 1\}$ which equals to 1 if and only if the current player (Alice if i is odd and Bob if i is even) has a wining strategy. We have the following recurrences for W:

$$W(i, c) = \begin{cases} \mathbb{1}[n \text{ is even}] & i = n + 1 \\ \bigvee_{j \in [3]} \left(c' = (c, j) \text{ is valid on } \{1, \ldots, i\} \wedge \neg W(i+1, c')\right) & i \leq n \end{cases}. \tag{1}$$

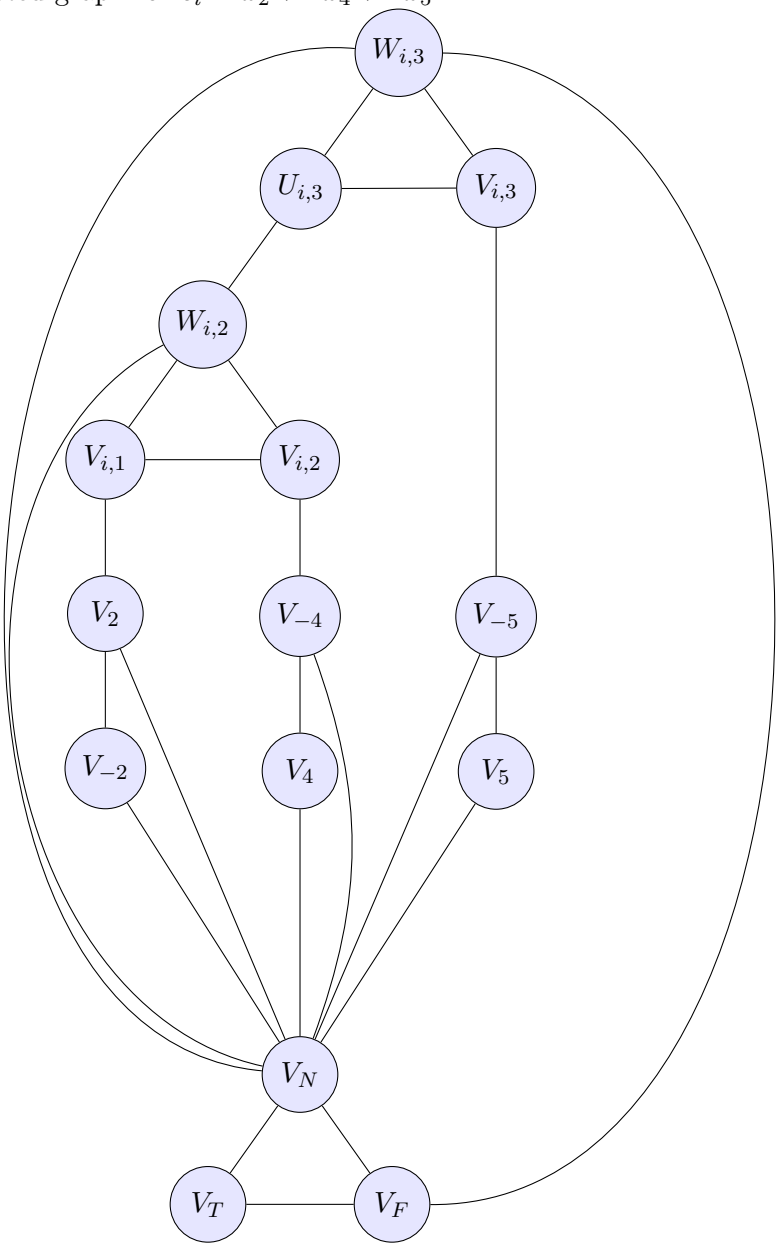
Our goal is to compute $W(1, ())$, which equals to 1 iff Alice has a winning strategy for the whole game. By (1), this can be simply done by a recursion. Since the recursion has depth at most $n + 1$, and we only need to store the partial colour c for the current state (as that for previous states is just a prefix), we conclude that $W(1, ())$ can be computed in $O(n)$ space, as desired.

To prove that 3-ColourGame is PSPACE-hard, we reduce TQBF to 3-ColourGame.

Given a TQBF instance $\forall x_1, \exists x_2, \forall x_3, \ldots, \exists x_n, \phi(x)$ (assume n is even for simplicity) where ϕ is a CNF. We construct a graph G in an almost same way as we do for reducing SAT to 3-Colourable. Specifically, we first construct a triangle as a palette, where we denote the three vertices by v_T, v_F, v_N respectively. Then for each variable x_i, we introduce two vertices v_i, v_{-i}, and construct a triangle with vertices v_i, v_{-i}, v_N. There are two ways to colour v_i and v_{-i}, and they have the meaning of $x_i = 1$ and $x_i = 0$ respectively.

For each clause $c_i = \ell_{i,1} \vee \cdots \vee \ell_{i,k}$, we "compute" the value of c_i by concatenating OR-gadgets given in the hint. Specifically, for each literal $\ell_{i,j}$, we identify $\ell_{i,j}$ with v_p if $\ell_{i,j} = x_p$ has positive sign and identify $\ell_{i,j} = v_{-p}$ if $\ell_{i,j} = \neg x_p$ has negative sign. We introduce a new vertex $v_{i,j}$ and add an edge $(v_{i,j}, \ell_{i,j})$. Then for each $2 \leq j \leq k$, we introduce a vertex $w_{i,j}$ and link it with v_N. For each $3 \leq j \leq k$, we introduce a vertex $u_{i,j}$ and build a triangle with vertices $u_{i,j}, v_{i,j}, w_{i,j}$. We also construct a triangle with vertices $v_{i,1}, v_{i,2}, w_{i,2}$ and link $u_{i,j}$ with $w_{i,j-1}$ for $j \geq 3$. Finally,

we link $w_{i,k}$ with v_F to ensure that the clause is satisfied. See the following figure for an example of the constructed graph for $c_i = x_2 \vee \neg x_4 \vee \neg x_5$.



Now consider the scenario that we have coloured $v_T, v_F, v_N, v_1, \ldots, v_n, v_{-1}, \ldots, v_{-n}$. It is easy to see there is valid colouring of the vertices in the OR-gadgets for c_i if and only if at least one of $\ell_{i,j}$ has the same colour as v_T.

Next, we need to specify an order of the vertices by giving a sequence S: S starts with v_T, v_F, v_N, then followed by $v_1, \ldots, v_n, v_{-1}, \ldots, v_{-n}$. Note that once the colours of $v_T, v_F, v_N, v_1, \ldots, v_n$ are determined, there exists a unique way to colour $v_{-1}, \ldots, v_{-n}$ Finally, we add vertices in the OR-gadgets to S in an arbitrary order. To ensure that all the vertices in OR-gadgets are coloured by Alice, we separate each pair of adjacent vertices which both belong to OR-gadgets by adding a dummy **isolated** vertex between them.

The reduction can be clearly realized in polynomial time. It remains to prove the correctness.

If $A \in$ TQBF, Alice has a strategy to choose $x_2, x_4, \ldots, x_n$ so that no matter how Bob chooses $x_1, x_3, \ldots, x_{n-1}$, $\phi(x)$ is satisfied. In words, $c_i(x) = 1$ for all i. By our construction,

this implies that Alice has a strategy to colour $v_2, v_4, \ldots, v_n$ so that no matter how Bob colours $v_1, v_3, \ldots, v_{n-1}$, the remaining graph is still 3-colourable. As a consequence, Alices win the game.

On the other hand, if $A \notin$ TQBF, Bob has a strategy to choose $x_1, x_3, \ldots, x_{n-1}$ so that no matter how Alice chooses $x_2, x_4, \ldots, x_n$, $\phi(x)$ is unsatisfied. In words, $c_i(x) = 0$ for some i. By our construction, this implies that Bob has a strategy to colour $v_1, v_3, \ldots, v_{n-1}$ so that no matter how Alice colours $v_2, v_4, \ldots, v_n$, the remaining graph is not 3-colourable. As a consequence, Bob wins the game.

In conclusion, 3-ColourGame is PSPACE-hard. As a consequence, 3-ColourGame is PSPACE-complete.