

Computational complexity – Homework

Marc CHEVALIER

November 3, 2014

1 NP-Hardness

1.1 Halting problem

Let be φ an instance of SAT problem. We denote by n the number of variables.

Let be M a TURING machine which tests in a cycle all the 2^n possible assignments of the previous formula : when M has tested all assignments, it starts again. This machine halts if and only if φ is satisfiable. This reduction is polynomial, therefore $SAT \leq_p HALT$, ie. $HALT$ is NP-hard since SAT is NP-hard.

$HALT$ is not NP-complete otherwise it was decidable by a TURING-machine, but $HALT$ is undecidable.

Theorem 1. $HALT$ is NP-hard but not NP-complete

1.2 TQBF

All instance of SAT problem is an instance of TQBF. Without transformation, we have a polynomial reduction, ie. $SAT \leq_p TQBF$ so TQBF is NP-hard.

This problem is known for being PSPACE-complete. It's maybe NP-complete.

Theorem 2. TQBF is NP-hard.

1.3 NAE – 3 – SAT

Lemma 3. $3 - SAT \leq_p NAE - 4 - SAT$

Proof. We define a single "reference variable" z for the entire NAE – SAT formula. Then we represent each variable x_i in the 3 – SAT formula F with a variable y_i in the NAE – SAT formula F_1 , where x_i is true or false if $y_i \neq z$ or $y_i = z$ respectively. Then we can represent a 3 – SAT clause $(x_i \vee x_j \vee x_k)$ with a NAE – 4 – SAT clause $(y_i \vee y_j \vee y_k \vee z)$, since this clause is satisfied if and only if at least one of y_i is different from z , ie., if at least one of the x_i is true. We can similarly represent x_i by y_i . \square

Lemma 4. $NAE - 4 - SAT \leq_p NAE - 3 - SAT$

Proof. To reduce NAE – 4 – SAT to NAE – 3 – SAT, we use variables w to break each 4-variable clause into two 3-variable clause:

$$y_i \vee y_j \vee y_k \vee z = (y_i \vee y_j \vee w) \wedge (w \vee y_k \vee z)$$

That works exactly by the same way as the proof of $SAT \leq_p 3 - SAT$. \square

Proposition 5. $SAT \leq_p NAE - 3 - SAT$.

Corollary 6. $NAE - 3 - SAT$ is NP-hard.

Theorem 7. $NAE - 3 - SAT$ is NP-complete

Proof. $NAE - 3 - SAT$ is clearly NP: a valid assignment is a sufficient witness. We can check in polynomial time if this assignment makes the formula true and if the "not all equal" condition is satisfied. Then $NAE - 3 - SAT$ is NP-complete. \square

1.4 MAXCUT

Let F be an instance of $NAE - 3 - SAT$

$$F = \bigwedge_{i=1}^m C_i$$

We produce a graph $G = (V, E)$ which has a vertex for each literal of F . There is a edge between two vertices if there is a clause which contains this two literals. So each clause is described by a triangle. Moreover, we add $|F|_{x_i}$ (the number of occurrences of x_i in F) edges between x_i and $\neg x_i$. The size of the cut we search is at least $5m$.

If we have an assignment of the $NAE - 3 - SAT$, we take the vertices which are true in S and the other in \bar{S} . So, we have $2m$ from the triangles due to the clauses and $3m$ from the edges between all pair $(x_i, \neg x_i)$.

Reciprocally, if we have a cut of size $\geq 5m$.

If we have no pair $(x_i, \neg x_i)$ on the same side, we have a valid assignment.

If there is a such pair, we can move one of them on the opposite side without decreasing the size of the cut. Let n_i the number of edges between x_i and $\neg x_i$. We note a the number of edges which x_i is an extremity and which the other is in the opposite side. We note b the number of edges between b and a vertex of the opposite size. We know that $a + b \leq 2n_i$. If we move x_i in the opposite size, the cut gains $n_i - a$ edges. If $\neg x_i$ go to the opposite side, it gains $n_i - b$. $\max(n_i - a, n_i - b) \geq 0$, so we can move one of these vertices to the opposite side without decreasing the size of the cut. We redo this transformation until we reach a cut of the first case (at most m times).

We proved that $NAE - 3 - SAT \leq_p \text{MAXCUT}$.

Moreover, MAXCUT is in NP. Indeed, a witness is the list of the vertices of S (or \bar{S}). The size is actually polynomial with respect of the size of G and we can check the solution in a polynomial time : we check easily that the cut has a size $\geq k$ in a quadratic time.

So, $\text{MAXCUT} \in \text{NP}$.

Theorem 8. MAXCUT is NP-complete

2 Reductions

Proposition 9. The set of the recursive languages is closed under polynomial-time KARP reduction.

Proof. That is obvious, isn't it ? \square

Proposition 10. DLOGTIME is not closed under polynomial-time KARP reduction.

Proof. Let $\mathcal{L} \in \text{DLOGTIME}$ such that $|\mathcal{L}| \geq 2$. For all $\mathcal{L}' \in \text{PTIME} \setminus \text{DLOGTIME}$ (there is at least one), there is a polynomial time function $g : \mathcal{L}' \rightarrow \{0, 1\}$, $x \in \mathcal{L}' \Leftrightarrow g(x) = 1$. Let φ a bijection between $\{0, 1\}$ and $\{a, b\}$ where $(a, b) \in \mathcal{L} \times \overline{\mathcal{L}}$. We have $x \in \mathcal{L}' \Leftrightarrow \varphi \circ g(x) = \varphi(a)$ ie. $x \in \mathcal{L}' \Leftrightarrow \varphi \circ g(x) \in \mathcal{L}$.

So, DLOGTIME is not close under polynomial-time KARP reduction. \square

Proposition 11. TAUTOLOGY is NP-hard.

Proof. Let M a TURING machine with oracle TAUTOLOGY . We will make that machine decide SAT in polynomial time.

This machine take a formula φ compute the negation $\neg\varphi$ and test with its oracle if $\neg\varphi$ is a tautology. If it is not, there is a assignment which make $\neg\varphi$ false, so this assignment make φ true and φ is satisfiable. If $\neg\varphi$ is a tautology, φ is not satisfiable.

So M decide SAT and TAUTOLOGY is NP-hard. \square

3 Difference of NP problems

Proposition 12. EXACTINDSET is in DP .

Proof. Let A be the set of all pairs (G, k) such that G has an independent set of size at least k , and let B be the set of all pairs (G, k) such G has a independent set of size at least $k + 1$. Then $\text{EXACTINDSET} = A \setminus B$ and A is in NP and B is in NP. Hence by definition of DP , EXACTINDSET is in DP . \square

Proposition 13. $\forall L \in DP, L$ is polynomial-time reducible to EXACTINDSET .

Proof.

Lemma 14. $\text{INDSET} \geq_p 3 - \text{SAT}$

Proof. Suppose we have an instance F of $3 - \text{SAT}$ problem where $F = \bigwedge_{i=1}^m C_i$ where C_i is the disjunction of 3 variables. We note x_1, \dots, x_n the variables. We create the graph G as follows:

- For each variable in each clause, create a vertex, which we will label with the name of the variable. Therefore there may be multiple vertices with the label x_i or $\neg x_i$, if these variables appear in multiple clauses.
- For each clause, add an edge between the three vertices corresponding the variables from that clause.
- For all i , add an edge between every pair of vertices with one is labelled with x_i and the other labelled with $\neg x_i$.

There is a independent set of size m in G if and only if F is satisfiable. \square

We note that this reduction from $3 - \text{SAT}$ to INDSET took an instance φ of $3 - \text{SAT}$ consisting of m clauses each of three literals and produced a graph G_φ with $3m$ vertices such that if φ is satisfiable then the largest independent set in G has m vertices, and if G is unsatisfiable then the largest independent set of G has at most $m - 1$ vertices.

Now suppose that A is in DP . We want to show that $A \leq_p \text{EXACTINDSET}$. By definition of DP , $A = L_1 \setminus L_2$ for $(L_1, L_2) \in NP^2$. Since $3 - \text{SAT}$ is NP-complete, there are polytime functions f_1, f_2 such that for $i = 1, 2$ and for all $x \in \{1, 2\}^*$ we have $x \in L_i \Leftrightarrow f_i(x) \in 3\text{SAT}$. Hence for each fixed $x \in \{1, 2\}^*$, setting $\varphi_i = f_i(x)$, we have $x \in L_i \Leftrightarrow \varphi_i$ is satisfiable. Thus from the above reduction to INDSET , there is a polytime function which maps x to a pair of graphs G_1, G_2 such that if m_i is the number of clauses

in φ_i , then for $i = 1, 2$, no independent set in G_i has more than m_i vertices, and $x \in L_i \Leftrightarrow$ the largest independent set in G_i has size m_i .

Now we use the notation $G \sqcup H$ for the disjoint union of graphs G and H . That is, the vertices in $G \sqcup H$ are the disjoint union of those in G and H , and similarly for the edges. Now let $G'_1 = G_1 \sqcup G_1$. Then a maximum independent set in G'_1 is the union of maximum independent sets in the two copies of G_1 . Thus $x \in L_1 \Rightarrow$ maximum independent set of G'_1 is $2m_1$ and $x \in \overline{L_1} \Rightarrow$ maximum independent set of G'_1 is $\leq 2m_1 - 2$. Now define G'_2 so that its vertices are those of G_2 together with $m_2 - 1$ new vertices, and the edges consist of those of G_2 together with an edge from each of the new vertices to every vertex of G_2 . Then we have designed G'_2 so that no independent set can contain both vertices of G_2 and new vertices, so $x \in L_2 \Rightarrow$ maximum independent set of G'_2 is m_2 , $x \in \overline{L_2} \Rightarrow$ maximum independent set of G'_2 is $m_2 - 1$. Now let $G_3 = G'_1 \sqcup G'_2$ and let $k = 2m_1 + m_2 - 1$. To finish the proof that $A \leq_p DP$, it suffices to show that

Lemma 15.

$$x \in A \Leftrightarrow (G_3, k) \in DP$$

(\Rightarrow): Suppose $x \in A$. Then $x \in L_1 \setminus L_2$, so we conclude the maximum independent set of $G_3 = G'_1 \sqcup G'_2$ is $2m_1 + m_2 - 1 = k$.

(\Leftarrow): Suppose $x \notin A$. There are three cases:

- $x \in L_1 \cap L_2 \Rightarrow \text{maxindset}(G_3) = 2m_1 + m_2 > k$.
- $x \notin L_1 \Rightarrow \text{maxindset}(G_3) \leq 2m_1 + m_2 - 2 < k$

□

Theorem 16. EXACTINDSET is DP-complete.

Proof. EXACTINDSET is in DP (proposition 12) and it is DP-hard (proposition 13). □

4 Classes with exponential resources

We name BOUNDEDHALT the language of 3-tuples $\langle M, x, k \rangle$ where the machine M halts on input x in k steps.

Theorem 17. BOUNDEDHALT is EXP-complete.

Proof.

Lemma 18. BOUNDEDHALT \in EXP.

Proof. Let $\langle M, x, k \rangle$ an instance of BOUNDEDHALT.

We simulate M on x for k steps and accepts if and only if M halts and rejects otherwise. The running time is $m^{O(1)}$. Let $n = |\langle \alpha, x, k \rangle| \geq \log k$. Therefore, the running time $\leq m^c = 2^{c \log m} \leq 2^{cn}$. □

Lemma 19. BOUNDEDHALT is EXP-hard.

Proof. For each language $\mathcal{L} \in \text{EXP}$, we need to give polytime reduction from \mathcal{L} to BOUNDEDHALT. For a given language $\mathcal{L} \in \text{EXP}$, we know there is a TURING MACHINE $M_{\mathcal{L}}$ that decides A in time $g(n) \leq 2^{n^c}$ for a $c \in \mathbb{N}$.

Let f such that $f(w) = \langle M_{\mathcal{L}}, w, m \rangle$ where $m = 2^{|w|^c}$.

f is polytime computable and $w \in \mathcal{L} \Rightarrow \langle M_{\mathcal{L}}, w, m \rangle \in \text{BOUNDEDHALT}$ and $w \notin \mathcal{L} \Rightarrow \langle M_{\mathcal{L}}, w, m \rangle \notin \text{BOUNDEDHALT}$ □

BOUNDEDHALT is in EXP (lemma 18) and it is EXP-hard (lemma 19). \square

Theorem 20.

$$L = P \Rightarrow PSPACE = EXP$$

Proof. Take an arbitrary $\mathcal{L} \in EXP$, decided by a TURING machine in time 2^{n^c} .

Let $L_{pad} := \{x \diamond^{2^{|x|^c}} \mid x \in \mathcal{L}\}$. We have $L_{pad} \in P$ and, by the assumption $P = L$, we found that $L_{pad} \in L$. So, there is some TURING machine M_0 deciding L_{pad} in space $\log n$. M_0 can be used to decide L : on input x , simulate M_0 on the padded input $x \diamond^{2^{|x|^c}}$ accept if and only if M_0 accepts.

The space we used on input x ($|x| = n$) is the space used by M_0 on the padded input $x \diamond^{2^{|x|^c}}$ of size $n + 2^{n^c}$, which is at most $\log(2^{n^c+1}) = n^c + 1$, a polynomial. Hence, we have that $\mathcal{L} \in PSPACE$.

If we actually write the padded input $x \diamond^{2^{|x|^c}}$ on the work tape, this would make the space usage exponential in n . But, we don't need to write entirely this padded input. We know what it looks like to the right of x . So we can simulate M_0 on the virtual padded input $x \diamond^{2^{|x|^c}}$ by using a counter which tells the position on the tape of M_0 . If M_0 enquires about the position to the right of x , we respond with the symbol \diamond (or the blank, if M_0 goes to the right of the entire padded input). Since the counter can assume the value at most 2^{n^c} , we need at most n^c bits for the counter. Thus, this new TURING machine uses space at most polynomial in n . \square

5 Downward self-reducibility

Notation 1. We note DSR the set of downward self-reducible languages.

Proposition 21. $P \subseteq DSR$

Proof. Without query to the oracle, we can decide every language of P . \square

Proposition 22. $SAT \in DSR$

Proof. Let F a CNF-formula.

$$F = \bigwedge_{i=1}^m \bigvee_{j=1}^{n_i} x_{i,j}^*$$

where $x_{i,j}^*$ represent $x_{i,j}$ or $\neg x_{i,j}$ according with the indexes i and j .

$F \in SAT$ if and only if one of the $\left(x_{m,k} \wedge \bigwedge_{i=1}^{m-1} \bigvee_{j=1}^{n_i} x_{i,j}^* \right)_{k \in \llbracket 1, n_m \rrbracket} \in (CNF^{<|F|})^{n_m}$ is in SAT. We can test that in polynomial time. \square

Corollary 23. $NP\text{-complete} \subseteq DSR$

Proof. Let \mathcal{L} be a NP language. There exists a polynomial reduction showing that $SAT \leq_p \mathcal{L}$. The polynomial p used is an element of $\mathbb{Z}[X]$ such that $\lim_{n \rightarrow +\infty} p(n) = +\infty$. Consequently, there is a cofinite number of integers n such that $p(n) < p(n+1)$. So, the downward self-reducibility as the same sense even after the reduction, except for a finite number of cases which can be hard-coded in a TURING machine. We conclude that $\mathcal{L} \in DSR$. \square

Proposition 24. $DSR \subseteq PSPACE$

Proof. Let $\mathcal{L} \in \text{DSR}$. Given an input x , we simulate the polytime computation that (with queries) decides \mathcal{L} , and recursively compute the answer to the each query as it is made. Since the recursive calls are all on strings shorter than $|x|$, we will eventually reach the base case in which we query strings of length 1. The program we are describing will simply have the answers to these (constant number of) length 1 queries hard-coded. The depth of the recursion is at most $|x|$, and at each level of recursion, we need to remember the state which requires space at most $p(|x|)$ where p is a polynomial. This last point holds because the basic computation runs in polynomial time, and hence polynomial space. Thus the overall procedure runs in PSPACE. \square

6 Space hierarchy theorem

Theorem 25. *There is a universal TURING machine \mathcal{U} such that $\mathcal{U}(\alpha, x) = M_\alpha(x)$ and \mathcal{U} halts on every input (α, x) after using at most $cS(|x|)$ space, where S is the computation space of M_α and c is some constant not depending on x .*

Theorem 26. *For every space-constructible function $f : \mathbb{N} \rightarrow \mathbb{N}$,*

$$\text{DSPACE}(o(f(n))) \subsetneq \text{DSPACE}(f(n))$$

Proof. Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a space constructible function and $g : \mathbb{N} \rightarrow \mathbb{N}$ such that $g = o(f)$

We note M a deterministic TURING machine. M 's computation:

- M writes $1^{f(n)}$ symbols on a work tape, where n is the input's size.
- Simulate $\mathcal{U}(\alpha, (\alpha, x))$, and stops at most when the machine go out of the previous marks.
- If the computation of $\mathcal{U}(\alpha, (\alpha, x))$ is not terminated M rejects.
- Otherwise, outputs the opposite result of $\mathcal{U}(\alpha, (\alpha, x))$

The first step takes a space $f(n)$. The second step takes also a space $f(n)$ (thanks to the restriction). Finally, M runs in space $\mathcal{O}(f(n))$.

We note $\mathcal{L}(M)$ the language recognized by M ($\mathcal{L}(M) \in \text{DSPACE}(f(n))$) and assume that there is a deterministic machine N such that $\mathcal{L}(M) = \mathcal{L}(N)$ and N runs in space $g(n)$. We denote by α_0 the code of N .

According to the theorem 25, for every word x , $\mathcal{U}(\alpha_0, (\alpha_0, x))$ uses less than $cg(|(\alpha_0, x)|)$ space. And, by hypothesis, $cg(|(\alpha_0, x)|) \leq f(|(\alpha_0, x)|)$ for large enough x .

Thus, for large enough x we have $M(\alpha_0, x) = \neg \mathcal{U}(\alpha_0, (\alpha_0, x)) = \neg N(\alpha_0, x)$. This is a contradiction since we have assumed $\mathcal{L}(M) = \mathcal{L}(N)$.

In conclusion, there exists no machine N that runs in space $\mathcal{O}(g(n))$ and recognizes $\mathcal{L}(M)$. Since we have proved $\mathcal{L}(M) \in \text{DSPACE}(f(n))$, we obtained $\text{DSPACE}(g(n)) \subsetneq \text{DSPACE}(f(n))$. \square

7 Polynomial hierarchy

Lemma 27. $\text{P}^{\text{SAT}} = \Delta_2^{\text{P}}$

Proof. Since SAT is NP-complete, all NP language is KARP-reducible to SAT , so $\text{P}^{\text{SAT}} = \text{P}^{\text{NP}}$ \square

Lemma 28. $\text{NP}^{\text{SAT}} = \Sigma_2^{\text{P}}$

Proof. $\text{NP}^{\text{SAT}} = \text{NP}^{\text{NP}}$ \square

Lemma 29. $\Delta_2^P = \Sigma_2^P \Rightarrow \Sigma_2^P \subseteq \Pi_2^P$

Proof. Indeed $\Delta_2^P \subseteq \Sigma_2^P \cap \Pi_2^P$. □

Lemma 30. $\Sigma_2^P \subseteq \Pi_2^P \Rightarrow \Sigma_2^P = \Pi_2^P$

Proof. Let us assume $\Sigma_2^P \subseteq \Pi_2^P$. We have equivalently $\Sigma_2^P \subseteq co\Sigma_2^P$, so $\Sigma_2^P = co\Sigma_2^P$ and $\Sigma_2^P = \Pi_2^P$. □

Theorem 31. $NP^{SAT} = P^{SAT} \Rightarrow \Sigma_2^P = \Pi_2^P$

ie. polynomial hierarchy collapse.

Proof. Just combine the previous lemmas. □