Réductions, Problèmes P, NP-complétude

Préparation au CAPES de Mathématiques, option Informatique

Mars 2017

Sources

— Cours de complexité M1 info FST 2013-16

— Cours d'informatique fondamentale 4A polytech Lille 2012.

1 Polynomial complexity

EXERCICE 1 ▶ **Shift**
Écrire une machine de Turing qui "shifte" son entrée d'une case vers la droite, et évaluer sa complexité. Quelle est la complexité du problème ?

Solution. La machine a déjà été réalisée dans le chapitre sur les machines de Turing. La complexité de la machine est $O(n)$ clairement (1 parcours ou 2 parcours du mot suivant que l'on revienne au début ou pas). La complexité du problème est linéaire car clairement, on ne peut pas faire mieux que parcourir le mot...

□

Soient les deux problèmes suivants :

— 2-SAT : Soit une forme normale conjonctive F dans laquelle chaque clause a exactement 2 littéraux. F est-elle satisfaisable ?

— 2-COLOR : Soit un graphe G = (V, E). Existe-t-il une fonction c : $V \to \{0, 1\}$ telle que pour toute arête (u, v) de G, $c(u) \neq c(v)$?

EXERCICE 2 ▶ **2-SAT dans P**
Prouver directement que 2-SAT est dans P, en essayant de construire un graphe, et en prenant l'exemple

$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_4) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_3) \wedge (\neg x_2 \vee x_4)$$

Solution. L'algorithme que l'on va décrire est celui d'Aspvall, Plass et Tarjan, et la rédaction vient de Philippe Gambette, http://philippe.gambette.free.fr/SCOL/Graphes/Gambette%20-%20Un%20graphe%20pour%20resoudre%202SAT.pdf

On considère le graphe ayant $2n$ sommets, dont n sont étiquetés par les variables, et les n autres par les négations des variables, illustré à la Figure 1. On représente les m clauses de la formule par $2m$ arêtes dans le graphe de la façon suivante : chaque clause de la forme $(y \wedge z)$ peut être vue comme l'implication $\neg y \Rightarrow z$ ou $\neg z \Rightarrow y$, donc on ajoute pour chaque clause ainsi notée les arêtes $(\neg y, z)$ et $(\neg z, y)$.

Calculons alors les composantes fortement connexes du graphe (en temps linéaire avec l'algorithme classique de Tarjan). La formule n'est pas satisfiable ssi il existe une composante fortement connexe contenant une variable x_i et $\neg x_i$ (puisque dans ce cas on a $x_i \Leftrightarrow \neg x_i$). Si aucune composante de ce type n'est trouvée, cherchons une valuation solution. L'algorithme

de Tarjan fournit un ordre topologique sur les composantes fortement connexe (c'est à dire un ordre $<$ tel que s'il existe une arête de S_i vers S_j alors $S_i < S_j$). De plus, une composante fortement connexe $S_i = \{x_1, \ldots, x_k\}$ est telle que $\{\neg x_1, \ldots, \neg x_k\}$ est aussi une composante fortement connexe, que l'on note $\neg S_i$. On considère alors les composantes dans l'ordre inverse, et on leur affecte des valuations : tant qu'il reste une composante non valuée S_i , affecter à tous ses noeuds la valeur VRAI et à tous les noeuds de $\neg S_i$ la valeur FAUX. Ceci assure que tous les noeuds sont finalement valués sans aucun chemin de type $VRAI \Rightarrow FAUX$.

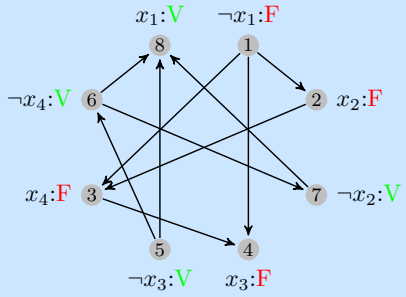


FIGURE 1 – Un exemple de graphe construit

Prenons maintenant un exemple, résolvons 2-sat sur

$$\phi = (x_1 \wedge x_2) \vee (x_3 \wedge \neg x_4) \vee (x_1 \wedge x_4) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_4)$$

Le graphe associé est celui de la figure 1 dans lequel chaque sommet une composante fortement connexe, les numéros à l'intérieur des noeuds indiquent un ordre topologique sur les composantes trouvées par l'algorithme de Tarjan. Comme il n'existe aucune composante fortement connexe contenant x_i et $\neg x_i$, la formule est satisfiable, et on peut effectuer les valuations, dans l'ordre topologique inverse : VRAI pour x_1 , donc FAUX pour $\neg x_1$, vrai pour $\neg x_2$, ... □

EXERCICE 3 ▶ **2-COLOR dans P**
Prouver directement que 2-COLOR est dans P.

Solution. G est deux-coloriable ssi il n'existe pas de cycle de longueur impaire. On en déduit un algorithme à base de parcours en profondeur (qui est polynomial) □

1.1 Appartenance à P via réduction polynomiale

On rappelle la définition d'une réduction polynomiale :

DÉFINITION 1. *Soient L_1 et L_2 deux langages de Σ^*. Une réduction polynomiale de L_1* vers *L_2 est une fonction $f : \Sigma^* \to \Sigma^*$* calculable en temps polynomial *telle que $x \in L_1$ ssi $f(x) \in L_2$. On note alors $L_1 \leq^P L_2$.*

Informellement, cela signifie que L_1 n'est pas plus difficile que L_2.
On a ensuite le résultat suivant

Proposition 1. *Si $L_1 \leq^P L_2$ et $L_2 \in P$, alors $L_1 \in P$.*

EXERCICE 4 ▶ **2-COLOR dans P, par réduction**
Démontrer que le langage 2-COLOR se réduit polynomialement à 2-SAT. En déduire qu'il est dans P.

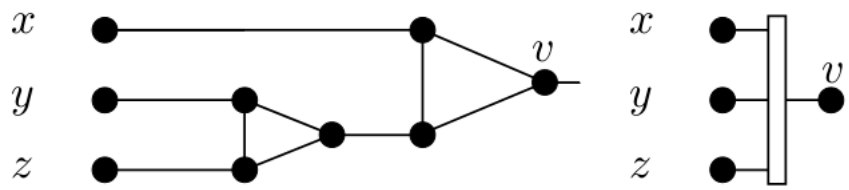
Solution. Soit $G = (V, E)$ un graphe. On doit construire une instance de 2-SAT. Classiquement, à chaque sommet u on associe un sommet x_u, et ensuite pour chaque couple (u, v) de sommets, $u \neq v$, on construit les deux clauses $x_u \wedge x_v$ et $\neg x_u \wedge \neg x_v$, l'instance de 2-SAT est l'union (et) de toutes ces clauses. L'algorithme de construction de l'instance est clairement polynomial en la taille du graphe (et construit une formule de taille poly !). Si G est 2-coloriable, alors en prenant la valuation induite par les couleurs : $\begin{cases} 1 \mapsto V \\ 0 \mapsto F \end{cases}$, on obtient une formule vraie. Si la formule est satisfiable, alors en prenant la coloration induite par la valuation "vraie" : $\begin{cases} x_u = V \mapsto c(u) = 1 \\ x_u = F \mapsto c(u) = 0 \end{cases}$ pour les on obtient bien G 2-coloriable. □

1.2 NP-complétude

EXERCICE 5 ▶ **NP-complétude de 3-COLOR**

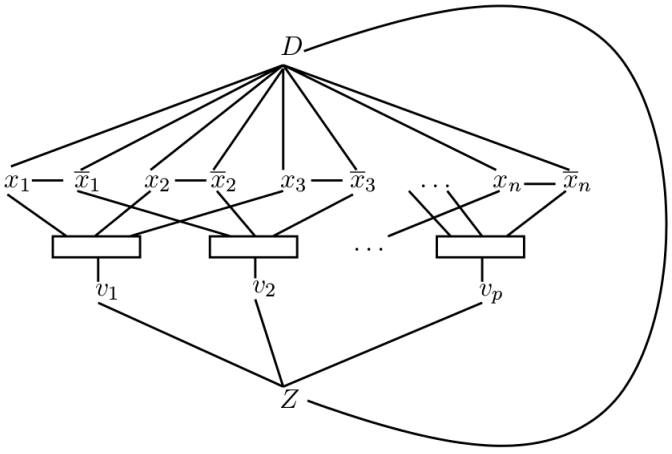
On va faire une réduction à partir de 3-SAT, montré NP-complet dans le cours.

Soit le gadget suivant (à droite son symbole dans la suite) :



1. Montrer que 3-COL est dans la classe NP.
2. Montrer que le gadget satisfait les 2 propriétés suivantes [1] :
   (a) Si $x = y = z = 0$ alors v est *nécessairement* colorié à 0.
   (b) Toute autre entrée (x, y, z) permet de colorier v à 1 ou 2 (au sens où l'on est capable d'exhiber un coloriage *tel que*...)

Soit I une instance de 3-SAT avec p clauses C_k (numérotées de 1 à p) et n variables x_i. On construit le graphe G suivant :



1. On pourra utiliser la notation suivante : $x, y \in \{0, 1, 2\}$ étant distincts, $\varphi(x, y)$ désigne le troisième élément de $\{0, 1, 2\}$.

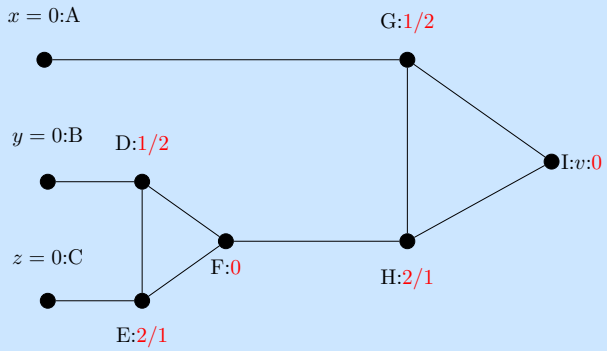
3. Vérifier que ce graphe a une taille polynomiale en l'instance I considérée.
4. Prouver que I a une solution ssi G est 3 coloriable.

Solution. Dans les solutions, les colorations suivront les conventions suivantes :

— valeurs en noir : valeurs fixées.

— valeurs en rouge : valeurs déduites.

1. Comme COLOR est dans NP, 3-COLOR est dans NP.
2. Si $x = y = z = 0$, alors on n'a pas énormément de choix pour le coloriage, en traitant tous les cas on trouve v à 0.



Pour la deuxième assertion, il faut encore bosser pas mal.

Passons maintenant à la preuve de NP-complétude.

3. Avec une instance de p clauses et n variables on construit un graphe de taille $1+2n+6p+1 = O(n+p)$ sommets. Donc une taille clairement polynomiale.
4. Prouvons que I_1 a une solution ssi G est 3 coloriable :

   $\Rightarrow$ On suppose que I_1 a une solution, ie une instantiation qui met toutes les clauses à *vrai*. On pose alors : $color(x_i) = \begin{cases} 1 & \text{si } x_i\text{vrai} \\ 0 & \text{sinon.} \end{cases}$, et $color(\overline{x_i}) = \begin{cases} 1 & \text{si } \overline{x_i}\text{vrai} \\ 0 & \text{sinon.} \end{cases}$. Comme il y a une variable à vrai dans chaque clause, dans chaque gadget il y a une entrée qui est non nulle. La propriété 2 du gadget permet d'avoir tous les v_i non nuls. En posant $color(D) = 2$ et $color(Z) = 0$ on obtient un coloriage de G à trois couleurs.

   $\Leftarrow$ On suppose que G a une coloration à 3 couleurs. En permutant les couleurs on peut supposer $color(D) = 2, color(Z) = 0$, et $color(x_i), color(\overline{x_i}) = (1,0)$ ou $(0,1)$. Comme $color(Z) = 0$, la couleur des v_i est 1 ou 2. La première propriété du gadget permet de conclure qu'il y a une variable à vrai pour chaque clause. Par conséquence, l'affectation $x_i = \begin{cases} 1 & \text{si } color(x_i) = 1 \\ 0 & \text{si} color(x_i) = 0 \end{cases}$, et $\overline{x_i} = \begin{cases} 1 & \text{si } color(\overline{x_i}) = 1 \\ 0 & \text{si } color(\overline{x_i}) = 0 \end{cases}$ donne une instantiation des x_i qui met toutes les clauses à vrai.

□

EXERCICE 6 ▶ **3-NAE "not all equal"**

— *Instance :* Un ensemble de clauses $C_1, \ldots, C_m$, chacune contenant exactement 3 littéraux.

— *Question :* Existe-t-il une instantiation des variables telle que chaque clause contient un littéral évalué à vrai et un littéral évalué à faux ?

Montrer que 3-NAE est NP-complet. *Indice : On pourra créer m+1 nouvelles variables, et construire une instance avec 2m clauses.*

Solution. Évidemment 3-SAT NAE est NP.

Pour montrer qu'il est NP-complet, et contrairement à toute attente, nous allons effectuer une réduction à partir de 3-SAT de la manière suivante :

Pour chaque clause, $a \vee b \vee c$ du problème 3-SAT, on crée les instances $a \vee b \vee x$ et $c \vee \overline{x} \vee f$ où f est global (ie f est la même pour chaque clause créée ainsi)

S'il existe une instantiation des variables qui met toutes les clauses à vrai, il en existe aussi une pour sa réduction, pour cela, il suffit de prendre $x = \overline{a \vee b}$, avec les notations ci-dessus, de plus si a ou b sont à vrai, x sera à faux, et donc pour la seconde clause sera mis a vrai par $\overline{x}$, on fini en prenant f toujours à faux, on a donc une bonne instantiation pour 3-SAT NAE. Réciproquement, si on a un bonne instanciation du problème 3-SAT NAE (réduction du problème 3-SAT) il existe un bonne instantiation pour le problème 3-SAT d'origine. En effet si f est à faux soit c est à vrai et donc la clause $a \vee b \vee c$ est vrai, soit c est à faux donc $\overline{x}$ est à vrai, x est donc à faux, d'où soit a soit b est à vrai et donc la clause $a \vee b \vee c$ est à vrai, par contre si f est à vrai, on prend toutes les variables du problème NAE et on prend leur négation, sans changer l'instance du problème. Cette instanciation, met toujours NAE à vrai, car comme "not all equal" une variable fausse au moins par clause qui est à vrai, et on recommence comme ci-dessus.

Ce qui précède montre que tout problème 3-SAT peut se réduire polynomialement (on multiplie le nombre de clause par deux, et on rajoute une variable plus une par clauses de départ) en un problème 3-SAT NAE équivalent, 3-SAT NAE est donc NP-complet. □

2 Réduction pour indécidabilité

Pour montrer qu'un problème est indécidable, soit on montre de manière directe (cf. cours), soit on effectue une réduction, dont on rappelle la définition :

Définition 2. *Soient L_1 et L_2 deux langages de Σ^*. Une* réduction *de L_1 à L_2 est une fonction récursive (*i.e. *calculable par machine de Turing) $\rho : \Sigma^* \to \Sigma^*$ telle que :*

$$w \in L_1 \text{ ssi } \rho(w) \in L_2.$$

Proposition 2. *Pour montrer qu'un langage L n'est pas récursif (i.e est indécidable), il suffit d'exhiber un langage L' non récursif tel que L' se réduit à L.*

En général, au lieu d'exhiber une machine de Turing qui réalise la réduction, on se contente de fournir un algorithme dans un langage "raisonnable"

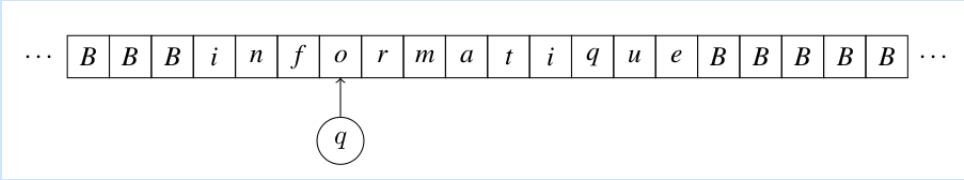
2.1 Une machine particulière

Une machine à k piles possède un nombre fini k de piles $r_1, \dots r_k$ qui correspondent à des piles d'éléments de Σ. Les instructions d'une machine à piles permettent seulement d'empiler un symbole sur l'une des piles, tester la valeur du sommet d'une pile, ou dépiler le symbole au sommet d'une pile. Si l'on préfère, on peut voir une pile d'éléments de Σ comme un mot w sur l'alphabet Σ. Empiler (*push*) le symbole a correspond à remplacer w par aw. Tester la valeur du sommet d'une pile (*top*) correspond à tester la première lettre du mot w. Dépiler (*pop*) le symbole au sommet de la pile correspond à supprimer la première lettre de w.

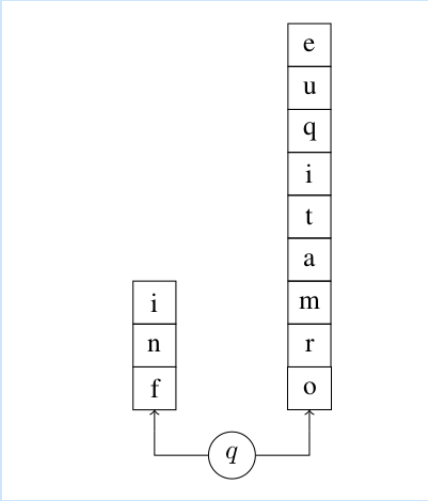
EXERCICE 7 ▶ **Simulation**

Montrer que tout machine de Turing peut être simulée par une machine à 2 piles. En déduire un problème indécidable pour les machines à 2 piles.

Solution. Soit une machine de Turing M, sur un certain alphabet, avec une certaine fonction de transition. Chaque configuration de cette machine :



peut être décrite par un triplet (w_1, q, w_2) avec w_i les mots à gauche et à droite de la machine de lecture. On décrit alors une machine à 2 piles avec les mêmes états, mais une configuration est maintenant :



(on empile par le bas) Ensuite il reste à exprimer chacune des transitions de la machine de Turing par une série d'opérations de la machine à 2 piles. Par exemple $(q, o, q', u, \rightarrow)$ à partir de la configuration dessinée de la pile se simule par :

- je regarde la lettre courante de la pile de droite (avec l'instruction `top`),
- si cette lettre est o, ce qui est le cas ici je la dépile (`pop`),
- j'empile u sur la pile de gauche, pour simuler le déplacement de la tête de lecture vers la droite (`push`).

En utilisant la définition de la machine de Turing M on arrive ainsi à définir une machine à 2 piles. Il est clair que cette machine à deux piles termine ssi M termine.

On peut donc en déduire que l'arrêt des machines à 2 piles est indécidable.

□