Le théorème de Ladner

Floriane Dardard

24 décembre 2009

Résumé

Le but de ce rapport est de comprendre les conséquences du théorème de Ladner : Pour cela, on commencera par l'énoncer et le démontrer, puis on donnera des exemples concrets. Ensuite, on survolera quelques résultats qui précisent ce théorème et lui donnent de la profondeur.

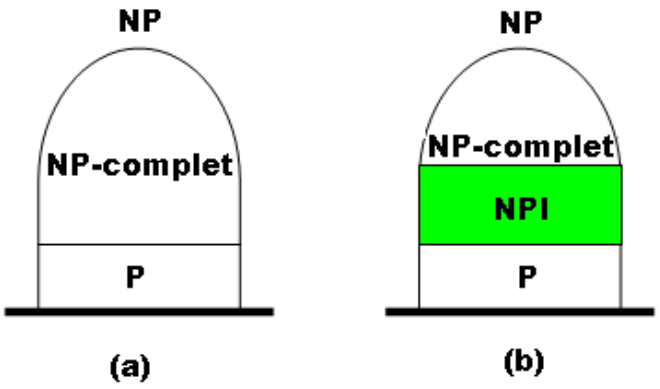
Table des matières

1 Introduction

Pour classer un problème dans la classe **NP**, il est souvent utile et plus simple de montrer qu'il est **NP**-complet. Cependant, parfois on rencontre des problèmes qui résistent aux démonstrations de complétude : par exemple le problème d'isomorphisme des graphes. Se pose alors la question de savoir si c'est parce que ces problèmes ne sont vraiment pas **NP**-complets, ou c'est parce que nous ne savons pas le démontrer. Existe-t-il des problèmes dans **NP** qui ne sont ni dans **P** ni **NP**-complets ?

Le point central, pour prouver ou réfuter l'existence de tels problèmes, est l'égalité de **P** et **NP**. En effet, si $\mathbf{P} = \mathbf{NP}$, alors tout problème dans

NP est dans **P** et est **NP**-difficile (On peut réduire polynomialement tout problème à tout autre), et alors **P** = **NP** = **NP**-complet. Par contre, si on suppose **P**≠**NP**, la question devient : la classe **NP** ressemble-t-elle à la figure (a) ou à la figure (b) ? La question est non triviale, et c'est Ladner qui le premier trouva la réponse en 1975.



2 Le théorème et sa démonstration

2.1 Le théorème

Théorème 2.1.1 *(Ladner, 1975) Si* ***P*≠*NP****, alors il existe un langage dans* ***NP*** *qui n'est ni dans* ***P*** *ni* ***NP****-complet. La classe de ces langages est appelée* ***NPI****, c'est-à-dire* ***NP****-Intermédiaire.*

2.2 Démonstration originale

Le principe de la preuve est, comme souvent en complexité, un argument de diagonalisation.

On va construire un langage L qui est quelquefois **SAT** (pour la non-appartenance à **P**)et quelquefois le langage vide (pour la non-complétude).

Soit M_1, M_2,... une énumération des machines de Turing déterministes calculables en un temps polynomial, avec répétitions éventuelles, telles que $M_i(x)$ se calcule en un temps $|x|^i$ (on les arrête si le calcul se prolonge plus). On a alors une manière de décrire tous les langages de **P**. On considère de même une énumération R_i des fonctions calculables en un temps polynomial.

On va créer une fonction $f : 1^* \rightarrow \mathbf{N}$ calculable en temps $\Theta(n)$.

On définit le langage A :

$$A = \{\omega / \omega \in \mathbf{SAT}, etf(|\omega|)estpair\} \quad (1)$$

On note K la machine de Turing calculant si ω est dans A.

Pour chaque i, on a deux critères à satisfaire :

$$\begin{aligned}(2i) & A \neq L_{M_i} \\ (2i+1) & \exists x, x \in \mathbf{SAT}, et R_i(x) \notin A, \\ & oux \notin \mathbf{SAT} et R_i(x) \in A\end{aligned}$$

On va alors définir f qui donne l'avancée de la satisfaction de tous ces critères. f va croître lentement. On définit :

$$f(0) = f(1) = 2 \tag{2}$$

Intuitivement :

- A l'étape 2i : On définit
$$f(n) = 2i \tag{3}$$
pour des n de plus en plus grands, jusqu'à ce que le critère $(2i)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors on aurait $A = L_{M_i}$ et A serait égal à **SAT** sauf sur un nombre fini d'instances, et cela contredirait le fait que **P**≠**NP**.
- A l'étape 2i+1 : On définit
$$f(n) = 2i + 1 \tag{4}$$
jusqu'à ce que le critère $(2i + 1)$ soit satisfait. Si jamais ce critère n'était jamais satisfait pour un certain i, alors A serait fini et **SAT** se réduirait à A par R_i, ce qui impliquerait que **SAT**∈**P**, et cela contredirait **P**≠**NP**.

Contruction formelle : Soit F la machine de Turing qui calcule f, on doit assurer à la fois la satisfaction des critères déjà mentionnés, et le fait que f va être calculé en $\Theta(n)$.
Pour cela, F va fonctionner en deux étapes distinctes pour le calcul de $f(n)$:

- *1ère étape : calcul rapide de f*
F calcule $f(0), f(1)$... jusqu'à avoir accompli n opérations. Supposons que la dernière valeur soit $f(i) = k$ (on a évidemment $i \leq n$) Alors on aura
$$f(n) = kouk + 1$$
selon la deuxième étape.
- *2ème étape : satisfaire les critères*
Si $k = 2i$ est pair, alors F calcule $\{M_i(z), SAT(z), F(|z|)\}$ où z est pris dans l'ordre lexicographique sur Σ^* jusqu'à avoir fait n opérations. F cherche un z tel que $K(z) \neq M_i(z)$. Si F trouve un tel z en n opérations, la condition $(2i)$ est satisfaite, et on définit $f(n) = k + 1$.

Sinon, $f(n) = k$ et F continue à chercher un z à l'étape d'après.
Si $k = 2i{+}1$ est impair, alors F calcule $\{R_i(z), SAT(z), SAT(R_i(z)), F(|R_i(z)|)\}$ où z est pris dans l'ordre lexicographique sur Σ^* jusqu'à avoir fait n opérations, comme dans le cas précédent. F cherche un z tel que $K(R_i(z)) \neq SAT(z)$ Si F trouve un tel z en n opérations, la condition $(2i)$ est satisfaite, et on définit $f(n) = k + 1$. Sinon, $f(n) = k$.

On obtient alors F qui calcule bien une fonction entière f en $\Theta(n)$.
De plus, f est croissante et n'est pas ultimement constante, sinon (comme on l'a vu intuitivement) cela contredirait **P**≠**NP**. Donc les critères sont tous satisfaits, et on a bien construit un langage $A \in$**NPI**.

On a ainsi montré un résultat qui éclaircit la structure de la classe **NP**. Néanmoins, le problème de cette démonstration est qu'elle produit des langages pas très "naturels", et on imaginerait difficilement une application pratique à ce résultat, sauf si on trouvait des problèmes plus intuitifs dans cette classe.

2.3 Problèmes dans NPI

Actuellement, il n'y a aucune démonstration prouvant l'appartenance d'un problème à cette classe. Cependant, certains langages particuliers sont "probablement" (comprendre que pendant 30 ans des informaticiens se sont acharnés à essayer de démontrer le contraire) dans cette classe, parmi lesquels :

2.3.1 FACTOR

Entrées : n entier positif et T une cible
Question : n a-t-il un facteur entre 2 et T (inclus) ?

On peut montrer que **FACTOR**∈**NP** ∩ **co-NP**.
En effet, **FACTOR**∈**NP** : Si la réponse est OUI, alors un facteur entre 2 et T est un certificat.
Pour voir que **FACTOR**∈**co-NP**, il faut envisager un certificat plus complexe : Si la réponse est NON, alors le certificat sera la décomposition en nombres premiers de n, combiné avec le certificat de que chaque facteur premier donné est vraiment premier (possible car PRIME∈**P**). L'algorithme de vérification regarde d'abord si la décomposition donnée redonne bien n en multipliant les facteurs, et si chaque facteur est premier. Cela confirmera que l'on a la bonne factorisation de n. Puis l'algorithme vérifie que le plus petit facteur premier est plus grand que T.

Donc **FACTOR**∈**NP** ∩ **co-NP**.
Si jamais **FACTOR** était **NP**-complet, alors ce serait un problème **NP**-complet dans **co-NP**. Cela impliquerait que **NP = co-NP**, ce qui est considéré comme faux (à la majorité informaticienne).
Si jamais **FACTOR** ∈ **P**, on pourrait casser RSA facilement...

Par conséquent, on pense que **FACTOR** n'est ni dans **P** ni **NP**-complet. Néanmoins, personne n'a encore réussi à le prouver.

2.3.2 GRAPHISOMORPHISM

Entrées : G et H, deux graphes
Question : G est-il isomorphe à H ?

GRAPHISOMORPHISM est un des candidats les plus naturels à la classe **NPI**
Intuitivement, on voit que si on change un tant soit peu les entrées d'une manière non intelligente, on va tout de suite perdre le caractère bijectif si on l'avait, ce qui n'est pas le cas de la majorité des problèmes **NP**-complets.

Ce problème est dans NP car on peut donner un isomorphisme d'une manière non déterministe, et vérifier si c'est vraiment un isomorphisme de manière déterministe et en temps polynomial. Par contre, savoir si ce problème est ou non dans **P** est un problème ouvert. D'autre part, on pense que **GRAPHISOMORPHISM** n'est pas **NP**-complet, car cela ferait s'effodrer de nombreux théorèmes (Time Hierarchy...).

3 Améliorations du théorème

3.1 Cardinal de NPI

Théorème 3.1.1 *Si* $\boldsymbol{P \neq NP}$, $\forall L \in \boldsymbol{NP} \setminus \boldsymbol{P}$, $\exists L' \in \boldsymbol{P} \setminus \boldsymbol{NP}$ *tel que* $L' \leq L$, *mais* $L \not\leq L'$.

Ce théorème a été montré par Ladner peu après l'autre, et la preuve est essentiellement la même pour les deux, sauf que cette fois, l'intersection se fait avec **L** et non avec **SAT**.

En itérant ce théorème, on montre facilement l'existence d'une chaîne infinie L_i de langages dans **NPI** tels que

$$\begin{aligned} \forall i, L_{i+1} &\leq L_i \\ L_i &\not\leq L_{i+1} \end{aligned}$$

Donc la classe **NPI** est infinie. On peut même montrer qu'elle est dense, dans le sens où :
Si A et$B$ sont dans **NPI** tels que

$$\begin{aligned} A &\leq B \\ B &\not\leq A \end{aligned}$$

Alors il existe $C \in$**NPI** un problème tel que

$$A \leq C$$
$$C \not\leq A$$
$$C \leq B$$
$$B \not\leq C$$

Avec ce résultat, plus aucun doute : on se trouve bien dans un de ces nombreux cas où l'intuition nous trahit : En supposant **P**≠**NP**, on a prouvé que la classe **NPI** avait une infinité d'éléments, infiniment proches les uns des autres.

3.2 Problèmes incomparables

Théorème 3.2.1 *(Balcazar-Diaz, 1982) Soit B et $L_1, ..L_m$ des langages tels que :*

- $L_i \in \boldsymbol{NP} \setminus \boldsymbol{P}$
- $L_i \leq B$
- $B \not\leq L_i$

Alors il existe un langage L_{m+1} tel que :

- $L_{m+1} \leq B$
- $B \not\leq L_{m+1}$
- L_{m+1} *et* L_i *sont incomparables en ce qui concerne les réductions en temps polynomial.*

Ce théorème se démontre lui aussi avec la même méthode que le théorème de Ladner.

L'intérêt de ce résultat est dans l'affirmation de la diversité dans la classe **NPI** : En effet, de ce théorème résulte l'existence de deux langages L et L' dans **NPI** qui sont incomparables. Par conséquent, la classe n'est pas une classe d'équivalence pour les réductions polynomiales, elle est donc bien plus complexe, à un point qu'il nous reste encore à découvrir.

4 Conclusion

Les résultats obtenus par Ladner et leurs améliorations on montré que les classes de complexité que nous utilisons régulièrement ont un structure très riche qui reste encore à découvrir. Si**P**≠**NP**, alors il existe une multitude de langages difficiles mais incomplets dans **NP**. Même si la preuve de l'existence de tels langages en construit quelques-uns, ces langages ne sont pas intuitifs. D'autre part, cela semble improbable que les langages naturels que l'on pense être dans **NPI** se réduisent en des langages aussi artificiels que ceux-là. Le théorème de Ladner est donc un théorème existentiel d'un

point de vue très mathématique, et ne donne pas d'exemples convaincants de problèmes concrets dans **NPI**, ce qui est regrettable. Pour définitivement clore le débat, il faudrait trouver un "vrai" problème dans **NPI**, ce qui ferait avancer significativement la recherche en complexité.

Références

[1] Wikipedia, *NP-Intermediate*
`http://en.wikipedia.org/wiki/NP-Intermediate`

[2] L. Fortnow *Diagonalization*
Bulletin of the European Association for Theoretical Computer Science, 71 :102-112, June 2000. Computational Complexity Column

[3] J. Papadimitriou, *Computational complexity*
Addison Wesley, 1994