Théorème de Mahaney

Marc Jeanmougin

24 décembre 2009

Introduction

Nous nous intéresserons ici aux complexités non uniformes en prenant pour nouveau modèle de calcul les circuits et non plus les machines de Turing. Les familles de circuits sont en effet bien plus efficaces que les machines de Turing puisqu'elles permettent d'avoir les appartenances à des langages non décidables... même si en contrepartie trouver le circuit est une opération non calculable (dommage...).

Même si ce sont donc des objets fortement théoriques ici (bien que très concrets dans les circuits imprimés...), les familles de circuit permettent d'obtenir des résultats en complexité standard : Nous nous intéresserons donc ici au théorème de Mahaney, qui donne une équivalence de la toujours ouverte question : $P \stackrel{?}{=} NP$.

On suppose connu les notions de base de complexité.

Table des matières

1 Définitions

1.1 Notion de circuits

Un *circuit* est un graphe acyclique orienté avec des portes appartenant à une base de portes.

Une porte d'entrée a pour degré entrant 0, et la porte de sortie a pour degré sortant 0.

La base la plus répandue et couramment utilisée est $B_0 = \{\neg, \wedge, \vee\}$, une base minimale (pour obtenir toutes les fonctions logiques) est $B_{min} = \{\neg\wedge\}$, une autre base utile est $B_1 = \{\neg, (\wedge^n)_{n\in\mathbb{N}}, (\vee^n)_{n\in\mathbb{N}}\}$.

Comme on peut s'y attendre, un circuit définit une fonction logique de $\{0,1\}^n \longrightarrow \{0,1\}$.

On a alors deux mesures de la *complexité* : la *profondeur* d'un circuit et sa *taille*. La taille d'un circuit est son nombre de portes, la profondeur le plus long chemin entre une porte d'entrée et une porte de sortie.

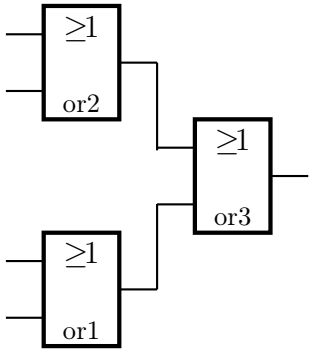


FIGURE 1 – Exemple de circuit modélisant $f : \{0,1\}^4 \to \{0,1\} : x \mapsto 1$ ssi x contient un 1.Taille : 3 ; profondeur : 2

Propriété 1 *Toute fonction peut être représentée par un circuit de B_1 de profondeur 3 (et de taille $< \Theta(2^n)$).*

Démonstration :
On utilise la forme normale conjonctive ou forme normale disjonctive de l'expression booléenne de la fonction du circuit. On note x^i la i^{eme} lettre de x.

$f(x) = \bigvee_{x'\in\{x\in\{0,1\}^n | f(x)=1\}} \bigwedge_{i=1}^{n} (x^i = x'^i)$
(ie "x est une solution de $f(x) = 1$")

$f(x) = \bigwedge_{x'\in\{x\in\{0,1\}^n | f(x)=0\}} \bigvee_{i=1}^{n} (x^i \neq x'^i)$
(ie "x n'est pas une solution de $f(x) = 0$")

Propriété 2 *Toute fonction peut être représentée par un circuit de B_0 de profondeur $\Theta(n)$ (et de taille $< \Theta(n2^n)$).*

Démonstration :
On avait auparavant des portes de degré entrant n de B_1, que l'on remplace simplement par $n-1$ portes de degré entrant 2. On avait également une porte de degré entrant au plus 2^n que l'on remplace par au plus $2^n - 1$ portes de degré entrant 2. On a bien une taille au pire en $\Theta(n2^n)$. En plaçant "intelligemment" les portes, on a une profondeur en $1 + \log(n) + log(2^n)$, soit $\Theta(n)$.

Définition 1 *Une famille de circuits $C = \{C_i\}_{i\in\mathbb{N}}$ est un ensemble infini de circuits tels que C_i a i portes d'entrées.*

Une famille de circuit définit alors une fonction $f : \{0,1\}^* \longrightarrow \{0,1\}$ tel que $f(x) = C_{|x|}(x)$ On a alors la propriété que tout langage, peu importe s'il est calculable, peut être reconnu par une famille de circuits.

Définition 2 *On définit alors des classes de complexité sur les circuits :*
- *$L \in SIZE(f)$ ssi Il existe une famille de circuits de B_0 telle que $taille(C_i) = O(f(i))$* [1]
- *$L \in DEPTH(f)$ ssi Il existe une famille de circuits de B_0 telle que $profondeur(C_i) = O(f(i))$* [2]

Définition 3 *Les familles uniformes de circuit sont celles pour lesquelles il existe une machine de Turing qui pour une entrée i a pour sortie une description de C_i. On ne s'intéressera désormais qu'aux circuits non-uniformes.*

1.2 Complexité des circuits et classe $P/Poly$

Définition 4 *On peut définir la classe $P/Poly$ de deux manières équivalentes :*
- *$P/Poly = \bigcup_{i\in\mathbb{N}} SIZE(n^i)$*
- *$P/Poly = \{L | \exists$une machine de Turing et une suite α_n de "conseils", $|\alpha_n| < p(n)$ et $M(x, \alpha_{|x|}) = \chi_L(x)\}$*

Ces deux définitions sont étonnamment équivalentes.
- $1 \Rightarrow 2$: Soit une famille de polynômes de taille bornée par un polynôme qui décide L. Alors on interprète C_i comme α_i, la suite de "conseils" pour la longueur i. Alors la machine de Turing pourra résoudre correctement le problème de l'appartenance.
- $2 \Rightarrow 1$: D'une machine de Turing et d'une suite de "conseils" bornés polynomialement on peut construire une famille de circuits.correspondant au parcours de la machine de Turing en suivant les "conseils" de la longueur du mot considéré. On peut alors borner polynomialement la taille du circuit.

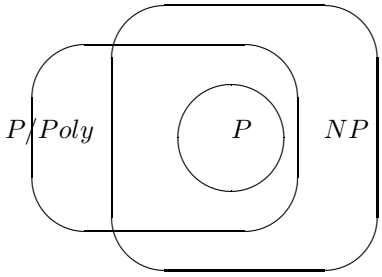
1. $SIZE(n2^n)$ n'a aucun intérêt (sur B_0).
2. tout comme $DEPTH(n)$

Propriété 3 *On a trivialement $P \subset P/Poly$ (en prenant la seconde définition, sans "conseil") et $P \neq P/poly$ ($P/Poly$ contient des langages indécidables [il contient tous les langages unaires]).*

La notion de circuit a quelques affinités avec la notion de Machine de Turing : Ainsi, on a les quelques propriétés suivantes, que nous citons à titre indicatif car elles n'interviennent pas dans la suite ni dans le théorème de Mahaney :

- $TIME(f) \subset SIZE(f \log f)$
- $NSPACE(f) \subset DEPTH(f^2)$

Par ailleurs, il est couramment supposé que $NP \not\subset P/Poly$. Si ce résultat était prouvé (si un problème NP(voire $NP - complet$?) se trouvait ne pas être dans $P/Poly$), on aurait démontré que $P \neq NP$.



?

2 Théorème de Mahaney

Définition 5 *Un langage est dit peu dense si* $|L \cup \{0,1\}^n| < p(n)$ *pour un certain polynôme p.*

2.1 Enoncé

Un langage $NP - Complet$ est réductible en temps polynomial à un langage peu dense si et seulement si $P = NP$

2.2 Démonstration

2.2.1 $\Longleftarrow$

Si $P = NP$, un langage $NP - complet$ est dans P... et on peut donc le ramener de manière polynomiale au problème d'appartenance au langage non-dense le plus simple, par exemple $\{1\}$.

2.2.2 $\Longrightarrow$

Schéma de la preuve Supposons qu'un langage $NP - Complet$ soit réductible en temps polynomial à un langage peu dense.
Soit LSAT un langage $NP - Complet$ que nous définirons plus loin.
Nous pouvons alors réduire LSAT à un langage peu dense en temps polynomial (en passant par L en temps polynomial).
On construira alors un algorithme... polynomial qui résoudra le problème SAT, d'où $P = NP$.

LSAT : On définit LSAT comme suit : Soit ϕ une expression booléenne à n variables, $x \in \{0,1\}^n$. $(\phi, x) \in LSAT \Longleftrightarrow \exists x' | x' \leq_{lexicographique} x$ et $\phi(x)$.

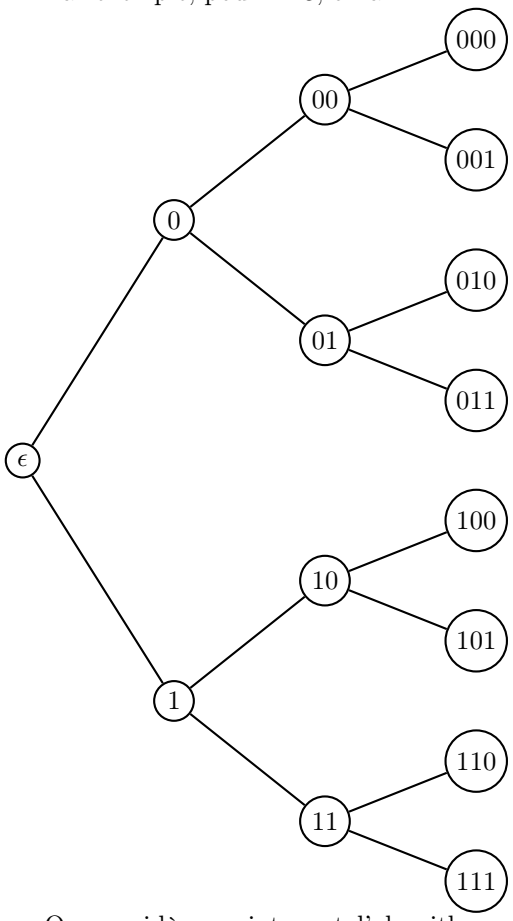
On a immédiatement $(\phi, 1^n) \in LSAT \Longleftrightarrow \phi \in SAT$, d'où LSAT est $NP - Complet$. Comme précisé dans le schéma de la preuve, on a LSAT réductible en

temps polynomial à un langage peu dense S, par la fonction f.

Puisque f est calculable en temps polynomial, on a une borne polynomiale (polynôme $p(n)$) sur la longueur de la sortie de f.
On pose $q(n) = |S \cup \{0,1\}^{\leq p(n)}|$, polynômial car S est peu dense et p polynomial (un polynôme de polynôme est un polynôme) .

Soit l'arbre ayant pour racine ϵ et à chaque noeud étiqueté par v ayant pour fils $v0$ et $v1$, de profondeur n.

Par exemple, pour n=3, on a :



On considère maintenant l'algorithme suivant pour résoudre le problème SAT en temps polynomial :

$LIVE_0 \leftarrow \{\epsilon\}$
for $i = 1$ à $n - 1$ **do**
  $LIVE_i \leftarrow \{$tous les fils de $LIVE_{i-1}\}$
  if $|LIVE_i| > q(n)$ **then**
    "élaguer" $LIVE_i$ jusqu'à ce qu'il contienne moins de $q(n)$ éléments.
  end if
end for
Renvoyer 1 ssi un des fils de $LIVE_{n-1}$ satisfait le problème.

L'élagage d'un ensemble de nœuds V est fait de la manière suivante : Calculer $Z_V = \{f(v)|v \in V\}$ puis répéter les étapes suivantes :

- Si $f(v_1) = f(v_2)$, retirer le plus grand des deux dans l'ordre lexicographique.
- Si Z_v contient plus de $q(n)$ valeurs, retirer le plus petit élément de V dans l'ordre lexicographique.

Il y a n niveaux de boucle et au plus $2q(n)$ noeuds considérés à chaque niveau, donc on "élague" au plus $2nq(n)$ fois, ce qui est polynomial. l'opération elle-même l'est également trivialement.

Il reste à prouver la correction de l'algorithme :
Montrons par induction qu'à chaque itération i, $LIVE_i$ contient un ancêtre du plus petit (dans l'ordre lexicographique) mot satisfaisant ϕ. C'est vrai pour $LIVE_0$.

Supposons la proposition au rang $i-1$. $LIVE_i$, avant l'élagage, satisfait toujours la condition...

On a ensuite deux possibilités :

- $f(v_1) = f(v_2)$ et $v_1 < v_2$. On retire v_2 mais si v_2 avait eu un descendant satisfaisant le problème, alors v_1 également, d'où v_2 ne peut pas être ancêtre du plus petit dans l'ordre lexicographique. On conserve donc la propriété par cette opération.
- Si Z_{LIVE_i} contient plus de $q(n)$ valeurs, alors nous savons qu'il en existe au moins une valeur dans Z qui n'est pas ancêtre d'une solution (il y en a au plus $q(n)$). Donc il y a une valeur dans $LIVE_i$ qui n'est pas ancêtre d'une solution de LSAT. Donc puisque v_α n'est pas satisfaite, si v_0 est le plus petit mot de V, v_0 n'est évidemment pas solution de LSAT. Donc ce n'est pas un ancêtre de la plus petite solution de ϕ et on peut donc le retirer.

Ainsi, on conserve la propriété à toutes les étapes de l'algorithme. Donc $LIVE_{n-1}$ contient le père d'une solution de SAT, et il n'y en a qu'un nombre polynomial, qui sont chacun polynomialement testables.

On a donc un algorithme polynomial qui résout SAT, donc

$$P = NP$$

.

On a donc démontré le théorème.□

Références

[1] Jonathan Katz, *Notes on Complexity Theory : Lectures 4 and 6*. 2005

[2] *Wikipedia, l'encyclopédie libre*. http ://en.wikipedia.org