Hiérarchie Polynomiale

Charles Fougeron

25 décembre 2010

Résumé

Les différentes manières de répondre au problème du voyageur de commerce, allant d'une simple inégalité sur la longueur du résultat, à un chemin explicite, motivent une extention de la classe de complexité **NP**. En généralisant cette extention, nous aboutirons à un affinage des classes de complexité entre **NP** et **PSPACE**. Ces classes intermédiaires seront d'autant plus intéressante qu'une séries de jolis résultats sur celle-ci nous inclinera à croire que $\mathbf{P} \neq \mathbf{NP}$.

1 Le problème du voyageur de commerce

Définition 1 (PVC)**.** Il y a plusieurs instances du Problème du Voyageur de Commerce, chacune d'une difficulté différente. L'objectif étant, pour chacune de celles-ci, d'obtenir des informations plus ou moins fortes, sur le plus court chemin passant par tous les sommets d'un graphe dont les arrêtes sont pondérées.

Énonçons ces différentes instances dans un "ordre croissant de difficulté" :

- PVC (D) détermine si il y a un parcours de poid inférieur à D dans le graphe.
- PVC EXACTE (D) détermine si la longueur du plus court chemin est de taille D.
- COÛT PVC retourne le coût du plus court chemin.
- PVC retourne le plus court chemin.

Remarque 2. Chacune de ces instances peut être réduite à la précédente. Pour les trois dernières c'est immédiat. Pour la deuxième, on remarque que la première instance est **NP**, or la seconde peut être réduite au problème du CHEMIN HAMILTONIEN, qui est **NP**-complet (Le détail est dans [2] ch. 9).
L'odre dans lequel les problèmes sont énoncés est donc bien croissant.

Le PVG EXACTE (D) peut se décomposer en deux partie, on détermine d'abord si la longueur du plus court chemin est $\leq D$ (PVC (D) $\sim$ **NP**) puis si elle est $> D$ ($\neg$ PVC (D) $\sim$ **coNP**). Ainsi ce problème est à l'intersection de deux langages, l'un **NP** et l'autre **coNP**. On appelle cette classe **DP**.

Définition 3 (**DP**)**.** Un langage L est dans la classe **DP** si il existe deux langages $L_1 \in$ **NP** et $L_2 \in$ **coNP** tels que $L = L_1 \cap L_2$.

Remarque 4. **DP** a peu de chance d'être dans **NP**, il est même très peu probable qu'il soit dans **NP** $\cup$ **coNP**. Par ailleurs EXACTE PVG est **DP**-complet.

Cette nouvelle classe de complexité décrit bien les problèmes de "coût exact" (elle contient par exemple les problèmes ENSEMBLE INDEPENDANT, KNAPSACK, MAX-CUT, MAX-SAT qui sont eux aussi **DP**-complets), et est très riche : par exemple elle comprend les problèmes de type "critique", ie qui déterminent si lorsque l'on supprime un élément, le problème a toujours une solution (entre autres SAT CRITIQUE, CHEMIN HAMILTONIEN CRITIQUE, 3-COLORIABILITE CRITIQUE).

2 Classes polynomiales

On peut regarder la classe introduite précédemment pour couvrir la complexité PVG EXACTE, d'une manière plus globale, comme si l'on avait introduit un nouveau type de machine de Turing qui pour accepter une entrée, test d'abord l'acceptation de celle-ci sur une machine **NP**, puis sur une machine **coNP**. On en vient donc naturellement à l'idée de généraliser ce type de machines qui font des tests à partir de machines données.
On se donne alors comme outil de travail une machine qui résout un problème fixé sans que cette résolution ne soit prise en compte dans le calcul de la complexité. On appelle une telle machine un oracle.

Définition 5 (Oracle)**.** Une machine de Turing $M^?$ avec un oracle, est une machine de Turing déterministe à plusieurs bandes, dont une spéciale, la bande d'oracle. Elle a en outre trois états suplémentaires : $q_?$, l'état pour consulter l'oracle, q_{OUI}, q_{NON}, les états de réponse.
Soit $A \subseteq \Sigma^\star$ un langage arbitraire. La machine à oracle A fonctionne comme une machine ordinaire, en dehors des états q : lorsqu'elle consulte l'oracle en se plaçant dans l'état $q_?$, les "pouvoirs surnaturels" de l'oracle font changer spontannement la machine d'état, la plaçant dans q_{OUI} si la chaine de caractère dans la bande d'oracle est dans A et dans q_{NON} sinon.

Ce nouveau concept a une très grande souplesse :

Remarque 6. Il existe des langages A et B, tels que, d'une part, $\mathbf{P^A} = \mathbf{NP^A}$ et d'autre, $\mathbf{P^B} \neq \mathbf{NP^B}$

Démonstration. Vu en TD. □

Au vu de la manière avec laquelle a été introduite **DP**, et fort du nouveau concept d'oracle, il est naturel d'étudier la classe de complexité $\mathbf{P}^{SAT}$, qui par **NP**-completude de SAT est aussi notée $\mathbf{P^{NP}}$.

Définition 7 ($\mathbf{FP^{NP}}$)**.** Cette classe est l'ensemble des fonctions, des chaines de caractères d'entrée dans les chaines de caractères de sortie, qui sont calculables par une machine de Turing dans $\mathbf{P^{NP}}$.

Le résultat suivant (dont nous ne donnerons pas la preuve), nous montre encore une fois la souplesse de la notion d'oracles, et l'intéret de cette nouvelle classe.

Théorème 8. *SORTIE-MAX, POIDS-MAX SAT et PVC sont* $\mathbf{FP^{NP}}$*-complet.*

Où

- SORTIE-MAX Étant donnée une machine de Turing N et un entier n, déterminer la longueur de la sortie la plus longue pour une entrée de taille n.
- POIDS-MAX Étant donné une formule logique sous forme normale conjonctive, et un poids pour chaque clause, nous cherchons une assignation des valeurs de vérité qui satisfait des clausses dont la somme des poids est maximale.

Définition 9 (La hiérarchie polynomiale)**.** On définit un nouvel ensemble de classes de manière séquencielle :

Tout d'abord, $\Delta_0\mathbf{P} = \Sigma_0\mathbf{P} = \Pi_0\mathbf{P} = \mathbf{P}$, et pour tout $i \geq 0$,

- $\Delta_{i+1}\mathbf{P}=\mathbf{P}^{\Sigma_i\mathbf{P}}$
- $\Sigma_{i+1}\mathbf{P}=\mathbf{NP}^{\Sigma_i\mathbf{P}}$
- $\Pi_{i+1}\mathbf{P}=\mathbf{coNP}^{\Sigma_i\mathbf{P}}$

On définit aussi la classe cumulée $\mathbf{PH} = \bigcup_{i\geq 0} \Sigma_i\mathbf{P}$

Comme $\Sigma_0\mathbf{P} = \mathbf{P}$, le premier niveau de la hiérarchie représente les classes qui nous sont déjà familières : $\Delta_1\mathbf{P} = \mathbf{P}$, $\Sigma_1\mathbf{P} = \mathbf{NP}$ et $\Pi_1\mathbf{P} = \mathbf{coNP}$. Le second niveau commence avec $\Delta_2\mathbf{P} = \mathbf{P}^{\mathbf{NP}}$ que nous venons d'étudier et continue avec $\Sigma_2\mathbf{P} = \mathbf{NP}^{\mathbf{NP}}$ et son complementaire $\Pi_2\mathbf{P} = \mathbf{coNP}^{\mathbf{NP}}$.
Naturellement, les trois classes à chaque niveau verifient les mêmes inclusions que celles connues du niveau 1, à savoir $\mathbf{P} \subseteq \mathbf{NP}$ et $\mathbf{P} \subseteq \mathbf{coNP}$. Et toute classe d'un certain niveau est inclue dans chaque classe des niveaux supérieurs.
La caractérisation des problèmes **NP** en terme de vérification d'une solution en temps polynomiale est conceptuellement intéressante. Les classes de hiérarchie polynomiale généralisent d'autant mieux **P** et **NP** qu'elles ont, elles aussi, un critère simple sur les solutions :

Théorème 10. *Soit L un langage, et $i \geq 1$. $L \in \Sigma_i\mathbf{P}$ si et seulement si il y a une relation R polynomialement équilibrée (ie il existe un $k \geq 1$ tel que, si $(x, y) \in R$ alors $|y| \leq |x|^k$) et telle que $\{x; y : (x, y) \in R\}$ soit dans $\Pi_{i-1}\mathbf{P}$ et $L = \{x : \exists y\ (x, y) \in R\}$*

Démonstration. Procédons par récurrence sur i. Pour $i = 1$, c'est un théorème du cours. Pour $i > 1$ montrons tout d'abord que $L \in \Sigma_i\mathbf{P}$. Nous devons donc trouver un machine non-déterministe qui décide en temps polynomial L avec un oracle dans $\Sigma_{i-1}\mathbf{P}$. C'est facile : prenons la machine non-déterministe qui parcourt tout les y possibles et teste si $(x, y) \in R$ avec son oracle, qui est alors bien dans $\Sigma_{i-1}\mathbf{P}$ par hypothèse car R est dans $\Pi_{i-1}\mathbf{P}$ (il ne nous restera plus qu'à tester si $(x, y) \notin R$).
Réciproquement, soit $L \in \Sigma_i\mathbf{P}$. On doit chercher la bonne relation R. Nous savons par hypothèse que L peut être décidé en temps polynomial par une machine de Turing non déterministe à oracle M^K avec $K \in \Sigma_{i-1}\mathbf{P}$. Comme $K \in \Sigma_{i-1}\mathbf{P}$ par hypothèse de récurrence, il existe une relation $S \in \Pi_{i-2}\mathbf{P}$ telle que $z \in K$ si et seulement si il existe w tel que $(z, w) \in S$.
Nous avons besoin d'un court "certificat" pour chaque $x \in L$ qui définira une relation. Or nous savons que $x \in L$ si et seulement si il y a un calcul acceptant dans M^K de x. Notre certificat sera donc une chaine de caractères y codant les étapes de ce calcul acceptant. Mais il faut faire un peu attention, certaines de ces étapes seront des demandes à l'oracle K, et certains vont avoir "oui" en réponse tandis que d'autre auront "non". Pour chaque demande z_i dont la réponse est positive, notre certificat inclura le certificat du calcul acceptant dans l'oracle, w_i

tel que $(z_i, w_i) \in S$. Definissons alors R de la sorte : $(x, y) \in R$ si et seulement si y correspond à un calcul acceptant dans M^K de x.
Alors vérifier si $(x, y) \in R$ peut être fait dans $\Pi_{i-1}\mathbf{P}$. Tout d'abord on doit montrer que toutes les étapes de calcul sont licites dans M^K, ce qui se fait en temps polynomiale. Ensuite, nous devont vérifier pour un nombre polynomial de paires, si $(z_i, w_i) \in S$; ce qui peut être fait dans $\Pi_{i-2}\mathbf{P}$ donc dans $\Pi_{i-1}\mathbf{P}$. Pour les demandes z_i à l'oracle répondus par "non", il faut vérifier que l'on a bien $z_i \notin L$. Or comme $K \in \Sigma_{i-1}\mathbf{P}$ par hypothèse, cette vérification est elle aussi dans $\Pi_{i-1}\mathbf{P}$. Enfin, $(x, y) \in R$ si et seulement si un nombre polynomial de problème dans $\Pi_{i-1}\mathbf{P}$ est vérifié, c'est donc grobalement dans $\Pi_{i-1}\mathbf{P}$. □

On a un énoncé "dual" pour $\Pi_i\mathbf{P}$:

Corollaire 11. *Soit L un langage et $i \geq 1$. $L \in \Pi_i\mathbf{P}$ si et seulement si il existe une relation binaire polynomialement équilibrée telle que $\{x; y : (x, y) \in R\}$ est dans $\Sigma_{i-1}\mathbf{P}$ et $L = \{x : \forall y \ (x, y) \in R\}$*

Démonstration. Il suffit de se rappeller que $\Pi_i\mathbf{P} = \mathbf{co}\Sigma_i\mathbf{P}$ □

Nous avons une deuxième description, qui est conséquence directe du théorème précédant, et qui crée une correspondance directe avec la logique mathématique, en liant la difficulté des calculs avec l'alternance des quanteurs :

Corollaire 12. *Soit L un langage et $i \geq 1$. $L \in \Sigma_i\mathbf{P}$ (resp. $\Pi_i\mathbf{P}$) si et seulement si il existe une relation polynomialement équilibrée, $i+1$-aire, R telle que*

$$L = \{x : \exists y_1 \forall y_2 \exists y_3 \dots Q y_i, \text{ avec } (x, y_1, \dots, y_i) \in R\}$$

Le ième quantificateur Q étant $\forall$ (resp. $\exists$) si i est pair, et $\exists$ (resp. $\forall$) sinon.

Démonstration. Il suffit de remplacer les langages dans $\Pi_j\mathbf{P}$ et dans $\Sigma_j\mathbf{P}$ par leur forme utilisant leur "certificat". □

Il est important, à ce point de l'exposé, de comprendre que cette hiérarchie dans les classes de complexité a été construite couche à couche, et que cela fait à la fois son intérêt mais aussi sa faiblesse. Le résultat suivant va nous montrer à quel point cette construction est fragile :

Théorème 13. *Si il existe un $i \geq 1$ tel que $\Sigma_i\mathbf{P} = \Pi_i\mathbf{P}$, alors pour tout $j > i$,*

$$\Sigma_j\mathbf{P} = \Pi_j\mathbf{P} = \Delta_j\mathbf{P} = \Sigma_i\mathbf{P} = \Pi_i\mathbf{P}$$

On dit que la hiérarchie polynomiale s'effondre au niveau i.

Démonstration. Il suffit de montrer $\Sigma_i\mathbf{P} = \Pi_i\mathbf{P}$ implique que $\Sigma_{i+1}\mathbf{P} = \Sigma_i\mathbf{P}$, le dual est alors immédiat, et on conclut ensuite par réccurence. Nous considérons donc un langage $L \in \Sigma_{i+1}\mathbf{P}$. D'après le Théorème 8, on a un relation R dans $\Pi_i\mathbf{P}$ telle que $L = \{x : \exists y (x, y) \in R\}$, par hypothèse R est alors aussi dans $\Sigma_i\mathbf{P}$. □

Exemple 14. Une application immédiate du théorème nous donne que si $\mathbf{P} = \mathbf{NP}$ ou même plus simplement si $\mathbf{NP} = \mathbf{coNP}$ la hiérarchie polynomiale s'effondre au niveau 1. Ce résultat nous montre à quel point cette hiérarchie repose sur le fait que $\mathbf{P} \neq \mathbf{NP}$ car dans le cas contraire, elle ne crée aucune distinction entre les problèmes.

Il y a une classe de problème intéressante, qui nous donne des problèmes $\Sigma_i\mathbf{P}$-complètes pour chaque i :

Définition 15 ($SATQ_i$, satisfiabilité quantifiée avec i itérations)**.** Soit ϕ une expression Booléenne avec des variables partitionnées en i ensembles $X_1, \ldots, X_i$, alors une instance de $SATQ_i$ a la forme suivante :

$$\exists X_1 \forall X_2 \exists X_3 \ldots Q X_i \ \phi$$

avec pour l'alternance des quanteurs : $\exists$ si i est impaire, $\forall$ si i est pair.

Théorème 16. *Pour tout $i \geq 1$ $SATQ_i$ est $\Sigma_i\mathbf{P}$-complet.*

Démonstration. Ce résultat repose fortement sur le Corollaire 12. $SATQ_i \in \Sigma_i\mathbf{P}$ en est une conséquence immédiate ; l'écrite de l'expression a été choisie pour rentrer dans les hypothèses.
Soit $L \in \Sigma_i\mathbf{P}$ réduisons le à $SATQ_i$. Nous utilisons alors L sous sa forme donnée par le Corollaire 12, en notant R la relation associée au langage. Il existe une machine de Turing déterministe M qui prend en entrée $x; y_1; \ldots; y_i$ et qui l'accepte si $(x, y_1, \ldots, y_i) \in R$. En se rappelant la preuve du fait que SAT est **NP**-complet, on trouve une formule ϕ qui traduit les calculs de la machine M. Il faut alors remarquer que la construction de la preuve nous donne des variables correspondants aux entrées ; puisqu'on utilise toutes les variables codant le fait que au i-ième calcul, la lettre a est en j-ème position sur la bande de la machine de Turing (notée $x_{i,j,a}$ dans le cours, cf [1] p. 171). On pose alors $i+1$ ensembles de variables. $X, Y_1, \ldots, Y_i$ qui correspondront chacun aux ensembles de variables introduites pour coder les entrées $x, y_1, \ldots, y_i$ de la machine, séparées par des ";". Le dernier ensemble (probablement bien plus gros) sera l'ensemble des autres variables Booléennes Z qui codent les autre aspects du calcul de la machine.
À présent, si l'on fixe les valeurs de $X, Y_1, \ldots, Y_i$, l'expression résultante est satisfiable si et seulement si les valeurs de vérité des éléments de ces ensembles codent pour des chaines de caractères dans le langage décidé par M donc est dans R. Soit à présent une chaine $x \in L$, on affecte les valeurs de vérité correspondantes au codage de x dans ϕ. Alors on sait que $x \in L$ si et seulement si il existe y_1 tel que, pour tout y_2, ... , si i est impair, il existe, et si i est pair, pour tout y_i tels quel $R(x, y_1, \ldots, y_i)$.
Ainsi comme les ensembles Y_k codent les entrées y_k, $x \in L$ si et seulement si $\exists Y_1 \forall Y_2 \ldots \exists Y_i \ \phi(X)$ est satisfiable (avec une bonne assignation des valeurs de vérité de Z). □

Regardons à présent jusqu'où va cette construction en étudiant la classe **PH**.

Remarque 17. Si il existe un problème **PH**-complet, la hiérarchie polynomiale d'effondre en un level fini.

Démonstration. Soit L un problème **PH**-complet, alors on a un $i \geq 0$ tel que $L \in \Sigma_i\mathbf{P}$, or pour chaque $L' \in \Sigma_{i+1}\mathbf{P}$ se réduit à L. Donc $L' \in \Sigma_i\mathbf{P}$ alors $\Sigma_i\mathbf{P} = \Sigma_{i+1}\mathbf{P}$. □

On a une borne supérieure pour cette classe. On voit facilement grâce au Corollaire 12, que chaque problème dans **PH** peut être résolu en espace polynomial, en cherchant les $y_1, \ldots, y_i$.

Remarque 18. **PH** $\subseteq$ **PSPACE**

Une fois cette hiérarchie introduite, le résultat suivant conclura notre exposé, en montrant un aspect de l'intérêt de cette classification. Introduisons tout d'abord quelques classes de complexité.
Il exite des classes de complexité tels que **RP** introduites pour décrire les algorithmes probabilistes, qui comprend les algorithmes qui répondent faussement "oui" à une entrée avec une probabilité inférieure à $\frac{1}{2}$. On formalise cette propriété en supposant les tirages aléatoires tous simulés par une machine non-déterministe.

Définition 19 (**RP**)**.** Soit N une machine de Turing non-déterministe dont l'exécution est à temps borné pour tout calcul. On suppose que N est *précise* c'est à dire qu'elle s'arrête après le même nombre d'étapes pour tout calcul. On suppose aussi qu'à chaque étape, il y a exatement deux choix non déterministes. Soit L un langage, il est dans la classe **RP** si il existe une machine telle que décrite au ci-dessus, dont le nombre d'étapes pour une entrée de taille n est polynomial en n, noté $p(n)$, et qui pour chaque entrée vérifie la propriété suivante : si $x \in L$, alors au moins la moitié des $2^{p(n)}$ calculs s'arrètent sur "oui". Si $x \notin L$ alors tous les calculs s'arrètent sur "non".

La classe **coRP** est alors celle décrivant les algorithmes répondant faussement "non" avec une probabilité inférieure à $\frac{1}{2}$ Une idée naturelle pour élargir ces deux classes qui acceptent un certain langage en répondant soit "oui" soit "non" faussement avec un certaine probabilité est de donner la bonne réponse quelle qu'elle soit avec une certaine probabilité (choisie égale à $\frac{1}{4}$ ici).

Définition 20 (**BPP**)**.** Cette classe contient tous les langages L pour lesquels il existe une machine de Turing normalisée de la même manière que pour **RP**, et qui vérifie cette propriété : pour tout $x \in L$, au moins $\frac{3}{4}$ des calculs accèptent x, pour $x \notin L$ au plus $\frac{1}{4}$ des calculs acceptent x.

Remarque 21. Il est important de remarquer que la constante $\frac{3}{4}$ est choisie arbitrairement, mais n'importe quelle constante entre $\frac{1}{2}$ et 1 strictement, donnerait la même classe.

Démonstration. Pour le montrer, utilisons une méthode probabiliste. Pour cela il faut remarquer que cette borne de $\frac{1}{4}$, $\frac{3}{4}$ est équivalante au fait que l'on donnera une réponse fausse avec une probabilité $\frac{1}{4}$. Introduisons tout d'abord l'inégalité de Chernoff.

Lemme 22 (La borne de Chernoff)**.** *Soient $x_1, \ldots, x_n$ des variables aléatoires indépendantes, prenant la valeur 1 et 0 avec une probabilité p et $1-p$ respectivement, et soit la somme $X = \sum_{i=0}^{n} x_i$. Alors pour tout $0 \leq \theta \leq 1$,*

$$\mathbf{P}[X \leq (1-\theta)pn] \leq e^{-\theta^2 pn}$$

(La preuve est calculatoire, et ne fait pas l'objet de cet exposé.)

Prenons à présent une machine qui répond faussement avec une probabilité égale à $\frac{1}{2} + \epsilon$. L'idée est de faire tourner la machine k fois, en acceptant si la majorité des calculs accepte. Pour utiliser la borne de Chernoff, nous prenons pour x_i des variables aléatoires qui prennent la valeur 1 si le calcul est juste, et 0 sinon. La probabilité d'erreur de cette nouvelle machine, est celle qu'une minorité de calculs de l'ancienne soit juste sur les k. D'où

$$\mathbf{P}[\text{la réponse est fausse}] = \mathbf{P}[X \leq \frac{n}{2}]$$

En prenant $\theta = \frac{\epsilon}{\frac{1}{2}+\epsilon}$, avec $p = \frac{1}{2} + \epsilon$, nous obtenons

$$\mathbf{P} \leq e^{-\frac{\epsilon^2}{\frac{1}{2}+\epsilon}n} \leq e^{-\epsilon^2 n}$$

car $\epsilon \leq \frac{1}{2}$. On peut donc choisir k assez grand pour avoir une probabilité d'erreur inférieure à $\frac{1}{4}$. □

On voit facilement que **NP** et **coNP** sont dans **BPP**. Montrons par une méthode probabiliste qu'elle est inclue dans le second niveau de la hiérarchie polynomiale.

Théorème 23. $\mathbf{BPP} \subseteq \Sigma_2\mathbf{P}$

Démonstration. Soit $L \in \mathbf{BPP}$. Il existe donc une machine de Turing M, avec des calculs de longueur $p(n)$ pour des entrées de longueur n. Pour chaque entrée x de longueur n, soit $A(x) \subseteq \{0,1\}^{p(n)}$ l'ensemble des calculs acceptant. On peut supposer la probabilité d'une réponse fausse inférieure à $\frac{1}{2^n}$ d'après la remarque précedente. Donc on a une machine telle que si $x \in L$ alors $|A(x)| \geq 2^{p(n)}(1-\frac{1}{2^n})$, et si $x \notin L$ alors $|A(x)| \leq 2^{p(n)}\frac{1}{2^n}$.
Soit U l'ensemble de toutes les chaines de bits de longueur $p(n)$. Pour $a, b \in U$ on définit l'opération $a \otimes b$ comme la résultant du "ou" exclusif sur chaque bit. Alors l'application "$\otimes b$" est une convolution et est donc en particulier injective. Soit t une chaine de bits de longueur $p(n)$. Considérons l'ensemble translaté de $A(x)$ par t : $A(x) \otimes t = \{a \otimes t : a \in A(x)\}$. Montrons alors le fait suivant : si $x \in L$, comme $A(x)$ est assez grand dans ce cas, on peut trouver un petit ensemble de translations de $A(x)$ qui décrivent la totalité de U ; à l'inverse si $x \notin A(x)$ il n'en existe pas.
Plus formellement, supposons $x \in L$, est considérons une suite de $p(n)$ chaines de bits, $t_1, \ldots, t_{p(n)} \in U$, obtenus de manière équiprobable sur les $p(n)^2$ possibilités. Soit $b\ inU$, on dit que cette suite couvre b si $\exists j \leq p(n), b \in A(x) \otimes t_j$. Calculons la probabilité pour qu'un point b soit couvert. Remarquons que $b \in A(x) \otimes t_j$ si et seulement si $b \otimes t_j \in A(x)$, or comme "$b\otimes$" est une bijection sur un ensemble équiprobable, $b \otimes t_j$ est une variable aléatoire identique à t_j. D'où $\mathbf{P}[b \notin A(x) \otimes t_j] = \frac{1}{2^n}$. Donc la probabilité que b ne soit couverte par aucun des t_j est $2^{-np(n)}$.
Ainsi, la probabilité qu'un point de U ne soit par couvert par la suite et $|U|.2^{-np(n)} = 2^{-(n-1)p(n)} < 1$. Donc les suites couvrent U avec un probabilité non nulle, il existe donc une suite qui couvre U.
A l'inverse, si $x \notin L$, $p(n).|A(x)| \leq 2^{p(n)}\frac{p(n)}{2^n} < |U|$ pour un n assez grand.
Finalement, on peut écrire L de la manière suivante :

$$L = \{x : \exists t_1, \ldots t_{p(n)} \in \{0,1\}^{p(n)}, \forall b \in U, \exists j \leq p(n), b \otimes t_j \in A(x)\}$$

Or ceci est exactement la forme de $\Sigma_2\mathbf{P}$ introduite dans le corollaire 12. Le dernier $\exists j$ pouvant ne pas être pris en compte dans la forme, car comme il est borné polynomialement, il peut être considéré comme un "ou". □

Remarque 24. La définition de **BPP** est clairement symétrique, donc elle est stable par complémentation. D'où :

$$\mathbf{BPP} \subseteq \Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$$

3 Conclusion

Ces nouvelles notions appuient donc l'hypothèse $\mathbf{P} \neq \mathbf{NP}$ si l'on croit en la difficulté des $SATQ_i$, du PVG ou des nombreux autres problèmes cités dans l'exposé. L'auteur de [2], C. H. Papadimitriou, l'un des grands chercheurs en complexité, qui a introduit le premier la classe **DP** et a travaillé entre autre sur la hiérarchie polynomiale ; annonce d'ailleurs clairement, au fil des théorèmes, qu'il est convaincu de ce résultat. Mais il faut rester prudent, et ne pas oublier que notre conception d'un problème difficile est floue, et ne colle pas nécessairement avec les classes de complexité que nous connaissons.

Références

[1] Olivier Carton. *Langages formels, calculabilité et complexité.* Vuibert, 2008.

[2] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.