Présentation de l'article : ARITHMETIZATION: A NEW METHOD IN STRUCTURAL COMPLEXITY THEORY

RANNOU Pierre

20 juin au 26 août 2006

1 Introduction

L'article que j'ai choisit a été co-ècrit par Làszlò Babai et Lance Fortnow.

Cet article introduit une nouveau modèle de calcul basé sur les polynômes construit inductivement à la manière des strates lines programmes. Les auteurs vont ainsi définir différent classe de polynômes équivalente à des classe de complexité générique pour ensuite redémontrer des résultat classique.

Le principal résultat de ce papier est l'égalité entre la classe de polynôme $PRAB$ et $\sharp P$. Les résultats classiques redémontrés ici sont:

- Le Théorème de TODA $PH \subseteq BP \oplus P \subseteq P^{\sharp P}$
- L'égalité $PSPACE = IP$
- Ainsi que quelques Théorèmes sur $\sharp_m P$.

L'approche en strate line programme présenté par les auteur donne une vision plus proche de la programmation que la vision des circuits ou des machine de turing.

Cette présentation de l'article ne sera pas exhaustive et ne présentera que les résultats les plus importants.

2 Préliminaires

2.1 Classe $\sharp P$

2.1.1 Classe NP

On va donner deux définition équivalente de la classe NP

Définition1: On dit qu'un problème est dans NP si il est reconnu en temps polynomial par une machine de turing non déterministe (Machine de turing

ou l'automate "*d'écriture et de lecture*" est non déterministe).
Définition2: On dit qu'un langage l est dans NP si il existe $l' \in P$ et un polynôme p_n tel que si il existe un couple $\langle x, y \rangle \in l'$ avec $|y| = p_n(|x|)$ alors $x \in l$ et réciproquement.

2.1.2 Arbre d'exécution d'un problème NP:

Définition: C'est un arbre binaire construit à partir de y de la manière suivante:

- Chaque niveau correspondant à un bit de y.
- Le premier nœud est étiqueté par n_0.
- Le nœud n_x du niveau $|x|$ est relié au nœud n_{x0} et n_{x1} du niveau $|x|+1$ qui correspondent à la valeur du bit $|x|+1$ de y.

Définition: Calculer le nombre de chemin acceptant d'un arbre d'exécution d'un problème NP est un problème $\sharp P$.

2.1.3 Propriété de $\sharp P$

$\sharp P$ admet des problèmes complets par réductions polynomiales: $\sharp sat$ et $\sharp 3-sat$ sont des problème $\sharp P$ complet.

2.1.4 La classe $\sharp_m P$

Définition: Soit $f \in \sharp P$ alors $f \ mod(m) \in \sharp_m P$.
Remarques: Il existe des problèmes $\sharp_m P$ complets comme le permanent d'une matrice. On notera $\oplus P$ la classe $\sharp_2 P$

2.1.5 La classe $Gap - P$

Définition: $f \in Gap - P$ si et seulement si il existe g et g' appartenant à $\sharp P$ tel que $f = g - g'$
Remarque: Il existe des problèmes $Gap - P$ complets

2.2 Classe probabiliste

2.2.1 Classe BPL

Définition: $l \in BPL$ si il existe $l' \in L$ tel que:

- si $x \in l$ alors pour plus des plus des deux tiers des chaînes y de taille $p_n(|x|)$ avec p_n un polynôme on a $\langle x, y \rangle \in l'$
- si $x \notin l$ alors pour plus des deux tiers des chaînes y de taille $p_n(|x|)$ avec p_n un polynôme on a $\langle x, y \rangle \notin l'$

Remarques: On peut remplacer la probabilité sur $\frac{2}{3}$ par $\frac{1}{2} + C$, C étant une constante appartenant à $[0, 1]$. On notera BPP la classe BPL où L est la classe des langages polynomiaux.

2.2.2 Classe PL

Définition: $l \in PL$ si il existe $l' \in L$ tel que:

- si $x \in l$ alors pour plus de la moitié des chaînes y de taille $p_n(|x|)$ avec p_n un polynôme on a $\langle x, y \rangle \in l'$
- si $x \notin l$ alors pour plus de la moitié des chaînes y de taille $p_n(|x|)$ avec p_n un polynôme on a $\langle x, y \rangle \notin l'$

Remarque: On notera PP la classe PL où L est la classe des langages polynomiaux.

2.3 Preuve interactive

Définition: On définit la classe des preuves interactive IP à l'aide d'un protocole entre deux machines de turing:

- La première est le prouveur P.
- La seconde est le vérifieur V.

Le prouveur est une machine de turing quelconque.

Le vérificateur est une machine de turing polynomial probabiliste qui demande si un x donné appartient à un langage l au prouveur et vérifie le résultat de ce dernier en respectant les propriétés suivantes:

1. Il existe un prouveur telque $\forall x \in l$ le vérifieur accepte la réponse du prouveur
2. Pour tout prouveur si $\forall x \notin l$ le vérifieur rejète avec une probabilité supérieur à un demi.

Remarque: il suffit d'une probabilité en $O(2^{n^c})$ avec c constant car on peut ensuite diminuer l'erreur en répétant le plusieur fois le protocole.

2.4 Hiérarchie polynomiale et la classe $PSPACE$

2.4.1 Hiérarchie polynomiale PH

Définition: Un langage l est dans la hiérarchie polynomiale si il existe un entier n constant, et une suite de polynôme $(p_i)_{i<n+1}$ et un langage $l' \in P$: tel que $(Q_i y_i)_{i<n+1}, \langle y_1, ..., y_n, x \rangle \in l'$ où la suite des $(Q_i y_i)_{i<n+1}$ est suite alternée de quantificateurs: $\forall$ et $\exists$ et où $|y_i|$ est major par $|p_i(|x|)|$.

2.4.2 Classe $PSPACE$

Définition: La classe $PSPACE$ est la classe des langages pour lesquels il existe une machine de turing qui calcul en espace polynomiale les fonctions de décision qui leur sont associée.
Remarque: Il existe des langages $PSPACE$ complet. Le langage des formule Booléenne totalement quantifié est $PSPACE$ complet.

3 Présentation de $PRAB$ et RAB

3.1 Classe $PRAB$

Définition: On définit inductivement le strait line programme correspondant à la classe $PRAB$ à l'aide des instructions suivantes:

1. p_k est une constante sur $\{0, 1\}$.
2. $p_k = x_i$ où x_i est une variable et $i \leq k$.
3. $p_k = 1 - x_i$ où x_i est une variable et $i \leq k$.
4. $p_k = x_i + x_j$ où $i, j < k$.
5. $p_k = x_i \times x_j$ où $i + j \leq k$.(multiplication retardée)
6. $p_k = p_j(x_i = 1)$ ou $p_k = p_j(x_i = 0)$ qui correspond au polynôme p_j dans lequel x_i est remplacé soit par un soit par zéro; on a $i, j < k$

La classe des strait line programme ainsi définit est appelé PRAB, on notera par la suite $p_{|x|}$ les straits lines programmes où k est de la forme $|x|^{0(1)}$.
Remarques:
Comme la multiplication est retardée on a le degré de $\widehat{p}_k$ inférieure à k
On note $\widehat{p}_k$ le polynôme correspondant au strait line programme p_k
Définition: Une famille $P_1, P_2, ..., P_n$ de strait line programme est dite uniforme si il existe une machine de Turing déterministe polynomial qui calcul le polynôme: $\widehat{P}_n$, ce polynôme $\widehat{P}_n$ ne dépendant que des variable $x_1, x_2, ..., x_n$.
Nous allons maitenant démontrer le principal théorème de l'article: l'égalité entre la classe $\sharp P$ et la classe des straits line programme $PRAB$.
Théorème: Soit f une fonction de $\{0, 1\}^* \longrightarrow \mathbf{Z}$ alors les deux proposition suivante sont équivalentes:

1. $f \in \sharp P$
2. il existe une famille P_n de strate line programme $PRAB$ uniforme telque pour tout chaîne $x \in \{0, 1\}^*$ on est:

$$f(x) = \widehat{P}_{|x|}(x)$$

Preuves: On va commencer la preuve par (1) $\Rightarrow$ (2) ensuite on démontrera (2) $\Rightarrow$ (1).

Pour prouver (1) $\Rightarrow$ (2) On va donner la construction d'une famille de strate line programme qui calcul $\sharp 3-sat$ comme ce dernier est $\sharp P$ complet on en déduira que $\sharp P \subseteq PRAB$. Pour ce faire on va associer à tout n une formule ϕ_n de $3-sat$, ainsi la fonction $f'(x)$ associé $\phi_n(x)$ s'évalue à 1 si et seulement si il existe un y telque $\phi(x)(y)$ soit vrai, par ailleur $\phi(x)(y)$ se vérifie dans P par $g_x(y)$. Ainsi $f(x) = \Sigma_{y\in\{0,1\}^*} g_x(y)$. On construit un polynôme q_x associé g_x (qui s'évalue à un si et seulement si g_x s'évalue à un et vaut zéro sinon).

1. On transforme chaque clause $C_i = v \vee u \vee \neg w$ en une formule ψ: disjonction de sept clause conjonctive qui décrive l'ensemble des valeur que preuve prendre u, v, et w pour valider C_i(en omettant bien sur la clause conjonctive $\neg C_i = \neg v \wedge \neg u \wedge w$).

2. Ensuite on transforme chaque formule ψ en un polynôme de degré trois en remplaçant les $\neg y_i$ par $1 - y_i$, les $\wedge$ par des multiplication et les $\vee$ par des additions. On remarque ainsi que chaque polynôme r_i correspondant a ψ_i vaut un si ψ_i s'évalue à vrai et zéro sinon.

3. Ensuite on prend le produit des r_i de degré trois pour former q_x.

On vient ainsi de construire q_x maintenant grâce au projection (instruction 6 du SLP) et aux addition on obtient $\Sigma_{y\in\{0,1\}^*} q_x(y)$ ainsi on calcul bien la valeur de $f(x)$. Pour démontrer (2) $\Rightarrow$ (1) on va faire un raisonnement par induction sur les instructions: à chaque instruction on va faire correspondre une opérationńation sur les arbre d'exécution binaire des problème NP.

1. A l'instruction un: la constante $p_j = c_i$, on fait correspondre l'arbre à un unique sommet acceptant uniquement si c_i vaut 1.

2. A l'instruction deux: la variable $p_j = y_i$, on fait correspondre le problème NP dont l'arbre d'exécution de profondeur deux, à un unique chemin acceptant dans lequel $y_i = 1$.

3. A l'instruction trois: la variable $p_j = 1 - y_i$, on fait correspondre le problème NP dont l'arbre d'exécution de profondeur deux, à un unique chemin acceptant dans lequel $y_i = 0$.

4. A l'instruction quatre:la somme $p_j = p_i + p_k$ on construit la machine de turing non déterministe qui exécute de façons non déterministe: soit la machine de turing non déterministe associé à p_i soit la machine

de turing non déterministe associé à p_j. L'arbre binaire d'exécution associé à cette MTND est celui qui correspond à l'instruction quatre.

5. A l'instruction cinq: le produit $p_j = p_i \times p_k$ on construit la machine de turing non déterministe qui exécute: la machine de turing non déterministe associé à p_i puis la machine de turing non déterministe associé à p_j et renvoie un si et seulement si les deux exécution on donné un. L'arbre binaire d'exécution associé à cette MTND est celui qui correspond à l'instruction cinq.

6. A l'instruction six: la projection $p_j = p_i(y_k = c)$ avec c constant on construit l'arbre binaire d'exécution à partir de celui de p_j dans lequel on supprime les chemins où $y_k = 1 - c$.

Remarque: Pour tout P_n on à la taille des chemins T_c (racine feuilles) de l'arbre d'exécution A inférieur à n plus particulièrement on a:

- Pour l'addition $P_i = P_j + P_k$ avec $i \geq max(j+1, k+1)$, de plus $T_c(A_i) = max(T_c(A_j)+1, T_c(A_k)+1)$ par construction et donc comme on a par induction $T_c(A_j) \leq j$ et $T_c(A_k) \leq k$ on peut en déduire $T_c(A_i) \leq i$.

- Pour la multiplication $P_i = P_j + \times P_k$ avec $i \geq j + k$, de plus $T_c(A_i) = T_c(A_j) + T_c(A_k)$ par construction et donc comme on a par induction $T_c(A_j) \leq j$ et $T_c(A_k) \leq k$ on peut en déduire $T_c(A_i) \leq i$.

Le nombre de chemin acceptant de la machine de turing associé à P_n est égale au nombre que l'on obtient en évaluant ce dernier. Fin.

3.2 Classe RAB

Définition: La classe RAB est définie en modifiant l'instruction 4 en $4'$ par:

$$p_k = x_i \pm x_j \text{ où } i, j < k$$

Remarque: On obtient une classe plus puissante que $PRAB$.

Théorème: Tout famille de SLP $P_1, ..., P_n \in RAB$ est équivalente à une famille de SLP $P'_1, ..., P'_{n'} \in RAB$ ($\widehat{P}_n = \widehat{P'}_n$) qui n'utilise qu'une instruction de soustraction et où $n' = O(n^c)$ avec c une constante, par une transformation uniforme.
Preuve: Pour une famille de SLP $P_1, ..., P_n \in RAB$ On va construit par induction sur les instruction une famille $P'_1, ..., P'_{n'} \in RAB$ en retardant l'utilisation de la soustraction:

1. L'instruction 1: la constante c_i est remplacé par: les instructions $c'_i = c_i$ et $c''_i = 0$.

2. L'instruction 2: la variable x_i est remplacée par: les instructions $x'_i = x_i$ et $x''_i = 0$
3. L'instruction 3: la variable $1 - x_i$ est remplacée par: les instructions $x'_i = 1 - x_i$ et $x''_i = 0$
4. L'instruction 4: la somme $p_j = p_i +_p k$ est remplacée par les instructions $p'_j = p'_i + p'_k$ et $p''_j = p''_i + p''_k$
5. L'instruction $4bis$: la soustraction $p_j = p_i - p_k$ est remplacée par $p'_j = p'_i + p''_k$ et $p''_j = p''_i + p'_k$
6. L'instruction 5: la multiplication $p_j = p_i \times p_k$ est remplacée par $p^3_j = p'_i \times p'_k$, $p^4_j = p''_i \times p''_k$, $p^5_j = p'_i \times p''_k$, $p^6_j = p''_i \times p'_k$, $p'_j = p^3_j + p^4_j$, et $p''_j = p^5_j + p^6_j$.
7. L'instruction 6: la projection reste inchangée.

Il est claire que la transformation se fait dans P et que $n' = O(n^c)$ avec c une constante. Maintenant on a pour tout j $p_j = p'_j - p''_j$, et donc on a $P_n = P_{n'-1} + P_(n'' - 2)$ et on a donc bien l'équivalence entre les deux SLP l'un n'utilisant qu'une seule soustraction.
Corollaire: La classe RAB est équivalente à $Gap - P$ uniforme.
Preuve: évidente à partir du théorème.
Corollaire: La classe $Gap-P$ est la clôture par la soustraction de la classe $\sharp P$.
Preuve: évidente à partir du corollaire précédent.

3.3 La Classe $\sharp_m P$

Petit résultat intéressant

Théorème: Soit f une fonction de $\{0,1\}^* \longrightarrow \mathbf{Z}$ alors les deux proposition suivante sont équivalentes:

1. $f \in \sharp_m P$
2. il existe une famille P_n de strate line programme $PRAB$ uniforme telque pour tout chaîne $x \in \{0,1\}^*$ on est:

$$f(x) = \widehat{P}_{|x|}(x)\ mod(m)$$

Preuve : évidente à partir de la caractérisation de $\sharp P$

Théorème: Soit f une fonction de $\{0,1\}^* \longrightarrow \mathbf{Z}$ alors les deux proposition suivante sont équivalentes:

1. $f \in Gap_m P$

2. il existe une famille P_n de strate line programme RAB uniforme telque pour tout chaîne $x \in \{0,1\}^*$ on est:

$$f(x) = \widehat{P}_{|x|}(x) \ mod(m)$$

Preuve: évidente d'après la caractérisation de $GapP$.

Théorème Pour toute famille P_n RAB $mod(m)$uniforme il existe une famille P'_n $PRAB$ $mod(m)$uniforme avec $n' = 0(n)$ pour c. **Preuves:** La seule différence entre les SLP RAB et $PRAB$ vient de l'instruction de soustraction qui n'existe pas dans le modèle $PRAB$ que l'on doit donc modéliser en un coût constant d'instruction. L'opposé $mod(m)$ d'un nombre s'obtient en le multipliant par $m-1$ donc en ajoutant l'opposé de ce nombre par la suite on obtient une soustraction par ce dit nombre donc finalement on obtient une instruction de soustraction ayant un coup majorée par $m-1$, m étant constant. **Corollaire:**$\sharp_m P = Gap_m P$

4 Les Classe IAB et AB

4.1 Définitions

Définition: On définit la classe IAB de SLP par induction à l'aide des instructions suivante:

1. p_k est une constante sur $\{0,1\}$
2. $p_k = 1 - p_i$ où $i \leq k$.
3. $p_k = x_i \times x_j$ où $i, j < k$.(multiplication non retardée)
4. $p_k = p_j(x_i = 1)$ où $p_k = p_j(x_i = 0)$ qui correspond au polynôme p_j dans lequel x_i est remplacé soit par un soit par zéro; on a $i, j < k$.

Le faite que la multiplication ne soit pas retarde rend le modèle beaucoup plus puissant.
Définition: On définit la classe AB en remplaçant l'instruction 3 de la classe précédente par une règle $3'$ de la forme suivante:

$$3' \ p_k = x_i - x_j$$

4.2 Classe $PSPACE$

On s'intéressera dans cette parie au relation entre les classe $PSACE$, IAB et AB **Théorème:** Soit f une fonction caractéristique d'un langage L alors les deux proposition suivante sont équivalente.

1. $L \in PSPACE$.
2. Il existe une famille de SLP: $P_n \in IAB$ uniforme où pour tout entré sur $\{0,1\}^*$,

$$f(x) = \widehat{P}_{|x|}(x)$$

Preuve: La preuve est similaire à celle de $\sharp P = PRAB$

On va commencer par montrer l'implication (1) $\Rightarrow$ (2) en utilisant $PSPACE$ complétude de $TQBF$. Pour cela on va transformer les formules de $TQBF$ en polynôme. On va commencé par transformer toute les disjonctions $c_i \vee c_j$ par $\neg(\neg c_i \wedge \neg c_j)$. Après on transforme les $\exists p_i$ en $\neg\forall\neg p_i$. Ensuite on transforme inductivement la formule obtenue:

1. Chaque $c_i \wedge c_j$ est remplacée par l'instruction 4, $p_k = p_j \times p_i$ où p_j est le polynôme correspondant à c_j et p_i celui qui correspond à c_i.
2. Chaque $\neg c_i$ est remplacée par l'instruction 3, $p_k = \neg p_i$ où p_i est le polynôme correspondant à c_i.
3. Chaque $\forall x_i, c_i$ (avec c_i est une variable booléenne) est remplacée par les instructions 4 et 6, $p_k = p'_i \times p''_i$ où p'_i est le polynôme correspondant c_i dans lequel x_i est instancié à vrai et p''_i est le polynôme où x_i est instancié à faux.
4. Les variables et les constantes restent inchangées

Cette famille est clairement uniforme. Ensuite l'article démontre l'implication (2) $\Rightarrow$ (1) pour cela l'auteur utilise les machines de turing alternée polynomiale qui définissent une classe incluse dans $PSPACE$. Cependant je ne suis pas convaincu par la preuve proposée, notament le passage où on simule l'instruction (4).

Théorème: Soit f une fonction caractéristique d'un langage L alors les deux proposition suivante sont équivalente.

1. $L \in PSPACE$.
2. Il existe une famille de SLP: $P_n \in AB$ uniforme où pour tout entré sur $\{0,1\}^*$,

$$f(x) = \widehat{P}_{|x|}(x)$$

Idée de démonstration:

L'implication (1) $\Rightarrow$ (2) est trivial à partir du théorème précédent.

L'implication (2) $\Rightarrow$ (1) est basé sur l'argument que les nombres exponentiels ce code en un nombre polynomiaux de bit et que les opérations sur ces nombres ce font en temps polynomiaux.

5 Les classes $BP \oplus P$, $P \oplus P$ et $PRAB$

5.1 Définition au moyen de $PRAB$:

Théorème: Soit f une fonction de $\{0,1\}^* \longrightarrow \mathbf{Z}$ alors les deux proposition suivante sont équivalentes:

1. $f \in BP \oplus P$

2. il existe une famille P_n de strate line programme $PRAB$ uniforme telque pour tout chaîne à $n + m(n)$ variable (avec m un polynôme) dans $\{0,1\}^*$ tel qu'on est:

$$f(x) = \widehat{P}_{|x|}(x, y) mod(2)$$

Pour plus de deux tiers des y sur $\{0,1\}^{m(n)}$

Théorème: On a le même théorème pour $P \oplus P$
Preuve du premier théorème: Pour toute fonction $f' \in \oplus P$ il existe une fonction $g \in \sharp P$ telque $\forall x, y, f'(x, y) = g(x, y) mod(2)$. Soit $f \in BP \oplus P$ alors il existe $f'' \in \oplus P$ telque $f''(x, y) = f(x)$ pour plus de deux tiers des entrée sur y. Il existe alors une fonction $g \in \sharp P$ telque $\forall x, y, f''(x, y) = g(x, y) mod(2)$, et donc par voie de fait $f(x) = g(x, y) mod(2)$ pour au moins deux tiers des y.

La démonstration du deuxième théorème est similaire.
Lemme de TODA: On à l'inégalité suivante:

$$P \oplus P \subseteq P^{\sharp P}$$

Preuve: Remarquons d'abord que le polynôme $T(x) = 4x^3 + 3x^4$ à la propriétété suivante pour tout couple x, ε, on a: $x = -\varepsilon\ mod(2^k)$ implique $T(x) = -\varepsilon\ mod(2^{2k})$.

Soit $f \in P \oplus P$ alors il existe une famille $P_1, P_2, ...$ de $PRAB$ uniforme qui vérifie les conditions du théorème précédent associé à f. On note $T^{(k)} = T \circ ... \circ T$ (K fois). Soit $s = O(log\ m'(n))$ avec m un polynôme, alors on pose $\widehat{Q}_n = T_{(s)} \widehat{P}_n$ qui est calculable par un $PRAB$ (En effet chaque T augmente d'un facteur quatre le coût totale mais comme il y en a un nombre logarithmique le coût n'augmente que de manière polynomial.)

Maintenant considérons pour $n = x$ et $s < log\ m(n)$ le polynôme suivant modulo $2^{m(n)}$:

$$S_n(x) = \sum_{r \in \{0,1\}^{m(n)}} -\widehat{Q}_n(x, r)$$

Qui compte selon r le nombre de fois où $\widehat{P}_n(x, r)$. est paire. Donc $f(x) = 1$ si et seulement si $S_{|x|}(x)$ est supérieur à 2^{s-1}. On a vu que $-S_n$ est décrit par un $PRAB$ uniforme et donc $-S_{|x|}(x)$ est calculable dans $\sharp P$, et donc $f \in P^{\sharp P}$.

6 Conclusion

L'article que j'ai étudié a été publié en 1991.

Pour me forger un opinion sur l'article j'ai regardé les démonstrations présentée ici, dans leurs articles initiaux. J'ai remarquée qu'elles étaient très similaire cependant la plus part utilisaient des polynômes. À mon avis cet article permet de présenter une méthode générique utilisé sur ces classes de façon plus formel et ainsi de rendre les futures preuve plus propres.

J'ai ensuite regardé le page rank de l'article et celui des articles qui l'on cité. Le page rank de cet article est de 43 mais celui des articles qui le cite est nettement supérieur. Finalement je pense que cet article est intéressant mais pas majeur: les nouveau résultat n'ayant qu'une faible porté.

Quant à la méthode proposée elle me semble plutôt jolie: en effet les définitions des classes de SLP me paraîssent très intuitives et suivent un même schéma d'instruction qui ne diffère pas que sur quelques instruction.

Pour conclure je dirais que cet article propose une méthode attractive pour rendre les preuves plus propre entre les classes combinatoires et $PSPACE$ mais je ne pense pas que ce soit un article majeur du domaine.