Modèle de Blum, Shub et Smale
Théorème de transfert pour la question P = NP dans $\mathbb{R}$

Fabrice Ben Hamouda

2009-2010

Résumé

Le problème P = NP standard est très difficile. Pour avoir une nouvelle approche de cette question, d'autres modèles de calcul ont été créés. Les machines de Blum, Shub et Smale (BSS) en sont un exemple. Ce sont des machines de Turing qui peuvent manipuler directement des réels. Selon les opérations permises, le problème P = NP est résolu, équivalent au problème standard ou de nature inconnue. Après une présentation du modèle de Blum, Shub et Smale, nous prouverons l'équivalence du problème P = NP dans $(\mathbb{R}, +, -, =, <)$ (sans constante) avec le problème standard.

Introduction

Pour avoir une nouvelle approche du problème P = NP, Stephen Smale, Michael Shub et Lenore Blum ont proposé en 1989, dans l'article [1], un nouveau modèle de calcul : les machines de Blum, Shub et Smale (BSS) qui sont des machines de Turing capables de manipuler directement des réels ou même des éléments d'une structure quelconque (c'est-à-dire un ensemble muni d'opérations et de relations). On est libre de choisir les opérations et les relations autorisées.

Dans ce modèle, les réels sont des entités à part entière, il n'y a pas de problème d'arrondis par exemple. Il s'agit bien sûr d'un modèle théorique qui n'est pas réalisable en pratique. Mais ce modèle ouvre de nombreuses perspectives. Il permet en particulier l'introduction de l'analyse, de la topologie et de l'algèbre (que l'on utilisera d'ailleurs beaucoup) en informatique théorique, ce qui offre de nouveaux outils pour aborder la question P = NP, par exemple.

Il est relativement facile de montrer par exemple que P $\neq$ NP dans la structure $(\mathbb{R}, +, -, =)$[1]. Dans les structures $(\mathbb{R}, +, \times, <)$ et $(\mathbb{C}, +, \times, =)$, le problème P = NP est encore ouvert. Et ce n'est qu'en 2004 qu'a été trouvé une structure (non triviale) dans laquelle P = NP (voir [3] et [4]).

Dans ce papier, nous nous intéresserons encore à une autre structure : $(\mathbb{R}, +, -, =, <)$. Après une présentation du modèle BSS, nous montrerons un résultat préliminaire important ($\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$) puis nous prouverons le théorème énoncé et démontré dans l'article [2] qui dit approximativement[2] que : P = NP dans $(\mathbb{R}, +, -, =, <)$ si et seulement si P = NP dans le cas standard.

1 Modèle

1.1 Machines de Blum, Shub et Smale

On se donne tout d'abord un ensemble de fonctions et de relations sur $\mathbb{R}$. Par exemple, on peut choisir de n'utiliser que les fonctions 2-aires et 1-aire suivantes : addition, passage à l'opposé et identité, et de n'utiliser que la relation 2-aire suivante : l'égalité. On dira alors qu'on travaille dans la structure $(\mathbb{R}, +, -, =)$, que les fonctions du langage sont l'addition et le passage à l'opposé et que les relations du langage sont l'égalité. On supposera toujours l'identité comme une fonction du langage.

Les machines de Blum, Shub et Smale sont des machines de Turing à k bandes bi-infinies dont l'alphabet d'entrée est $\mathbb{R}$ et l'alphabet de bande est $\mathbb{R} \cup \{\#\}$ où # est le symbole de case vide. Chaque

[1] Notation pour dire que l'alphabet de bande est $\mathbb{R}$ et que les opérations permises sont l'addition, le passage à l'opposé et le test d'égalité.

[2] Pour un énoncé complet, il faut d'abord présenter le modèle BSS que l'on utilise.

bande possède une tête de lecture et chaque tête de lecture est indépendante. Les cases des bandes sont numérotés par des éléments de $\mathbb{Z}$.

Chaque machine peut utiliser un nombre fini de réels $c_1, ..., c_p$ appelés *paramètres*. On suppose que 1 est toujours un paramètre de la machine.

Pour plus de clarté, le système de commande n'est pas un automate fini mais une liste finie d'instructions numérotées (de 1 à n par exemple). Voici les instructions possibles :

- *stop* : arrêter la machine.
- *déplace(i,d)* : déplacer la tête de lecture de la $i^{ème}$ bande à gauche (si $d = -1$) ou à droite (si $d = 1$).
- *calcule(f,i,j)* : f étant une fonction n-aire de la structure, lire les n valeurs $a_1, ..., a_n$ immédiatement à droite de la tête de lecture de la $i^{ème}$ bande, calculer $f(a_1, ..., a_n)$ et stocker le résultat dans la case pointée par la tête de lecture de la $j^{ème}$ bande. Comme l'identité est une fonction du langage, il est ainsi possible de copier une case dans une autre.
- *test(r,i,q,q')* : r étant une relation n-aire de la structure, lire les n valeurs $a_1, ..., a_n$ immédiatement à droite de la tête de lecture de la $i^{ème}$ bande et si ces valeurs satisfont la relation r, aller à l'instruction q sinon aller à l'instruction q'.
- *testvide(i,q,q')* : si la case pointée par la tête de lecture de la $i^{ème}$ bande est vide, aller à l'instruction q sinon aller à l'instruction q'.
- *ecrit(i,j)* : écrit le paramètre c_j ou 0 (si $j = -1$) ou # (si $j = -2$) dans la case pointée par la tête de lecture de la $i^{ème}$ bande.

Au départ, chaque tête de lecture est sur la case 0 de sa bande et toutes les bandes sont vides sauf la première bande qui contient l'entrée de taille finie n. La liste d'instructions est exécutée dans l'ordre naturel (en partant de l'instruction numéro 1) sauf quand il y un test. Si la machine s'arrête, on obtient le résultat du calcul en lisant la dernière bande.

Quand elle s'arrête, une machine qui résout un problème de décision (savoir si l'entrée est dans un langage donné) doit nécessairement retourner 0 ou 1 comme résultat, c'est-à-dire que la dernière bande contient un 0 ou un 1 en position 0 et que toutes les autres cases sont vides. On suppose de plus (sans perdre de généralité) que la machine se termine toujours par un test suivi (selon le résultat) d'une écriture d'un 0 ou d'un 1 sur la dernière bande (par une instruction *ecrit*).

Pour éviter que la machine ne se bloque, le symbole # est considéré comme un 0 quand il est donné en argument d'une fonction ou d'une relation.

Comme on ne s'intéressera qu'aux classes de complexité, on supposera de plus que les machines s'arrêtent sur toutes les entrées. Le temps de calcul sur une entrée de taille n est le maximum du nombre d'instructions exécutées (avant que la machine ne s'arrête) pour toutes les entrées de taille n.

Remarque 1.1.1. Il est possible de représenter une concaténation de deux entrées ($a = (a1, ..., a_n)$ et $b = (b_1, ..., b_n)$ où les a_i et les b_j sont des réels) avec l'alphabet de bande $\mathbb{R}$:

$$< a, b >= (a_1, 0, a_2, 0, ..., 0, a_n, 0, b_1, 1, ..., b_n, 1)$$

Remarque 1.1.2. On notera $\mathbb{R}^\infty$ l'ensemble des suite finies de réels de $\mathbb{R}$ (pour ne pas confondre avec $\mathbb{R}^*$). Mais on continuera à noter $\{0,1\}^*$ l'ensemble des suites finies de booléens.

On notera aussi souvent $\vec{x} = (x_1, \ldots, x_n)$ un mot de $\mathbb{R}^\infty$ ou de $\{0,1\}^*$.

1.2 Lien avec les machines standards

Il est facile de voir que dans la structure $(\{0,1\}, =)$, les machines définies ci-dessus sont équivalentes aux machines de Turing déterministes classiques (et qui s'arrêtent sur toutes les entrées) avec comme alphabet d'entrée $\{0,1\}$ et comme alphabet de bande $\{0,1,\#\}$. De plus le temps d'un calcul de taille n sur la machine transformée (dans un sens comme dans l'autre) dépend polynomialement du temps de calcul sur la machine initiale.

Une machine de Turing classique qui s'exécute en temps polynomial peut être transformée en une machine de Turing classique avec comme alphabet d'entrée $\{0,1\}$ et comme alphabet de bande $\{0,1,\#\}$ qui s'exécute en temps polynomial (il suffit de numéroter les symboles de l'alphabet de bande et d'écrire leur numéro en binaire plutôt que le symbole lui-même - sauf pour le symbole case vide #).

1.3 Classes de complexité

On note P et NP les classes de complexité standard.

Définition 1.3.1. Un problème (c'est-à-dire un langage) L est dans $\mathrm{P}_{\mathbb{R}}$ si et seulement s'il existe une machine BSS qui détermine si un mot x de taille n est dans le langage L en un temps majoré par un polynôme en n.

Définition 1.3.2. Un problème L est dans $\mathrm{NP}_{\mathbb{R}}$ si et seulement si :
- il existe un langage L' tel que $L = \{x \in \mathbb{R}^\infty \mid \exists y \in \mathbb{R}^\infty, (x,y) \in L'\}$ et tel que la taille maximale d'un y pour un x de taille n est majorée par un polynôme de taille n.
- et il existe une machine BSS qui prend en entrée un couple $< x, y >$ et qui détermine si le mot (x,y) est dans le langage L' en un temps polynomial en n, la taille de x.

Le y (qui n'est pas forcément unique) associé à un x est appelé *certificat* d'appartenance de x à L.

Définition 1.3.3. Un problème L est dans $\mathrm{NBP}_{\mathbb{R}}$ si et seulement s'il est dans $\mathrm{NP}_{\mathbb{R}}$ et qu'on peut n'utiliser que des *certificats booléens* (ou *binaires* - c'est-à-dire dans $\{0,1\}^*$).

On définit $\mathrm{P}^0_{\mathbb{R}}$, $\mathrm{NP}^0_{\mathbb{R}}$ et $\mathrm{NBP}^0_{\mathbb{R}}$ comme $\mathrm{P}_{\mathbb{R}}$, $\mathrm{NP}_{\mathbb{R}}$ et $\mathrm{NBP}_{\mathbb{R}}$ sauf que l'on impose aussi que la machine BSS considérée n'ait que 1 comme paramètre.

Il est évident que $\mathrm{NBP}_{\mathbb{R}} \subset \mathrm{NP}_{\mathbb{R}}$ car un problème $\mathrm{NBP}_{\mathbb{R}}$ admet un certificat booléen qui est en particulier un certificat de $\mathbb{R}^\infty$.

On remarque que pour tout problème $\mathrm{NBP}_{\mathbb{R}}$, il existe une machine BSS, qui s'exécute en temps exponentiel, qui le résout : il suffit de prendre une machine qui énumère et teste tous les certificats booléens possibles (par ordre de taille puis lexicographique par exemple). Pour un problème $\mathrm{NP}_{\mathbb{R}}$ quelconque, cela n'est a priori pas possible : car on ne peut pas énumérer tous les certificats réels ! Cela rend d'autant plus étonnant le résultat de la section suivante : $\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$ dans $(\mathbb{R}, +, -, =, <)$.

2 $\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$ dans $(\mathbb{R}, +, -, =, <)$

Désormais nous nous intéresserons uniquement à la structure $(\mathbb{R}, +, -, =, <)$.

Le but de cette partie est de montrer un résultat préliminaire nécessaire à la preuve du théorème 3.0.3 : $\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$ (et $\mathrm{NP}^0_{\mathbb{R}} = \mathrm{NBP}^0_{\mathbb{R}}$). L'inclusion $\mathrm{NBP}_{\mathbb{R}} \subset \mathrm{NP}_{\mathbb{R}}$ est évidente. L'idée de la preuve de l'inclusion inverse est de considérer un langage L de $\mathrm{NP}_{\mathbb{R}}$ et de remplacer, pour chaque mot du langage $(x_1, \ldots, x_n)$, son certificat réel $(y_1, \ldots, y_m)$ par une matrice $(\tilde{y})_{i,j}$ de rationnels (codés en binaire) qui représente :

$$\left(\sum_{j=1}^{n} \tilde{y}_{1,j} x_i + \sum_{j=1}^{p} \tilde{y}_{1,n+j} c_j, \ldots, \sum_{j=1}^{n} \tilde{y}_{m,j} x_j + \sum_{j=1}^{p} \tilde{y}_{m,n+j} c_j \right)$$

et de telle sorte que :

$$y_i \approx \sum_{j=1}^{n} \tilde{y}_{i,j} x_i + \sum_{j=1}^{p} \tilde{y}_{i,n+j} c_j$$

au sens où remplacer y_i par $\sum_{j=1}^{n} \tilde{y}_{i,j} x_i + \sum_{j=1}^{p} \tilde{y}_{i,n+j} c_j$ ne va pas changer le comportement de la machine BSS chargée de vérifier le certificat.

2.1 Remarques préliminaires

On peut remarquer qu'à une étape t' du calcul d'une machine BSS sur une entrée $(x_1, \ldots, x_n)$, les cases mémoires sont des combinaisons linéaires des paramètres $(c_1, \ldots, c_p)$ et des entrées $(x_1, \ldots, x_n)$ à coefficients entiers de taille au plus $t' + 1$. Cela se montre aisément par récurrence.

Il en résulte que les comparaisons effectuées par une machine BSS sur une entrée $(x_1, \ldots, x_n)$ en un temps t sont de la forme (quitte à changer les termes de membre) :
- ou bien $\lambda_1 x_1 + \ldots + \lambda_n x_n = \mu_1 c_1 + \ldots + \mu c_p$
- ou bien $\lambda_1 x_1 + \ldots + \lambda_n x_n < \mu_1 c_1 + \ldots + \mu c_p$

avec les λ_i et les μ_j des entiers de taille au plus t.

C'est pourquoi nous nous intéresserons à la manipulation des entiers et des rationnels par des machines classiques et nous verrons quelques lemmes d'algèbre linéaire avant de prouver le théorème : $\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$.

2.2 Manipulation des entiers et des rationnels

Définition 2.2.1. On appelle taille d'un entier n la partie entière supérieure de $\log_2 |n|$. C'est le nombre de chiffres de l'entier écrit en binaire (sans le signe). La taille d'un rationnel est le maximum de la taille de son numérateur et de son dénominateur quand il est écrit sous forme irréductible.

Un entier peut être représenté par la suite des bits de sa valeur absolue (suite de taille au plus n) plus un bit pour le signe. Un rationnel peut être représenté comme un couple de deux entiers (numérateur et dénominateur).

Regardons maintenant quelques propositions sur la manipulation des rationnels écrits sous forme binaire par une machine BSS. Ces propositions sont démontrées en annexe page 14.

Proposition 2.2.2. *Soit $\mathcal{M}$ une machine* BSS *qui s'exécute en temps polynomial en la taille de son entrée. Alors il existe une machine* BSS *$\mathcal{M}'$ qui s'exécute en temps polynomial, qui prend en entrée une suite finie de rationnels représentés sous forme binaire et qui simule le fonctionnement de $\mathcal{M}$ sur ces rationnels.*

On peut de plus imposer que $\mathcal{M}'$ utilise les mêmes paramètres que M.

Corollaire 2.2.3. *Soit $\mathcal{M}$ une machine* BSS *qui s'exécute en temps polynomial en la taille de son entrée. On suppose que l'entrée de $\mathcal{M}$ est un couple $< \vec{x}, \vec{y} >$ ($\vec{x} = (x_1, \ldots, x_n)$ et $\vec{y} = (y_1, \ldots, y_m)$). Alors il existe une machine* BSS *$\mathcal{M}'$ qui s'exécute en temps polynomial, qui prend en entrée un vecteur de réels $\vec{x}$ une matrice $(y_{i,j})$ de rationnels représentés sous forme binaire (au lieu d'une suite finie de réels représentés par des réels) et qui effectue le même calcul que $\mathcal{M}$ effectuerait sur le couple*

$$< \vec{x}, (y_{1,1}x_1 + \ldots + y_{1,n}x_n, \ldots, y_{m,1}x_1 + \ldots + x_{m,n}y_n) >$$

On peut de plus imposer que $\mathcal{M}'$ utilise les mêmes paramètres que M.

Proposition 2.2.4. *Soit $\mathcal{M}$ une machine* BSS *ayant 1 pour seul paramètre et qui s'exécute en temps polynomial en la taille de son entrée. Alors il existe une machine de Turing classique $\mathcal{M}'$ qui s'exécute en temps polynomial, qui prend en entrée une suite finie de rationnels représentés sous forme binaire (au lieu d'une suite finie de réels représentés par des réels) et qui effectue le même calcul que $\mathcal{M}$ (autrement dit $\mathcal{M}'$ simule $\mathcal{M}$ sur un vecteur de rationnels écrits en binaire).*

Avant de voir quelques lemmes d'algèbre linéaire nous avons encore besoin du lemme suivant.

Lemme 2.2.5. *Le déterminant d'une matrice carrée de taille n composée d'entiers de taille au plus L est de taille au plus $n(m + \log n) + 1$.*

Démonstration. Le déterminant d'une telle matrice est une somme de $n!$ produits de n coefficients de la matrice. Comme chaque entier est de taille au plus L, chaque produit est inférieur ou égal à $\left(2^L\right)^n = 2^{Ln}$ et le déterminant est inférieur ou égal à $n!2^{Ln}$. Sa taille est donc au plus $\lceil \log_2 n!2^{Ln} \rceil \leq n(\log_2 n + L) + 1$. □

2.3 Lemmes d'algèbre linéaire

Nous avons besoin d'un premier lemme d'algèbre linéaire (lemme 2.3.1 que nous prouverons en annexe page 15) pour montrer que tout système d'inéquations et d'équations linéaires à coefficients entiers pas trop grands admet une solution combinaison linéaire des seconds membres à coefficients entiers pas trop grands (lemme 2.3.2).

Lemme 2.3.1. *Considérons un système linéaire de la forme (d'inconnues $x_1, \ldots, x_n$) :*

$$\begin{cases} \lambda_{1,1}x_1 + \ldots + \lambda_{1,n}x_n & = & a_1 \\ \vdots & \vdots & \vdots \\ \lambda_{m,1}x_1 + \ldots + \lambda_{m,n}x_n & = & a_m \end{cases}$$

où $a_i \in \mathbb{R}$ pour tout i et où $\lambda_{i,j} \in \mathbb{R}$ pour tout couple (i,j). Si ce système a une solution positive, on peut remplacer certaines inconnues par 0 de façon à ce que le système résultant ait une unique solution positive.

Lemme 2.3.2. *Considérons un système S de la forme (d'inconnues $x_1, \ldots, x_n$) :*

$$\begin{cases} \lambda_{1,1}x_1 + \ldots + \lambda_{1,n}x_n & \geq & a_1 \\ \vdots & \vdots & \vdots \\ \lambda_{m,1}x_1 + \ldots + \lambda_{m,n}x_n & \geq & a_m \\ \mu_{1,1}x_1 + \ldots + \mu_{1,n}x_n & > & b_1 \\ \vdots & \vdots & \vdots \\ \mu_{p,1}x_1 + \ldots + \mu_{p,n}x_n & > & b_p \end{cases}$$

où $a_i, b_j \in \mathbb{R}$ pour tout i,j et où $\lambda_{i,j}, \mu_{i,j}$ sont des entiers de taille L. Ce système a une solution sous la forme d'un vecteur de combinaisons linéaires d'au plus $2n+1$ des a_k et b_k, et à coefficients rationnels de taille au plus $4n(L + \log(2n)) + 4$.

Démonstration. Ajoutons d'abord une inconnue t et multiplions toutes les inéquations et équations par $1+t$. Pour chaque inconnue x_i, ajoutons deux nouvelles inconnues u_i et v_i et remplaçons x_i par $\frac{u_i - v_i}{1+t}$ dans S. Ajoutons aussi une inconnue s_j par inégalité large telle que : $s_j = (1+t)(\lambda_{j,1}x_1 + \ldots + \lambda_{j,n}x_n - a_j)$ et une inconnue w_j par inégalité stricte telle que : $w_j = (1+t)(\mu_{j,1}x_1 + \ldots + \mu_{j,n}x_n - b_j) - 1$

On pose S' le système suivant :

$$\begin{cases} \lambda_{1,1}(u_1 - v_1) + \ldots + \lambda_{1,n}(u_n - v_n) & = & (1+t)a_1 + s_1 \\ \vdots & \vdots & \vdots \\ \lambda_{m,1}(u_1 - v_1) + \ldots + \lambda_{m,n}(u_n - v_n) & = & (1+t)a_m + s_m \\ \mu_{1,1}(u_1 - v_1) + \ldots + \mu_{1,n}(u_n - v_n) & = & (1+t)b_1 + w_1 + 1 \\ \vdots & \vdots & \vdots \\ \mu_{p,1}(u_1 - v_1) + \ldots + \mu_{p,n}(u_n - v_n) & = & (1+t)b_p + w_p + 1 \end{cases}$$

On remarque que :

- si S a une solution $(x_1, \ldots, x_n)$, alors, pour t suffisamment grand,

$$(1+t)(\mu_{1,1}x_1 + \ldots + \mu_{1,n}x_n) - (1+t)b_1 > 1$$

  car $\mu_{1,1}x_1 + \ldots + \mu_{1,n}x_n - b_1 > 0$. Et donc S' a une solution positive (en faisant les changements de variables : $u_i = (1+t)\max(0, x_i)$, $v_i = (1+t)\max(0, -x_i)$, et en choisissant w_j et s_j comme précisé plus haut).
- si S' a une solution positive, alors S a une solution $(x_1, \ldots, x_n)$, il suffit de poser $x_i = \frac{u_i - v_i}{1+t}$.

Appliquons maintenant le lemme 2.3.1 au système S. En annulant certaines inconnues, on obtient un système S'' qui n'a qu'une seule solution positive. S'il reste des inconnues w_j ou s_j dans S'', elles n'apparaissent que dans au plus une équation. On peut donc exprimer facilement w_j en fonction des autres inconnues et on suppose donc que les w_j et les s_j n'apparaissent pas dans S''.

Écrivons le système S'' sous forme matricielle : $A\vec{Y} = \vec{B}$ avec $\vec{Y}$, $\vec{B}$ le vecteur des seconds membres (chaque coordonnée de $\vec{B}$ est de la forme $b_j + 1$ ou a_j) et A une matrice. Comme le système a une solution, on peut extraire de A une sous-matrice A' inversible. Posons $\vec{B'}$ la matrice extraite de $\vec{B}$ correspondante.

D'après la règle de Cramer, Y_j (la $j^{ème}$ coordonnée de l'unique solution $\vec{Y}$ du système) est de la forme $\dfrac{\det \tilde{A'}_j}{\det A'}$ où $\tilde{A'}_j$ est la matrice A où la $j^{ème}$ colonne est remplacée par $\vec{B'}$. En développant selon la $j^{ème}$ colonne et en notant $\tilde{A'}_{i,j}$ la matrice $\tilde{A'}$ privée de sa $i^{ème}$ ligne et de sa $j^{ème}$ colonne, on a :

$$Y_j = \sum_{i=1}^{n} (-1)^{i+j} \frac{\det \tilde{A'}_{i,j}}{\det A'} b_j$$

La matrice $\tilde{A'}$ a au plus $2n+1$ colonnes car les inconnues restantes (celles correspondant aux Y_j) ne peuvent être que u_j, v_j (pour $j \in [\![1, n]\!]$) ou t. Donc les Y_j (et en particulier t) sont des combinaisons

linéaires d'au plus $2n+1$ des a_k et b_k, et à coefficients rationnels de taille au plus $2n(L+\log(2n))+1$. Comme on a : $x_j = \frac{u_j - v_j}{1+t}$, les x_j sont des combinaisons linéaires d'au plus $2n+1$ des a_k et b_k, et à coefficients rationnels de taille :

$$(2n(L+\log(2n))+1+1)+(2n(L+\log(2n))+1+1) = 4n(L+\log(2n))+4$$

. □

Corollaire 2.3.3. *Le lemme précédent est aussi vrai dans le cas d'un système linéaire de n inconnues où les coefficients sont de taille au plus L et où l'on autorise des inégalités larges ($\leq$ et $\geq$), des inégalités strictes ($<$ et $>$) et aussi des égalités ($=$).*

Démonstration. On remarque en effet que :
- $\alpha \leq \beta \iff -\alpha \geq -\beta$
- $\alpha < \beta \iff -\alpha > -\beta$
- $\alpha = \beta \iff \alpha \geq \beta \quad$ et $\quad \alpha \leq \beta$

□

2.4 $\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$

Théorème 2.4.1. *Dans la structure $(\mathbb{R}, +, -, =, <)$, on a :*

$$\mathrm{NP}_{\mathbb{R}} = \mathrm{NBP}_{\mathbb{R}}$$

Démonstration. L'inclusion $\mathrm{NBP}_{\mathbb{R}} \subset \mathrm{NP}_{\mathbb{R}}$ est évidente. Montrons l'inclusion inverse.

Soit L un problème de $\mathrm{NP}_{\mathbb{R}}$ et $\mathcal{M}$ une machine BSS qui vérifie la propriété de la définition 1.3.2. Soit $\vec{x} = (x_1, \ldots, x_n)$ un mot de L et $\vec{y} = (y_1, \ldots, y_m)$ un certificat de ce mot (lorsqu'on fait varier n, m dépend polynomialement de n). Soit t le temps d'exécution de $\mathcal{M}$ sur une entrée de taille n.

D'après les remarques de la sous-section 2.1, les tests effectués par la machine sont de la forme :
- ou bien $\lambda_{i,1}y_1 + \ldots + \lambda_{i,m}y_m = a_i$
- ou bien $\lambda_{i,1}y_1 + \ldots + \lambda_{i,m}y_m < a_i$

avec les $\lambda_{i,j}$ des entiers de taille au plus t et les a_i des combinaisons linéaires des x_k et des paramètres c_k de $\mathcal{M}$.

On construit un système S d'équations et d'inéquations en ajoutant, pour chaque $(m+1)$-uplet $(\lambda_{i,1}, \ldots, \lambda_{i,m}, a_i)$ qui apparaît dans un test de la machine $\mathcal{M}$, l'équation (ou inéquation) vérifiée par $\vec{y}$ parmi les trois suivantes :
- $\lambda_{i,1}y_1 + \ldots + \lambda_{i,m}y_m < a_i$
- $\lambda_{i,1}y_1 + \ldots + \lambda_{i,m}y_m = a_i$
- $\lambda_{i,1}y_1 + \ldots + \lambda_{i,m}y_m > a_i$

Ce système S vérifie les hypothèses du corollaire 2.3.3 donc admet une solution sous la forme d'un vecteur de combinaisons linéaires des a_i à coefficients rationnels de taille au plus $4m(t+\log(2m))+4$ ce qui est majoré par un polynôme en n. On note ce vecteur :

$$\vec{\tilde{y}} = (\tilde{y}'_{1,1}a_1 + \ldots + \tilde{y}'_{1,N}a_N, \ldots, \tilde{y}'_{m,1}a_1 + \ldots + \tilde{y}'_{m,N}a_N)$$

où $N \leq t$ est le nombre d'équations et inéquations du système S (et donc le nombre de a_i).

Comme les a_i des combinaisons linéaires des x_k et des paramètres c_k de $\mathcal{M}$, on peut aussi écrire :

$$\vec{\tilde{y}} = \left(\sum_{j=1}^{n} \tilde{y}_{1,j}x_i + \sum_{j=1}^{p} \tilde{y}_{1,n+j}c_j, \ldots, \sum_{j=1}^{n} \tilde{y}_{m,j}x_j + \sum_{j=1}^{p} \tilde{y}_{m,n+j}c_j \right)$$

On donne alors comme certificat booléen la matrice $(\tilde{y}_{i,j})$, matrice représentable par une suite booléenne majorée par un polynôme en n car N, m et la taille des $y_{i,j}$ le sont déjà.

On remarque que $\vec{\tilde{y}}$ est aussi un certificat de $\vec{x}$ pour $\mathcal{M}$ car les tests par $\mathcal{M}$ seront les mêmes et auront le même résultat que l'entrée soit $< \vec{x}, \vec{y} >$ ou $< \vec{x}, \vec{\tilde{y}} >$ (on le montre par l'absurde, en considérant la première étape de calcul non identique entre $< \vec{x}, \vec{y} >$ et $< \vec{x}, \vec{\tilde{y}} >$). On conclut par le corollaire 2.2.3 : il existe une machine $\mathcal{M}'$ qui convient. □

Corollaire 2.4.2. *Dans la structure* $(\mathbb{R}, +, -, =, <)$*, on a :*

$$\mathrm{NP}^0_{\mathbb{R}} = \mathrm{NBP}^0_{\mathbb{R}}$$

Démonstration. Dans la preuve précédente, on remarque que $\mathcal{M}'$ utilise les mêmes paramètres que $\mathcal{M}$ (voir énoncé du corollaire 2.2.3 en particulier). □

3 Un théorème de transfert

Nous pouvons maintenant nous intéresser à la démonstration du théorème de transfert annoncé :

Théorème 3.0.3. *Dans la structure* $(\mathbb{R}, +, -, =, <)$*,* $\mathrm{P}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}}$ *si et seulement si* $\mathrm{P} = \mathrm{NP}$*.*

3.1 $\mathrm{P}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}} \Longrightarrow \mathrm{P} = \mathrm{NP}$

Cette implication est relativement aisée à démontrer.

Démonstration. Supposons $\mathrm{P}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}}$ et considérons un langage booléen L dans NP. Ce langage est aussi dans $\mathrm{NP}^0_{\mathbb{R}}$ car il est aisé de transformer une machine de Turing classique en une machine BSS (n'utilisant que le paramètre 1) en multipliant le temps d'exécution par une constante. Donc L est un langage $\mathrm{P}^0_{\mathbb{R}}$ et il existe une machine BSS $\mathcal{M}$ qui s'exécute en temps polynomial et qui reconnaît L.

Grâce à la proposition 2.2.4 appliquée dans le cas où les seules entrées possibles sont des suites de 0 et 1 (et non pas des rationnels ou des réels), il existe une machine de Turing classique qui reconnaît L en temps polynomial et $L \in \mathrm{NP}$.

On a ainsi montré $\mathrm{NP} \subset \mathrm{P}$ et donc $\mathrm{NP} = \mathrm{P}$. □

3.2 $\mathrm{P} = \mathrm{NP} \Longrightarrow \mathrm{P}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}}$

L'idée de la démonstration est la suivante : on considère un problème L de $\mathrm{NBP}^0_{\mathbb{R}}$ (car $\mathrm{NBP}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}}$ d'après le corollaire 2.4.2) et on montre que l'on peut calculer, pour tout vecteur de réels $\vec{x} = (x_1, \ldots, x_n)$, un vecteur de rationnels $\vec{\tilde{x}} = (\tilde{x}_1, \ldots, \tilde{x}_n)$ suffisamment proche de $\vec{x}$ pour que : $\vec{x} \in L$ si et seulement si $\vec{\tilde{x}} \in L$. Comme les certificats utilisés sont binaires, on peut alors appliquer la proposition 2.2.4 et on en déduit qu'il existe une machine capable de dire en temps polynomial (car on a supposé $\mathrm{P} = \mathrm{NP}$) si $\vec{\tilde{x}} \in L$ ou non.

Pour cela, nous avons besoin d'une définition et d'un lemme.

Définition 3.2.1. Un *oracle* (booléen de NP) est un mécanisme qui permet à une machine de Turing classique ou BSS d'interroger à un coût unitaire un langage L fixé de classe NP, autrement dit de savoir si un mot écrit sur une de ses bandes est dans le langage L ou non.

Remarque 3.2.2. Comme dans toute cette sous-section et la suivante, on suppose $\mathrm{NP} = \mathrm{P}$, on peut utiliser des oracles NP dans toute machine de Turing classique ou BSS (mais l'entrée de l'oracle est toujours composée de 0 et de 1).

Remarque 3.2.3. L'utilisation d'oracle nécessite parfois de convertir des entiers encodés comme des réels (qui prennent une case d'une machine BSS) en entiers encodés en binaire (qui prennent plusieurs cases d'une machine de Turing classique ou BSS) et vice versa.

La conversion d'un entier binaire en un réel est expliquée dans la preuve de la proposition 2.2.2. La conversion inverse (d'un réel x) s'effectue par un algorithme similaire à celui présenté un peu plus loin (page 8) où L' est simplement le langage des réels strictement inférieurs à x (un test du type $y \in L'$ est une simple comparaison).

Dans la suite, la plupart du temps, il suffira d'utiliser la proposition 2.2.4 pour vérifier que les oracles considérés sont bien des langages NP.

Lemme 3.2.4. *Prenons les notations du corollaire 2.3.3 et plaçons nous sous ses hypothèses, dans le cas où les seconds membres sont des entiers de taille L (comme les coefficients des équations et inéquations). Supposons aussi que L est majoré par un polynôme en n (le nombre d'inconnues) et que le nombre d'équations et inéquations aussi. Si* $\mathrm{P} = \mathrm{NP}$ *alors il existe une machine de Turing classique capable de trouver, en temps polynomial, un rationnel solution du système S considéré. Ce rationnel est alors nécessairement de taille majorée par un polynôme en n.*

Démonstration. Comme L est majoré par un polynôme en n il existe k et K entiers tels que : $L < Kn^k$ (prendre k le degré du polynôme et K la somme de ses coefficients plus 1 par exemple).

Nous savons déjà grâce au corollaire 2.3.3 qu'il existe une solution de S, combinaison linéaire des seconds membres et à coefficients rationnels de taille majorée par un polynôme en n. Cette solution est rationnelle de taille majorée par un polynôme en n car les seconds membres sont des rationnels de taille au plus L. De plus nous pouvons même majorer ce polynôme par un terme de la forme $K'n^{k'} - 1$ avec K' et k' deux nouvelles constantes.

Le problème de savoir pour un entier l (de taille au plus $K'n^{k'}$) s'il existe une solution de S telle que le numérateur de la première coordonnée est strictement inférieur à cet entier l est dans la classe NP = P (le certificat est la solution correspondante). Notons L' le langage correspondant à ce problème. Si on note l_0 la plus petite première coordonnée possible, on a : $L' = \{l \in \mathbb{Z} | l > l_0\}$ et $l_0 = \min L'$. Ainsi par dichotomie on peut trouver un numérateur possible pour la première coordonnée. Voici un algorithme qui convient :

$l \leftarrow -1$
for $j = 1$ to $K'n^{k'}$ **do**
  $l \leftarrow l + l$ {à la fin de la boucle $l = -2^{K'n^{k'}}$}
for $i = K'n^{k'}$ to 0 **do**
  $l' \leftarrow 1$
  for $j = 1$ to i **do**
    $l' \leftarrow l' + l'$ {à la fin de la boucle $l' = 2^i$}
  if $l + l' \notin L'$, ce qui est équivalent à $l + l' \leq l_0$ **then**
    $l \leftarrow l + l'$
return l

L'idée de cet alorithme est de partir de $l = -2^{K'n^{k'}}$ et de maintenir l'invariant suivant dans la boucle **for i** : $1 - 2^i \leq l - l_0 \leq 0$.

Cette recherche dichotomique s'effectue en temps logarithmique par rapport à la taille de l'intervalle d'entiers à tester ($[\![-2^{K'n^{k'}}, 2^{K'n^{k'}}]\!]$) et donc en temps polynomial en n.

On dit que l'on a utilisé un oracle NP (l'oracle correspondant au langage L') pour résoudre le problème demandé.

Ensuite on fixe le numérateur trouvé. Le problème de savoir pour un entier l (de taille au plus $K'n^{k'}$) s'il existe une solution de S telle que le denominateur de la première coordonnée est strictement inférieure à cet entier l (le numérateur de la première coordonnée est donnée en entrée) est dans la classe NP = P. Ainsi par dichotomie on peut trouver un dénominateur possible pour la première coordonnée et ainsi de suite pour les n coordonnées du vecteur solution cherché.

Cet algorithme s'effectue en temps polynomial d'où le résultat. □

Démonstration de $\mathrm{P} = \mathrm{NP} \Longrightarrow \mathrm{P}^0_{\mathbb{R}} = \mathrm{NP}^0_{\mathbb{R}}$. Supposons que P = NP et considérons un problème L dans $\mathrm{NP}^0_{\mathbb{R}} = \mathrm{NBP}^0_{\mathbb{R}}$ (d'après corollaire 2.4.2). Soit $\mathcal{M}$ une machine BSS qui vérifie la propriété de la définition 1.3.2. Notons $t = t(n)$ le temps d'exécution de $\mathcal{M}$ sur une entrée de taille n ($t = t(n)$ dépend polynomialement de n). Soit $\vec{x} = (x_1, \ldots, x_n) \in L$ le mot à tester et $\vec{y}$ un certificat booléen de ce mot.

Plaçons nous dans l'espace affine $\mathbb{R}^n$.

Considérons $\mathcal{H}_n = \{h_1, \ldots, h_m\}$ l'ensemble des hyperplans affines d'équations $\lambda_1 x_1 + \ldots \lambda_n x_n = b$ avec λ_i et b des entiers de taille au plus t.

Chaque hyperplan h_i d'équation $\lambda_1 x_1 + \ldots + \lambda_n x_n = b$ définit deux *demi-espaces* d'équations $\lambda_1 x_1 + \ldots + \lambda_n x_n > b$ et $\lambda_1 x_1 + \ldots + \lambda_n x_n < b$. On note h_i^+ l'un de ces demi-espaces et h_i^- l'autre.

Nous savons déjà d'après les remarques de la sous-section 2.1, que l'ensemble des tests effectués par $\mathcal{M}$ forme un système d'équations et d'inéquations linéaires à coefficients entiers de taille au plus t et à second membre entier de taille aussi au plus t. Or une inéquation (stricte) de cette forme n'est qu'un test du type : $x \in h_i^+$ ou $x \in h_i^-$ pour un certain i et une équation n'est q'un test de la forme : $x \in h_i$. D'où cette idée d'introduire l'ensemble $\mathcal{H}_n$ qui représente, d'une certaine manière, l'ensemble des tests possibles pour $\mathcal{M}$ sur une entrée de taille n.

On définit une relation d'équivalence $\sim$ sur les points de $\mathbb{R}^n$ de la façon suivante :

$$x' \sim x'' \iff \forall i \in [\![1, m]\!], \begin{cases} x'' \in h_i^+ & \text{si} \quad x' \in h_i^+ \\ x'' \in h_i & \text{si} \quad x' \in h_i \\ x'' \in h_i^- & \text{si} \quad x' \in h_i^- \end{cases}$$

Les classes d'équivalence de cette relation sont appelées des *cellules*. Sur le schéma ci-contre, le segment vert (privé de ses extrémités) et le triangle vert sont des cellules de l'ensemble des droites noires (les hyperplans en dimension 2).

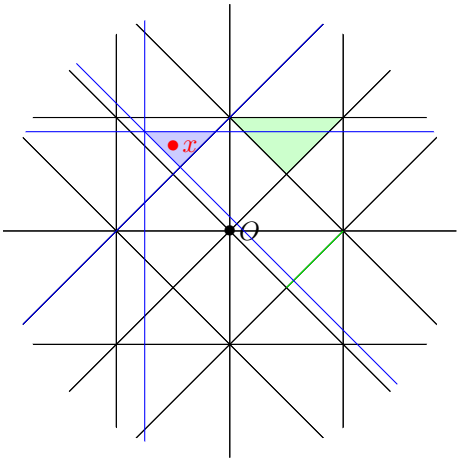

FIG. 1 – Cellules en dimension 2.

Chaque classe d'équivalence est soit toute entière incluse dans le langage L soit d'intersection vide avec L. En effet les éléments d'une même classe d'équivalence ont les mêmes résultats à chacun des tests de la machine $\mathcal{M}$.

Supposons que nous possédons une machine BSS capable de trouver la cellule de x, ou plus précisément capable de trouver, en temps polynomial en n, un système d'équations et d'inéquations affines à coefficients entiers de taille polynomiale en n telle que l'ensemble des points vérifiant ce système est contenu dans une cellule et contient x. Nous dirons qu'une telle machine résout le problème de *localisation d'un point par rapport à* $\mathcal{H}$. Nous montrerons l'existence de cette machine dans la sous-section 3.3 car cette preuve est technique et la présenter ici nuirait à la clarté de l'exposé. Comme la machine s'exécute en temps polynomial en n, la taille du système est aussi polynomial en n.

Sur l'exemple ci-dessus, un système d'inéquations possible est représenté par les droites bleues (les droites bleues sont les hyperplans ; les demi-espaces à choisir sont facile à voir).

Il ne reste plus qu'à vérifier si l'ensemble solution du système trouvé est inclu dans L ou non. Pour cela, commençons par trouver un rationnel solution de taille majorée par un polynôme en n (ce qui est faisable par une machine de Turing classique d'après le lemme 3.2.4).

Ensuite on remarque que la proposition 2.2.4 affirme l'existence d'une machine de Turing classique qui prend comme entrée un couple $< \vec{x'}, \vec{y'} >$ avec $\vec{x'}$ un vecteur de rationnels et $\vec{y'}$ un certificat booléen et qui vérifie si $\vec{y'}$ est bien un certificat booléen pour le vecteur $\vec{x'}$ (considéré comme vecteur de réels) au sens de la machine BSS initiale $\mathcal{M}$.

On peut donc vérifier, en temps polynomial, si le rationnel calculé plus haut est ou non dans L ce qui revient au même, d'après ce qui précède, à vérifier si x est dans L ou non.

□

On a ainsi démontré les deux implications du théorème 3.0.3.

3.3 Localisation d'un point

Il nous reste à montrer le théorème suivant pour que la preuve soit complète.

Théorème 3.3.1. *Reprenons les notations de la démonstration précédente. Si* P = NP*, il existe une machine* BSS *capable de localiser un point par rapport à* $\mathcal{H}$ *en temps polynomial en* n*.*

La preuve étant très technique, nous nous appuierons beaucoup sur des dessins. Elle sera bien moins détaillée (que les démonstrations précédentes) et utilisera des parties de d'autres preuves.

Désormais nous considérerons le vecteur $\vec{x}$ comme un point de l'espace affine $\mathbb{R}^n$ et nous le noterons x.

Avant de s'intéresser à la démonstration du théorème, nous avons besoin d'une nouvelle définition et de quelques lemmes.

Définition 3.3.2. Un réel r est un *granularité* (*coarseness* en anglais dans les articles [2] et [5]) d'un ensemble d'hyperplans (affines) $h_1, \ldots, h_m$ de $\mathbb{R}^n$ si pour toute boule de rayon r ou bien $\{h_i | h_i \cap B \neq \emptyset\} = \emptyset$ ou bien $\bigcap_{h_i \cap B \neq \emptyset} \neq \emptyset$.

Cela signifie que pour toute boule de rayon r, ou bien aucun hyperplan ne coupe cette boule, ou bien tous les hyperplans coupant cette boule ont un point commun. Notons que si r est une granularité alors tous les réels strictement plus petits en sont aussi une.

On suppose aussi que $t(n)$ est un polynôme à coefficients entiers en n quitte à remplacer $t(n)$ par un de ses majorants.

Dans l'article [2], il est expliqué que l'on peut choisir r_n tel que :

$$\frac{1}{r_n} = n^{n^2} 2^{2n^2 t(n) + O(n^2)}$$

Ce résultat est vraiment démontré dans [5]. Comme $n^{n^2} = 2^{n^2 log_2(n)} = 2^{O(n^3)}$, on en déduit le lemme suivant :

Lemme 3.3.3. *Il existe un entier κ tel que si r_n est tel que :*

$$\frac{1}{r_n} = 2^{2n^2 t(n) + \kappa n^3}$$

$\sqrt{n}\frac{r_n}{2}$ est une granularité de $\mathcal{H}_n$.

On pose r_n tel que : $\frac{1}{r_n} = 2^{2n^2 t(n) + \kappa n^3}$.

Remarque 3.3.4. Soit C un hypercube de $\mathbb{R}^m$ $(m \leq n)$ de côté r_n, C est inscrit dans une boule de rayon $\frac{\sqrt{m} r_n}{2} \leq \frac{\sqrt{n} r_n}{2}$ donc ou bien $\{h_i | h_i \cap C \neq \emptyset\} = \emptyset$ ou bien $\bigcap_{h_i \cap C \neq \emptyset} \neq \emptyset$.

Notons que l'on a pris soin de choisir $\frac{1}{r_n}$ comme puissance entière de 2 ce qui facilitera les recherches par dichotomie. Notons aussi que la taille de $\frac{1}{r_n}$ est polynomiale en n. Mieux encore, pour tout réel x on peut calculer, en temps polynomial, $\frac{1}{r_n} x$ par l'algorithme suivant :

Require: x : un réel
  for $i = 1$ to $2n^2 t(n) + \kappa n^3$ **do**
    $x \leftarrow x + x$
  return x

et comme $2n^2 t(n) + \kappa n^3$ est un polynôme en n à coefficients entiers, on peut le calculer aussi en temps polynomial (on commence par additionner n fois n à lui-même pour obtenir n^2 puis on additionne n^2 fois n à lui-même pour obtenir n^3 etc jusqu'à n^d où d est le degré du polynôme - tout cela a un coût polynomial en n - ; ensuite on additionne ces différents monômes pour obtenir le polynôme voulu ce qui se fait en coût constant car les coefficients du polynôme sont fixés une fois pour toute).

Lemme 3.3.5. *La localisation d'un point dans l'hypercube $[-1, 1]^n$ peut s'effectuer en temps polynomial si* P = NP.

Démonstration. Commençons par le cas $n = 1$. Les hyperplans sont alors des points. Il s'agit de localiser le point x sur le segment $[-1, 1]$. Découpons virtuellement le segment $[-1, 1]$ en $\frac{1}{r_n}$ segments (ouverts à gauche sauf le dernier) de taille r_n. Par une recherche dichotomique, on peut trouver dans lequel de ces petits segments est x, autrement dit on peut trouver $k \in [\![-\frac{1}{r_n}, \frac{1}{r_n} - 1]\!]$ tel que $x \in [\![k r_n, (k+1) r_n]\!]$ (privé de son extrémité gauche sauf pour $k = \frac{1}{r_n} - 1$).

Comme il n'est pas possible de diviser pour faire la recherche dichotomique, on commence par multiplier x par $\frac{1}{r_n}$ et il n'y a plus qu'à faire une simple recherche dichotomique comme dans la démonstration du lemme 3.2.4 page 8.

Ensuite deux cas se présentent :

– Si le $[\![k r_n, (k+1) r_n]\!]$ ne contient aucun point de $\mathcal{H}_1$, alors retournons le système d'inéquations :

$$\begin{cases} \frac{1}{r_n} \quad x \quad < \quad k+1 \\ \frac{1}{r_n} \quad x \quad > \quad k \end{cases}$$

– Sinon, par définition de la granularité, tous les points de $[\![kr_n, (k+1)r_n]\!] \cap \mathcal{H}_1$ ont un point commun, autrement dit, il n'y a qu'un seul point de $\mathcal{H}_1$ dans $[\![kr_n, (k+1)r_n]\!]$. Pour le trouver, la méthode est la même que pour trouver un point rationnel solution d'un système d'équation : on utilise un oracle NP pour savoir si le point cherché est à gauche ou à droite d'un point donné en argument et on fait une recherche dichotomique (voir démonstration du lemme 3.2.4 page 8). Soit $\frac{p}{q} \in \mathcal{H}_1$, le point cherché.
  – Si $x = \frac{p}{q} \in \mathcal{H}_1$, on renvoie l'équation : $qx = p$
  – Sinon on renvoie le système d'inéquations suivant (dans le cas où $x < \frac{p}{q}$, l'autre cas étant symétrique) :

$$\begin{cases} q & x & < & p \\ \frac{1}{r_n} & x & > & k \end{cases}$$

Comme l'on peut savoir par un oracle NP dans quel cas on est, on a bien réussi à localiser x dans $[-1, 1]$.

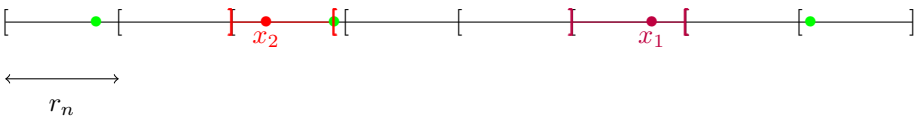

FIG. 2 – Schéma montrant les deux cas de la localisation d'un point dans $[-1, 1]$ - les points verts sont les hyperplans considérés (normalement ce sont tous les points rationnels de numérateur et de dénominateur de taille bornée mais pour des raisons de clarté, nous avons sélectionné moins de points) et les crochets rouges et violets indiquent quelles inéquations ont été choisies pour localiser les points x_1 et x_2.

Étudions maintenant le cas $n > 1$.
L'idée est de projeter intelligemment x sur une face[3] bien choisie de l'hypercube $[-1, 1]^n$. On se retrouve alors dans $[-1, 1]^{n-1}$ et on peut appliquer le même algorithme par récurrence.
Plus précisément, voici l'algorithme utilisé.

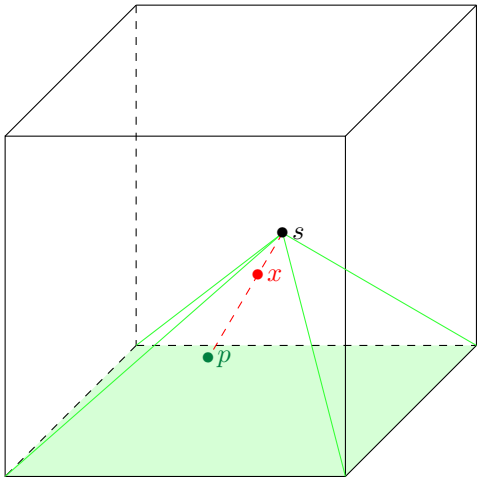

FIG. 3 – Exemple dans $\mathbb{R}^3$ où x est projeté sur la face du bas du cube.

[3]Dans toute cette démontration, une face d'un hypercube de dimension n est de dimension $n-1$. Pour l'hypercube $[-1, 1]^n$, une face est définie par (ou incluse dans) un hyperplan d'équation $x_i = 1$ ou $x_i = -1$ pour un certain $i \in [\![1, n]\!]$.

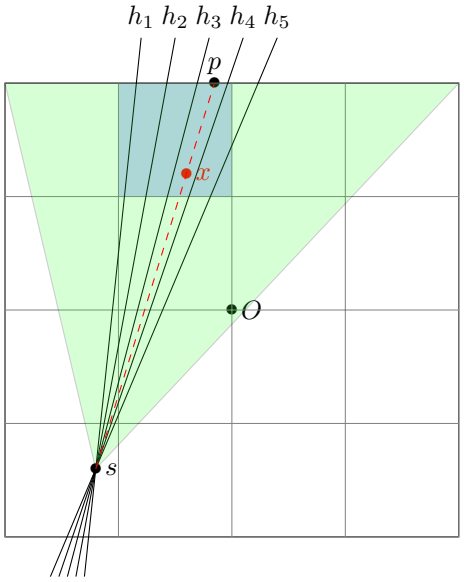

FIG. 4 – Exemple dans $\mathbb{R}^2$ du découpage de l'hypercube $[-1,1]^2$ (un carré).

On commence par découper l'hypercube $[-1,1]^n$, en de petits cubes de taille r_n (comme dans le cas $n=1$). Par recherche par dichotomie sur chacune des coordonnées de x, on peut trouver un petit hypercube noté C dans lequel se trouve x.
La remarque 3.3.4 nous assure qu'on est dans l'un des deux cas suivants (distingables comme précédemment par un oracle NP (dont l'entrée contient les coordonnées de l'hypercube C et $\frac{1}{r_n}$, le reste s'en déduit - il est évidemment impossible de donner la liste de tous les hyperplans de $\mathcal{H}$ car celle-ci est de taille exponentielle) :

- aucun hyperplan ne coupe C et alors les $2n$ hyperplans définissant les $2n$ faces de C donnent un système d'inéquations localisant x dans $\mathcal{H}_n$.
- sinon, soit $\mathcal{H'}_n = \{h \in \mathcal{H} | h \cap C \neq \emptyset\} \neq \emptyset$ l'ensemble des hyperplans coupant C et soit I leur intersection ($I \neq \emptyset$).

Notons que comme toute intersection de sous-espace affine est un sous-espace affine ou l'ensemble vide, I est un sous-espace affine.

Trouvons maintenant un point s de I. Pour cela, construisons une suite de sous-espaces affines $(E_1, \ldots, E_j)$ de dimension strictement décroissante (et donc $j \leq 1$) tels que $I \subset E_i$ pour tout i et $I = E_j$.

Par un oracle NP et par dichotomie, on trouve un hyperplan $h_1 = E_1$ de $\mathcal{H'}_n$.

Supposons E_i déjà construit. On remarque que pour tout hyperplan h de $\mathcal{H'}_n$, ou bien $\dim(E_i \cap h) = \dim(E_i) - 1$ ou bien $\dim(E_i \cap h) = \dim E_i$ (car $E_i \cap h \neq \emptyset$). Si on peut trouver (toujours par un oracle NP) un hyperplan h de $\mathcal{H'}_n$ tel que $\dim(E_i \cap h) = \dim E_i - 1$, alors on pose $E_{i+1} = E_i \cap h$. Sinon, c'est que $\dim(E_i \cap h) = \dim E_i$ pour tout hyperplan h de $\mathcal{H'}_n$ donc $E_i \subset h$ pour tout h de $\mathcal{H'}_n$ donc $E_i = E_j = I$.

Comme on connaît un système d'au plus n équations de $E_j = I$ (par construction), il suffit d'utiliser un oracle NP pour trouver un point à coordonnées rationnelles (de taille polynomiale en n) solution de ce système (comme dans le cas à une dimension - grâce au lemme 3.2.4). Notons $s = (s_1, \ldots, s_n) = (\frac{p_1}{q_1}, \ldots, \frac{p_n}{q_n})$ ce point de I. Si $x = s$ ($q_i x_i = p_i$ pour tout i), le système d'équation :

$$\begin{cases} q_1 & x & = & p_1 \\ \vdots & & \vdots & \vdots \\ q_n & x & = & p_n \end{cases}$$

convient et on arrête l'algorithme.

Sinon, on trouve la face f de l'hypercube telle que x est dans la pyramide de sommet s et de base la face considérée (la pyramide verte de la figure 3). Cette fois-ci nous n'avons pas besoin d'un oracle NP, il suffit de tester les $2n$ faces possibles de l'hypercube (en regardant à chaque fois si x est bien du "bon" côté des $2n - 2 + 1$ faces de la pyramide). Notons h l'hyperplan (qui a une équation du type $x_i = \pm 1$) contenant la face f.

Les $2n$ hyperplans définissant les $2n$ faces de C donnent un système d'inéquations définissant C. Ainsi si nous localisons x par rapport à $\mathcal{H'}_n$, en ajoutant au système obtenu le système d'inéquations de C, on aura localisé x dans $\mathcal{H}_n$ car tous les points de C sont dans la même classe d'équivalence par rapport à l'ensemble d'hyperplans $\mathcal{H}_n \setminus \mathcal{H'}_n$.

Comme sur les figues 3 et 4 notons p le projeté de x sur f (ou h ce qui revient au même) par rapport à la droite (sx). Si nous localisons p par rapport à $\mathcal{H''}_n = \{h \cap h' | h' \in \mathcal{H}_n \text{ et } s \in h\}$, nous aurons aussi localisé x par rapport à $\mathcal{H'}$. En effet soit h' un hyperplan de $\mathcal{H'} \subset \mathcal{H}_n$, on a :

– $x \in h'$ si et seulement si $(sx) \subset h'$ (car $s \in h'$) si et seulement si $(sp) \subset h'$ si et seulement si $p \in h'$ si et seulement si $p \in h' \cap h$.
– si $x \in h'^+$, la demi-droite ouverte $]s, x)$ est dans h'^+ (sinon l'un de ses points serait sur h' et s ne pourrait plus être sur h') et donc p (qui est sur $]s, x)$) est dans h'^+ et donc dans $(h' \cap h)^+$ si on se place dans le sous-espace affine h.
– la réciproque du point précédent est similaire (on regarde la demi-droite $]s, p) =]s, x)$).
– on peut remplacer h'^+ par h'^- dans les affirmations ci-dessus.

Malheureusement, il n'est pas possible de calculer les coordonnées de p car la structure de machine BSS considérée ne contient pas la multiplication. Si c'était possible, il suffirait d'appliquer l'algorithme précédent (par récurrence) en dimension $n-1$ avec ${\mathcal{H}''}_n$ au lieu de $\mathcal{H}_n$ (ce qui ne change pas grand chose car il est aisé de savoir par un algorithme polynomial si un hyperplan de $\mathcal{H}_n$ est dans ${\mathcal{H}''}_n$: il suffit de regarder si s est dans l'hyperplan ou non).

On peut contourner ce problème de la manière suivante : à chaque fois qu'un test de la forme $x \in h'$, ou $x \in h'^+$ ou $x \in h'^-$ (avec h' un hyperplan du sous-espace h, à coefficients entiers) est nécessaire, on le remplace par un test dans $\mathbb{R}^n$ avec l'hyperplan de $\mathbb{R}^n$ contenant h' et passant par s. Cela se montre comme précédemment en regardant la demi-droite $]s, c) =]s, p)$.

Il reste à voir que le r_n choisi au début pour $\mathcal{H}_n$ est toujours valide pour ${\mathcal{H}''}_n$. Cela est montré dans [5] et cité dans [2].

□

Il ne reste plus qu'à déduire le théorème 3.3.1 du lemme 3.3.5.

Démonstration du théorème 3.3.1. Soit $x = (x_1, \ldots, x_n)$ un point de $\mathbb{R}^n$. Plaçons nous maintenant dans $\mathbb{R}^{n+1}$ et posons $\tilde{x} = (x_1, \ldots, x_n, 1)$.

Soit $\tilde{\mathcal{H}}_n$ l'ensemble des hyperplans linéaires de $\mathbb{R}^n$ d'équations $\lambda_1 x_1 + \ldots \lambda_n x_n - b x_{n+1} = 0$ avec λ_i et b des entiers de taille au plus t. Tous ces hyperplans ont un point commun : $s = 0$.

Comme dans la démonstration précédente, page 3.3, trouvons la face f (contenue à un hyperplan affine h) de l'hypercube $[-1, 1]^{n+1}$ telle que la pyramide de base f et de sommet $s = 0$ contient $\tilde{x}$ (on suppose la pyramide non bornée par sa base comme dans le dessin ci-contre).
Soit p le projeté de $\tilde{x}$ sur f.
Localiser x dans $\tilde{\mathcal{H}}_n$ revient à localiser p dans l'ensemble d'hyperplans affines $\bar{\mathcal{H}}_n = \{h' \cap h | h' \in \tilde{\mathcal{H}}_n\}$, comme dans la démonstration précédente. Et la même astuce que dans la démonstration précédente rend inutile le calcul exact de p.

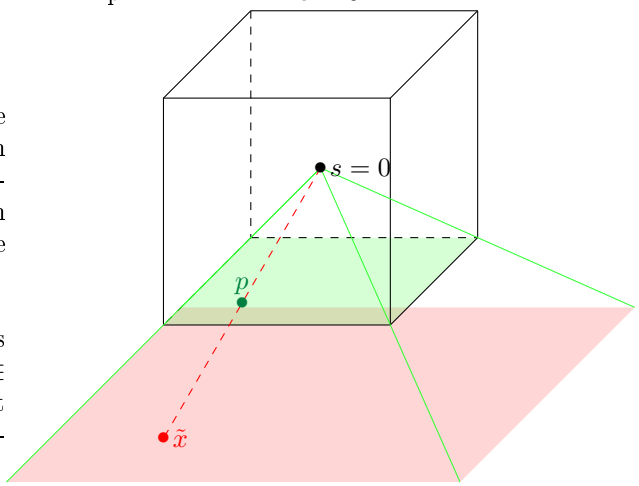

FIG. 5 – Exemple dans $\mathbb{R}^3$ où $\tilde{x}$ est projeté sur la face f du bas du cube.

□

A Démonstration des propositions de manipulation des entiers et des rationnels

Démonstration de la proposition 2.2.2. Notons $(x_1 = \frac{p_1}{q_1}, \ldots, x_m = \frac{p_m}{q_m})$ l'entrée de la machine que l'on veut construire et

$$P = \prod_{j=1}^{m} q_j \quad \text{et} \quad P_i = \prod_{j=1,\, j\neq i}^{m} q_j$$

Posons aussi $\tilde{x}_i = P\ x_i = P_i\ p_i$ et $\tilde{c}_i = P\ c_i$. On remarque que si le produit P des q_i est positif :

- $\lambda_1 x_1 + \ldots + \lambda_m x_m = \mu_1 c_1 + \ldots + \mu_p c_p \iff \lambda_1 \tilde{x}_1 + \ldots + \lambda_m \tilde{x}_m = \mu_1 \tilde{c}_1 + \ldots + \mu_p \tilde{c}_p$
- $\lambda_1 x_1 + \ldots + \lambda_m x_m > \mu_1 c_1 + \ldots + \mu_p c_p \iff \lambda_1 \tilde{x}_1 + \ldots + \lambda_m \tilde{x}_m > \mu_1 \tilde{c}_1 + \ldots + \mu_p \tilde{c}_p$

et que sinon :

- $\lambda_1 x_1 + \ldots + \lambda_m x_m = \mu_1 c_1 + \ldots + \mu_p c_p \iff \lambda_1 \tilde{x}_1 + \ldots + \lambda_m \tilde{x}_m = \mu_1 \tilde{c}_1 + \ldots + \mu_p \tilde{c}_p$
- $\lambda_1 x_1 + \ldots + \lambda_m x_m > \mu_1 c_1 + \ldots + \mu_p c_p \iff \lambda_1 \tilde{x}_1 + \ldots + \lambda_m \tilde{x}_m < \mu_1 \tilde{c}_1 + \ldots + \mu_p \tilde{c}_p$

Cela permet de résoudre le problème de la division. Voici une preuve plus formelle :

Quitte à doubler le nombre de bandes de $\mathcal{M}$(et à en ajouter une) et quitte à multiplier par une constante son temps d'exécution, on peut supposer que certaines bandes (que l'on dira de type 1) contiennent uniquement des combinaisons linéaires à coefficients entiers des paramètres c_i et que d'autres (que l'on dira de type 2) contiennent uniquement des combinaisons linéaires des entrées x_i. D'après le début de cette section, les comparaisons sont des tests d'égalité ($=$) ou d'inégalité stricte ($<$) entre deux cases : une d'une bande de type 1 et une autre d'une bande de type 2 (quitte à ajouter une case valant 0 au d'une bande de type 1 et d'une de type 2).

On peut en effet à chaque étape séparer les combinaisons linéaires de c_i de celles des x_i car le résultat de la machine est conditionné par un test final (voir page 2).

Construisons maintenant $\mathcal{M}'$. Cette machine commence par calculer les constantes $\tilde{c}_i$ sur une nouvelle bande. Si l'entrée est de taille n, alors a fortiori, les x_i sont de taille au plus n et $m \leq n$. Donc la taille de Π est $n \times n = n^2$. Ainsi la multiplication des constantes c_i par Π pour obtenir $\tilde{x}_i$ s'effectue en temps polynomial (la multiplication apprise en classe élémentaire adaptée au cas binaire s'effectue en $O(n^3)$ pour deux nombres de taille n).

Ensuite, la machine calcule, en temps polynomial, les $\tilde{x}_i$ (qui sont des listes de 0 et 1 et qui occupent n cases dans la machine) et les transforme en vrais réels $\bar{x}_i$ (qui occupe donc une seule case dans la machine) grâce à l'algorithme suivant :

Require: $z = \overline{z_n \ldots z_1}$ un entier écrit en base 2
  $\bar{z} \leftarrow 0$
  for $i = n$ downto 1 **do**
    $\bar{z} \leftarrow \bar{z} + \bar{z}$
    if $z_i = 1$ **then**
      $\bar{z} \leftarrow \bar{z} + 1$
  return $\tilde{z}$

Enfin, on effectue les mêmes calculs que la machine $\mathcal{M}$ en utilisant les nombres $\tilde{c}_i$ et $\bar{x}_i$ au lieu des c_i et x_i en prenant soin d'inverser toutes les inégalités si le produit P est négatif (précaution qui ne fait que multiplier le temps d'exécution par une constante).

Ainsi on a bien créé une machine $\mathcal{M}'$ qui s'exécute en temps polynomial sur des entrées rationnelles et qui effectue le même calcul que $\mathcal{M}$. □

Démonstration du corollaire 2.2.3. La même preuve que précédemment fonctionne sauf que $\tilde{x}_i = P_{i,1} y_{i,1} x_1 + \ldots + P_{i,n} y_{i,n} x_n$ en posant :

$$y_{i,j} = \frac{p_{i,j}}{q_{i,j}}\ ,\ P = \prod_{(i,j)\in[\![1,n]\!]^2} q_{i,j} \text{ et } P_{k,l} = \prod_{(i,j)\in[\![1,n]\!]^2\setminus(k,l)} q_{i,j}$$

□

Démonstration de la proposition 2.2.4. La transformation de $\mathcal{M}$ en $\mathcal{M}'$ est très proche de celle décrite dans la preuve de la proposition 2.2.2. La différence est que l'on ne transforme par les rationnels en vrais réels et que l'on fait tous les calculs avec les numérateurs écrits sous forme binaire. Cela est possible car

tous les nombres manipulés sont soit des entiers écrits sous forme binaire (numérateur et dénominateur des entrées) soit la constante 1 (qui est d'ailleurs "remplacée" par le produit P des dénominateurs, au début de l'exécution de la machine $\mathcal{M}'$).

De plus cette machine reste en temps polynomial car l'addition et la soustraction d'entiers sous forme binaire est linéaire en la taille de ces entiers et que la taille des entiers manipulés est polynomial en la taille de l'entrée (pour la même raison que les coefficients λ_i et μ_i de la sous-section 2.1 sont de taille polynomiale en la taille de l'entrée). □

B Démonstration du lemme 2.3.1

Avant de démontrer le lemme 2.3.1, nous avons besoin du lemme de Carathéodory.

Lemme B.0.6 (lemme de Caratheodory)**.** *Soit E un $\mathbb{R}$-espace vectoriel et $\vec{x}$ un vecteur de E. Supposons que $\vec{x}$ s'écrive comme combinaison linéaire à coefficients positifs de vecteur de E $\vec{x} = \sum_{k=1}^{p} \lambda_k \vec{a}_k$ (avec $\vec{a}_k \in E$ et $\lambda_k \in \mathbb{R}^+$ pour tout k).*

Alors il existe une sous-famille libre de $(\vec{a}_1, \ldots, \vec{a}_p)$ telle que $\vec{x}$ est aussi combinaison linéaire à coefficients positifs de cette sous-famille.

Démonstration. On raisonne par récurrence sur p.

- Pour $p = 1$, $(\vec{a}_1)$ forme déjà une famille libre de E.
- Supposons le résultat vrai pour $p-1$. Alors si $(\vec{a}_1, \ldots, \vec{a}_p)$ est libre, on a fini. Sinon, il existe $(\mu_1, \ldots, \mu_p)$ tels que : $0 = \sum_{k=1}^{p} \mu_k \vec{a}_k$. Soit i tel que $\mu_1 \neq 0$ et que $\left|\frac{\lambda_i}{\mu_i}\right|$ est minimum. On a :

  $$\vec{a}_i = -\sum_{k=1, k\neq i}^{p} \frac{\mu_k}{\mu_i} \vec{a}_k$$

  et donc :

  $$x = \sum_{k=1, k\neq i}^{p} \left(\lambda_k - \lambda_i \frac{\mu_k}{\mu_i}\right) \vec{a}_k$$

  Comme $\left|\frac{\lambda_i}{\mu_i}\right| \leq \left|\frac{\lambda_k}{\mu_k}\right|$, on a :

  $$\lambda_k - \lambda_i \frac{\mu_k}{\mu_i} \geq \lambda_k - \lambda_i \frac{|\mu_k|}{|\mu_i|} \geq 0$$

  et donc $\vec{x}$ est combinaison linéaire à coefficients positifs de $p-1$ vecteurs. Grâce à l'hypothèse de récurrence, on en déduit la propriété voulue.

□

Démonstration du lemme 2.3.1. Notons $(x'_1, \ldots, x'_n)$ une solution positive. Le vecteur $(a_i)_{i\in[\![1,m]\!]}$ est combinaison linéaire à coefficients positifs (qui sont les x'_i) des vecteurs $\vec{\Lambda}_j = (\lambda_{i,j})_{i\in[\![1,m]\!]}$. Donc par le lemme B.0.6, $(a_i)_{i\in[\![1,m]\!]}$ est combinaison linéaire d'une sous-famille libre de $(\vec{\Lambda}_1, \ldots, \vec{\Lambda}_n)$. En ne gardant que les inconnues x_j correspondant aux $\vec{\Lambda}_j$ de la sous-famille et en annulant les autres, on obtient un système linéaire dont l'unique solution est $(x'_1, \ldots, x'_n)$. En effet si $(x''_1, \ldots, x''_n)$ était une autre solution, on aurait pour tout i :

$$\lambda_{i,1}(x'_1 - x''_1) + \ldots + \lambda_{i,n}(x'_n - x''_n) = a_i - a_i = 0$$

(dans la somme il n'y a que les $\lambda_{i,j}$ sélectionnés par la sous-famille) ce qui contredirait le caractère libre de la sous-famille considérée des $\vec{\Lambda}_j$. □

Remerciements

Je remercie chaleureusement Monsieur Sylvain Perifel pour son aide précieuse à l'élaboration de ce dossier et pour sa relecture.

Références

[1] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers : NP-completeness, recursive functions and universal machines. *American Mathematical Society*, 21(1), 1989. 1

[2] H. Fournier and P. Koiran. Lower bounds are not easier over the reals : Inside PH. *Lecture notes in computer science*, pages 832–843, 1999. 1, 10, 13

[3] C. Gaßner. Eine Struktur endlicher Signatur mit Identitätsrelation und mit P= NP. 2004. 1

[4] A. Hemmerling. P= NP for some structures over the binary words. *Journal of Complexity*, 21(4) :557–578, 2005. 1

[5] Friedhelm Meyer auf der Heide. Fast algorithms for n-dimensional restrictions of hard problems. *J. ACM*, 35(3) :740–747, 1988. 10, 13

[6] B. Poizat. *Les petits cailloux*. Aléas, 1995. 16

Pour ceux qui veulent aller plus loin, je leur conseille vivement de commencer par lire le livre de B. Poizat [6] qui est une excellente introduction aux machines BSS.