Problème P vs NP dans le modèle de Blum, Shub, Smale

Guilhem Gamard

Décembre 2010

Résumé

Dans ce travail de rédaction, nous introduisons le modèle de calcul de Blum, Shub et Smale sur les nombres réels. Nous commencerons par donner informellement quelques définitions et propriétés, ainsi qu'une justification de l'intérêt d'un tel modèle. Nous formaliserons ensuite les définitions et résultats abordés en première partie, avant de montrer l'équivalence entre la célèbre question $P = NP$ et une forme analogue de cette question dans le modèle BSS.

1 Introduction

Les machines de Turing constituent un modèle bien connu de la notion de *calcul* et de *fonction calculable*. Ce modèle présente un certain nombre de restrictions, mises en place afin de ne pas aboutir à une classe de fonctions calculables trop grossière : l'alphabet de bande et le nombre d'états de la machine doivent être finis. Cependant, ces restrictions nous interdisent également l'étude des problèmes portant sur les nombres réels exacts. On peut en effet représenter une approximation, arbitrairement fine, des réels sur une machine de Turing : il suffit de prendre un rationnel (couple d'entiers) arbitrairement proche du réel voulu. Mais cette idée n'est pas satisfaisante dans tous les cas. Considérons par exemple l'étude de la méthode de Newton, un algorithme d'approximation un zéro d'un polynôme. Il procède itérativement, augmentant la précision du résultat à chaque tour de boucle. Si le modèle de calcul dans lequel on se place introduit lui-même des approximations sur la représentation des données, l'étude de ce calcul devient très complexe.

Pour pallier à ce problème, M. Blum, Shub et Smale ont proposé en 1989 un modèle de calcul très similaire aux machines de Turing, la différence résidant dans l'alphabet de bande qui se trouve être l'ensemble $\mathbb{R}$ des réels. Chaque machine BSS possède également un ensemble d'**opérations légales**, qui définit ce que la machine a le droit d'écrire sur la bande. Lors de l'exécution, l'entrée est écrite au début de la bande, qui est ensuite remplie avec un symbole spécial $\perp$, puis le calcul est effectué transition par transition. Une machine BSS, tout comme une machine de Turing, peut être utilisée en mode « acceptation » ou en mode « calcul ». Enfin, le nombre d'états d'une telle machine doit être fini. Le concept sera formalisé en section 2.1, mais on peut dores et déjà observer un premier exemple pour en saisir le comportement général :

Exemple. On définit une machine BSS qui accepte les suites $(y, x_1, \ldots, x_n)$ vérifiant $y = \sum_i x_i$.

Cette machine possèdera quatre états : Q_i l'état initial, Q_1, Q_2, Q_3 et deux opérations légales : $+$ et $=$. La première ajoute à la case courante le contenu de la case immédiatement à droite ; la seconde teste l'égalité entre la case courante et celle immédiatement à droite. Elle écrit 1 si elle est réalisée, 0 sinon.

- Dans l'état initial, on lit l'entrée sans la modifier et on se déplace vers la droite jusqu'à trouver un zéro. On passe alors dans l'état Q_1.

- Dans l'état Q_1, on écrit la somme de la case courante et de celle immédiatement à droite, puis on se déplace à gauche, jusqu'à arriver au début de la bande. On passe alors dans l'état Q_2.
- Dans l'état Q_2, on effectue un test d'égalité entre la case courante (à ce stade de l'algorithme, c'est la première case de la bande) et celle immédiatement à droite, puis passe dans l'état Q_3.
- L'état Q_3 met fin au calcul en acceptant si le contenu de la case courante est 1, et bloque la machine sinon.

On a supposé que les machines BSS étaient capables de détecter le début de la bande, par exemple grâce à un symbole $\perp$ placé à l'indice -1. On remarque en outre qu'une machine décide de son acceptation ou de son rejet en faisant un test.

Les descriptions plus formelles de machines BSS s'expriment sous la forme d'algorithmes ne faisant intervenir que les opérations légales ou bien sous forme de tableaux donnant l'ensemble des transitions.

Le modèle réel étant conceptuellement très proche des machines de Turing[1], il est naturel de vouloir tenter de définir des classes de complexité, spatiales ou temporelles, analogues aux familières P, NP, et ainsi de suite. Ces définitions ne posent aucun problème pour les complexités en temps, et ouvrent ainsi de nouvelles questions du style « P est-il égal à NP sur les nombres réels ? ». Il est alors tentant de chercher à relier ces questions au fameux problème « P versus NP » dans le modèle de Turing, afin de fournir une nouvelle approche de la résolution de cette question. En revanche, on montre que tout calcul peut être effectué en espace constant sur les nombres réels, aussi les classes de complexité spatiales ne sont plus pertinentes ici : on ne s'en préoccupera pas par la suite.

Le résultat essentiel de ce travail de rédaction montrera l'équivalence entre le problème P = NP dans le modèle de Turing et ce même problème dans une certaine version du modèle BSS. De manière intuitive, les conséquences de ce résultat sont mitigées. D'une part, on peut voir que P = NP dans BSS implique P = NP dans le modèle de Turing, ce qui nous autorise à travailler sur cette dernière question à l'aide de machines BSS. On dispose alors d'un de tous les outils algébriques et analytiques sur $\mathbb{R}$ ou $\mathbb{C}$, ce qui semble ouvrir de nombreuses perspectives originales dans la résolution de cette grande question de complexité. D'autre part, on aura réciproquement que P = NP dans le modèle de Turing implique la même égalité dans le modèle BSS. En d'autres termes, la question P versus NP ne se révèle pas plus simple à résoudre dans le modèle réel que dans le modèle booléen.

Il est à noter que les machines BSS ne sont pas l'unique type de machine à calculer sur des réels. On pourra par exemple considérer le modèle de Valiant (1979), dont l'étude dépasse le cadre de cette rédaction. Le sujet est traité dans [4].

La plupart des modèles réels se généralisent sans peine à d'autres corps, comme $\mathbb{Q}$ ou $\mathbb{C}$.

1. que l'on nommbera « modèle booléen » par la suite

2 Formalisation du modèle de Blum, Shub et Smale

Les définitions présentées ici rappelleront fortement leurs homologues booléennes. Il peut être commode de faire le parallèle pour bien comprendre, mais certaines subtilités propre au monde de BSS doivent être prises en compte.

2.1 Définitions fondamentales

Nous commençons par définir les machines BSS et les calculs effectués par ces machines.

Définition (Machine BSS, configuration)**.** Une **machine BSS** est la donnée d'un 5-uplet (Q, q_i, F, Γ, t) où :
- Q est un ensemble fini d'états,
- $q_i \in Q$ est l'état initial,
- $F \subset Q$ est l'ensemble des états acceptants,
- Γ est l'ensemble des opérations légales, qui sont des fonctions $\mathbb{R}^2 \to \mathbb{R}$,
- et t est une fonction (de transition) de $Q \times \mathbb{R}$ dans dans $Q \times \Gamma \times \{+1, 0, -1\}$.

Une **configuration** est un triplet (q, i, B) où :
- $q \in Q$ est l'état actuel de la machine,
- $i \in \mathbb{N}$ est l'indice de la tête de lecture,
- $B \in \mathbb{R}^{\mathbb{N}}$ est l'état de la bande.

Une configuration définit l'état, au sens intuitif, d'une machine BSS à un instant donné. Celui-ci est totalement déterminé par l'état dans lequel se trouve la tête de lecture, la position de celle-ci (son indice sur la bande) et le contenu de la bande, qui peut être vu comme une suite de réels. On passe alors d'une configuration à l'autre en faisant un pas de calcul. C'est l'objet de la définition suivante.

Définition (Configuration suivante)**.** On définit une **configuration suivante** d'une configuration (q, i, B) comme suit :

Soit $(q', f, n) = t(q, B_i)$, avec t la fonction de transition. La configuration suivante correspondant à (q, i, B) est alors donnée par $(q', i + n, B_0 B_1 \dots B_{i-1} f(B_i, B_i + 1) B_{i+1} \dots)$. En d'autres termes, il s'agit de la configuration à laquelle on aboutit après un pas de calcul.

On remarquera que toutes les opérations légales sont binaires. Elles sont systématiquement appelées sur deux cases consécutives, puis le résultat est écrit dans la première des deux. Ces restrictions permettent d'éviter que l'on se serve d'une des cases de la bande pour stocker un pseudo-état, et ainsi émuler une machine ayant une infinité non-dénombrable d'états... Ce qui mènerait à une notion bien trop large de fonction calculable.

Forts de cette définition d'un pas de calcul, nous pouvons définir un calcul complet.

Définition (Calcul)**.** Un **calcul** est une suite $(U_i)_{i \in \mathbb{N}}$ de configurations telles que pour tout i, U_{i+1} est une configuration suivante de U_i. On dit qu'un calcul **termine** si l'une de ces configurations n'a pas de transition possible, qu'il **accepte** si cette dernière configuration a un état appartenant à F et qu'il **refuse** sinon.

Exemple. On décrit une machine BSS sur les réels qui résout le *problème de la sous-somme*, formulé comme suit :

Étant donnés une famille finie de réels $x_1, \dots, x_n$, déterminer s'il existe un ensemble d'indices $I \subset \{1, \dots, n\}$ tel que $\sum_{i \in I} x_i = 0$.

Ce problème peut également se formuler dans le monde booléen avec des entrées entières ; il est alors NP-complet (voir [2]). Il existe donc une machine de Turing non-déterministe qui résout

ce problème en temps polynomial et une machine de Turing déterministe qui *vérifie* une solution donnée en temps polynomial (il s'agit simplement de calculer une somme). Pour résoudre le problème de façon non-déterministe, on procède alors comme suit :

- choisir de façon non-déterministe une partie de l'entrée,
- vérifier que la somme des éléments cette partie vaut bien la valeur recherchée.

Si au moins l'un des calculs accepte, alors la machine entière accepte, ce qui est bien le cas si et seulement s'il existe une solution au problème.

Cependant, nous n'avons pas défini la notion de « machine non-déterministe » dans le monde des réels. C'est tout à fait possible : il s'agit essentiellement de machines ayant plusieurs transitions réalisables pour une même configuration, qui acceptent si et seulement s'il existe un calcul acceptant. À partir de là, on pourrait imaginer résoudre le problème de la sous-somme dans les réels de la façon suivante :

État courant	Réel lu	Nouvel état	Réel écrit	Direction
Q_0	$\bot$	Q_1	$\bot$	$\Leftarrow$
Q_0	x_i	Q_0	x_i	$\Rightarrow$
Q_0	x_i	Q_0	0	$\Rightarrow$
Q_1	$\bot$	Q_2	$\bot$	$\Rightarrow$
Q_1	x_i	Q_1	$x_i + x_{i+1}$	$\Leftarrow$

FIGURE 1 – Tableau de transitions non-déterministe

L'état Q_2 passe dans l'état Q_f s'il lit un 1 sur la bande, et bloque la machine sinon.

Cette machine parcourt la bande en plaçant certains éléments à 0, de façon non-déterministe. Ensuite, elle parcourt son entrée pour en calculer la somme et accepte si cette dernière est nulle. Ainsi, si le I recherché existe, la machine s'arrêtera nécessairement, et bloquera sinon.

Une autre approche équivalente, retenue dans la suite de ce travail, est de considérer que toutes les machines sont déterministes, et à donner une définition plus subtile de la classe $\mathrm{NP}^0_{\mathbb{R}_{ovs}}$, comme on le verra en section 2.2. On ne considère donc plus que les machines BSS « déterministes », comportant une seule transition possible par configuration.

Remarque. Toutes les généralisations connues des machines de Turing, telles que les machines bi-infinies, les machines à K-bandes, avec diverses restrictions sur la lecture et l'écriture via ces bandes, peuvent se définir sans peine sur des machines BSS. Ces différents modèles de machine ne sont toutefois pas équivalents dans le mode réel. On pourra consulter [3].

2.2 Classes de complexité

La notion de complexité dans le modèle BSS dépend des opérations légales que l'on s'autorise. Il est aisé de voir que les machines munies d'addition, soustraction, multiplication et ordre ont nettement plus de puissance de calcul que les machines pouvant uniquement effectuer des additions et des tests d'égalité. Certains problèmes sont même décidables par les premières, ce qui n'est pas le cas avec les secondes, d'où la nécessité de se placer dans un cadre particulier pour la suite de cet article.

Nous ne considèrerons donc plus que les machines munies de l'addition, de la soustraction, de l'égalité ($=$) et de l'ordre ($<$). On appelle ce modèle $\mathbb{R}_{ovs}$, pour Ordered-Vector-Space. Les définitions des classes de complexité s'étendent très bien à d'autres styles de machines BSS, mais les résultats étudiés par la suite ne se généralisent pas. En outre, nos machines ne disposeront d'aucune constante, c'est-à-dire qu'elles ne peuvent pas écrire un réel donné, arbitraire, sur la bande. Seuls les réels issus de tests ou de sommes peuvent être écrits. Ceci ce note $\mathbb{R}^0_{ovs}$, le 0 donnant le nombre de constantes autorisées dans les transitions.

Définition (Temps polynomial)**.** On dit qu'un langage est dans $\mathrm{P}^0_{\mathbb{R}ovs}$ s'il est décidé par une machine dont le temps d'exécution (au sens nombre d'étapes de calcul) est majoré par un polynôme de la taille de son entrée.

De même, un problème est dans $\mathrm{P}^0_{\mathbb{R}ovs}$ s'il est résolu par une machine de cette classe.

Rappelons que toutes les machines BSS que nous considérons sont déterministes.

Exemple. La première machine BSS donnée en exemple, section 1, s'exécute systématiquement en temps linéaire. Le problème consistant à vérifier la somme d'une suite finie de réels non-nuls est donc bien dans $\mathrm{P}^0_{\mathbb{R}ovs}$.

Définition (Temps non-déterministe polynomial)**.** Un langage L est dans $\mathrm{NP}^0_{\mathbb{R}ovs}$ si et seulement s'il existe un langage $A \in \mathrm{P}^0_{\mathbb{R}ovs}$, et un polynôme entier $r(n)$ tels que :

$$x \in L \Leftrightarrow \exists z \in \{0,1\}^{r(|x|)} < z, x > \in A$$

Le sens intuitif de cette condition se saisit en faisant le parallèle avec la caractérisation de NP dans le modèle de Turing. Étant donné que toutes les machines BSS sont supposées déterministes, on effectue les « choix » non-déterministes avant même l'exécution, puis on les passe en entrée. S'il existe une série de choix — donc un calcul possible — qui mène à une acceptation, la machine doit accepter. On conclut donc que $x \in L$ dès lors qu'il existe un choix menant à l'acceptation de x par A. On appelle *certificat* un tel choix.

L'ensemble des problèmes $\mathrm{NP}^0_{\mathbb{R}ovs}$ se définit de la même manière.

Exemple. La machine BSS donnée en exemple dans la section 2.1 s'exécute toujours en temps polynomial. On se donne un certificat de longueur n, donc une suite 0 de et de 1 de la même longueur que l'entrée. Il détermine quelle transition choisir dans l'état Q_0, lorsque deux transitions sont possibles : écrire 0 ou écrire le réel lu. On en déduit une machine qui prend en entrée z en plus des x_i et qui effectue les transitions suivant ce schéma.

Le problème de la sous-somme est donc bien un problème de $\mathrm{NP}^0_{\mathbb{R}ovs}$.

Remarque. En termes de langages (ou de problèmes), on a l'inclusion $\mathrm{P}^0_{\mathbb{R}ovs} \subset \mathrm{NP}^0_{\mathbb{R}ovs}$.

L'argument permettant de le montrer est similaire au cas booléen, il découle des définitions. Les machines déterministes étant un cas particulier de machines non-déterministes, cette inclusion est claire.

À ce stade du développement, nous avons toutes les définitions nécessaires pour formuler le problème $\mathrm{P} = \mathrm{NP}$ sur les machines $\mathbb{R} - ovs$. Il s'agit simplement du problème $\mathrm{P}^0_{\mathbb{R}ovs} = \mathrm{NP}^0_{\mathbb{R}ovs}$. Nous pouvons désormais nous poser la question de l'équivalence avec la question $\mathrm{P} = \mathrm{NP}$ classique.

3 Problème P vs NP dans le modèle BSS $\mathbb{R}_{ovs}$

Maintenant que la définition des principales classes de complexité sur les machines $\mathbb{R} - ovs$ sont connues, nous nous proposons de montrer que les problèmes $\mathrm{P}^0_{\mathbb{R}ovs} = \mathrm{NP}^0_{\mathbb{R}ovs}$ et $\mathrm{P} = \mathrm{NP}$ sont équivalents. Le sens *modèle réel* vers *modèle booléen* sera facile, puisqu'il suffira de montrer que les machines BSS, sous certaines restrictions fortes, se comportent comme des machines de Turing. Nous aurons alors la partie dite positive du résultat : « on peut travailler sur $P = NP$ avec des nombres réels, l'addition et l'ordre... »

L'autre sens demandera, quant à lui, davantage de travail et quelques résultats intermédiaires. Il nous donnera la partie dite négative : « ...mais ce sera tout aussi difficile que dans le monde booléen. »

3.1 Sens réel $\Rightarrow$ booléen

Pour montrer que $\mathrm{P}^0_{\mathbb{R}ovs} = \mathrm{NP}^0_{\mathbb{R}ovs} \Rightarrow \mathrm{P} = \mathrm{NP}$, on considère les machines BSS travaillant uniquement sur l'ensemble $\{0,1\}$ avec les bonnes restrictions pour se retrouver dans le modèle classique de Turing. Les définitions présentées dans la section précédente nous permettrons alors de conclure immédiatemment.

Proposition. *Les machines BSS dont les entrées sont restreintes à $\{0,1\}^*$ et dont toutes les opérations légales sont des fonctions de $\{0,1\}^2 \to \{0,1\}$ ignorant leur second argument ont la même puissance que les machines de Turing.*

Démonstration. Il s'agit de considérer la définition des machines BSS avec les restrictions demandées et de vérifier qu'elle est bien équivalente à celle des machines de Turing dont l'alphabet de bande est $\{0,1\}$. On remarque ensuite qu'il existe une machine de Turing universelle dont l'alphabet de bande possède seulement deux symboles (voir [5] et [6]). Les MT dont l'alphabet de bande est $\{0,1\}$ sont donc bien aussi puissantes que les MT en toute généralité, ce qui permet de conclure. □

Corollaire immédiat Si $\mathrm{P}^0_{\mathbb{R}ovs} = \mathrm{NP}^0_{\mathbb{R}ovs}$, alors $\mathrm{P} = \mathrm{NP}$.

Reste donc à établir l'implication dans l'autre sens : si $\mathrm{P} = \mathrm{NP}$, alors on a $\mathrm{P}^0_{\mathbb{R}ovs} = \mathrm{NP}^0_{\mathbb{R}ovs}$. L'inclusion $\mathrm{P} \subset \mathrm{NP}^0_{\mathbb{R}ovs}$ a déjà été vue en section 2.2. On se propose donc de montrer l'inclusion $\mathrm{NP}^0_{\mathbb{R}ovs} \subset \mathrm{P}^0_{\mathbb{R}ovs}$ sous l'hypothèse $\mathrm{P} = \mathrm{NP}$. On a besoin pour celà de quelques résultats intermédiaires.

3.2 Hyperplans à coefficients entiers dans $\mathbb{R}^n$

L'idée intuitive, pour montrer l'inclusion voulue, consiste à remarquer que les machines BSS que nous considérons ne peuvent pas effectuer de tests $<$ ou $=$ sur n'importe quels réels. Tous les réels impliqués dans ces tests sont des combinaisons linéaires à coefficients entiers des réels de l'entrée. Or ce sont les résultats de ces tests qui déterminent l'acceptation — ou non — de la machine.

On remarque alors que tout test effectué par une machine BSS sur $\mathbb{R}^0_{\mathbb{R}ovs}$ au bout de K étapes de calcul peut être ramené à une question du style « x est-t-il devant ou derrière l'hyperplan h ? ». L'hyperplan h sera déterminé par des équations linéaires à coefficients dans $\{-2^K, \ldots, 2^K\}$. À partir de là, nous « discrétiserons » l'espace en le découpant en zones délimitées par de tels hyperplans, et nous nous ramènerons au cas booléen. La conclusion vient alors rapidement par l'hypothèse $\mathrm{P} = \mathrm{NP}$.

Nous allons ainsi montrer, dans un premier temps, que les machines BSS perçoivent l'espace dans lequel elles travaillent comme un ensemble fini de **cellules**, et, qu'en un temps polynomial, on peut déterminer dans quelle zone se situe tout point x.

Soit $H = \{h_1, \ldots, h_n\}$ une famille finie d'hyperplans de $\mathbb{R}^n$. Chacun de ces hyperplans donne une partition de $\mathbb{R}^n$ en trois espaces que l'on appelera h_i^+, h_i^- et h_i le plan lui-même. On définit alors la quantité z_i pour tout point $x \in \mathbb{R}^n$ comme suit :

- $z_i(x) = -1$ si $x \in h_i^-$,
- $z_i(x) = 0$ si $x \in h_i$,
- $z_i(x) = +1$ si $x \in h_i^+$.

On introduit alors la relation d'équivalence $x \sim y \Leftrightarrow \forall i, z_i(x) = z_i(y)$; on appelle les classes d'équivalences les **cellules** du plan. Il est aisé de vérifier, par définition de $\sim$, que de telles cellules sont des compacts connexes. Une description topologique plus détaillée des cellules est donnée dans [1].

Nous allons maintenant introduire l'algorithme essentiel de la fin de la preuve : l'algorithme de **localisation**. Étant donné un point $x \in \mathbb{R}^n$ et une famille finie d'hyperplans $h_1, \ldots, h_n$, le but est de déterminer la cellule dans laquelle se trouve x. L'algorithme proposé s'exécute en temps polynomial sous l'hypothèse $P = NP$ booléenne. On commencera par donner son fonctionnement sur un exemple :

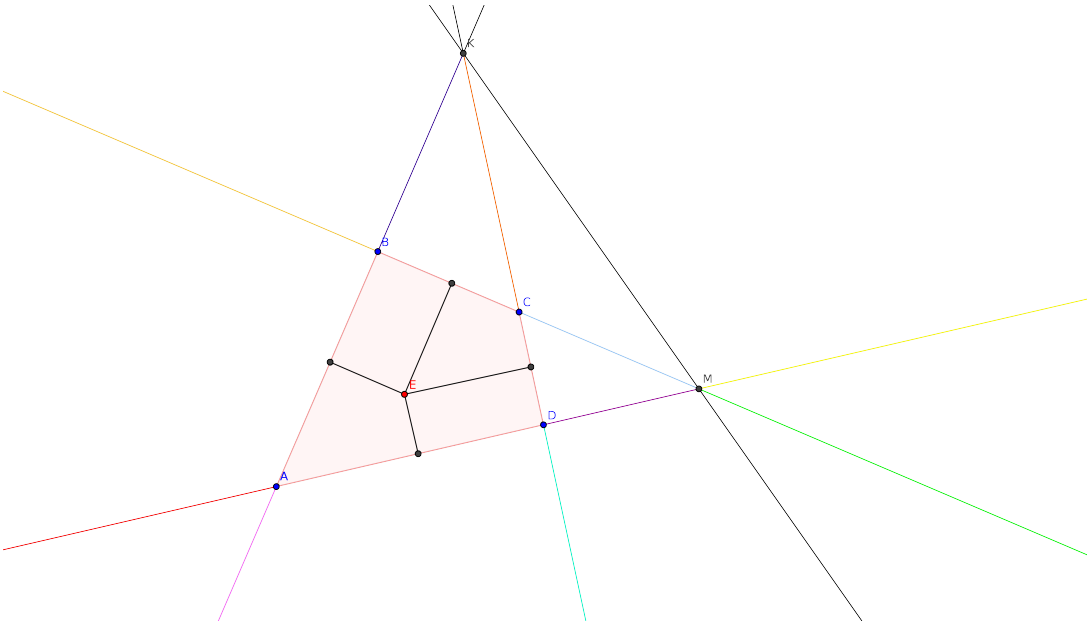

FIGURE 2 – Exemple d'instance du problème de localisation

Il s'agit de localiser le point E dans $\mathbb{R}^2$, les h_i étant les droites visibles sur la figure 2.

- On choisit une droite, par exemple (AD), et on projette E dessus ;
- on localise récursivement la projection de E sur la droite (AD) ;
- on sait donc que le projeté de E est compris entre A et D, donc que E est compris entre les droites (AB) et (CD) ;
- on localise le projeté de E sur (AB) ou sur (CD) ;
- on sait donc que le projeté est compris entre A et B, donc que E est compris entre les droites (AD) et (BC) ;
- les droites (AB), (AD), (CD) et (BC) délimitent bien une cellule du plan, l'algorithme est terminé.

Le problème de localiser un réel sur une droite est polynomial, puisqu'il suffit de comparer x à chacun réels qui déterminent les hyperplans. L'algorithme pourrait s'écrire, plus généralement :

Algorithme 1 Algorithme de localisation simplifié

Localisation

1: $(h'_2, \ldots, h'_n) \leftarrow (h_2, \ldots, h_n) \cap h_1$
2: $x' \leftarrow \text{Projection}(x, h_1)$
3: $A \leftarrow \text{Localiser}(x', h'_2, \ldots, h'_n)$
4: $i \leftarrow \text{face}(A)$
5: $(h''_2, \ldots, h''_{n-1}) \leftarrow (h_1, \ldots, h_{i-1}, h_{i+1}, \ldots, h_n) \cap h_i$
6: $x'' \leftarrow \text{projection}(x, h_i)$
7: $B \leftarrow \text{localiser}(x'', h''_1, \ldots, h''_{n-1})$
8: **return** $A \cap B$

L'étude complète de cet algorithme se trouve dans [1]. On y montre sa correction ainsi que la borne polynomiale de son temps d'exécution, sous l'hypothèse P = NP.

Cet algorithme est important pour la suite de la preuve dans la mesure où, étant donnée une machine BSS M et un entier K, on peut trouver des hyperplans délimitant des cellules sur lesquelles la machine a un comportement identique pendant K étapes de calcul. En d'autres termes, une machine lancée sur deux points d'une même cellule trouvera le même résultat à chacun des tests $<$ ou $=$ durant l'exécution, jusqu'à la K^{ieme} transition.

Voyons comment déterminer ces hyperplans. On remarque en premier lieu que la bande de la machine, après K étapes de calcul, ne peut contenir que des combinaisons linéaires à coefficients entiers compris entre 1 et K des réels de l'entrée. Appelons A_K l'ensemble de tels réels. Les tests que la machine peut effectuer après K étapes de calcul peuvent donc tous se réécrire comme « x est-il devant ou derrière l'hyperplan h ? », où h est déterminé par une équation linéaire dont les coefficients sont tous dans A_K. On pose alors H l'ensemble des hyperplans déterminés par une équation linéaire à coefficients dans A_K et on obtient le résultat souhaité.

Ainsi, étant donnés une machine BSS M et un entier N, on peut toujours déterminer un ensemble H d'hyperplans délimitant des cellules sur lesquelles la machine se comporte de façon uniforme sur toute entrée de longueur au plus N. Il suffit de poser $K = T(n)$ le temps d'exécution maximal de M sur une entrée de taille N, et d'utiliser la procédure du paragraphe précédent.

3.3 Sens booléen ⇒ réel

Nous sommes maintenant armés pour montrer l'inclusion souhaitée, $NP^0_{\mathbb{R}ovs} \subset P^0_{\mathbb{R}ovs}$.

Note : dans cette preuve, deux calculs seront considérés équivalents si les tests qu'ils effectuent ont tous le même résultat. On travaillera alors sur ces classes d'équivalence plutôt que sur les calculs en eux-mêmes.

Soit un langage $L \in \text{NP}^0_{\mathbb{R}ovs}$. Par la définition donnée en section 2.2, il existe un langage $A \in \text{P}^0_{\mathbb{R}ovs}$ et un polynôme entier $r(n)$ tels que :

$$(x_1, \ldots x_n) \in L \Rightarrow \exists z \in \{0,1\}^{r(n)} / < x, z > \in A$$

Comme $A \in \text{P}^0_{\mathbb{R}ovs}$, il existe une machine BSS T qui décide ce langage en temps polynomial. La condition précédente peut alors se formuler comme : « $\exists z \in \{0,1\}^{r(n)}$, $\exists c$ chemin de calcul de T tels que le chemin de calcul c accepte l'entrée $< x, z >$. Rappelons que l'on entend ici « chemin de calcul » au sens « série de résultats de tests ».

Pour tout $x \in \mathbb{R}^n$ et tout $z \in \{0,1\}^{r(n)}$, on peut vérifier en temps polynomial t par machine BSS sur $\mathbb{R}ovs$ si $< x, z > \in A$. On notera que le polynôme t ne dépend que de A.

On appelle alors $H_1, \ldots, H_n$ l'ensemble des hyperplans de $\mathbb{R}^n$ déterminés par des équations linéaires à coefficients dans $\{-2^{t(n)}, \ldots, 2^{t(n)}\}$ et A l'ensemble des cellules délimitées par les H_i. Comme montré dans la section 3.2, la machine BSS décidant L se comportera de façon identique pour tous les points d'un même élément de A.

On donne alors un algorithme en temps $\mathrm{P}^0_{\mathbb{R}ovs}$ qui décide L :

- Localiser l'entrée $x = (x_1, \ldots, x_n)$, à l'aide de l'algorithme vu dans la section 3.2, qui s'exécute en un temps polynomial. Le résultat est un système d'inéquations linéaires à coefficients dans $\{-2^{r(n)}, \ldots, 2^{r(n)}\}$ qui détermine un élément de A. On appelera S cet élément.
  On doit maintenant vérifier si $S \subset L$ ou bien $S \subset L^c$. Vu que l'on a fait l'hypothèse $\mathrm{P} = \mathrm{NP}$, on peut utiliser pour cela un algorithme (booléen) en temps NP.
- Choisir de façon non-déterministe un $z \in \{0,1\}^{r(n)}$ et un chemin calcul c de T. On appelle alors P l'ensemble des x tels que l'entrée $< x, z >$ suit le chemin c sur la machine T ».
- L'ensemble des entrées suivant un même calcul est nécessairement une cellule ou une union de cellules. On accepte donc l'entrée si et seulement si P inter $S \neq 0$.
  Il s'agit d'un problème de programmation linéaire vérifiable en temps polynomial par une machine BSS.

Cet algorithme s'exécute bien en temps polynomial sur une machine BSS $\mathbb{R}_{ovs}$ déterministe, qui décide un langage de $NP^0_{\mathbb{R}ovs}$. Comme aucune hypothèse n'a été faite sur ce langage, on a bien montré l'inclusion $NP^0_{\mathbb{R}ovs} \subset P^0_{\mathbb{R}ovs}$, et, par suite, le résultat voulu.

□

3.4 Conclusion

L'article original ayant inspiré ce travail de rédaction ([1]) donne, outre des explications plus rigoureuses, une généralisation du précédent résultat à l'ensemble de la hiérarchie polynomiale. Le livre [3] donne quant à lui une introduction générale au modèle de BSS.

L'équivalence du problème $\mathrm{P} = \mathrm{NP}$ dans le modèle de Turing et le modèle R*ovs* tend à montrer que ce dernier est un cadre de travail tout à fait raisonnable pour les questions classiques de complexité. On pourra enfin s'intéresser aux modèles comprenant davantage d'opérations légales.

Je tiens à remercier Sylvain Perifel pour m'avoir fait découvrir ce sujet et pour m'avoir aidé tout au long de ce travail.

Références

[1] H. Fournier, P. Koiran, *Lower bounds are not esier over the reals.* 2000.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to algorithms.* 2001.

[3] B. Poizat, *Les petits cailloux.* 1995.

[4] L. G. Valiant, *Completeness classes in algebra.* 1979.

[5] A. M. Turing, *On computable numbers, with an Application to the Entscheidungsproblem.* 1936.

[6] M. Minsky, *Size and structure of universal turing machines using Tag systems.* 1962.