Complexité Effective.

Dr Serge Burckel, Université de la Réunion

Nous proposons une notion de complexité en temps des machines de Turing plus proche de ce que nous attendons d'un programme effectif. Nous montrons que les classes P et NP interprétées dans ce modèle de complexité effective sont distinctes.

1. Introduction.

Les machines de Turing sont des automates ayant un nombre fini d'états qui se déplacent de lettre en lettre sur un mot pour y lire ou écrire ou ajouter ou changer d'état suivant un programme codé dans une fonction de transition. Ces machines sont des modèles mathématiques de nos programmes. Chaque transition de la machine de Turing compte pour une étape dans le temps et nous souhaitons que cette complexité en temps soit la plus petite possible en fonction de la taille de l'entrée initiale. Une fonction polynomiale est ainsi souhaitée pour caractériser l'efficacité du calcul. Par ailleurs, le fait de se déplacer sans rien changer sur le mot pour y chercher une information éloignée ne correspond pas vraiment à la réalité de nos ordinateurs qui ont la possibilité d'un accès direct à une case mémoire et notamment dans un espace de mémoire à lecture seule (ROM). Ces aspects pratiques de la complexité en temps des machines de Turing nous conduisent aux définitions suivantes.

Définition. *La complexité effective d'un calcul de machine de Turing M est le nombre d'étapes de calcul sans compter les déplacements et les lectures dans une zone mémoire du ruban considérée comme une ROM. Ces étapes en ROM sont dites neutres et les autres sont dites effectives.*

Cette notion de temps de calcul est plus proche de celle de nos ordinateurs disposant d'accès directs à leurs cases mémoires. Notez que l'on a restreint la définition standard uniquement aux accès à une ROM car cela sera suffisant pour le reste de notre exposé. On peut aussi noter qu'une machine de Turing ayant k états ne peut effectuer sur une zone de ROM de taille t qu'un nombre limité d'étapes neutres successives sans entrer ensuite dans une boucle perpétuelle. Cette limite est clairement égale à $k.t$.

Le deuxième point concernant l'efficacité des calculs nous conduit à une redéfinition des classes de complexités polynomiales. Pour cela on se fixe un entier K suffisamment grand comme $K = 6$ ou encore $K = 10^{100000}$. Le dernier cas n'est pas très réaliste mais reste cependant dans le cadre de la suite de notre exposé.

Définition. *Etant fixé un nombre K, une machine de Turing M (déterministe ou non) est dite effectivement polynomiale si pour tout entier n et tout mot w de longueur n, tout calcul de M à partir du mot w prend au plus Kn^K étapes effectives avant d'atteindre un état acceptant ou rejetant. La classe P^K (resp. NP^K) est l'ensemble des langages décidés par machines de Turing déterministes (resp. non déterministes) qui sont effectivement polynomiales.*

2. Résultat.

Comme pour le résultat d'indécidabilité du problème de l'arrêt, nous aurons besoin de simuler le calcul d'une machine de Turing à partir d'un codage de celle-ci.

Définition. *Etant donnée une machine de Turing M déterministe ou non, le code de M est un mot noté $[M]$ qui contient la liste finie représentative de sa fonction de transition. La simulation du calcul d'une machine de Turing consiste à effectuer, à partir du code $[M]$ et d'un mot initial w, les étapes de calcul de M sur le mot w. Cette simulation est dite neutre si elle ne fait que chercher des informations dans le code $[M]$ sans le modifier au cours du calcul.*

Observons que la simulation neutre de p étapes effectives de calcul d'une machine de Turing M prend également p étapes effectives puisque les étapes neutres de recherche de la transition à effectuer ne comptent pas.

Proposition. *Pour tout entier $K \geq 2$, $P^K \neq NP^K$.*

Preuve. Soit l'ensemble L des codes $[M]$ pour lesquels il existe un calcul rejetant de M à partir du mot $[M]$ de longueur n en au plus Kn^K étapes effectives.

Nous allons montrer que le langage L est dans NP^K mais n'est pas dans P^K.

Supposons que $L \in P^K$. Il existe alors une machine de Turing déterministe M effectivement polynomiale qui décide donc L en au plus Kn^K étapes effectives sur des entrées de longueur n. Soit alors l'entrée $W = [M]$ de longueur N.

• Si $W \in L$ alors il existe un calcul rejetant de M à partir de W en au plus KN^K étapes effectives. Puisque M est déterministe, ce calcul est unique et il n'existe pas de calcul acceptant. Donc $W \notin L$.

• Si $W \notin L$ alors il n'existe pas de calcul rejetant de M à partir de W en au plus KN^K étapes. Puisque ce nombre d'étapes est supposé suffisant pour cette machine particulière M, elle est en mesure de conclure et d'accepter W. Mais alors $W \in L$.

Dans les deux cas nous obtenons une contradiction.

Montrons que $L \in NP^K$. Considérons pour cela une machine de Turing non déterministe S qui étant donnée une entrée w réalise le programme non déterministe suivant :

Dans un choix de calcul, rejeter immédiatement le mot w.

Dans un autre choix de calcul, si w est un code $[M]$ de longueur n d'une machine de Turing M, préserver ce code $[M]$ en ROM et en faire une copie sur un espace de travail, deviner un calcul rejetant en au plus Kn^K étapes effectives de M sur w et si une vérification par simulation neutre montre que ce calcul est bien rejetant alors accepter w.

La machine S décide bien le langage L. Par ailleurs, elle ne tombe pas dans les contradictions précédentes pour $W = [S]$. On aura $W \in L$. En effet, nous avons défini S de telle sorte qu'elle puisse immédiatement rejeter tous les mots. Elle peut donc rejeter en particulier le mot W et aisément le deviner et le vérifier. Elle acceptera W dans l'autre choix de calcul. Cette existence d'un calcul acceptant est la condition nécessaire et suffisante de décision pour une machine de Turing non déterministe comme S.

■

3. Conclusion.

Premièrement, tout ce qui a été fait ici peut également être appliqué aux classes de complexités exponentielles EXP et $NEXP$ interprétées dans un même cadre effectif. Deuxièmement, pour obtenir le même résutat $P \neq NP$ dans leurs définitions historiques, le point technique de ce travail qu'il s'agit d'adapter concerne la simulation. En effet, simuler les Kn^K étapes de calcul d'une machine de Turing M sur son propre code $[M]$ de longueur n peut multiplier par n cette complexité puisqu'il faut chercher à chaque fois dans le code la transition à effectuer. La notion de simulation neutre nous permet d'éviter ce décalage.

Après des années de recherches dans les directions $P = NP$ ou $P \neq NP$, nous pensons que le schéma de preuve présenté dans ce travail est minimal et peut servir de modèle à une preuve plus formelle dans le cadre classique de l'informatique théorique. Mais il nous semble aussi que les définitions historiques pourraient elles aussi être adaptées à l'évolution des mathématiques et des techniques. La question n'est peut-être pas close mais le débat est ouvert.