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.

Nous considérons un modèle de machines de Turing qui se rapproche de celui de nos ordinateurs disposant d'accès directs à la mémoire. Pour se faire, tout en restant le plus général possible, nous considérons qu'une machine de Turing dispose de deux rubans : un ruban de travail classique et un ruban à lecture seule (ROM) sur lequel est écrit le mot d'entrée. La tête de lecture sur ce ruban à lecture seule ne fait que se déplacer, lire et changer d'état et nous ne comptons pas ces opérations.

Définition. *On suppose qu'une machine de Turing dispose toujours de deux rubans : un ruban à lecture seule (ROM) et un autre ruban classique de travail. Le mot initial est écrit sur les deux rubans : sur le ruban ROM pour y être préservé et sur le ruban de travail pour y être modifié. La complexité effective d'un calcul d'une machine de Turing est le nombre d'étapes effectuées sur le ruban de travail (étapes effectives) sans compter celles effectuées en ROM (étapes neutres).*

On peut noter qu'un calcul terminant sur un mot d'entrée de taille n d'une machine de Turing ayant s états peut effectuer au maximum $n.s$ étapes neutres successives sinon il entre dans une boucle perpétuelle. Cette limitation linéaire pourrait à elle seule justifier que les étapes neutres ne soient pas comptées dans la complexité que nous souhaitons polynomiale dans un objectif d'efficacité. Cependant, les définitions classiques

$$P = \bigcup_k DTIME(n^k) \qquad NP = \bigcup_k NTIME(n^k)$$

ne posent aucune limite sur le paramètre k. Pour être plus réaliste et effectif, nous fixons une limite K à ce paramètre. Nous pouvons la choisir suffisamment grande comme avec $K = 16$ ou encore $K = 10^{100000}$. Le dernier cas n'est cepedant pas très réaliste non plus mais reste toutefois dans le cadre de la suite de notre exposé.

Définition. *Etant fixée une paire $E = (C, K)$ de nombres entiers positifs, une machine de Turing (déterministe ou non) est dite effectivement polynomiale si pour tout entier n et tout mot w de longueur n, tout calcul à partir du mot w prend au plus $C.n^K$ étapes effectives pour atteindre un état acceptant ou rejetant. La classe P^E (resp. NP^E) 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 $[M]$ représentatif de sa fonction de transition. La simulation du calcul d'une machine de Turing M consiste à effectuer, à partir de son 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.*

Nous avons défini les notions de ROM et de simulation neutre afin d'être le plus effectif possible sans avoir à considérer les fonctions de complexité à leurs limites asymptotiques.

Théorème. Pour toute paire $E = (C, K)$ d"entiers strictement positifs, $P^E \neq NP^E$.

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 $C.n^K$ étapes effectives.

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

Supposons que $L \in P^E$. Il existe alors une machine de Turing déterministe M effectivement polynomiale qui décide L en au plus $C.n^K$ 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 $C.N^K$ 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 $C.N^K$ étapes effectives. Puisque ce nombre d'étapes est supposé suffisant pour cette machine particulière M, elle doit être en mesure de conclure et d'accepter W. Mais alors $W \in L$.

Dans les deux cas nous obtenons une contradiction.

Montrons que $L \in NP^E$.

Soit une machine de Turing non déterministe S qui, étant donnée une entrée w de longueur n écrite sur sa ROM et son ruban de travail, 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 $[M]$ d'une machine de Turing M, deviner un calcul rejetant en au plus $C.n^K$ 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 est effectivement polynomiale et 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 $P \neq NP$ avec les définitions historiques, il s'agirait d'adapter un point technique concernant la simulation. En effet, simuler les $C.n^K$ 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. L'idée de simulation neutre avec une ROM nous a permis d'éviter ce décalage. Une adaptation de cette idée ainsi qu'une extension de la définition du langage L pourraient être envisagées dans le cas de la complexité standard. Nous pensons que le schéma de preuve présenté dans ce travail est minimal et peut servir de modèle à une preuve plus formelle de $P \neq NP$ dans le cadre classique de l'informatique théorique. Mais il nous semble aussi que les définitions historiques pourraient être adaptées à l'évolution de l'informatique et de ses techniques. La question n'est peut-être pas close mais le débat est ouvert.