JACQUES STERN

Quelques aspects du problème $P = NP$

Publications du Département de Mathématiques de Lyon, 1982, fascicule 1B
« Quelques thèmes de la théorie des algorithmes », , p. 15-26

<http://www.numdam.org/item?id=PDML_1982___1B_15_0>

QUELQUES ASPECTS DU PROBLEME P = NP

JACQUES STERN
UNIVERSITÉ DE CAEN

Les vingt dernières années ont vu un rapide développement de recherches motivées par la pratique des calculs sur machines. Contrairement à une idée parfois admise, certaines de ces recherches débouchent sur des problèmes mathématiques précis et difficiles. Le but du présent exposé est de présenter le problème P = NP qui est devenu, semble-t-il, un des grands problèmes mathématiques de la fin du XX^{e} siècle.

Nous avons choisi de présenter très rapidement le problème de façon informelle puis d'introduire progressivement les outils mathématiques permettant une approche rigoureuse fondée sur les machines de Turing. Nous donnons ensuite quelques résultats sur le problème P = NP relativisé et les hiérarchies de Meyer et Stockmeyer, dans le but d'indiquer la difficulté du problème.

Il s'agit là d'un exposé d'information : aucun résultat n'est dû à l'auteur ; il s'adresse à un public aussi large que possible de mathématiciens et ne requiert pas de connaissance préalable du sujet.

§1. PRESENTATION DU PROBLEME

On s'intéresse dans ce qui suit à des problèmes de décision : un tel problème se présente comme la description générique d'une situation à laquell. est attachée une question. Voici deux exemples :

Exemple 1 : On donne quatre entiers m, n, p, q ; a-t-on $mn = pq$?

Exemple 2 : On donne une suite finie d'entiers $n_1,\ldots,n_p$; existe-t-il une partie A de $\{1,\ldots,p\}$ telle que

$$\sum_{i \in A} n_i = \sum_{j \notin A} n_j \ ?$$

Si on dispose d'un algorithme pour résoudre l'un ou l'autre de ces problèmes, il est clair que chaque système de données conduit à un certain temps de calcul. On ne va pas donner -pour l'instant- de définition précise de cette notion mais on va se borner à considérer qu'il s'agit, en gros, du nombre d'opérations élémentaires que l'algorithme conduit à effectuer successivement. Bien entendu, le temps de calcul croît avec la taille des données ; une fonction $T : \mathbb{N} \to \mathbb{N}$ est dite borner le coût ou la complexité d'un algorithme si, sur les entrées de taille k, le calcul se déroule en temps $\leq T(k)$. Un algorithme a un coût polynômial si sa complexité est bornée par un polynôme. On note P la classe des problèmes résolubles en temps polynômial.

Tout ceci reste informel et le lecteur peut légitimement s'interroger, par exemple, sur la façon de mesurer la taille d'une donnée. On va éluder cette question en invitant le lecteur à faire appel dans chaque cas particulier à son sens de l'économie ; c'est ainsi que, dans le problème 1, il est raisonnable d'écrire les nombres m, n, p, q dans une base de numération (2 par exemple) et d'évaluer la taille du problème par

$$k = \log(m) + \log(n) + \log(p) + \log(q)$$

où log représente le logarithme de base 2. Un moment de réflexion montre alors qu'il existe un algorithme de complexité $O(k^2)$ pour résoudre le problème.

Un problème est résoluble de façon non déterministe en temps polynômial s'il existe un algorithme permettant d'obtenir toute réponse positive en un temps polynômial, moyennant l'utilisation de données numériques auxiliaires "devinées" en cours de calcul.

Le problème 2 admet un algorithme non déterministe polynômial : si la taille des données est estimée par

$$k = \sum_{i=1}^{p} \log (n_i),$$

on procède comme suit :

1. Deviner un entier $q \leqslant p$ et des entiers $i_1, \ldots, i_q$
2. Vérifier que $2 \sum_{j=1}^{q} n_{i_j} = \sum_{i=1}^{p} n_i$

Bien entendu, un calcul non déterministe n'est pas susceptible d'être réalisé par un ordinateur. Le non déterminisme se révèle cependant un outil conceptuel intéressant et naturel. Il conduit d'une part à la définition de la classe des problèmes NP qui, comme celui de l'exemple 2 sont résolubles de façon non déterministe en temps polynômial ; d'autre part, il correspond à ce qui se passe dans tout processus créateur ; c'est ainsi que le mathématicien, par exemple, n'explore pas systématiquement tous les théorèmes possibles, mais développe des lignes de recherche qui lui paraissent intéressantes ; c'est là une attitude non-déterministe.

§2. PROBLEMES NP COMPLETS

2.1. La notion de problème présentée plus haut n'a pas un caractère mathématique. On va maintenant lui substituer celle de langage ; soit Σ un ensemble fini non vide qu'on convient d'appeler alphabet ; un mot de Σ est une suite finie d'éléments de Σ ; on note Σ^* l'ensemble des mots construits sur Σ. Toute partie L de Σ^* est appelée un langage d'alphabet Σ.

A un problème Π, on associe par un codage adéquat (et économique !) un alphabet Σ et un sous-ensemble $\mathcal{D}_\pi$ de Σ^* correspondant à tous les systèmes de données possibles ; $\mathcal{D}_\pi$ est lui-même partagé en $\mathcal{D}_\pi^+$ et $\mathcal{D}_\pi^-$ suivant que la réponse au problème est oui ou non. Résoudre le problème est alors reconnaître si un mot x de Σ^* est ou non élément du langage $\mathcal{D}_\pi^+$.

Dans l'exemple 1 ci-dessus, on peut choisir un alphabet comprenant trois symboles 0, 1, # et associer au problème l'ensemble des mots de la forme

$$\#\,\mathrm{bin}(m)\,\#\,\mathrm{bin}(n)\,\#\,\mathrm{bin}(p)\,\#\,\mathrm{bin}(q)\,\#$$

où bin(m) désigne la représentation binaire de m.

2.2. Réductions polynômiales, problèmes NP complets

Soient Σ et Γ deux alphabets ; une fonction $f : \Sigma^* \to \Gamma^*$ est dite calculable en temps polynômial s'il existe un algorithme qui, pour tout élément x de Σ^* de longueur k, calcule $f(x)$ en temps borné par un polynôme de k. Une fois encore, il s'agit là d'une définition informelle qui sera précisée ultérieurement. Un langage $L \subseteq \Sigma^*$ est réductible à $L' \subseteq \Gamma^*$ s'il existe une fonction $f : \Sigma^* \to \Gamma^*$ calculable en temps polynômial telle que

$$L = f^{-1}(L')$$

Un langage L appartient à NP si on peut reconnaître de façon non déterministe, en temps polynomial, qu'un mot x est dans L.

Un langage L appartenant à NP est NP-complet si tout langage NP lui est réductible. Un problème est NP-complet si le langage associé est NP complet.

2.3. Exemples de problèmes NP-complets

De très nombreux problèmes combinatoires sont NP-complets ; en voici un échantillon. On peut se reporter à [4] pour en savoir plus.

2.3.1. Le problème SAT

Donnée : Une formule Φ du calcul propositionnel (construite à partir de variables propositionnelles $p_0, p_1, \ldots$ par négations ($\neg$), conjonctions ($\wedge$) disjonctions ($\vee$), implications ($\to$)).

Question : Φ est-elle satisfaisable, c'est-à-dire, existe-t-il une distribution de valeurs de vérité 0,1 pour les variables de Φ telle que Φ soit vraie ?

Remarque : Le problème reste NP-complet même si on se restreint aux formules Φ qui sont des conjonctions de formules du type

$$a \vee b \vee c$$

avec a, b, c de la forme p_i ou $\neg p_i$.

2.3.2. Le problème V.C (vertex cover)

Donnée : Un graphe G, un entier p.

Question : Existe-t-il un ensemble C d'au plus p sommets du graphe tel que toute arête ait au moins un de ses sommets dans C ?

2.3.3. Le problème H.C.

Donnée : Un graphe G.

Question : Existe-t-il un circuit hamiltonien dans G, c'est-à-dire une énumération des sommets tels que deux sommets consécutifs soient joints par une arête de même que le dernier et le premier ?

2.3.4. Le problème de partition

C'est l'exemple 2 du §1.

2.3.5. Le problème du voyageur de commerce

Donnée : Un ensemble fini $c_1,\ldots,c_n$ (de villes) ; une fonction distance $d(c_i,c_j)$ symétrique à valeurs entières ; un entier B.

Question : Existe-t-il une permutation σ des n premiers entiers, telle que

$$\sum_{i=1}^{n-1} d(c_{\sigma(i)}, c_{\sigma(i+1)}) + d(c_{\sigma(n)}, c_{\sigma(1)}) \leq B \ ?$$

§3. MACHINES DE TURING

On entame maintenant la partie proprement formelle de l'exposé en faisant le choix d'un type de calculateur. Le calculateur dont il s'agit appelé machine de Turing est purement théorique ; de cette façon, on se trouve affranchi des limitations inhérentes aux machines réelles.

3.1. Machine de Turing déterministe

Une machine de Turing se compose :

1. d'un nombre fini de bandes divisées en cases contiguës et illimitées dans une direction. Chaque case peut contenir un symbole pris dans un alphabet fini Σ ou être vierge ;

2. de têtes de lecture se déplaçant sur chacune des bandes ;

3. d'un dispositif de contrôle dont la configuration interne nous importe peu mais qui peut se trouver dans un certain nombre d'états $q_0,\ldots,q_n$.

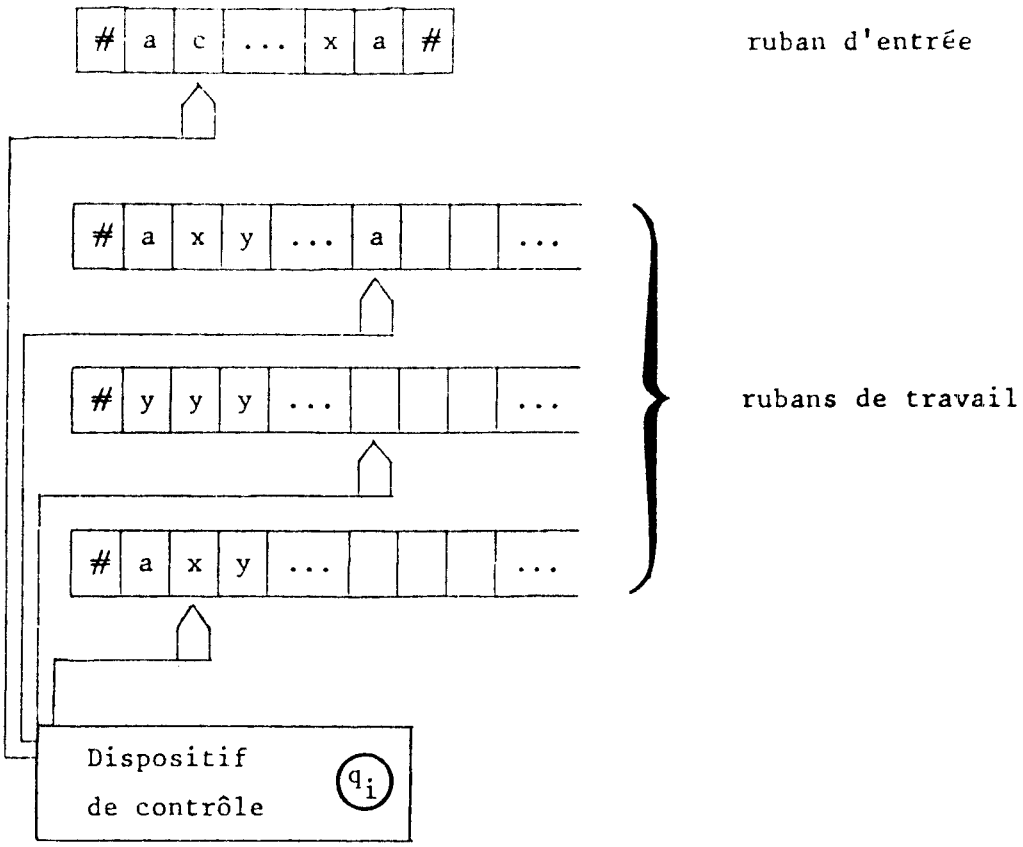

Fig. Une machine de Turing à 4 rubans

L'un des symboles noté # a un rôle particulier de séparateur ; deux états notés q_Y et q_N jouent également un rôle particulier. La machine fonctionne comme suit : une bande particulière contient la donnée qui est une suite finie d'éléments de $\Sigma - \{\#\}$ encadrée par deux symboles # . Les autres bandes sont vierges. Le dispositif de contrôle est dans l'état q_o. La machine réalise alors une suite déterminée d'opérations successives , chaque opération étant complètement définie par l'état de la machine et par les symboles lus par chacune des têtes de lecture. Chaque opération comprend trois actions :

i) effacement éventuel des symboles lus par les têtes de lecture et impression éventuelle d'autres symboles.

ii) déplacement éventuel de chaque tête de lecture d'une case vers la gauche ou la droite.

iii) transition vers un nouvel état q_j.

Le fonctionnement de la machine est en somme représentable par diverses fonctions de domaines finis. Soit Σ l'alphabet d'une machine de Turing,

b un symbole non dans Σ (destiné à indiquer qu'une case est vierge), $\tilde{\Sigma}$ l'ensemble $\Sigma \cup \{b\}$, Q l'ensemble des états et Δ un ensemble à 3 éléments $\{R,L,S\}$, permettant d'indiquer un mouvement à droite, à gauche ou une absence de mouvement. Un moment de reflexion permet de se rendre compte qu'une machine à 4 rubans est représentée par des fonctions :

$$\varphi : \tilde{\Sigma}^4 \times Q \longrightarrow \tilde{\Sigma}^4$$
$$\delta : \tilde{\Sigma}^4 \times Q \longrightarrow \Delta^4$$
$$\theta : \tilde{\Sigma}^4 \times Q \longrightarrow Q$$

correspondant respectivement aux actions d'effacement, impression, de déplacement et de transition évoquées plus haut. C'est en ce sens qu'une machine de Turing (M.T.) est un objet purement mathématique.

Le fonctionnement de la machine est tel que certaines règles sont respectées ; ces règles se traduisent facilement sur les fonctions φ, δ, θ.

i) On ne peut traverser un indicateur de fin de bande #, ni l'effacer ;

ii) On ne peut imprimer un symbole # ;

iii) On ne peut modifier les données ;

iv) si l'un des états particuliers q_Y ou q_N est atteint, il n'y a plus d'effacement, de déplacement, ni de transition vers un autre état ; la machine s'arrête donc ; si q_Y est atteint, on dit que la machine accepte la donnée ; si elle atteint q_N elle la refuse.

3.2. Machine non déterministe

Une machine de Turing non déterministe (MTND) se présente de la même façon qu'une MT à cela près qu'à chaque étape, les trois actions de la machine (effacement / impression, déplacement, transition) sont à choisir parmi un nombre fini de combinaisons.

Ainsi à une donnée est attachée, non un calul unique, mais une série de calculs parallèles, chaque nouvelle opération ouvrant un branchement.

3.3. Langage reconnu par une machine

Soit Σ un alphabet fini et $L \subset \Sigma^*$ un langage. L est reconnu par une MT, M si

$$\forall x \in \Sigma^* \quad (x \in L \text{ si } M \text{ accepte } x)$$

L est reconnu par une MT en temps polynômial s'il existe un polynôme P et une machine M telle que :

(i) M s'arrête sur tout donnée de longueur n en au plus P(n) étapes

(ii) M reconnait L.

L est reconnu par une MTND en temps polynômial, s'il existe une machine MTND, M et un polynôme P, tels que :

(i) tout calcul réalisable par M s'arrête après au plus P(n) étapes

(ii) $x \in L$ ssi il existe au moins un calcul acceptant pour la donnée x.

On note P la classe des langages reconnus par une MT en temps polynômial et NP la classe des langages reconnus par une MTND en temps polynômial. Il est clair que P est inclus dans NP ; on ignore si cette inclusion est stricte ; c'est précisément là le problème P = NP.

3.4. Retour sur la notion de réduction polynômiale

On peut aisément définir des MT qui calculent des fonctions de Σ^* dans Γ^* (où Σ, Γ sont des langages) en adjoignant un ruban de sortie où l'impression seule est permise. Une transformation polynômiale est alors une application f de Σ^* dans Γ^* pour laquelle existe une MT, M et un polynôme P tels que M calcule f(x) en moins de $P(|x|)$ étapes.

On peut maintenant substituer la notion de transformation polynômiale à la notion informelle de fonction calculable en temps polynômial introduite dans la section 2.2 ; ceci permet de dire précisément ce qui est une réduction polynômiale et autorise la définition précise de la classe des langages NP-complets.

En l'état actuel, la question P = NP est ouverte ; si donc, on reconnait qu'un problème est NP-complet, on présume qu'il n'existe pas d'algorithme pour résoudre ce problème, qui fonctionne en temps polynômial. Les algorithmes exponentiels ne pouvant essentiellement pas être mis en œuvre, même sur les machines les plus puissantes (sauf pour de très petites valeurs des entrées), on a souvent recours pour traiter ces problèmes à des techniques de résolution approchée ou à des méthodes heuristiques.

§4. VERSIONS RELATIVES DU PROBLEME P = NP

4.1. Machine de Turing avec oracle

Une MT avec oracle est une MT à plusieurs rubans dont un ruban est particularisé et appelé ruban d'oracle. La machine a trois états particuliers $q?$, q^+ et q^- et fonctionne dès qu'est fixé un langage $A \subset \Sigma^*$. A chaque fois qu'un état $q?$ est atteint, la machine passe sans effacement, impression ni mouvement à l'état q^+ ou q^- selon que le contenu du ruban d'oracle appartient ou non à A. Au niveau coût, cette opération compte pour une unité. Le reste du fonctionnement se déroule suivant les règles habituelles, suivant le mode déterministe ou non déterministe.

Cette nouvelle définition ne conduit pas nécessairement à sortir du cadre des machines de Turing. On peut en fait introduire de nouvelles classes de langages en imaginant, par exemple, une machine opérant en temps polynômial avec pour oracle un langage reconnu par une MT opérant en un temps bien plus grand.

4.2. On définit les classes P^A et NP^A de façon analogue à P et NP.

Théorème.- (Baker - Gill - Solovay [2])

i) il existe une partie A de $\{0,1\}^*$ telle que $P^A = NP^A$

ii) il existe une partie A de $\{0,1\}^*$ telle que $P^A \neq NP^A$.

La construction d'un oracle A tel que $P^A \neq NP^A$ est donnée en appendice ; avec un peu plus de soin, on peut faire en sorte que A soit reconnu par une M.T. en temps exponentiel.

4.3. Sens de ces résultats

Il est bien clair que le résultat de Baker, Gill et Solovay n'est pas d'un grand secours d'un point de vue concret. Il doit être considéré comme une indication de la difficulté du problème P = NP. En effet, une preuve de l'égalité P = NP se présentant comme un moyen mécanique de passer d'un calcul non déterministe polynômial à un calcul déterministe polynômial se transférerait vraisemblablement au cas $P^A = NP^A$, ce qui est impossible. De même une preuve de $P \neq NP$, utilisant une méthode plus ou moins standard de diagonalisation amènerait à $P^A = NP^A$ ce qui n'est pas possible non plus. Il semble donc que la solution du problème P = NP nécessite l'introduction de techniques nouvelles.

§5. LA HIERARCHIE DE STOCKMEYER

5.1. Pour tenter d'éclairer le problème $P = NP$, Meyer et Stockmeyer ont construit toute une hiérarchie de classes de langages, qui présente une certaine analogie avec la hiérarchie de Kleene en théorie de la récursion.

On définit par récurrence la classe Σ_n^P par

i) $\Sigma_o^P = P$

ii) Σ_{n+1}^P la classe des langages reconnus par une machine NP^A avec $A \in \Sigma_n^P$

On constate que Σ_1^P est exactement la classe NP ; on note Π_n^P la classe des langages dont le complémentaire est dans Σ_n^P.

5.2. Rien n'est connu sur les propriétés de cette hiérarchie, sinon les inclusions triviales :

$$\cdots \quad \begin{matrix} \Sigma_n^P \\ \Pi_n^P \end{matrix} \subseteq \Sigma_{n+1}^P \cap \Pi_{n+1}^P \subseteq \begin{matrix} \Sigma_{n+1}^P \\ \Pi_{n+1}^P \end{matrix} \quad \cdots$$

et le résultat suivant ([8]).

Proposition.- Si pour un entier i, on a $\Sigma_i^P = \Pi_i^P$, alors pour tout entier $j \geq i$, on a $\Sigma_j^P = \Pi_j^P$; en particulier, si $P = NP$, alors pour tout i, on a $\Sigma_i^P = \Pi_i^P = P$.

5.3. Si un oracle A est fixé, on peut définir une hiérarchie relativisée dont les classes sont notées $\Sigma_n^{P,A}$, $\Pi_n^{P,A}$ ou simplement Σ_n^A, Π_n^A. De fait, même en présence d'oracles, on ne sait guère contrôler le comportement de la hiérarchie. On sait construire des oracles réalisant les situations suivantes :

(1) $P^A \neq \Sigma_1^A = \Pi_1^A$ (Baker, Gill, Solovay [2])

(2) $\Sigma_1^A \neq \Pi_1^A$ (Baker, Gill, Solovay [2])

(3) $\Sigma_2^A \neq \Pi_2^A$ (Baker, Selman [3])

L'inégalité (2) implique $\Sigma_1^A \subsetneq \Sigma_2^A$; en fait, on peut aussi réaliser :

(4) $\Sigma_1^A \subsetneq \Sigma_2^A \cap \Pi_2^A$ (Santha [5])

et également

(5) $\Sigma_1^A \neq \Pi_1^A$ et $\Sigma_2^A = \Pi_2^A$ (Santha [5])

Rien d'analogue n'est connu pour les classes de niveau ≥ 3.

Appendice : Construction d'un oracle A tel que $P^A \neq NP^A$.

On fixe une énumération M_i des machines déterministes à oracle et une énumération p_j des polynômes à cœfficients entiers. On énumère les couples (i,j).

A chaque étape k on construit un entier n_k et on décide de placer dans A un élément de longueur k ou aucun. On note $A^{(k)}$ l'oracle formé des éléments placés jusqu'à l'étape k.

On choisit n_{k+1} tel que $n_{k+1} > 2^{n_k}$ et tel que si (i,j) est le $k+1^{ie}$ couple $p_j(n_{k+1}) < 2^{n_{k+1}}$. Ecrivons n pour n_{k+1}.

(1) Si l'on obtient un calcul acceptant à partir de 0^n pour la machine M_i avec l'oracle $A^{(k)}$ en temps $\leq p_j(n)$, on n'ajoute aucun élément de longueur n.

(2) Sinon on ajoute à $A^{(k)}$ un unique élément de longueur n non appelé par l'oracle durant l'opération de M_i sur $p_j(n)$ pas. Noter que : M_i ne peut appeler l'oracle que $p_j(n)$ fois (i.e. $< 2^n$ fois).

Soit $L = \{0^n : A$ admet un élément de longueur $n\}$.

Pour vérifier si $0^n \in L$ on devine un "élément" de A de longueur n et on "questionne" l'oracle, ce qui prouve que L appartient bien à NP^A.

Si maintenant $L \in P^A$, on peut trouver i et j tels que L soit reconnu par la machine M_i fonctionnant avec l'oracle A en temps $\leq p_j$. Si (i,j) a été traité au k^{ie} pas et si $n = n_k$, on constate que M_i fonctionne à partir de 0^n avec A comme avec $A^{(k)}$. Le cas (1) conduit à une contradiction de même que le cas (2).

BIBLIOGRAPHIE

[1] A.V. AHO, J.E. HOPCROFT, J.D. ULLMAN, The design and analysis of computer algorithms, Addison-Wesley, Reading (1974).

[2] T.P. BAKER, J. GILL, R.M. SOLOVAY, Relativizations of the P = ? NP question, SIAM J. Comput 4, (1975) 431-442.

[3] T.P. BAKER, A.L. SELMAN, A second step towards the polynomial hierarchy, Proc. 17 th Ann. Symp. on Foundations of Computer Science, IEEE Computer Society Long Beach (1976) 71-75.

[4] M.R. GAREY, D.S. JOHNSON, Computers and intractability. A guide to the theory of NP-completeness, W.H. Freeman, San Francisco (1979).

[5] J.E. HOPCROFT, J.D. ULLMAN, Introduction to automata theory, languages and computation, Addison - Wesley, Reading (1979).

[6] M. SANTHA, résultats non publiés.

[7] J. STERN, Séminaire : Algorithmes et complexité, Université de CAEN (1982)

[8] L.J. STOCKMEYER, The polynomial time hierarchy, Theor. Comput. Sci. 3 (1976) 1-22.