Astérisque

P. FLAJOLET
J. M. STEYAERT

Hiérarchies de complexité et réductions entre problèmes

Astérisque, tome 38-39 (1976), p. 53-72
<http://www.numdam.org/item?id=AST_1976__38-39__53_0>

Société Mathématique de France
Astérisque 38-39 (1976) p.53-72

HIÉRARCHIES DE COMPLEXITÉ ET RÉDUCTIONS ENTRE PROBLÈMES

par

P. FLAJOLET - J.M. STEYAERT

INTRODUCTION.

L'étude des questions de complexité en algorithmique consiste essentiellement en l'évaluation et la comparaison d'algorithmes, numériques et non numériques ; cette étude conduit naturellement à se poser la question de l'optimalité d'un algorithme déterminé et donc à rechercher des bornes inférieures à la complexité de tout algorithme résolvant le problème. Un grand nombre de problèmes spécifiques ont été étudiés de ce point de vue en calcul numérique, manipulation de données non numériques, par exemple en théorie des graphes (cf. les autres exposés). Ce qui est présenté ici est essentiellement l'étude et la comparaison des différentes notions de complexité qu'il est naturel de définir : temps ou mémoire de calcul, difficulté de vérification d'une solution, c'est-à-dire complexité non déterministe. La présentation que nous adopterons ici comporte trois parties :

I.- Les modèles de calculabilité et les mesures de complexité,

II.- Les mesures non déterministes et la réductibilité entre problèmes,

III.- Les bornes inférieures sur les procédures de décision en mathématiques.

Dans chaque cas, nous ne ferons que citer les résultats existants en décrivant informellement les méthodes de preuve utilisées. De même, nous nous bornerons à citer en référence des articles d'intérêt général facile-

ment accessibles ; une bonne présentation de nombreux résultats de complexité a été donnée par Borodin (1973). Les preuves détaillées de la plupart des théorèmes cités ici apparaissent dans Flajolet-Steyaert (1975).

1.- MODÈLES DE CALCULABILITÉ ET MESURES DE COMPLEXITÉ

L'intérêt de disposer d'une formalisation précise de la notion de calculabilité est fondé sur la possibilité, dès lors, de partager les problèmes entre ceux qui sont calculables et ceux qui ne le sont pas : cette formalisation, déjà ancienne, a été introduite par les logiciens vers 1930 (cf. Kleene (1952)). L'approche introduite par Turing en 1936 est intéressante ; elle consiste à définir un processus calculable comme une manipulation, à chaque étage et de manière séquentielle, d'une quantité finie d'informations selon un programme fini prédéterminé (sur les différentes notions de calculabilité cf. Minsky (1967), Yasuhara (1971)).

D'un point de vue informatique, cette formalisation apparaît comme l'un des modèles de calculabilité : celui où la mémoire est susceptible d'un accès séquentiel par paquet d'information de taille fixée. Cette approche est non dépourvue d'intérêt pour les deux raisons suivantes :

- pour de nombreux problèmes (multiplication de nombres, reconnaissance de langages,..) les meilleurs algorithmes connus n'utilisent pas, de manière fondamentale, la possibilité d'un accès indexé *;
- il existe un lien relativement étroit entre temps et mémoire de calcul par machine à accès indexé - RAM - et les machines de Turing - TM -. L'écart peut être important pour des problèmes de complexité peu élevée (par exemple la recherche en table) où le choix d'une bonne structure de données est primordial ; à partir d'un certain niveau de complexité (classe NP, Γ), il est équivalent d'étudier les problèmes en termes de RAM ou de TM ; en revanche les TM sont plus manipulables (cf. infra).

* par accès indexé nous traduirons le terme "random access" parfois rendu par "accès aléatoire".

1.1.- Définitions

Une machine de Turing (TM) consiste en un programme déterminé contrôlant un nombre fixe de mémoires ou bandes accessibles séquentiellement, dont chaque case peut contenir une information élémentaire appartenant à un ensemble fini (par exemple binaire $\{0,1\}$). Les instructions sont :

(a) - les instructions d'échange avec la mémoire :

Ecrire (a,i) : écrire a sur la i° mémoire

Lire (i) : lire la i° mémoire

(b) - les instructions d'exploration de la mémoire :

Droite (i)
Gauche (i) } déplacement sur la i° mémoire

(c) - les instructions de contrôles habituelles : branchement, branchement conditionnel.

Les temps et mémoire de calcul sont définis de manière évidente. Par la suite, les machines de Turing introduites, sont les machines à nombre quelconque de mémoires : les machines à une bande présentent en effet une trop grande restriction sur l'accès à l'information pour que les résultats de complexité puissent être considérés comme significatifs de la complexité intrinsèque des problèmes étudiés.

Une machine à accès indexé - RAM - consiste en un programme fini opérant sur une suite de cases de mémoire susceptibles chacune de contenir un nombre entier.

Les instructions sont :

(a) - les instructions arithmétiques : $X_i \leftarrow X_j \otimes X_k$ où $\otimes$ appartient à l'ensemble d'opérations arithmétiques permises par la machine ;

(b) - les instructions d'échange avec la mémoire :

$X_i \leftarrow X_{x_j}$ (chargement indirect)

$X_{x_i} \leftarrow X_j$ (rangement indirect)

(c) - les instructions de contrôle habituelles : branchement, branchement

conditionnel, entrée/sortie.

Notation : RAM $[+,-]$ désigne l'ensemble des machines telles que $\otimes \in \{+,-\}$;
RAM $[+,-,\times]$ désigne l'ensemble des machines telles que
$\otimes \in \{+,-,\times\}$, etc...

Il est possible de définir alors un coût de chaque instruction : ce coût peut être choisi :

- soit fixe : ce sont les c-RAM ou RAM à coût constant, indépendant de la taille des opérandes,
- soit variables : ce sont les v-RAM dont le coût par opération dépend ici
  - de manière proportionnelle - de la taille des opérandes supposés représentés en binaire (RAM à coût logarithmique).

Pour une étude de ces machines, on se référera à Cook-Reckhow 1973.

Remarque 1.- Dans tous les cas, la complexité d'un problème est évaluée en fonction de la taille, non de la valeur de la donnée : un problème est résoluble en temps T(n) par une machine M ssi pour toute donnée de longueur n, M utilise au plus T(n) étapes de calcul. Dans le même esprit, à propos de bornes de complexité, on abrègera $\lim_{n \to \infty} T(n) = \infty$ par $T \to \infty$.

Remarque 2.- Afin d'éliminer des phénomènes non significatifs provenant seulement de l'ordre de grandeur des résultats des calculs, nous établirons les propriétés des classes de complexité même dans le cas où le résultat du calcul ne vaut que 0 ou 1 ; il s'agit alors d'algorithmes de solution de problèmes ou encore de reconnaissance de langages ; ces deux termes seront employés indifféremment l'un pour l'autre par la suite.

1.2.- Théorèmes de hiérarchie

Les résultats suivants montrent que, pour les mesures envisagées, il existe à chaque niveau des problèmes difficiles ; de plus, l'existence de ces problèmes difficiles sera utilisée en III pour démontrer des résultats de borne inférieure sur certaines procédures de décision en mathématiques.

THÉORÈME 1 : Soient T_1 et T_2 deux bornes de complexité (*) telles que $\frac{T_2}{T_1 \log T_1} \to \infty$; alors il existe un problème reconnaissable par TM en temps T_2, mais non reconnaissable en temps T_1.

Preuve.- Hennie et Stearns (1966). Il s'agit d'obtenir une machine universelle pour la classe des machines à nombre quelconque de bandes opérant en temps T_1, puis d'appliquer une construction diagonale appropriée. La preuve repose sur un algorithme non trivial qui simule un nombre quelconque de bandes opérant en temps T par une bande unique (ainsi qu'une certaine mémoire de calcul auxiliaire) opérant en temps T log T.

De manière plus simple, pour la complexité en mémoire, on a le théorème analogue :

THÉORÈME 2.- Soient L_1 et L_2 deux bornes de complexité telles que $\frac{L_2}{L_1} \to \infty$; alors il existe un problème reconnaissable par TM en mémoire L_2 mais non reconnaissable en mémoire L_1.

Dans le cas des machines à accès aléatoire, la structure d'accès à la mémoire rend plus aisées les constructions de machines universelles ce qui permet un résultat légèrement plus "serré" que dans les cas des machines de Turing :

THÉORÈME 3.- Soient T_1 et T_2 deux bornes de complexité telles que $\frac{T_2}{T_1} \to \infty$; alors, il existe un problème reconnaissable par v-RAM $[+,-]$ ou c RAM $[+,-]$ en temps T_2 et non reconnaissable par la même classe de machines en temps T_1.

Ainsi donc pour une large classe de mesures, il existe des problèmes traitables e-g en ressources n^3 mais dont une borne inférieure sur la complexité est $n^{2,99}$.

1.3.- Relations entre mesures

Les mesures introduites précédemment ne sont pas indépendantes. Par exemple

---

(*) les fonctions utilisées comme bornes de complexité sont supposées "honnêtes". Une définition précise n'est pas donnée ici mais toutes les fonctions usuelles, e-g les combinaisons de polynômes, exponentielles, factorielles, logarithmes sont honnêtes.

Cook et Reckhow ont démontré :

THÉORÈME 4.- (a) Si A est reconnu en temps T par une v-RAM [+,-], A est reconnaissable en temps T^2 par TM ;

(b) Si A est reconnaissable en temps T par une TM, A est reconnaissable en temps T log T par une v-RAM [+,-].

Preuve : Facile : il suffit d'évaluer le coût de simulation d'un accès aléatoire par un accès séquentiel et vice-versa.

Ceci montre un lien relativement étroit entre RAM et TM, qui rend notamment significatives les bornes inférieures de complexité exponentielles obtenues par l'intermédiaire des TM en III. Des résultats très voisins sont valables pour tous les modèles raisonnables de v-RAM en particulier v-RAM[+,-,×].

Par contre, des résultats différents ont été obtenus concernant les c-RAM [+,-,×], c'est-à-dire des RAM dont le temps de multiplication est indépendant de la taille des opérandes.

Cette assignation de coût correspond à un degré de parallèlisme important et en fait Simon [1974] a démontré l'équivalence de ces machines avec les machines vectorielles de Pratt-Rabin-Stockmeyer [1974] qui constituent une sorte de "machine APL". Pratt et al. ont notamment démontré :

THÉORÈME 5.- Pour les machines vectorielles :

(a) ce qui est effectuable en mémoire S par machine de Turing est effectuable en temps S par machine vectorielle ,

(b) ce qui effectuable en temps T par machine vectorielle est effectuable en mémoire T^2 par machine de Turing.

Il y a donc correspondance entre le temps de calcul par machine vectorielle, ou encore par c-RAM [+,×] et la mémoire de calcul par machine de Turing. Ce résultat entraîne que des problèmes difficiles vis-à-vis des TM,

ont vis-à-vis de la mesure associée aux c-RAM multiplicatives une complexité bien moins élevée, mettant ainsi en évidence le gain correspondant au parallèlisme.

1.4.- Compléments

Nous ne discuterons pas ici des problèmes de la relation entre temps et mémoire de calcul pour une même classe de machines (TM ou RAM). On se contentera de noter que pour les TM, le temps est borné inférieurement par l'encombrement mémoire et supérieurement par une exponentielle de la mémoire. C'est un problème ouvert de savoir si chacune de ces inégalités est stricte ; sur ce problème, consulter Cook (1973) et le résultat récent inédit de W. Paul (1975).

Signalons également que la théorie de la complexité axiomatique due à Rabin et Blum permet de prévoir la forme des résultats concernant les mesures : théorèmes de hiérarchie et relation entre mesures en sont un exemple ; par contre, elle ne fournit pas l'évaluation des fonctions de relation qui reste à déterminer par l'étude des mesures particulières ; cette théorie est exposée par exemple dans Yasuhara (1971).

Enfin, en complément au théorème 4 sur le lien entre accès séquentiel (TM) et accès indexé (RAM), il convient de citer le résultat suivant, dû à Cook (1971) :

THÉORÈME 6.- Il existe une classe d'algorithmes fonctionnant en temps n^2 utilisant une mémoire séquentielle restreinte (*) pour lesquels une mémoire à accès aléatoire permet la construction d'algorithmes équivalents fonctionnant en temps linéaire.

Il en résulte l'existence d'algorithmes non triviaux opérant en temps linéaire sur RAM pour un grand nombre de problèmes de traitement de caractè-

---

(*) Cette classe est celle des 2 DPDA - twoway deterministic pushdown automata-qui en plus de la donnée explorable dans les deux sens sans écriture, utilisent une mémoire de pile auxiliaire.

re ("string matching") ; cf. Aho Hopcroft Ullman (1974).

2.- LES MESURES NON DÉTERMINISTES

2.1.- Le concept de non déterminisme

La solution d'un grand nombre de problèmes naturels s'exprime simplement en terme d'algorithmes non déterministes. On peut introduire, afin de formaliser cette notion, un certain nombre d'approches conceptuellement voisines et mathématiquement équivalentes, cf. Cook (1971), Karp (1972), Karp (1975).

(a) un algorithme non déterministe comporte en plus des instructions usuelles, des instructions de choix. On peut considérer l'exécution d'une telle instruction comme un dédoublement du programme en deux sous-programmes concurrents (parallèlisme) ; le programme principal se termine en fournissant un résultat dès que l'un des deux sous-programmes se termine. Ainsi le programme non déterministe suivant teste si un nombre est composite (= non premier).

```
COMPOSITE (b)
    soit b = (b_o b_1 ... b_n)_2
pour i ← 0 à n faire
    d_i ← choix (0,1) ;
soit d = d_o d_1 ... d_n)_2
    si d ≠ 1 et d ≠ b et d | b alors résultat ← vrai ;
```

De même qu'en 1.1, une définition précise peut s'obtenir au moyen de la notion de machine non déterministe ; celle-ci déjà ancienne provient des premiers travaux de théorie des langages et des automates (Rabin et Scott (1959)). Aux algorithmes non déterministes correspondent ainsi les machines de Turing non déterministes - susceptibles de plusieurs évolutions dans un état donné. Une telle machine accepte une donnée dès qu'il existe une suite (un choix) d'évolutions qui conduit à l'acceptation ; le temps de calcul non déterministe est alors la longueur de la plus courte (la "première" dans

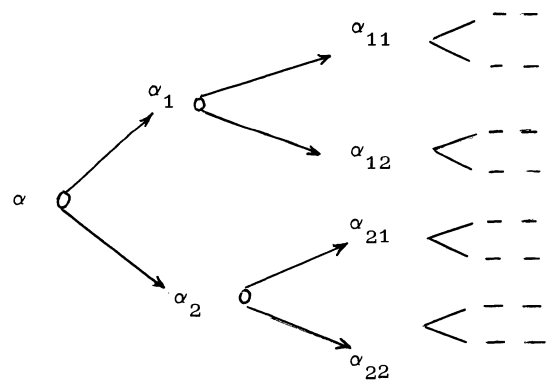
l'approche parallèle) suite de choix conduisant à l'acceptation. De même la mémoire de calcul non déterministe est l'encombrement mémoire minimum d'un calcul conduisant à l'acceptation.

(b) un <u>système de preuve</u> : un certain nombre de problèmes peuvent être décrits par un système formel ou système de preuve : les formules valides du calcul propositionnel (tautologies). Le temps de calcul de (a) a ici comme homologue le nombre d'étapes d'une plus courte démonstration et la mémoire de calcul la taille du plus grand mot intervenant dans une preuve.

(c) les <u>problèmes de recherche arborescente</u> : correspondent au "schéma" suivant :

RECHERCHE (α) = si condition (α) alors vrai
sinon RECHERCHE (α_1) ou
RECHERCHE (α_2).

La procédure de recherche se développe donc selon un arbre de possibilités dans lequel on recherche une branche conduisant à un sommet satisfaisant la condition terminale.



<u>Exemple</u> : la recherche de la satisfabilité d'une expression booléenne peut s'écrire :

SAT ($F(x_1,x_2,\ldots,x_n)$) = si $n = 0$ alors F
sinon
SAT ($F(0,x_2,\ldots,x_n)$) ou
SAT ($F(1,x_2,\ldots,x_n)$) ;

Il est facile de vérifier qu'un problème de recherche exprimé sous cette forme est résoluble par un algorithme non déterministe qui choisit la "bonne" branche de l'arbre ; le temps de calcul de l'algorithme non déterministe est alors égal à la profondeur de l'arbre de possibilités. L'équivalence entre (a) et (c) est ainsi claire.

Un algorithme de recherche par essai-retour ("backtracking") est défini comme explorant successivement chacune des branches, s'arrêtant et revenant en arrière pour modifier son dernier choix s'il rencontre une impossibilité ; c'est donc un algorithme déterministe qui résout le problème de recherche ; remarquons que contrairement à l'algorithme non déterministe équivalent, l'exploration de toutes le branches correspond à un coût exponentiel en la profondeur de l'arbre des possibilités. On a ainsi :

THÉORÈME 7.- Si un problème est reconnaissable en temps T(n) par machine de Turing non déterministe à choix binaire, il est reconnaissable en temps $2^{T(n)}$ par machine de Turing déterministe.

Dans le cas de la mémoire, on a par contre le résultat suivant dû à Savitch (1970) :

THÉORÈME 8.- Si un problème est reconnaissable en mémoire L(n) par machine de Turing non déterministe, il est reconnaissable en mémoire $L^2(n)$ par machine de Turing déterministe.

Preuve.- pour une preuve simplifiée, cf. Flajolet-Steyaert (1975).

La question se pose alors naturellement à propos du théorème 6 de l'inéluctabilité du coût exponentiel en temps de la réduction du non déterminisme. Il est possible de l'exprimer de la manière suivante :

Problème 1.- Existe-t-il un polynôme p(x) (par opposition à l'exponentielle du théorème 6) tel que tout problème résoluble en temps T par machine non déterministe, soit résoluble en temps p(T) par machine déterministe ?

Il est raisonnable d'étudier l'incidence de ce problème par les classes

de complexité de niveau le plus bas, dans lesquelles se trouvent un grand nombre de problèmes courants. En particulier, on définira :

P = la classe des problèmes résolubles en temps polynômial par machine déterministe.

NP = la classe des problèmes résolubles en temps polynômial par machine non déterministe (donc clairement $P \subset NP$).

Grâce aux résultats de la section I, on se rendra compte facilement que les classes introduites sont indépendantes du modèle particulier de machine utilisée ; on peut prendre indifféremment TM, c-RAM additives, v-RAM quelconques.

En fait, P représente la classe des problèmes traitables de manière efficace par l'une des méthodes classiques de programmation : exploration simple d'une structure de donnée adequate - arbre ou graphe, méthodes tabulaires de la programmation dynamique ... NP représente, selon (c) ci-dessus la classe de tous les problèmes résolubles par un backtracking de profondeur polynômialement borné (en fonction de la taille de la donnée).

Exemples : P contient les problèmes de tri, les calculs arithmétiques usuels, les problèmes de recherche, certains problèmes de graphe : plus courtes distances et flot maximum...

NP contient en plus des problèmes énumérés ci-dessus, un certain nombre de problèmes classiques de théorie des graphes : coloriage, problèmes de commis-voyageurs, ainsi que différents problèmes de recouvrement et partition d'ensembles, habituellement résolus par des algorithmes "combinatoires".

D'après ce qui a été dit plus haut, les seuls algorithmes déterministes connus pour les problèmes énumérés ci-dessus utilisent un temps qui est l'exponentielle d'un polynôme. Une instance particulière du problème 1 est alors :

Problème 1' : A-t-on P = NP ?

En plus de son importance pratique évidente, celui-ci tire son intérêt de ce qu'on peut montrer que toute réponse à (1') fournit une réponse à (1).

2.2.- Réduction et problèmes complets

Cook (1971) a montré l'important résultat qu'il existe des problèmes "caractéristiques" de la classe NP tout entière : ces problèmes qui sont dans NP, sont tels que tout élément de NP leur est "facilement" réductible ; il en découle que P = NP ssi les problèmes caractéristiques de NP sont dans P ; ou encore : pour étudier NP, il suffit d'étudier la complexité de l'un quelconque des problèmes caractéristiques.

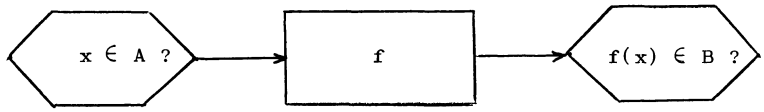
On peut donner un fondement précis à la notion de "facilement réductible" en posant :

Définition.- A est dit p-réductible à B, s'il existe une fonction f telle que :

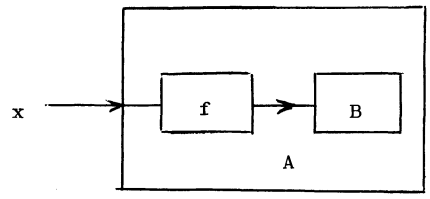
i) $\forall x \quad x \in A \longleftrightarrow f(x) \in B$

ii) f est calculable en temps polynômial (par exemple : par machine de Turing déterministe).

La fonction f de réduction transforme donc les questions sur A en questions sur B selon le schéma :



Il en découle que tout algorithme pour B permet de construire un algorithme pour A :



f étant "facile", dire que A est réductible à B revient à dire que B est au moins aussi compliqué que A.

Des notions intuitives voisines réductibilité étaient connues : c'est ainsi que sachant calculer la fonction carré = $x \to x^2$, on en"déduit facilement" un mode de calcul de la fonction produit $x,y \to x \cdot y$; les problèmes qui "se traitent par" les méthodes de la programmation linéaire, ont été largement étudiés...

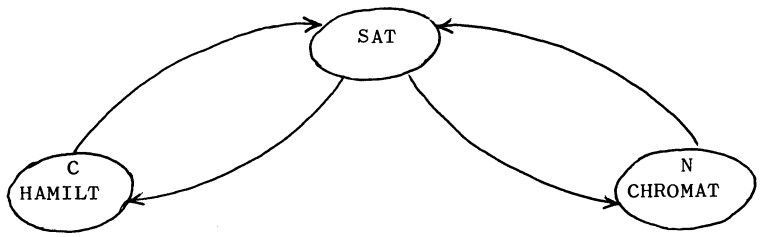
Ces définitions posées, on a alors :

THÉORÈME 9.- Le problème de la satisfiabilité des expressions booléennes est tel que tout problème dans NP lui est réductible (on dit encore que c'est un problème NP-complet).

Corollaire.- P = NP ssi le problème de la satisfiabilité des expressions booléennes est résoluble en temps polynômial.

Preuve.- Elle utilise la définition formalisée de NP en terme de machine de Turing non déterministe ; on montre alors qu'on peut décrire de manière concise les calculs de machines non déterministes au moyen des expressions booléennes.

Ce théorème permet ensuite de prouver l'équivalence du point de vue de la complexité d'un grand nombre de problèmes combinatoires. Par exemple, la recherche d'un circuit hamiltonien dans un graphe est (p-) équivalente au problème de détermination de nombre chromatique. La preuve utilise le diagramme suivant :



où les flèches représentent la p-réductibilité :

C. HAMILT $<$ SAT } découle, par le th. 8 de ce que
N. CHROMAT $<$ SAT } C. Hamilt et N. Chromat sont dans NP.

SAT $<$ C. HAMILT } utilisent des constructions particulières
SAT $<$ N. CHROMAT }

Suivant Karp (1972) et Sahni (1973), on peut énoncer :

THÉORÈME 10.- (i) un grand nombre de problèmes combinatoires sont équivalents du point de vue complexité.

(ii) P = NP ssi il existe un algorithme polynômial pour résoudre l'un quelconque d'entre eux. Parmi ces problèmes, citons : les problèmes de recouvrement minimaux d'ensemble, les problèmes de programmation en nombres entiers, les problèmes de coloriage de graphe, certains problèmes de flots, de séquencement...

Enfin, Garey, Johnson et Stockmeyer (1974) ont essayé de trouver des problèmes les plus simplifiés possible qui soient encore NP-complets : c'est le cas, par exemple, de la détermination de coloriabilité avec trois couleurs des graphes planaires...

2.3.- Programmation heuristique et programmation approchée

Selon toute vraisemblance, en tout cas dans l'état actuel de nos connaissances, les problèmes NP-complets sont difficiles à traiter puisque seuls en sont connus des algorithmes utilisant un temps exponentiel. On peut pratiquement proposer, pour éviter cet inconvénient, deux approches que nous appellerons par commodité programmation heuristique et programmation approchée.

Programmation heuristique.- ceci consiste à élaborer des tests qui permettent, dans la plupart des cas, d'éliminer une partie des recherches dans l'arbre des possibilités ; sans que le cas le pire cesse d'utiliser un temps exponentiel, on parvient parfois à améliorer notablement en moyenne le com-

portement de l'algorithme. Cette approche est utilisée en théorie des jeux en démonstration automatique...

Programmation approchée.- Elle s'applique aux problèmes qui s'expriment comme recherche d'un optimum, il existe donc une mesure d'évaluation du degré d'acceptabilité d'une solution. On peut alors rechercher non la solution optimale, mais une solution quasi-optimale, en un sens à préciser.

C'est cette approche que nous étudierons maintenant ; les résultats de Johnson (1973), Karp (1975), Carey-Johnson (1975) montrent que des situations très diverses peuvent se produire :

(a) Il existe un algorithme efficace (= dans P) qui fournit toujours une solution proche de l'optimum. C'est le cas pour l'organisation en paquets ("binpacking") dont une heuristique fournit une solution de coût au plus les 11/9 de l'optimum.

(b) Il existe une suite d'algorithmes qui fournissent une solution quasi-optimale ; cependant le temps de calcul des algorithmes approchés croît rapidement lorsqu'on cherche à réduire l'écart entre les solutions quasi-optimales admises et l'optimum.

C'est le cas du problème de réalisation d'une commande ("Subset Sum" ou "Knapsack") : pour chaque k, il existe un algorithme qui donne une solution approchée à au $\frac{1}{k}$ de l'optimum mais dont le temps de calcul est $O(n^k)$.

(c) Aucun algorithme approché n'est connu (et probablement n'existe) ; c'est le cas du problème de coloriage de graphe : Carey et Johnson ont en effet montré que la détermination approchée du nombre chromatique d'un graphe même à un facteur de 2 près était un problème NP-complet, donc aussi difficile que le problème exact dont on était parti.

3.- COMPLEXITÉ DE PROCÉDURES DE DÉCISION MATHÉMATIQUES

Les problèmes de la classe NP sont tels que les seuls algorithmes connus pour les résoudre opèrent en temps exponentiel ; cependant, aucune borne

inférieure de ce type n'est connue sur leur complexité. Nous montrons ici qu'il existe des problèmes naturels dont la complexité est intrinsèquement exponentielle.

Dans la partie 1, on a démontré l'existence de problèmes complexes à tous niveaux ; dans la partie 2 le concept de réductibilité a été introduit. Il est possible de combiner les méthodes de 1 et 2 de la manière suivante : si une classe C de problèmes (contenant des problèmes difficiles d'après 1) est réductible facilement (par une méthode analogue au théorème 9 dans 2) à un problème π donné, on peut en déduire que le problème π est lui-même aussi difficile que le problème le plus complexe de C. De manière simplifiée, si un problème difficile se réduit facilement à un problème π, alors π est lui-même difficile.

Les premiers résultats en ce sens ont été obtenus par Meyer-Stockmeyer (1972), (1973) auquel nous empruntons l'exemple suivant :

Expressions avec carré : on considère l'ensemble E des expressions formées sur un alphabet binaire $\{a,b\}$ au moyen des opérations de :

- union notée +
- concaténation notée .
- carré : A^2 dénote A.A.

Ainsi : $a(a+b)$ dénote l'ensemble des mots $\{aa,bb\}$

$(ab+ba)^2 + a^2b^2 + b^2a^2$ dénote les mots de longueur 4 contenant deux a et deux b.

L'équivalence de deux expressions de E est clairement décidable puisque seuls sont dénotés des ensembles finis ; un algorithme évident consiste à développer complètement les expressions et comparer les ensemble de mots obtenus, mais utilise un temps plus qu'exponentiel. Ce comportement, dans le cas le pire, est inévitable car Meyer et Stockmeyer ont démontré :

THÉORÈME 11.- Tout algorithme décidant de l'équivalence d'expressions dans E nécessite un temps exponentiel.

Preuve.- On démontre que tout problème résoluble en temps exponentiel par machine de Turing est p-réductible à l'équivalence d'expressions dans E ; ce résultat s'établit par un lemme de description des calculs de machines de Turing opérant en temps exponentiel. On conclut en utilisant le théorème 1 de hiérarchie.

Remarquons que ce résultat de borne inférieure établi pour le temps de calcul par machine de Turing est encore valable pour de larges classes de machines grâce au théorème 4 de relation entre mesures.

La méthode de preuve fournit également le résultat surprenant suivant :

PROPOSITION.- Tout algorithme qui décide de l'équivalence sur E est améliorable dans une proposition exponentielle en une infinité de points (phénomène de speed-up).

Ainsi tout algorithme est intrinsèquement long mais peut toujours être amélioré partiellement (i.e. il n'existe pas d'algorithme optimal). D'autres résultats du même ordre ont été obtenus concernant des problèmes de mots ou d'équivalence algébrique.

Ces méthodes ont été appliquées avec succès à l'étude de la complexité d'un certain nombre de procédures de décision en mathématiques. On sait, par exemple, depuis Presburger que les propriétés de nombres entiers vis-à-vis de l'addition (= la théorie du 1° ordre de la structure $< N, + >$) sont décidables. Fischer et Rabin (1974) ont démontré :

THÉORÈME 12.- Toute procédure décidant les propriétés des entiers vis-à-vis de l'addition nécessite un temps au moins $2^{2^{cn}}$ dans le cas le pire.

Des bornes inférieures du même ordre ont aussi pu être établies pour un grand nombre de théories logiques ; d'autre part, l'examen des procédures de décision connues montre que l'encadrement obtenu est assez étroit ; cf. Rackoff (1975). On peut montrer que les mêmes phénomènes de "speed up" que ci-dessus se produisent. Enfin, des bornes inférieures même sur la complexité

non déterministe peuvent être établies : elles s'appliquent donc à la longueur de preuve dans tout système formel et il en résulte que même avec des heuristiques parfaites ou un opérateur qui guide parfaitement la machine, la complexité des problèmes étudiés reste aussi élevée. La pertinence de ces résultats est discutée dans Rabin (1974).

CONCLUSION.-

Nous avons donné ici l'esquisse d'une classification assez générale des problèmes rencontrés en algorithmique non numérique. Celle-ci est fondée sur un concept de calculabilité précis et manipulable qui prend en compte toutes les instructions effectuées par la machine : le concept de machine de Turing ; on a vu que les résultats se transposaient ensuite facilement à une large classe de machines très voisines des calculateurs réels. Cette classification prend le relais, à partir d'un certain niveau de complexité, de celle fondée sur l'étude des structures de données, qui apparaît dans Knuth (1968-73). Notons également que le concept de réduction entre problèmes qui permet d'évaluer la complexité d'un certain nombre de procédures de décision mathématiques, réduit significativement le nombre de questions indépendantes (cf. Even Tarjan (1975) pour un problème de graphes complet vis-à-vis d'une classe plus élevée que NP).

Enfin, si la réponse à P = NP semble très difficile à trouver, d'autres questions d'importance peuvent être abordées : recherche de types généraux de problèmes qui soient dans P (cf. le théorème 6 de Cook), ce qui rejoint une étude plus générale des méthodologies en programmation; étude des approximations aux problèmes NP et évaluation des heuristiques d'après le comportement moyen des algorithmes.

* * *

RÉFÉRENCES

[1] AHO,HOPCROFT, ULLMAN (1974) : The design and Analysis of Computer Algorithms, Addison Wesley.

[2] BORODIN (1973) : Computational Complexity, Theory and Practice ; in Currents in the Theory of Computing, Prentice Hall.

[3] COOK (1971) : The Complexity of theorem proving Procedures, 3rd ACM SIGACT.

[4] COOK (1971) : Linear time simulation of deterministic two-way pushdown automata, IFIP-Ljubljana.

[5] COOK (1973) : An observation on Time-Storage Trade-off, 5^{th} ACM-SIGACT.

[6] COOK, RECKHOW (1973) : Time bounded Random Access Machines, JCSS 7.

[7] EVEN,TARJAN (1975) : A combinatorial problem which is Polynomial Space Complete, in 7^{th} ACM SIGACT.

[8] FISCHER, RABIN (1974) : Super-exponential Complexity of Presburger Arithmetic, MIT C.C. Rep.

[9] FLAJOLET, STEYAERT (1975) : Complexité logique des Algorithmes. Notes de l'Ecole IRIA de Complexité, Berder 1974.

[10] GAREY, JOHNSON, STOCKMEYER (1974) : Some simplified NP-complete problems, 6^{th} ACM. SIGACT.

[11] HENNIE, STEARNS (1966) : Two tape simulation of multitape Turing machines, JACM 13.

[12] KARP (1972) : Reducibility among Combinatorial Problems, in Complexity of Computer Computations, Plenum Press.

[13] KARP (1975) : On the Computational Complexity of Combinatorial problems, Network 5.

[14] KLEENE (1952) : Introduction to metamathematics, North Holland.

[15] KNUTH (1968-73) : The Art of Computer Programming I,II,III, Addison Wesley.

[16] MEYER, STOCKMEYER (1973) : The equivalence problem for regular expression with squaring requires exponential space, in 13^{th} IEEE SWAT.

[18] MINSKY (1967) : Computation : Finite and Infinite Machines, Prentice Hall.

[19] PRATT, RABIN, STOCKMEYER (1974) : A Characterisation of the Power of Vector Machines.

[20] RABIN (1974) : Theoretical Impediments to Artificial Intelligence, in IFIP Stockholm.

[21] RACKOFF (1975) : The Computational Complexity of some logical theories, Thèse MIR.

[22] SAHNI (1972) : Some related problems from network flows, game theory and integer programming, in 13th IEEE SWAT.

[23] SAVITCH (1970) : Relationships between non-deterministic and deterministic tape complexities, JCSS 4.

[24] SIMON (1974) : On the power of multiplication in Random Access Machines, Cornell Uni. CS Rep.

[25] YASUHARA (1971) : Recursive Function theory and logic, Academic Press.

[26] HOPCROFT, PAUL, VALIANT (1975) : On time versus space and related problems, 16th F.O.C.S.

P.FLAJOLET et J.M.STEYAERT
Institut de Recherche d'Informatique
et d'Automatique
Domaine de Voluceau - Rocquencourt
78150 LE CHESNAY