Complexité des algorithmes

par

Robert Rolland

R. Rolland, Aix–Marseille Université,
Institut de Mathématiques de Marseille – I2M
Luminy Case 930, F13288 Marseille CEDEX 9
e-mail : robert.rolland@acrypta.fr

1. Introduction et notations

Lorsqu'on essaie de donner un sens à la notion de procédure effective ou d'algorithme on est amené aux notions de langage, grammaire, automate fini, automate à pile, machine de Turing.

Nous nous contentons ici d'étudier la notion de machine de Turing qui semble la plus adaptée à la notion intuitive de procédure effective et quelques variations autour de cette notion, en vue de définir les outils de complexité qui nous sont nécessaires pour étudier les algorithmes utilisés en cryptographie.

Un alphabet V est un ensemble fini de symboles. Les mots construits sur cet alphabet sont les suites finies d'éléments de V. On notera V^* l'ensemble de tous les mots, y compris le mot vide ϵ. Tout mot $w \in V^*$ a une longueur, notée $l(w)$, qui est la longueur de la suite finie qui représente w.

On définit la concaténation de deux mots, w_1 et w_2, comme étant la suite constituée des symboles de w_1 suivi des symboles de w_2. On notera w_1w_2 la concaténation des mots w_1 et w_2.

Un langage est un sous-ensemble de V^*. Plusieurs problèmes se posent alors : comment spécifier un langage ? Quand un langage a été défini, comment reconnaître qu'un mot est ou n'est pas dans ce langage ?

2. Rappels sur les machines de Turing

2.1. Description intuitive. — Commençons par donner une vision intuitive de ce qu'est une machine de Turing. Une machine de Turing est constituée d'une mémoire infinie qu'on peut considérer comme un tableau infini à droite indexé par $\mathbb{N}$. C'est le ruban. On peut écrire sur le ruban avec un alphabet dit alphabet de ruban (un symbole par case mémoire). On dispose aussi d'une tête de lecture qui se déplace sur le ruban, d'un ensemble d'états du système comportant un état initial et un ensemble d'états accepteurs.

À l'instant initial le ruban contient un mot d'entrée écrit avec un sous-alphabet de l'alphabet de ruban qu'on appelle l'alphabet d'entrée. Dans toutes les autres cases il y a un symbole spécial de l'alphabet de ruban : le blanc. La tête de lecture se trouve sur la première case.

Une étape consiste à lire le symbole qui est sous la tête de lecture et en fonction de ce qui est lu et de l'état de la machine, à écrire un symbole à la place de celui qu'on a lu (éventuellement réécrire le même symbole), à changer l'état de la machine et à déplacer la tête de lecture d'une case à droite ou à gauche.

2.2. Formalisation. — Une machine de Turing déterministe est donnée par :

$$M = (Q, \Gamma, \Sigma, \delta, q_0, F),$$

où :

— Q est un ensemble fini d'état,
— Γ est un alphabet (alphabet de ruban),
— $\Sigma \subset \Gamma$ (alphabet d'entrée),

— $q_0 \in Q$ est l'état initial,
— $F \subset Q$ est l'ensemble des états accepteurs,
— δ fonction d'une partie D de $Q \times \Gamma$ dans $Q \times \Gamma \times \{L, R\}$ est la fonction de transition.

On suppose en outre l'existence d'un symbole spécial $B \in \Gamma \setminus \Sigma$ appelé le caractère blanc. Remarquons encore que L, R représentent les deux directions (gauche et droite) de déplacement de la tête de lecture.

Une configuration est un élément de $Q \times \Gamma^* \times \mathbb{N}$. C'est donc la donnée de l'état à l'instant considéré, du mot sur le ruban à cet instant et de la position de la tête. Remarquons que le mot sur le ruban partira de la première case et ira jusqu'au dernier caractère non blanc. C'est bien un mot (suite finie) car à l'instant initial on part d'un mot.

Une dérivation en une étape à partir d'une configuration (q, w, n) est définie lorsque la valeur $\delta(q, w[n])$ est définie, et lorsqu'on peut bouger la tête de lecture dans le sens voulu. Cette dernière condition exclut le cas pour lequel $n = 0$ et $\delta(q, w[n]) = (q_1, b, L)$.

Si la dérivation en une étape est définie, et si $\delta(q, w[n]) = (q_1, b, X)$ on obtient à partir de la configuration de départ la configuration (q_1, w_1, n_1) où :

— w_1 est le mot obtenu en remplaçant dans w le terme $w[n]$ par b et en limitant le mot au dernier symbole non blanc ;
— n_1 vaut $n + 1$ si $X = R$ et $n - 1$ si $X = L$.

On notera :

$$(q, w, n) \vdash (q_1, w_1, n_1),$$

cette dérivation en un pas.

A partir de là on définit la dérivation en un nombre fini d'étapes et on note :

$$(q, w, n) \vdash^* (q\prime, w\prime, n\prime)$$

lorsqu'il existe une suite finie $(q_i, w_i, n_i)_{i=0,\cdots,k}$ de telle sorte que :

$$(q_0, w_0, n_0) = (q, w, n) \text{ et } (q_k, w_k, n_k) = (q\prime, w\prime, n\prime),$$

et que pour tout i tel que $0 \leqslant i < k$ on ait :

$$(q_i, w_i, n_i) \vdash (q_{i+1}, w_{i+1}, n_{i+1})$$

.

Un mot w est accepté par une machine de Turing M s'il existe une configuration $(q\prime, w\prime, n\prime)$ avec $q\prime \in F$ de telle sorte que :

$$(q_0, w, 0) \vdash^* (q\prime, w\prime, n\prime).$$

On définit le langage accepté par la machine de Turing M comme étant l'ensemble $L(M)$ des mots acceptés par M.

Une machine de Turing peut

— s'arrêter :
  — dans un état accepteur,
  — car on tombe sur une configuration pour laquelle la fonction de transition n'est pas définie,
  — car on ne peut pas déplacer la tête de lecture comme indiqué,
— ne jamais s'arrêter.

Un mot est accepté si la machine s'arrête dans un état accepteur.

Remarque 2.1. — Si pour un mot d'entrée w_0 et un état initial q_0 la machine ne s'arrête pas, il existe une suite infinie $(q_i, w_i, n_i)_{i\geqslant 0}$ telle que $(q_i, w_i, n_i) \vdash (q_{i+1}, w_{i+1}, n_{i+1})$ et pour laquelle quel que soit i la fonction de transition soit définie, on puisse déplacer la tête dans le sens requis et l'état q_i ne soit pas accepteur.

Remarque 2.2. — Il existe d'autres modèles de machines de Turing équivalents à celui que nous venons de décrire. En particulier, dans certains modèles le ruban est « infini à droite et à gauche ».

2.3. Machines de Turing non-déterministes. — Une machine de Turing non-déterministe est définie comme une machine de Turing déterministe à l'exception de la définition de la fonction de transition δ qui devient une fonction de transition Δ d'une partie D de $Q \times \Gamma$ dans l'ensemble des parties de $Q \times \Gamma \times \{L, R\}$. Avec cette nouvelle définition on définit aussi la dérivation en un pas. Pour cela on part d'une configuration (q, w, n) et on regarde si Δ est définie sur $(q, w[n])$. Si oui alors on peut poursuivre et on dira que (q_1, w_1, n_1) dérive en un pas de (q, w, n) s'il existe un triplet (q_1, b, X) dans l'image $\Delta(q, w[n])$ de

telle sorte que w_1 soit le mot obtenu en remplaçant dans w le terme $w[n]$ par b et en limitant le mot au dernier symbole non blanc et que n_1 soit $n+1$ si $X = R$ et $n-1$ si $X = L$.

On définit aussi la notion de dérivation en un nombre fini d'étapes, puis la notion de langage accepté. La différence avec le cas des machines déterministes est que maintenant à chaque étape il peut y avoir plusieurs dérivations possibles. Une dérivation dépend donc d'une séquence de choix.

Là encore lors d'une dérivation une machine de Turing non-déterministe peut :

— s'arrêter :
 — dans un état accepteur,
 — si on tombe sur une configuration pour laquelle la fonction de transition n'est pas définie,
 — si on ne peut pas déplacer la tête de lecture comme indiqué,
— ne jamais s'arrêter.

Un mot est accepté s'il existe une séquence de choix pour laquelle la machine s'arrête dans un état accepteur.

Remarquons qu'à partir d'un mot accepté, il peut y avoir des séquences de choix pour lesquelles la machine s'arrête dans un état qui n'est pas accepteur, ou même ne s'arrête pas.

Théorème 2.3. — *Si $\mathcal{L}$ est le langage accepté par une machine de Turing non-déterministe, il existe une machine de Turing déterministe dont le langage accepté est aussi le langage $\mathcal{L}$.*

2.4. Décidabilité. — Un problème désigne la description générale d'une question. Par exemple : trouver un facteur non trivial d'un entier (si ce facteur existe). Quand on se donne une valeur particulière de l'entier en question on a une instance du problème.

Pour résoudre un tel problème on essaie de décrire un algorithme. C'est cette notion qu'on pense formaliser au mieux grâce à la notion de machine de Turing. Encore faut-il se préoccuper de l'effectivité de la procédure (permet-elle de répondre effectivement à la question ?) et aussi de l'efficacité (le calcul est-il vraiment réalisable ?).

Dans la suite nous considérons les problèmes de décision (ou décisionnels), c'est-à-dire ceux pour lesquels la réponse est oui ou non.

Le problème précédent de la recherche d'un facteur non trivial d'un entier admet une version sous la forme d'un problème de décision : étant donnés deux entiers positifs n et k, l'entier n a-t-il un facteur m vérifiant $2 \leqslant m \leqslant k$?

Étant donné un problème de décision, $\mathcal{L}$ le langage formé par l'ensemble des représentations des entrées possibles, on désigne par $\mathcal{L}_y$ le sous-ensemble de $\mathcal{L}$ correspondant aux entrées pour lesquelles la réponse est oui.

Définition 2.4. — Nous dirons qu'une machine de Turing déterministe résout le problème si elle possède un état q_y tel que la machine s'arrête dans l'état q_y si et seulement si le mot initial est dans $\mathcal{L}_y$.

Si la réponse est non, la machine peut s'arrêter, et alors ce sera dans un état autre que q_y, ou ne pas s'arrêter.

Définition 2.5. — Nous dirons qu'une machine de Turing non-déterministe résout le problème s'il existe une séquence de choix et un état q_y tels que la machine s'arrête dans l'état q_y si et seulement si le mot initial est dans $\mathcal{L}_y$.

De plus, même si la réponse est oui, pour certaines séquences de choix elle peut s'arrêter dans un autre état que q_y ou ne pas s'arrêter. Si la réponse est non, pour toute séquence de choix elle s'arrête dans un état différent de q_y ou ne s'arrête pas.

On introduit le complémentaire d'un problème de décision. Pour ce problème complémentaire, la réponse est oui quand la réponse au problème initial est non et la réponse est non quand la réponse au problème initial est oui.

Définition 2.6. — Un problème de décision est décidable s'il existe une machine de Turing qui le résout et une machine de Turing qui résout son complémentaire.

Un problème de décision est décidable s'il existe une machine de Turing qui le résout et qui n'a pas d'exécution infinie.

On comprend bien que cette notion est importante, car s'il y a des exécutions infinies on ne peut jamais savoir si l'exécution ne va pas s'arrêter avec une réponse positive (sauf si on connaît *a priori* une majoration du nombre de pas pour obtenir une réponse positive).

3. Temps d'exécution, complexité

Nous allons maintenant nous préoccuper de l'efficacité d'une procédure. Pour cela introduisons les notions indispensables suivantes :

— la taille de l'entrée est le nombre de symboles nécessaires pour décrire les données initiales. Par exemple si on suppose que l'alphabet d'entrée est $\{0,1\}$ et si l'entrée doit représenter un entier n alors la taille est $k = \lfloor \log_2(n) \rfloor + 1$. Le plus souvent, une évaluation de l'ordre de grandeur de la taille suffit, nous prendrons alors $\log_2(n)$ comme valeur approchée de la taille.

— le temps d'une exécution est le nombre de pas (éventuellement infini) effectués lors d'une exécution. Remarquons que si la machine est non-déterministe le temps d'une exécution dépend non seulement de l'entrée mais aussi de la séquence de choix utilisée.

3.1. Classes de problèmes. — On distingue plusieurs classes de problèmes suivant les coûts des algorithmes qui les résolvent. On dit qu'un algorithme effectuant un calcul le fait en temps polynomial s'il existe un entier d tel que le nombre d'opérations sur les symboles de l'alphabet (le plus souvent les bits) pour mener à bien le calcul sur des entrées de taille au plus k soit un $\mathcal{O}(k^d)$.

Un algorithme est en temps exponentiel si son temps d'exécution est un $\mathcal{O}(e^{ck})$ (où $c>0$).

Un algorithme est en temps sous-exponentiel si son temps d'exécution est un $\mathcal{O}(e^{f(k)})$ où $f(k)$ est un $o(k)$. Par exemple $f(k) = ck^\gamma(\log(k))^{1-\gamma}$ avec $0 < \gamma < 1$ ou encore $f(k) = \frac{k}{\log(\log(k))}$.

3.2. La classe P. —

Définition 3.1. — Un problème de décision est dans la classe P des problèmes en temps polynomial s'il existe une machine de Turing déterministe qui le résout en temps polynomial lorsque la réponse est oui.

Définition 3.2. — Un problème de décision est dans la classe co-P si le problème complémentaire est dans la classe P.

Il est facile de voir que P=co-P.

3.3. La classe NP. —

Définition 3.3. — Un problème de décision est dans la classe NP des problèmes en temps non-déterministe polynomial s'il existe une machine de Turing non déterministe qui le résout en temps polynomial lorsque la réponse est oui. C'est-à-dire qu'il existe une constante A et un entier $d \geqslant 1$ tels que lorsque la réponse est oui, il existe une séquence de choix qui donne la solution en un nombre de pas $\leqslant Ak^d$ où k est la taille du mot initial.

Définition 3.4. — Un problème de décision est dans la classe co-NP si le problème complémentaire est dans la classe NP.

Il est clair que $P \subset NP$. On ne sait pas si $P = NP$. On ne sait pas non plus si NP=co-NP.

Remarque 3.5. — Un problème de décision est dans la classe NP si étant donné une instance du problème pour laquelle la réponse est oui, quelqu'un disposant d'une puissance de calcul exponentielle peut fournir un moyen de vérifier la réponse en temps polynomial (donnant ainsi un certificat).

3.4. Réduction d'un problème à un autre. —

Définition 3.6. — En résolvant un problème $\mathcal{P}_1$ nous dirons que nous faisons appel à un $\mathcal{P}_2$—oracle si l'algorithme que nous utilisons crée à partir de l'instance initiale du problème $\mathcal{P}_1$, une instance du problème $\mathcal{P}_2$.

Définition 3.7. — Si $\mathcal{P}1$ et $\mathcal{P}_2$ sont deux problèmes, nous dirons que $\mathcal{P}1$ se réduit à $\mathcal{P}_2$ (en temps polynomial) s'il existe un algorithme en temps polynomial pour $\mathcal{P}1$ qui utilise au plus un nombre polynomial d'appels à un $\mathcal{P}_2$—oracle.

Définition 3.8. — Un problème $\mathcal{P}$ est NP-dur si tout problème de la classe NP est réductible à $\mathcal{P}$. Si de plus $\mathcal{P}$ est dans la classe NP, nous dirons qu'il est NP-complet.

4. Probabilisation des algorithmes non-déterministes. Classes ZPP, RP, BPP

Nous avons vu qu'une machine de Turing non-déterministe dépend d'une séquence de choix. Nous allons supposer maintenant que ces choix successifs dépendent d'un tirage aléatoire dans un espace probabilisé bien choisi que nous allons décrire.

Nous partons d'une machine de Turing non-déterministe vue de manière particulière :

$$\mathcal{T} = (Q, \Gamma, \Sigma, \Delta, q_0, F)$$

où

- — Q est l'ensemble fini des états de la machine ;
- — Γ est l'alphabet de ruban ;
- — $\Sigma \subset \Gamma$ est l'alphabet d'entrée ;
- — Δ est une application d'un sous ensemble D de $Q \times \Gamma$ dans l'ensemble $\mathcal{P}_2\left(Q \times \Gamma \times \{L, R\}\right)$ des parties à deux éléments de $Q \times \Gamma \times \{L, R\}$;
- — q_0 est l'état initial ;
- — F est l'ensemble des états accepteurs.

Remarquons que la particularité ici est d'imposer que Δ soit une application dans l'ensemble des parties à deux éléments de $Q \times \Gamma \times \{L, R\}$ et non pas dans l'ensemble de toutes les parties. En fait on peut toujours représenter une machine de Turing non-déterministe sous cette forme (on pourra s'en convaincre en pensant que dans un arbre, un noeud qui a plusieurs fils peut toujours être remplacé par une succession de noeuds ayant deux fils (on itère la dichotomie : le premier de la liste et la liste des suivants).

Revenons sur le fonctionnement d'une telle machine. Pour cela on doit définir la notion de configuration de la machine de Turing. Une configuration est un triplet (q, w, n) où q est un état, w est un mot écrit avec l'alphabet de ruban, et n un entier $\geqslant -1$ (on donne ici une petite variante de la définition des machines non-déterministes donnée précédemment de manière à prendre en compte plus simplement le cas où la machine s'arrête car elle ne peut pas exécuter le déplacement demandé de la tête de lecture). Au départ, la configuration initiale est $(q_0, w_0, 0)$. On passe éventuellement d'une configuration (q, w, n) à une autre grâce à la fonction Δ. Plus précisément :

1. si on est dans un des trois cas suivants :
   (a) $q \in F$;
   (b) $(q, w[n]) \notin D$;
   (c) $n = -1$;
   alors la machine s'arrête, c'est-à-dire qu'on ne peut plus avoir de transition vers une configuration ultérieure ;
2. sinon, on passe à une configuration suivante en prenant n'importe lequel des deux élément $(q', b, X) \in \Delta(q, w[n])$ et on construit la configuration (q', w', n') en prenant le mot w' obtenu en remplaçant dans le mot w le symbole $w[n]$ par le symbole b et en coupant la chaîne obtenue après le dernier caractère non blanc ; quant à n' il est égal à $n + 1$ si $X = R$ (si on déplace la tête de lecture vers la droite) et à $n - 1$ si $X = L$ (si on déplace la tête de lecture vers la gauche). On dira que (q', w', n') dérive en un pas de (q, w, n).

On voit qu'à partir d'une configuration (q, w, n), on peut passer (en un pas) à deux configurations différentes en utilisant à chaque étape les deux transitions possibles appartenant à $\Delta(q, w[n])$. On numérotera par 0 et par 1 ces deux transitions, ce qui fait qu'une transition sera entièrement définie par les valeurs q et $w[n]$ de la configuration d'origine (q, w, n), ainsi que par son numéro 0 ou 1. Ainsi à partir d'un mot d'entrée w_0 en prenant de proche en proche les deux transitions d'une configuration aux configurations suivantes on va construire un arbre appelé l'arbre d'exécution de la machine à partir du mot w_0. Certaines des branches de cet arbre peuvent être infinies :

$$(q_0, w_0, 0) \vdash (q_1, w_1, n_1) \vdash \cdots \vdash (q_i, w_i, n_i) \vdash \cdots$$

où le signe $\vdash$ signifie que la configuration qui est à droite du symbole découle de la configuration qui est à gauche du symbole par une dérivation en un pas. Les autres sont finies et de la forme :

$$(q_0, w_0, 0) \vdash (q_1, w_1, n_1) \vdash \cdots \vdash (q_k, w_k, n_k),$$

où la configuration (q_k, w_k, n_k) n'a pas de suivante pour l'une des trois raisons invoquées précédemment. La machine de Turing s'arrête alors (sur cette branche) et dans le cas où $q \in F$ elle s'arrête avec une sortie valide qui est le mot w_k. Sinon elle s'arrête sans donner de résultat. Lorsque la machine de Turing s'arrête sur une branche, cette branche est finie et le temps d'exécution relatif à cette branche est le nombre de transitions (ou de segments) qui la composent (nombre de noeuds moins un).

À partir de cette machine de Turing non-déterministe $\mathcal{T}$ on définit une machine de Turing probabiliste $\mathcal{T}_P$ de la façon suivante : pour chaque paire $(q, a) \in D$ on dispose de la probabilité $P_{(q,a)}$ équirépartie sur l'ensemble à deux éléments $\Delta(q, a)$. Ainsi, quand on se trouve dans une configuration (q, w, n) qui a des successeurs pour la dérivation en un pas, la probabilité de passer dans la configuration (q', w', n') est $1/2$ où la configuration (q', w', n') est construite à partir de la configuration (q, w, n) et de la transition $(q', b, X) \in \Delta(q, w[n])$ comme indiqué précédemment. Maintenant, l'arbre d'exécution a la propriété

suivante : chaque segment de branche qui fait passer d'un nœud à un nœud suivant par une dérivation en un pas est pondéré par le nombre $P_{(q,w[n])}(q', b, X) = 1/2$.

Nous noterons E_{w_0} l'ensemble des branches (finies ou non) de l'arbre d'exécution de la machine de turing à partir de l'entrée w_0.

L'arbre d'exécution de la machine $\mathcal{T}_P$ est défini de la façon suivante : un nœud a ou bien deux successeurs ou aucun si la machine s'arrête là. Si on note $A = \{0, 1\}$, on va définir l'application f de $A^{\mathbb{N}}$ dans l'ensemble E_{w_0} des branches de l'arbre d'exécution à partir de w_0 :

$$f \ : \ A^{\mathbb{N}} \longrightarrow E_{w_0}$$

définie par

1. $$f(x_0, x_1, \cdots) = (e_0, e_1, \cdots)$$
   si les nœuds $(e_0, e_1, \cdots)$ de cette branche infinie se déduisent les uns des autres grâce aux transitions numérotées par $x_0, x_1, \cdots$ (les x_i valent 0 ou 1)
2. $$f(x_0, x_1, \cdots) = (e_0, e_1, \cdots, e_k)$$
   si les nœuds jusqu'à e_k se déduisent les uns des autres grâce aux transitions numérotés $x_0, x_1, \cdots, x_{k-1}$ mais qu'aucun nœud suivant de e_k se déduise de e_k par la transition numérotée x_k.

Cette fonction jouera un rôle important par la suite. Si on munit A de la topologie discrète et $A^{\mathbb{N}}$ de la topologie produit, alors $A^{\mathbb{N}}$ est un espace topologique compact (compact de Cantor). Si on considère que $A = \{0, 1\}$ est un groupe est que $A^{\mathbb{N}}$ est le groupe produit, alors $A^{\mathbb{N}}$ est un groupe compact dont nous noterons μ la mesure de Haar de masse totale 1. On peut remarquer par exemple que

$$\mu\left(\{x_0\} \times \cdots \times \{x_{n-1}\} \times A \times A \times \cdots\right) = \left(\frac{1}{2}\right)^n.$$

On définit alors la probabilité P_{w_0} sur E_{w_0} comme probabilité image de la probabilité μ par la fonction f. Ainsi lorsque la machine s'arrête, la suite des nœuds constituant les configurations successives de

cette branche finie $(e_0, e_1, \cdots, e_n)$ est obtenue grâce aux transitions successives $(x_0, x_1, \cdots, x_{n-1})$. La probabilité de cette exécution est alors $(1/2)^n$.

Une telle machine de Turing sera appelée machine de Turing probabiliste. Soit une machine de Turing probabiliste qui résout un problème $\mathcal{P}$ et qui pour toute donnée initiale w_0 s'arrête pour P_{w_0}-presque toute séquence de choix (c'est-à-dire que la probabilité d'une exécution infinie est nulle). On définit le coût moyen $M(w_0)$ pour la donnée w_0 comme la moyenne de tous les temps d'exécution à partir du mot initial w_0 pour toutes les séquences de choix. C'est-à-dire que si on note $E_{w_0,t}$ l'événement : l'algorithme partant de l'entrée w_0 s'arrête en un temps t, alors :

$$M(w_0) = \sum_{t=0}^{\infty} t P_{w_0}(E_{w_0,t}) = \sum_{t=0}^{\infty} \frac{t}{2^t}.$$

On définit alors le coût probabiliste de l'algorithme pour une entrée de taille $\leqslant k$:

$$T(k) = \max_{taille(w)\leqslant k} M(w).$$

4.1. La classe ZPP. — Un problème de décision appartient à la classe ZPP (Zero-sided Probabilistic Polynomial) s'il existe une machine de Turing probabiliste qui le résout et qui s'arrête en temps probabiliste polynomial (que la réponse soit oui ou non), c'est-à-dire que le coût probabiliste $T(k)$ est majoré par un polynôme en la variable k et en particulier est fini pour tout k. Mais attention il peut très bien arriver que pour une certaine entrée w, l'algorithme ait certaines exécutions sur cette entrée qui soient infinies, bien que ceci ait une probabilité nulle, plus précisément l'événement $E_{w,\infty}$: l'algorithme partant de l'entrée w ne s'arrête pas, est tel que $P_w(E_{w,\infty}) = 0$.

Voici un exemple typique. L'entrée est un entier n de taille k. On tire au sort un entier $1 \leqslant w \leqslant n$ en supposant que tous les entiers de l'intervalle considéré sont équiprobables. Supposons qu'on dispose d'un algorithme en temps majoré par $T(n) \leqslant Ck^d$ qui s'applique à w. Cet algorithme avec la probabilité $0 < p < 1$ donne un résultat qui permet de conclure et avec la probabilité $1-p$ donne un résultat qui ne permet

pas de conclure. Tant qu'on ne peut pas conclure on retire au sort un nouveau w (éventuellement un nombre déjà tiré). Alors la probabilité d'avoir une exécution infinie est nulle. De plus si on compte le nombre moyen $I(n)$ de tirages de w faits avant la conclusion on a :

$$I(n) = \sum_{t=1}^{\infty} t(1-p)^{(t-1)}p = \frac{1}{p}.$$

Donc le coût probabiliste de l'algorithme est $\leqslant \frac{C}{p}k^d$.

4.2. La classe RP. — Un problème de décision appartient à la classe RP (Random Polynomial) s'il existe une machine de Turing probabiliste qui le résout et qui s'arrête toujours en temps maximum polynomial (que la réponse soit oui ou non) c'est-à-dire :

$$\max_{\substack{taille(w)\leqslant k \\ c\in choix}} t(k,w,c) = \mathcal{O}(k^d),$$

où $t(k,w,c)$ est le temps d'exécution à partir du mot w de taille $\leqslant k$ pour une séquence de choix c. De plus, lorsque la donnée implique la réponse oui, la machine s'arrête dans l'état q_y avec une probabilité $p \geqslant 1/2$, lorsque la réponse est non la donnée n'est pas acceptée.

Ainsi, si la machine répond oui (s'arrête dans l'état q_y), on est sûr que la réponse est oui, en revanche si la machine répond non, la vraie réponse n'est pas nécessairement non. La probabilité conditionnelle pour que, sachant que la donnée implique une réponse oui, on obtienne une réponse non est $1-p$. On laisse au lecteur le soin de chercher la probabilité de se tromper lorsqu'on obtient pour réponse non, c'est-à-dire la probabilité conditionnelle que la donnée implique la réponse oui, sachant que la machine a répondu non.

4.3. La classe BPP. — Un problème de décision appartient à la classe BPP (Bounded Probabilistic Polynomial) s'il existe une machine de Turing non-déterministe qui le résout et qui s'arrête toujours en temps maximum polynomial (que la réponse soit oui ou non), et qui de plus, lorsque la donnée implique la réponse oui, s'arrête dans l'état q_y

avec une probabilité $\geqslant 2/3$, et qui lorsque la réponse est non s'arrête aussi dans l'état q_y avec une probabilité $\leqslant 1/3$.

4.4. Relations entre ces classes. — On sait que :

$$ZPP = co - ZPP,$$

$$BPP = co - BPP,$$

$$P \subset ZPP \subset RP \subset BPP,$$

$$RP \subset NP,$$

$$ZPP = RP \cap co - RP.$$

Un algorithme qui montre qu'un problème est dans ZPP est un algorithme de Las Vegas. Un tel algorithme s'exécute en temps probabiliste polynomial, il peut ne pas se terminer, mais s'il s'arrête, il donne la bonne réponse.

Un algorithme qui montre qu'un problème est dans RP est un algorithme de Monte-Carlo. Un tel algorithme s'exécute en temps maximum polynomial, il se termine donc toujours, mais la réponse qu'il donne n'est pas toujours correcte : s'il répond oui, on est sûr que la réponse est oui, en revanche s'il répond non la réponse n'est pas sûre. On peut en général rendre, à partir d'un tel algorithme, la probabilité de se tromper très petite.

Un algorithme qui montre qu'un problème est dans BPP est un algorithme d'Atlantic City. Un tel algorithme se termine toujours en temps maximum polynomial, mais si la réponse est oui, il peut répondre non, et répondre oui si la réponse est non. Bien entendu la probabilité d'une réponse erronée est plus faible que la probabilité d'une réponse correcte. Là encore on peut souvent, à partir d'un tel algorithme, rendre la probabilité d'une réponse fausse très petite.

4.5. Modèle uniforme, modèle non uniforme. — Toutes les définitions que nous avons données concernant les classes de complexité, l'ont été en utilisant, pour résoudre un problème, une seule machine de Turing quelle que soit la taille de l'instance du problème. Dans ce cas nous nous intéressons à une seule ressource : le temps d'exécution. La

taille du programme (c'est-à-dire la taille des ensembles qui définissent la machine de Turing) reste constante lorsqu'on change d'instance du problème. C'est ce qu'on appelle le modèle uniforme. Il est possible de définir d'autres modèles de calcul. En particulier le modèle non uniforme. Dans le modèle non uniforme, un problème est résolu par une suite $(M_k)_k$ de machines de Turing : toute instance de taille k est soumise à la machine M_k dont la taille (taille des ensembles qui la définissent) est $g(k)$. Cette machine M_k s'exécute sur l'instance x de taille k en temps $T(k, x)$. Dans ce cas nous nous intéressons à deux types de ressources : le temps d'exécution et la taille du programme. En imposant des contraintes sur ces deux types de ressources on obtient des classes de complexités non uniformes. Dans toutes les définitions qui suivent nous nous placerons toujours dans le cadre du modèle uniforme.

5. Fonctions à sens unique

Une fonction à sens unique est une fonction « facile à calculer »et « dure à inverser ». Il y a plusieurs variantes dans la formalisation de cette notion.

Nous dirons qu'une fonction est facile à calculer s'il existe un algorithme de la classe BPP qui effectue le calcul.

Remarque 5.1. — Nous avons défini les diverses classes pour des problèmes de décision. Mais souvent (pas toujours) le calcul d'une fonction se ramène à un problème de décision. Par exemple si on veut calculer une fonction f de $\mathbb{N}$ dans $\mathbb{N}$ et si la taille τ de $f(n)$ est un $\mathcal{O}(t^k)$, où $t = \log(n)$ est la taille de n, alors le calcul de $f(n)$ se réduit au problème de décision suivant : n et m étant donnés, $f(n)$ est-il $\geqslant m$? Pour montrer cette réduction, il suffit de raisonner par dichotomie.

Nous dirons qu'une fonction ν de $\mathbb{N}$ dans $\mathbb{N}$ est à décroissance rapide si elle converge plus vite vers 0 que l'inverse de tout polynôme, c'est-à-dire, si quel que soit $m > 0$:

$$\lim_{k\to\infty} (\nu(k)k^m) = 0.$$

Définition 5.2. — Une fonction f de $\{0,1\}^*$ dans $\{0,1\}^*$ est à sens unique si elle possède les deux propriétés suivantes :

a) Il existe un algorithme de la classe BPP qui permet de calculer f ;

b) pour tout algorithme non-déterministe probabiliste A en temps maximum polynomial, il existe une fonction à décroissance rapide ν et un k_0 tels que pour tout $k \geqslant k_0$ et tout $x \in \{0,1\}^k$ on ait :

$$P\left(f\left(A(1^k, f(x))\right) = f(x)\right) \leqslant \nu(k).$$

Cette définition appelle quelques remarques. D'une part la notation $A(1^k, f(x))$ signifie qu'on applique l'algorithme A aux données 111...1 et $f(x)$ (le mot 111...1 étant de longueur k). Le mot 1^k de longueur k est mis artificiellement en entrée de l'algorithme A pour imposer que l'entrée de A soit au moins de longueur k, même s'il se trouve que $f(x)$ soit beaucoup plus court que x. Pour comprendre l'intérêt d'une telle contrainte, donnons l'exemple suivant. Définissons la valeur de f sur un mot de longueur k par :

$$f(x) = f(x_1, x_2, \cdots, x_k) = (x_1, x_2, \cdots, x_t)$$

où $t = \lfloor \log_2(k) \rfloor + 1$. Il est très facile de trouver un algorithme A donnant une préimage de $f(x) = (x_1, x_2, \cdots, x_t)$, puisqu'il suffit de compléter ce mot en un mot de longueur k par les symboles qu'on veut. Cependant si on suppose que l'entrée de A est $f(x)$, c'est-à-dire un mot de longueur t, la simple écriture d'une solution, donc de k symboles, est exponentielle. Pourtant on voit bien que la solution est « facile »et qu'on ne peut choisir une telle fonction pour fonction à sens unique. Le dysfonctionnement provient du fait que la longueur de $f(x)$ est bien plus courte que celle de x. Pour éviter cet avatar, on impose à l'entrée de A d'être de longueur au moins k.

Index

Révisé 17 Février 2015

Robert Rolland