Un peu de Complexité pour la cryptographie

Robert Rolland

November 25, 2001

1 Introduction - 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**, **machines de Turing**.

Nous ne rentrons pas ici dans toute cette théorie des langages et de la calculabilité, nous nous contentons de regarder 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 définit w.

On définit la **concaténation** de deux mots w_1, w_2 comme étant la ssuite constituée des symboles de w_1 suivi des symboles de w_2. On notera $w_1 w_2$ 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.

A l'instant initial le ruban contient un mot d'entrée écrit avec un sous alphabet de l'alphabet de ruban. 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, 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'intant 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 $\delta(q, w[n])$ est défini, et lorsque on peut bouger la tête de lecture dans le sens voulu. Cette dernière condition exclut le cas où $\delta(q, w[n]) = (q_1, b, L)$ et $n = 0$.
Si la dérivation en une étape est définie, alors 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 et où 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

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

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 acceptateur,

- ▷ 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 acceptateur.

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. Un 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 acceptateur,
  - ▷ 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 s'il existe une séquence de choix pour laquelle la machine s'arrête dans un état acceptateur.

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 acceptateur, ou même qui ne s'arrête pas.

Théorème 2.1 *Si 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 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?)

Pour simplifier un peu nous considèrerons les **problèmes de décision**, 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 $2 \leq m \leq k$?

Etant donné un problème de décision, L le langage formé par l'ensembles des représentation des entrées possibles, L_y le sous ensemble de L correspondant aux entrées pour lesquelles la réponse est oui.

Définition 2.1 *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 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 2.2 *Nous dirons qu'une machine de Turing non déterministe* **résout le problème** *si elle possède un état q_y tel que la machine s'arrête dans l'état q_y pour une séquence de choix si et seulement si le mot initial est dans 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 2.3 *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 à 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 celà introduisons les notions indispensables suivantes.

La **taille de l'entrée** est le nombre de symboles nécessaires à 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 $k = log_2(n)$.

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 Les échelles de croissance

De manière intuitive on peut dire qu'un algorithme effectuant un calcul le fait en **temps polynômial** s'il existe un entier d tel que le nombre d'opérations

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

Un algorithme est **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 polynômial s'il existe une machine de Turing déterministe qui le résout en temps polynômial lorsque la réponse est oui.*

Définition 3.2 *Un problème de décision est dans la classe co-*$\mathbf{P}$ *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 polynômial s'il existe une machine de Turing non déterministe qui le résout en temps polynômial lorsque la réponse est oui. C'est-à-dire il existe une constante A et un entier $d \geq 1$ tels que lorsque la réponse est oui, il existe une séquence de choix qui donne la solution en un nombre de pas $\leq Ak^d$ où k est la taille du mot initial.*

Définition 3.4 *Un problème de décision est dans la classe co-*$\mathbf{NP}$ *si le problème complémentaire est dans la classe* **NP**.

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

Remarque: 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 polynômial (donnant ainsi un **certificat**).

3.4 Réduction d'un problème à un autre

Définition 3.5 *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 3.6 *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 polynômial) s'il existe un algorithme en temps polynômial pour $\mathcal{P}1$ qui utilise au plus un nombre polynômial d'appels à un $\mathcal{P}_2-$oracle.*

Définition 3.7 *Un problème $\mathcal{P}$ est* **NP-complet** *ou encore* **NP-dur** *si tout problème de la classe NP est réductible à $\mathcal{P}$.*

3.5 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. Nous noterons μ la probabilité.

Soit une machine de Turing non déterministe probabilisée (à chaque étape le choix de la transition est probabilisé) qui résout un problème $\mathcal{P}$. et qui pour toute donnée initiale w_0 s'arrête pour μ-presque toute séquence de choix. 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

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

Remarquons qu'il y a une certaine ambiguité dans la mesure où nous n'avons pas encore précisé la probabilité μ utilisée. Cela dépend de la façon dont est organisée au cours du déroulement de l'algorithme le tirage au sort de la séquence de choix. L'exemple du paragraphe suivant éclairera un peu ce point.
Le coût probabiliste de l'algorithme pour une entrée de taille $\leq k$ est alors

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

3.5.1 La classe ZPP

Un problème de décision appartient à la classe ZPP (Zero-sided Probabilistic Polynomial) s'il existe une machine de Turing non déterministe qui le résout et qui s'arrête en **temps probabiliste polynômial** (que la réponse soit oui ou non). Mais attention l'algorithme pourrait très bien ne pas s'arrêter, bien que ceci ait une probabilité nulle.

Voici un exemple typique. L'entrée est un entier n de taille k. On tire au sort un entier $1 \leq w \leq n$, en supposant que tous les entiers de l'intervalle considéré sont équiprobables, et on dispose alors d'un algorithme en temps majoré par $T(n) \leq Ck^d$ 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. 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 $\leq \frac{C}{p}k^d$.

3.5.2 La classe RP

Un problème de décision appartient à la classe RP (Random Polynomial) s'il existe une machine de Turing non déterministe qui le résout et qui s'arrête toujours en **temps maximum polynômial** (que la réponse soit oui ou non) c'est-à-dire

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

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

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 vrai réponse n'est

pas nécessairement non. La probabilité pour que sachant que la donnée implique une réponse oui on obtienne une réponse non est $1-p$. **Exercice:** Chercher la probabilité de se tromper lorsqu'on obtient pour réponse non, c'est-à-dire la probabilité conditionnelle que la donnnée implique la réponse oui, sachant que la machine a répondu non.

3.5.3 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 polynômial** (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 $\geq 2/3$, et qui lorsque la réponse est non s'arrête aussi dans l'état q_y avec une probabilité $\leq 1/3$.

3.5.4 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 algorithme qui montre qu'un problème est dans RP est un algorithme de **Monte-Carlo**

Un algorithme qui montre qu'un problème est dans BPP est un algorithme d' **Atlantic City**

4 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: Nous avons défini les diverses classes pour des problèmes de décision. Mais souvent 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 $\geq m$? (Pour voir celà, il suffit de raisonner par dichotomie)

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

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

Définition 4.1 *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 polynômial, il existe une fonction à décroissance rapide ν et un k_0 tels que pour tout $k \geq k_0$ et tout $x \in \{0,1\}^k$ on ait

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

Remarque: le 1^k signifie que l'entrée de l'algorithme A doit être au moins de longueur k, même s'il se trouve que $f(x)$ est plus court que x.