Complexité algébrique et cryptographie

Alexandre Guillemot

10 décembre 2022

Table des matières

Chapitre 1

Problèmes difficiles en théorie des nombres

1.1 Complexité et cryptographie

1.1.1 Introduction

Idée : mesurer la "difficulté" algorithmique d'un problème.

Définition 1.1.1. (Problème de décision) Un problème de décision est une collection d'instances qui sont des ensembles de données qui admettent exactement une des deux réponses "oui" ou "non".

Ex 1.1.1. 1. Problème SAT (Satisfaisabilité)

Instance : Une fonction à variables booléenne $F : \{0,1\}^n \to \{0,1\}$ construite avec les connecteurs logiques $\vee, \wedge, \neg$. Par exemple,

$$f(x_1, x_2, x_3, x_4) = (\neg(x_1 \wedge (\neg x_3))) \vee (x_1 \wedge x_2 \wedge (\neg x_1))$$

Question : existe-t-il $x_1, \cdots, x_n \in \{0,1\}$ temls que $F(x_1, \cdots, x_n) = 1$?

Algorithme : recherche exhaustive sur $(x_1, \cdots, x_n)$, la complexité est en $\mathcal{O}(2^n)$.

2. FBQ (Formes Booléennes Quantifiées)

Instance : Une formule booléenne avec quantificateur e.g. $\forall x_i \exists x_j \cdots F(x_1, \cdots, x_n)$ (F est une fonction booléenne comme dans SAT)

Question : Cette formule est-elle vraie ?

Algorithme : Recherche exhaustive ($\mathcal{O}(2^n)$).

3. Equations diophantiennes ($10^{\text{ème}}$ problème de Hilbert)

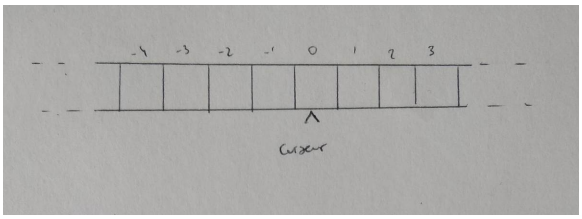
Instance : Une équation polynomiale à plusieurs inconnues et à coefficients entiers
Question : Cette équation admet-elle des solutions entières ?
Algorithme : Matyasevich, 1971 : il n'y a pas d'algo qui répond à cette question.

1.1.2 Calculabilité au sens de Turing

Turing : Cryptanalyse d'Enigma, construction de machines dédiées à la cryptanalyse d'Enigma, Machine de Turing.

Modèle de Turing

On dispose d'un ruban infini



Chaque case contient un symbôle (dans un alphabet fini Σ que l'on peut supposer être $\{0,1\}$, ou le symbôle blanc b). Le ruban v aêtre lu case par case par le curseur, la machine est à chaque instant dans un état $q_i \in Q$, où Q est l'ensemble fini des états possibles.

Définition 1.1.2. (Machine de Turing) Une opération élémentaire est entièrement déterminée par le symbôle lu par le curseur, et par l'état actuel q_i :

1. Le curseur remplace le symbôle par un élément de $\Sigma \cup \{b\}$
2. Le curseur de déplae soit d'une case vers la gauche, soit d'une case vers la droiten, soit reste sur place.
3. La machine passe de l'état q_i à l'état q_j.

Une machine de Turing est donc la donnée d'une fonction

$$M : (\Sigma \cup \{b\}) \times Q \to (\Sigma \cup \{b\}) \times \{-1, 0, 1\} \times Q$$

Définition 1.1.3. (Calcul déterministe) Le calcul déterministe d'une entrée x avec une machine de Turing M est la suite d'opération suivante :

1. La machine est commence par être dans l'état q_0
2. Le curseur est placé sur la case 1
3. x est écrite sur les cases $1, \cdots, n$ du ruban, les autres contenant b.

4. On applique itérativement M, le calcul se terine lorsque la machine atteint l'état final q_F. La sortie y est alors la donnée inscrite sur le ruban lorsque la machine termine.

Terminologie : L'ensemble des suites finies de symbôles de Σ est noté Σ^*. Un mot est un élément de Σ^*. Une fonction $f : \Sigma^* \to \Sigma^*$ est turing calculable s'il existe une machine de Turing M qui sur tout entrée $x \in \Sigma^*$ calcule $y = f(x)$.

1.1.3 Complexités en temps et classes P, NP

Définition 1.1.4. La longeuru d'un calcul sur une entrée $x \in \Sigma^*$ pour une machine de Turing M est le nombre $t_M(x)$ d'opération élémentaires qui composent le calcul. Ainsi on défini la complexité en temps d'une machine de Turing comme

$$\begin{array}{rccc} T_M : & \mathbb{N} & \to & \mathbb{N} \\ & n & \mapsto & \max_{\substack{x \in \Sigma^* \\ |x|=n}} \{t_M(x)\} \end{array}$$

Définition 1.1.5. Un algorithme polynomial $\mathcal{A}$ pour calculer f est une machine de Turing M qui calcule f et telle qu'il existe un polynôme p tel que $\forall n \in \mathbb{N}$, $T_M(n) \leq p(n)$. On appelle classe P l'ensemble des problèmes de décision admettant un algorithme polynomial.

Définition 1.1.6. On dit qu'un pb de décision est calculable par un algo non déterministe polynomial s'il existe une machien de Turing M et un polynome p tel que

1. La réponse est oui pour l'entrée x ssi il existe $y \in \Sigma^*$ (certificat) tel que M calcule 1 lorsqu'on met $x \in \Sigma^*$ dans les cases 1 à n et y dans les cases -1 a $-m$.
2. Pour tout x donnant la réponse 1, M calcule 1 et temps $\leq p(n)$

On appelle classe NP la classe des problèmes de décision admettant un algorithme non déterministe polynomial.

Rq 1.1.1. $P \subseteq NP$.

1.1.4 Problèmes NP-complets

Définition 1.1.7. On dit que le problème de décision p_1 se réduit au problème de décision p_2 s'il existe une fonction $\varphi : \Sigma^* \to \Sigma^*$ calculable en temps polynomial telle que la réponse à p_1 est oui pour l'entrée x si et seulement si la réponse à p_2 est oui pour l'entrée $\varphi(x)$.

Notation. On note $p_1 \ltimes p_2$.

Proposition 1.1.1. *$p_1 \in P$ et $p_1 \ltimes p_2 \Rightarrow p_1 \in P$*

Définition 1.1.8. Un problème Π est NP-complet ssi $\forall p \in NP$, $p \ltimes \Pi$.

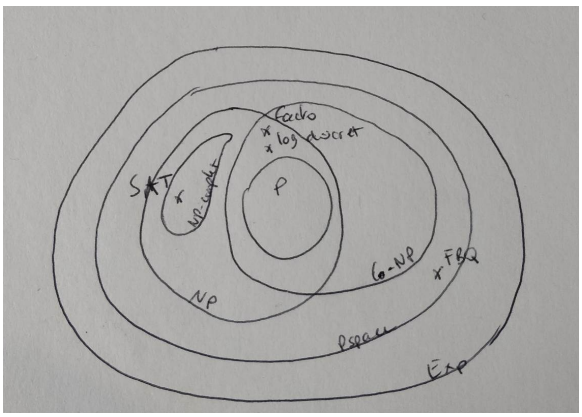
Théorème 1.1.1. *(Cook, 1971) SAT est NP-complet.*

Rq 1.1.2. Si $SAT \ltimes P$, alors P est NP-complet.

Ex 1.1.2.
1. SAT
2. 3-SAT
3. Circuit hamiltonien
4. 3-coloriabilité d'un graphe
5. TSP
6. Pb du sac à dos
7. Système de n équations quadratiques sur un $\mathbb{F}_2$.

On conjecture que $P \neq NP$. Astuce de Levin : Supposons que $P = NP$: alors on peut construire un algorithme polynomial pour résoudre SAT. Considérons les machines de Turing $M_1, M_2, \cdots$ qui prennent en entrée une instance de SAT : Alors on fait tourner les machines simultanément, en faisant tourner de une étape M_1, puis en faisant tourner de une étape M_2, puis M_1, puis M_3, M_2 et M_1, etc.

Résumé de la hiérarchie des complexités algorithmiques :



Rq 1.1.3. Co-$NP \cap NP$-complet $= \emptyset$.

1.2 Factorisation

1.2.1 Complexité

Problème de décision FACTM (problème des facteurs majorés)

- Instance : n entier, $M \leq n$.
- Question : Existe-t-iil un diviseur de n qui est $\leq M$.

Si on a un algo polynomial de factorisation, alors on peut résoudre FACTM en temps polynomial. Inversement, Supposons $\mathcal{A}$ algo polynomial pour FACTM. Comment factoriser n ? Soit p le plus petit facteur premier de n, alors

- On applique $\mathcal{A}(n, \sqrt{n})$. Si l'algorithme répond non, alors on termine et on répond non (car n est alors premier)
- Sinon, on applique $\mathcal{A}(n, \sqrt{n}/2)$. Si l'algo répond non, alors $p \in [\sqrt{n}/2, \sqrt{n}]$, et sinon $p \in [1, \sqrt{n}/2]$.
- On continue la dichotomie jusqu'à ce que la taille de l'intervalle obtenu soit plus petite que 1.

L'algorithme termine dès que $\sqrt{n}/2^k < 1$, où k est le nombre d'appels de $\mathcal{A}$. Ainsi il y a $k = \log_2(\sqrt{n})$ est donc de l'ordre de $\log n$. Une fois p trouvé, on recommence l'algo avec n/p. On va recommencer le nombre de facteurs premiers de n (comptés avec leur multiplicité). Mais

$$n = \prod_i p_i^{\alpha_i} \geq \prod_i 2^{\alpha_i} = 2^{\sum \alpha_i}$$

donc $\sum \alpha_i < \log_2 n$, et c'est aussi le nombre de facteurs premiers de n (comptés avec leur multiplicité). Au total, l'algorithme est polynomial.

1.2.2 Idée de Fermat

- L'idée naïve est d'essayer de diviser par les entiers successifs $\leq n$. C'est en $\mathcal{O}(n)$ donc exponentiel en la taille de l'entier.
- On peut aussi s'arrêter avant $\sqrt{n}$, mais l'algorithme reste exponentiel.
- On peut aussi diviser par les nombres premiers $\sqrt{n}$. D'après le théorème des nombres premiers (Hadamard, de la Vallée-Poussin), le cardinal des entiers premiers plus petits que x est aymptotiquement équivalent à $x/\ln x$. Ainsi l'algo est en $\mathcal{O}(\sqrt{n}/\log n)$, qui reste exponentiel en la taille de n.