Logique, Complexité et Vérification

Richard Lassaigne
Logique mathématique,
CNRS-Université Paris 7

Logique et Complexité (S. Cook, R. Karp)

Classes de complexité en temps et en espace polynomial

$\mathcal{P} = \mathcal{NP}$, **Problèmes** $\mathcal{NP}$**-complets**

Classes de complexité probabilistes

Problèmes de **calcul** :

- produit de deux matrices
- plus court chemin dans un graphe
- clôture transitive d'une relation binaire

Problèmes de **décision** :

- problème d'accessibilité dans un graphe
- existence d'un couplage parfait dans un graphe biparti
- satisfaction d'une formule propositionnelle
- problème du voyageur de commerce
- colorabilité d'un graphe à l'aide de 3 couleurs

Machine de Turing (déterministe) : $M = (Q, \Sigma, q_0, \delta, F)$

- alphabet Σ
- ensemble d'états Q, q_0 état initial
- $F = \{q_Y, q_N\} \subseteq Q$ états finaux (acceptant, rejetant)
- m rubans, m têtes de lecture-écriture
- fonction de transition $\delta : Q \times \Sigma^m \longrightarrow Q \times \Sigma^m \times \{-1, 0, +1\}^m$

$L \subseteq \Sigma^*$ est décidable s'il existe une machine M d'alphabet Σ t. q. :

- pour tout $x \in L$, M accepte l'entrée x,
- pour tout $x \notin L$, M rejette l'entrée x,

Langage L_π associé au problème de décision π :
Ensemble des codes de données positives pour le problème π
Le problème π est décidable si son langage associé est décidable.

Codage d'un problème de décision

On suppose que : $\Sigma \supseteq \{0, 1, (,), [,], \#\}$

Un codage convenable d'un problème de décision π :
$\{$données du problème $\pi\} \longrightarrow \{$chaînes structurées$\}$

L'ensemble des chaînes structurées est défini par induction :

- la représentation binaire d'un entier est une chaîne structurée
- si x est structurée, alors $[x]$ est structurée
- si $x_1, x_2, ..., x_k$ sont structurées , alors $(x_1 \# x_2 \# ... \# x_k)$ est structurée

Codage d'un problème de décision

Exemple : Un graphe G à n sommets peut être représenté

- soit par le couple (n, M) où M est la matrice d'adjacence
- soit par une suite de n listes d'adjacence

La taille de la donnée d'un problème est la longueur de la chaîne structurée codant cette donnée suivant un codage convenable

Exercice :
Les tailles d'une même donnée suivant 2 codes convenables sont liées polynomialement

Complexité en temps

- Temps de calcul $t_M(x)$ de M sur une entrée x :

nombre de transitions effectuées avant l'arrêt

- Fonction de complexité en temps T_M de M :

$T_M(n) = sup\{t_M(x) \ / \ x \in \Sigma^*$ de longueur $|x| = n\}$

- Une machine M calcule, ou décide, en temps polynomial

s'il existe un polynôme p (à coeff. entiers) t.q. pour tout n,

$T_M(n) \leq p(n)$

Temps de calcul polynomialement borné : **indépendant**

- du codage (convenable) et
- du modèle de calcul (Turing, RAM ou algorithme)

$\mathcal{P}$: classe des langages décidables en temps polynomial.

Un problème π est décidable en temps polynomial si

L_π est dans $\mathcal{P}$

Complexité en espace

Machine de Turing avec un ruban d'entrée et un (ou plusieurs) rubans de travail

- Espace $e_M(x)$ de M sur une entrée x :
  nombre de cases utilisées sur les rubans de travail avant l'arrêt
- Fonction de complexité en espace E_M de M :
  $E_M(n) = sup\{e_M(x) \ / \ x \in \Sigma^*$ de longueur $|x| = n\}$
- Une machine M calcule, ou décide, en espace polynomial
  s'il existe un polynôme p (à coeff. entiers) t. q. pour tout n,
  $E_M(n) \leq p(n)$

$\mathcal{P}ESPACE$: classe des langages décidables en espace polynomial
Un problème est décidable en espace polynomial si
L_π est dans $\mathcal{P}ESPACE$

Le problème du voyageur de commerce

Donnée :

- Graphe $G = (E, R)$ complet à n sommets $(i = 1, 2, ..., n)$
- Coûts $c(i, j)$ entiers (pour chaque couple (i, j))
- Entier C

Propriété :

Il existe un parcours permettant de visiter tous les sommets du graphe (une fois et une seule), dont le coût total $\leq C$,
i.e. il existe une permutation σ sur $\{1, 2, ..., n\}$ t.q.
$\sum_{i=1}^{n-1} c(\sigma(i), \sigma(i+1)) + c(\sigma(n), \sigma(1)) \leq C$

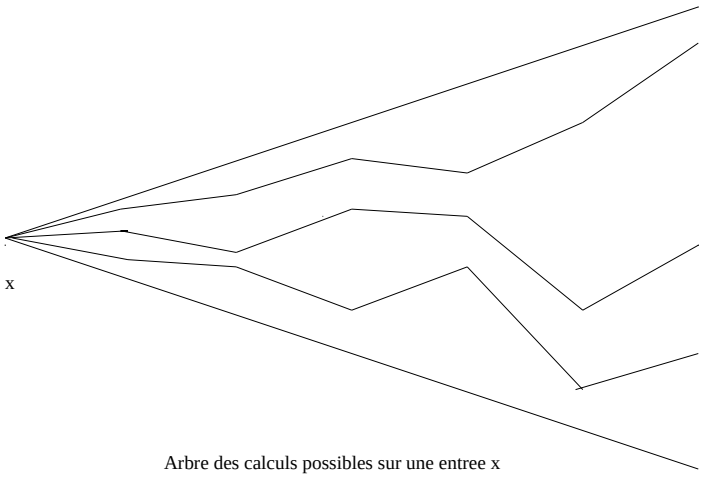
R. Karp (1972) : liste de 21 problèmes (graphes, logique,...) pour lesquels on ne connaît pas d'algorithme en temps polynomial

Machine de Turing non déterministe(1e version)

$M' = (Q, \Sigma, q_0, \Delta, F)$ où Δ est la relation de transition :
$\Delta : (Q \times \Sigma^m) \times (Q \times \Sigma^m \times \{-1, 0, +1\}^m)$

A chaque étape de calcul, M a le choix d'effectuer une transition parmi un nombre fini de transitions possibles

En fait, il existe un entier $k \leq |Q| \cdot |\Sigma|^m \cdot 3^m$ qui borne le nombre de transitions possibles à chaque étape

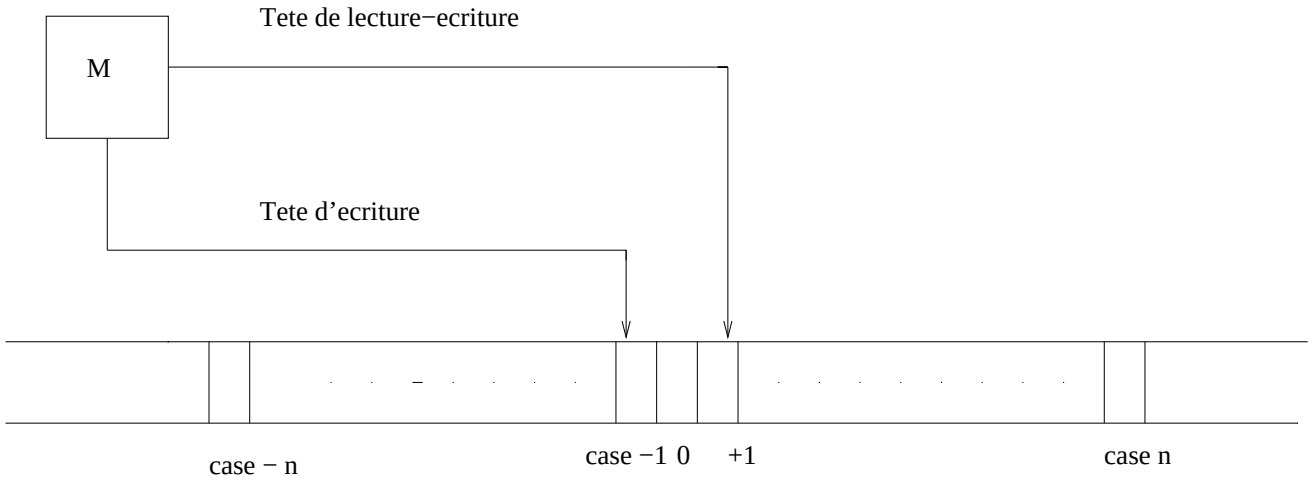


Arbre des calculs possibles sur une entree x

Machine de Turing non déterministe(2e version)

Machine M dotée :

- d'une tête d'écriture (sur la partie gauche du ruban)
- d'une tête de lecture-écriture (sur tout le ruban)



- 1e phase (supposition) : engendrer
  de manière non déterministe une suite y
- 1e phase (vérification) : vérifier de manière déterministe
  si la suite y représente une solution pour l'entrée x

Machine de Turing non déterministe

Proposition

Une machine non déterministe du 1er type peut être simulée en temps polynomial par une machine du 2e type, et inversement

Le problème du voyageur de commerce est vérifiable en temps polynomial sur une machine non déterministe

Classe $\mathcal{NP}$

- Une machine non déterministe M (d'alphabet Σ) vérifie un langage L ssi pour tout mot $x \in \Sigma^*$, $x \in L$ ssi il existe un chemin acceptant x

- Un langage L est vérifiable en temps polynômial

s'il existe une machine non déterministe M vérifiant L et un polynôme p (à coefficients entiers) t.q. tout chemin de M sur une entrée x avec $|x| = n$ est de longueur $\leq p(n)$

- $\mathcal{NP}$: classe des langages L vérifiables en temps polynomial sur une machine non déterministe

Relation entre les classes $\mathcal{P}$ et $\mathcal{NP}$

Proposition :

Pour chaque machine non déterministe N
(à k fonctions de transition),
il existe une machine déterministe qui, pour tout m,
produit les configurations de M obtenues en m pas de calcul
Le coût d'une telle simulation est proportionnel à $m.k^m$

Corollaire

Si un problème est dans la classe $\mathcal{NP}$,
alors il existe un polynôme p t. q.
ce problème puisse être résolu par une machine déterministe
dont la fonction de complexité est un $O(2^{p(n)})$

Hiérarchie de Complexité

Soit $f : \mathbb{N} \longrightarrow \mathbb{N}$ une fonction croissante

Il est alors possible de définir les classes de complexité :

- $DTEMPS(f(n))$: classe des problèmes décidables en temps borné par $f(n)$
- $DESPACE(f(n))$: classe des problèmes décidables en espace borné par $f(n)$
- $NTEMPS(f(n))$: classe des problèmes vérifiables en temps borné par $f(n)$
- $NESPACE(f(n))$: classe des problèmes vérifiables en espace borné par $f(n)$

Exemples : Fonctions de complexité intéressantes

$f_k(n) = n^k (k \in \mathbb{N})$ $\quad g(n) = log_2(n)$

$\mathcal{L} = DESPACE(log_2(n))$ $\quad \mathcal{NL} = NESPACE(log_2(n))$

Hiérarchie de Complexité

Théorème :

Soit $f : \mathbb{N} \longrightarrow \mathbb{N}$ une fonction croissante. Alors

- $DTEMPS(f(n)) \subseteq NTEMPS(f(n))$ et
  $DESPACE(f(n)) \subseteq NESPACE(f(n))$
- $DTEMPS(f(n)) \subseteq DESPACE(f(n))$ et
  $NTEMPS(f(n)) \subseteq NESPACE(f(n))$
- $NTEMPS(f(n)) \subseteq DESPACE(f(n))$
- Si $L \in NESPACE(f(n))$, alors il existe c t.q.
  $L \in DTEMPS(c^{log\ n+f(n)})$

Corollaire : $\mathcal{L} \subseteq \mathcal{NL} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PESPACE}$

Réductions

Languages $L_1 \subseteq {\Sigma_1}^*$ et $L_2 \subseteq {\Sigma_2}^*$

Une fonction $f : {\Sigma_1}^* \longrightarrow {\Sigma_2}^*$ est

une réduction polynomiale de L_1 à L_2 si

- pour tout $x \in {\Sigma_1}^*$, $x \in L_1$ ssi $f(x) \in L_2$
- f est calculable en temps polynomial.

Soient P_1, P_2 des problèmes de décision

et L_1, L_2 les langages associés

P_1 est réductible en temps polynomial à P_2 ($P_1 \prec P_2$)

s'il existe une réduction polynomiale de L_1 à L_2.

Réductions

Exemple : le problème du **circuit hamiltonien**

Donnée : Graphe $G = (E, R)$ à n sommets $(i = 1, 2, ..., n)$

Propriété : Il existe un circuit passant par

tous les sommets du graphe (une fois et une seule)

Ce problème est réductible à celui du **voyageur de commerce**

Proposition 1 :

Si $P_1 \prec P_2$ et si le problème P_2 est dans la classe $\mathcal{P}$ (resp.$\mathcal{NP}$),

alors le problème P_1 est aussi dans la classe $\mathcal{P}$ (resp. $\mathcal{NP}$)

Proposition 2 :

La relation de réductibilité entre problèmes est transitive

Problèmes $\mathcal{NP}$-complets

Un problème P de la classe $\mathcal{NP}$ est dit $\mathcal{NP}$-complet
si tout problème de la classe $\mathcal{NP}$ lui est réductible

Proposition 3 :
Si P_1 et P_2 sont dans la classe $\mathcal{NP}$, si P_1 est $\mathcal{NP}$-complet
et si $P_1 \prec P_2$, alors P_2 est $\mathcal{NP}$-complet

Problèmes $\mathcal{NP}$-complets

Un exemple générique : le problème **SAT**

Donnée : Une formule propositionnelle F sous forme CNF

$F = \bigwedge_{j=1}^{m} c_j \quad c_j = \bigvee_{i=1}^{k} \varepsilon_i p_i \quad (\varepsilon_i p_i = p_i \text{ ou } \neg p_i)$

- n variables propositionnelles $\{p_1, p_2, \ldots, p_n\}$
- m clauses c_j $(1 \leq j \leq m)$

Propriété : Il existe une valuation satisfaisant F

Théorème : (S. Cook,1971)

Le problème SAT est $\mathcal{NP}$-complet

Idée sur la preuve :

- Le problème SAT est bien dans la classe $\mathcal{NP}$

  On définit une machine non déterministe N :

  Phase de supposition : elle devine une valuation

  Phase de vérification : elle vérifie si cette valuation satisfait la formule

Cette vérification peut s'effectuer en en temps linéaire par rapport à nm (n nb de variables et m le nb de clauses)

- Il reste à montrer que tout problème dans $\mathcal{NP}$

peut se réduire en temps polynomial au problème SAT

Fonctionnement d'une machine sur une entrée x en temps polynomial :

le nombre de pas de calcul et de symboles écrits $\leq p(n)$ où p est un polynôme et $|x| = n$

Réduction au problème SAT :

Phase de vérification : décrite par l'état de la machine,
la position de la tête de lecture-écriture et le contenu des cases
Caractéristiques de la machine :
états q_j $(0 \leq j \leq r)$, symboles s_l $(0 \leq l \leq t)$

On introduit des variables propositionnelles :

- $Q_{i,j}$: l'état de la machine à l'instant i est q_j
  $(0 \leq i \leq p(n)$ et $0 \leq j \leq r)$
- $P_{i,j}$: la tête se trouve à l'instant i devant la case de n° j
  $(0 \leq i \leq p(n)$ et $-p(n) \leq j \leq p(n)+1)$
- $S_{i,j,l}$: à l'instant i, la case de numéro j contient le symbole s_l
  $(0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)+1$ et $0 \leq l \leq t)$

Réduction au problème SAT :

Toute exécution acceptante de la machine M sur une entrée x :
définit une valuation sur l'ens. de ces var. prop.
Inversement, on détermine un ensemble de clauses t.q.
toute valuation satisfaisant ces clauses
corresponde à une exécution acceptante

Cette correspondance est calculable en temps polynomial

Réduction au problème SAT :

L'ensemble de clauses exprime :

- à chaque instant, la machine est dans un état et un seul
- à chaque instant, la tête est dans une position et une seule
- à chaque instant, chaque case contient un symbole et un seul
- la configuration initiale correspond à l'entrée x
- à l'instant $p(n)$, la machine est dans l'état acceptant
- les transitions de la machine à l'aide de la fonction de transition

3-SAT

Donnée : Une formule propositionnelle F sous forme CNF

- n variables propositionnelles $\{p_1, p_2, \ldots, p_n\}$
- m clauses c_j $(1 \leq j \leq m)$ à 3 littéraux

Propriété : Il existe une valuation satisfaisant F

Recouvrement de sommets

Donnée : Un graphe $G = (E, R)$ et un entier $k \leq |E|$

Propriété : Il existe un recouvrement des sommets de G de taille $\leq k$, i.e. un ensemble $E' \subseteq E$ t. q. $|E'| \leq k$ et $\forall (a, b) \in R$ $a \in E'$ ou $b \in E'$

Colorabilité d'un graphe

Donnée : Un graphe $G = (E, R)$ et un entier $k \geq 3$

Propriété : Il existe un k-coloriage du graphe G, i.e. une fonction $f : E \longrightarrow \{1, 2, ..., k\}$ t.q. $\forall (a, b) \in R$ $f(a) \neq f(b)$

N machine non déterministe travaillant
en temps polynomialement borné
On peut toujours supposer :
• N précise, i.e. tous les calculs sur une entrée x
s'arrêtent après le même nb d'étapes (polynomial)
• le degré de non déterministe à chaque étape,
i.e. le nb de choix possibles, est restreint à 2

Machine probabiliste : non déterministe avec tirage aléatoire

Classe $\mathcal{RP}$ (Randomised Polynomial Time) :
Langages L pour lesquels il existe une machine probabiliste A fonctionnant en temps polynomial t.q. pour toute entrée x

- si $x \in L$, alors $Prob_\Omega(A \text{ accepte } x) \geq \frac{1}{2}$
- si $x \notin L$, alors $Prob_\Omega(A \text{ accepte } x) = 0$

Ω est l' espace probabiliste des tirages de la machine

Classe $co\mathcal{RP}$:
Langages L dont le complémentaire est dans $\mathcal{RP}$

Exemple : Le problème de primalité est dans $co\mathcal{RP}$

Classe $\mathcal{RP}$

- Algorithmes de Monte-Carlo (avec erreur d'1 seul côté) :

$$Prob_{\Omega}(x \in L \text{ et } A \text{ rejette } x) < \frac{1}{2}$$

- Réduction (exponentielle) de la probabilité d'erreur :

En itérant k fois l'algorithme A, on obtient un algorithme A' t.q.

$$Prob_{\Omega}(x \in L \text{ et } A' \text{ rejette } x) < \frac{1}{2^k}$$

Test de primalité
(Solovay-Strassen)

$\mathbb{Z}_n = \mathbb{Z}/n\mathcal{Z}$ et $\mathbb{Z}_n^* = \{a \in \mathbb{Z} \ / \ 1 \leq n$ et $pgcd(a,n) = 1\}$

Symbole de Legendre de $a \in \mathbb{Z}_p^*$ (p premier) : $[\frac{a}{p}] = a^{\frac{p-1}{2}}$

Soit n un entier impair de décomposition $n = \prod_{i=1}^{k} p_i^{\alpha_i}$

Symbole de Jacobi de $a \in \mathbb{Z}_n^*$: $[\frac{a}{n}] = \prod_{i=1}^{k} [\frac{a}{p_i}]^{\alpha_i}$

Proposition :

Le symbole de Jacobi est calculable en temps polynomial

Test de primalité

Idée de l'algorithme :

- Si n est premier, alors pour tout $a \in \mathbb{Z}_n^*$, $[\frac{a}{n}] \equiv a^{\frac{n-1}{2}} \pmod{n}$
- Si n est composé, alors il existe une proportion importante ($\geq 1/2$) de $a \in \mathbb{Z}_n^*$ t.q. $[\frac{a}{n}] \not\equiv a^{\frac{n-1}{2}} \pmod{n}$

Algorithme : l'entrée est un nombre entier impair

- *choisir aléatoirement (de manière uniforme) un* $a \in \mathbb{Z}_n - \{0\}$
- *calculer le* $pgcd(a, n)$
- *si* $pgcd(a, n) \neq 1$, *alors retourner composé*
- *calculer* $[\frac{a}{n}]$ *et* $a^{\frac{n-1}{2}}$
- *si* $[\frac{a}{n}] \not\equiv a^{\frac{n-1}{2}} \pmod{n}$, *alors retourner composé*
  sinon retourner premier

Références

- M.R. Garey et D.S. Johnson : *Computers and Intractability*
  Freeman and Co, 1979

- C.H. Papadimitriou : *Computational Complexity*
  Addison Wesley, 1994

- R. Lassaigne et M. de Rougemont : *Logic and Complexity*
  Springer-Verlag, 2004