07/04
2015

Classes de complexité : exemples

Motivation : la mise en œuvre d'un algo nécessite l'utilisation de ressources physiques (temps, espace) ; on appelera complexité l'approximation asymptotique de ces grandeurs en fonction de la taille de l'entrée. On aimerait alors classifier les problèmes en fonction de la complexité minimale des algorithmes qui permettent de les résoudre

I Définitions

1 Complexité des algorithmes

Cadre On algorithme sera ici représenté par une machine de Turing (MT) déterministe ou non, effectuant ses calculs sur un nombre quelconque de rubans, et sur un alphabet Σ possédant au moins deux caractères.

On ne considèrera que des MT qui s'arrêtent sur toutes leurs entrées.

Def 1 Soit M une MT, w un mot sur Σ. On considère l'exécution $\mathcal{E}(M,w)$ de M sur w. On appelle :
- $t_M(w)$ le nombre d'étapes de calcul de $\mathcal{E}(M,w)$
- $s_M(w)$ le nombre de cases mémoire uniquement utilisées par $\mathcal{E}(M,w)$.

On peut alors définir, pour $n \in \mathbb{N}$, les complexités de M
- $t_M(n) = \sup_{|w|=n} t_M(w)$: en temps
- $s_M(n) = \sup_{|w|=n} s_M(n)$: en espace

2 Classes de complexité

Proposition 2 (Théorèmes d'accélération, de compression)
Soit $c \in \mathbb{N}^*$, M une machine de Turing. Il existe
- une machine M' telle que $t_{M'}(n) \leq t_M(n)/c + n \quad \forall n \in \mathbb{N}$
- une machine M'' telle que $s_{M''}(n) \leq s_M(n)/c + 2 \quad \forall n \in \mathbb{N}$

où M' et M'' sont équivalentes à M, et déterministes ssi M l'est

Proposition 3 Toute machine de Turing M est équivalente à une MT déterministe M' telle que $t_{M'}(n) = 2^{O(t_M(n))}$

Définition 4 (Classes de complexité) : Soit $f : \mathbb{N} \to \mathbb{N}$, et $L \subset \Sigma^*$ un langage. On dira que
- $L \in TIME(f(n))$ (resp. $NTIME(f(n))$) si il existe une MT déterministe (resp. non-déterministe) qui décide L en temps $O(f(n))$
- $L \in SPACE(f(n))$ (resp $NSPACE(f(n))$) si il existe une MT déterministe (resp non-déterministe) qui décide L en espace $O(f(n))$

Définition 5 On définit enfin des classes plus générales :
- $P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$: langages décidés en temps polynomial par une MT déterministe
- $NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$: langages décidés en temps polynomial par une MT non-déterministe
- $PSPACE = \bigcup_{k \in \mathbb{N}} SPACE(n^k)$, $NPSPACE = \bigcup_{k \in \mathbb{N}} NSPACE(n^k)$

[PAPA] [CAR] [CAR]

II. Généralités sur les classes

1 Exemples de problèmes

On assimilera par la suite un langage à l'ensemble des instances positives d'un problème de décision

Proposition 6 : SAT { entrée : une formule propositionnelle ; sortie : oui ssi elle admet un modèle }

est dans NP (on dit aussi NP-difficile)

Proposition 7 : COLOR { entrée : un graphe G, un entier k ; sortie : oui ssi il existe une k-coloration de G }

est dans NP

Proposition 8 : HAM-PATH : { entrée : un graphe G ; sortie : oui ssi il existe un cycle qui passe une unique fois par chaque noeud }

est dans NP

Proposition 9 : PVC : { entrée : un graphe complet, aux arêtes pondérées, une borne M ; sortie : oui ssi il existe un chemin hamiltonien de poids inférieur à M }

est dans NP

Proposition 10 : Soit G une grammaire algébrique sous forme normale quadratique

IN(G) { entrée : un mot w ; sortie : oui ssi w est engendré par G }

est dans P

Proposition 11 :

QBF-SAT { entrée : une formule propositionnelle quantifiée ; sortie : oui ssi elle admet un modèle }

est dans PSPACE

Remarque 12 : La formulation exacte du problème, ainsi que la façon dont l'entrée est encodée, sont très importants !

Par exemple, si un entier est codé par l'ensemble de ses facteurs premiers, on peut décider en temps linéaire s'il est premier !

2 Reconnaître la classe d'appartenance

Définition 13 (Réduction) Soient A et B deux problèmes sur un alphabet Σ. Une réduction de A à B est une fonction $f : \Sigma^* \to \Sigma^*$ calculable telle que

$\forall w \in \Sigma^*$, $w \in A$ ssi $f(w) \in B$

Une réduction est dite polynomiale si elle est calculable en temps polynomial.

Si il existe une réduction polynomiale de A vers B, on note $A \leq_p B$.

Définition 14 Soit $\mathcal{C}$ une classe de langage, $\mathcal{L}$ un langage

- $\mathcal{L}$ est $\mathcal{C}$-difficile (pour les réductions polynomiales) si $\forall \mathcal{L}' \in \mathcal{C}$, $\mathcal{L}' \leq_p \mathcal{L}$
- $\mathcal{L}$ est $\mathcal{C}$-complet si $\mathcal{L} \in \mathcal{C}$ et $\mathcal{L}$ est $\mathcal{C}$-difficile

Proposition 15 : . Si $A \leq_p B$ et $B \in P$, alors $A \in P$

. Si A est NP-difficile et $A \leq_p B$, alors B est NP-difficile

Théorème 16 (de Cook) SAT est NP-complet [DEV]

Proposition 17 : On montre, par des réductions à SAT, que COLOR, HAM-PATH et PVC sont NP-complets

Proposition 18 : Si $B \in PSPACE$, $A \leq_p B$, et A est PSPACE-complet, alors B est PSPACE-complet.

3 Relations entre les classes

Définition 19 Soit $\mathcal{C}$ une classe de langages. On appelle co-$\mathcal{C}$ la classe des complémentaires des langages de $\mathcal{C}$

Proposition 20 Si $\mathcal{C}$ est une classe déterministe, on a $\mathcal{C} = co\text{-}\mathcal{C}$

Théorème 21
- $P \subseteq NP \cap coNP$ (cf. Annexe)
- $NP \cup co\text{-}NP \subsetneq PSPACE$
- $PSPACE = NPSPACE$ (Théorème de Savitch)
- On ne sait pas si $P = NP$, ni si $NP = co\text{-}NP$!

Proposition 22 Si $NP = co\text{-}NP$, alors $P \neq NP$ [HU]

Proposition 23 $NP = co\,NP$ ssi il existe un problème NP complet qui est aussi dans co-NP [PAPA]

Proposition 24 Si L est NP-complet, alors $\bar{L} = \Sigma^* - L$ est co-NP-complet [PAPA]

Proposition 25 : PRIME [entrée : un entier n écrit en binaire; sortie : oui ssi n est premier] est dans $NP \cap coNP$

III Résoudre polynomialement les problèmes NP

1 Entrées particulières

En se restreignant à certaines entrées, on peut obtenir des algorithmes polynomiaux pour des problèmes NP-difficiles dans le cas général.

Proposition 26 : 2-SAT [entrée : une formule propositionnelle sous forme normale conjonctive, avec au plus deux littéraux par clause; sortie : oui ssi la formule est satisfiable] est dans P

Proposition 27 : Le problème COLOR restreint aux graphes d'intervalles est dans P

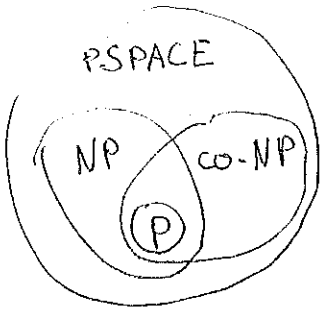
2 Algorithmes d'approximation

Définition 28 : Soit A un problème d'optimisation, x une instance, appelons $F(x)$ l'ensemble des solutions possibles. À toute solution $s \in F(x)$ est associé un score $c(s)$, et soit $OPT(x)$ le score d'une solution optimale.
Soit M un algorithme. Si pour toute instance x, $M(x) \in F(x)$, M est appelé algorithme d'approximation. Si de plus il existe $\varepsilon > 0$ tq $\forall x$, $\frac{|c(M(x)) - OPT(x)|}{\max(c(M(x)), OPT(x))} \leq \varepsilon$, M est dit d'ε-approximation [PAPA]

Propriété 28 : Si il existe $\varepsilon > 0$, et un algo d'ε-approximation à PVC, alors $P = NP$

Propriété 29 : Pour tout $\varepsilon > 0$, il existe un algorithme d'ε-approx de SAC-A-DOS

[DEV]



On peut aussi mettre
- d'autres classes (L, NL, EXPTIME) NEXPTIME
- plein d'autres problèmes
- caractérisation de NP par les vérificateurs

Sources :
[CAR] : Carton
[HU] : Hopcroft - Ullman
[PAPA] : Papa

GAR

Approximation de problèmes NP

2014-2015

1 Problème du sac à dos

Une *instance* du problème du sac à dos (SAC) est définie par
— Un ensemble de couples d'entiers $(p_i, v_i), 1 \leq i \leq n$, représentant le poids et la valeurs de divers objets.
— Une capacité maximale P_{max}
On cherche à trouver un sous-ensemble S de $\{1, \ldots, n\}$ maximisant $\sum_{i \in S} v_i$, et respectant la contrainte $\sum_{i \in S} p_i \leq P_{max}$

Théorème 1. SAC est NP-complet.

Théorème 2. Pour tout $\varepsilon > 0$, on peut trouver un algorithme polynomial qui retourne une solution ε-proche de la solution optimale d'une instance quelconque de SAC.

Un algorithme pseudo-polynomial ... L'algorithme de programmation dynamique suivant résoud le problème SAC de manière optimale.

Soit $V = \max\{v_1, \ldots, v_n\}$, on construit un tableau $(P(i,v))_{0 \leq i \leq n, 1 \leq v \leq nV}$ qui vérifie : $P(i,v)$ est le poids minimum atteignable en choisissant $S \subset \{1, \ldots, i\}$ tel que $\sum_{i \in S} v_i = v$.

On procède de la manière suivante :
— $\forall 1 \leq v \leq nV, \quad P(0,v) = +\infty$
— $\forall 1 \leq i \leq n, 1 \leq v \leq nV \quad P(i+1,v) = \min\{P(i,v), P(i, v - v_{i+1}) + p_{i+1}$
On choisit alors le plus grand v tel que $P(n,v) \leq P_{max}$, et on obtient bien une solution optimale à notre probème. Cet algorithme peut sembler polynomial, mais il n'en est rien ! En effet, sa complexité est en $\mathcal{O}\left(n^2 V\right)$, or V peut être rendu exponentiel en la taille de l'entrée.

...dont on va approximer l'entrée ! En effet, en oublient les b premiers bits des v_i, on fait artificiellement diminuer V, et on va ainsi obtenir un algo polynomial.

Soit b un entier, $x = (p_1, \ldots, p_n, P_{max}, v_1, ..., v_n)$ une instance de SAC, on considère l'instance $x' = (p_1, \ldots, p_n, P_{max}, v'_1, ..., v'_n)$, où $v'_i = 2^b \lfloor \frac{v_i}{2^b} \rfloor$. On peut résoudre x' en temps $\mathcal{O}\left(\frac{n^2 V}{2^b}\right)$, en faisant tourner l'algorithme présenté en oubliant le facteur 2^b des v'_i.

Soit maintenant S (resp. S') une solution optimale de x (resp. x'), explicitons l'erreur d'approximation. On a : $\forall i, v_i \geq v'_i \geq v_i - 2^b$, d'où

$$\sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S} (v_i - 2^b) \geq \sum_{i \in S} v_i - n2^b$$

La seconde inégalité étant vraie puisque S' est une solution optimale de x'. On a ainsi majoré l'erreur d'approximation par $n2^b$. On sait de plus que V est une borne minimale de la meilleure solution (puisque qu'on peut supposer que $p_i \leq P_{max}$ pour tout i), ce qui nous un facteur d'approximation $\varepsilon \leq \frac{2n^b}{V}$.

Prenons finalement $\epsilon > 0$, et posons $b = \lceil \log \frac{\epsilon V}{n} \rceil$, on obient un algorithme d'approximation de complexité $\mathcal{O}\left(\frac{n^2 V}{2^b}\right) = \mathcal{O}\left(\frac{n^3}{\varepsilon}\right)$, ce qui est bien polynomial en la taille de l'entrée !

2 Problème du voyageur de commerce

Une *instance* du problème du voyageur de commerce (PVC) est donnée par un nombre de villes n, et un tableau $\mathrm{d}(i,j)_{1 \leq i,j \leq n}$ donnant la distance entre deux villes.

Une *solution* à une instance de PVC est une permutation π des villes telle que $\sum_{i=1}^{n} d(\pi(i), pi(i+1)$ soit minimale.

Théorème. Si il existe un d'algorithme polynomial d'ε-approximation de PVC pour $\varepsilon > 0$, alors P = NP.

Preuve. Soit $G = (V, E)$ un graphe connexe, soit $\varepsilon > 0$. Supposons que l'on dispose d'un algorithme A d'ε-approximation de PVC, on va montrer qu'on peut alors résoudre HAM-PATH sur G en temps polynomial. HAM-PATH étant NP-complet, on aura ainsi montré P = NP.

Posons, pour i et j dans V

— $\mathrm{d}(i,j) = 1$ si $(i,j) \in E$

— $\mathrm{d}(i,j) = \frac{|V|}{1-\varepsilon}$ sinon.

Appliquons A à (V, d). Si la permutation trouvée à un poids exactement $|V|$, alors G possède un cycle hamiltonien. Sinon, la longueur totale de la permutation trouvée est supérieure à $\frac{|V|}{1-\varepsilon}$. Comme notre algorithme est une ε-approximation, on peut être sûr qu'il n'existe pas de permutation de poids inférieur ou égale à $1 - \varepsilon \frac{|V|}{1-\varepsilon} = V$, et on sait ainsi que G ne possède pas de chemin Hamiltonien.

Voilà donc une méthode simple et effica pour résoudre en temps polynomial tout problème NP.

Source

Papadimitriou

Théorème de Cook

Langages Formels. Calculabilité et complexité.
Olivier CARTON

2011-2012

Théorème 0.1 : de Cook, 1971
Le problème SAT *est NP-complet.*

Démonstration. Soit A un problème de NP, et soit $\mathscr{M}$ une machine de Turing non-déterministe qui décide A en temps polynomial.

Pour chaque entrée w, on va construire une formule φ_w qui sera satisfiable *si et seulement si* $\mathscr{M}$ accepte w.

On note $n = |w|$. On peut supposer que chaque calcul acceptant sur w est de longueur exactement n^k (quitte à rajouter des transitions inutiles).

La machine utilise donc au plus n^k cellules de sa bande de travail, et donc les configurations sont de longueur au plus n^k : de même, on les prendra de longueur exactement n^k, quitte à rajouter des symboles blancs.

On les note dans un tableau :

Conf.	0	1	2	3	$\cdots$	n^k
$C_0 =$	q_0	w_1	w_2	w_3	$\cdots$	$\sharp$
$C_1 =$	w'_1	q_1	w_2	w_3	$\cdots$	$\sharp$
$C_2 =$	w'_1	w_2	q_2	w_3	$\cdots$	$\sharp$
$C_2 =$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\sharp$
$\vdots$						$\vdots$
$C_{n^k} =$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

On va donc coder une formule φ_w qui code l'existence d'un tel tableau.

On définit les variables $x_{i,j,a}$ pour $i, j \in [\![0, n^k]\!]$ et a symbole de $A = \Gamma \cup Q$ qui codent le fait que la variable a se trouve dans la case i, j. Il y a $|A|n^{2k+2}$ telles variables.

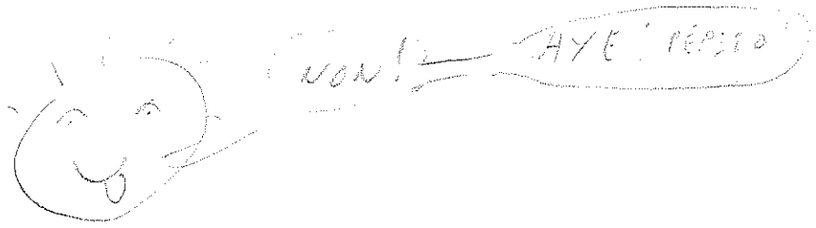
On décompose notre formule φ_w en quatre formules φ_0, φ_1, φ_2 et φ_3, qui vont chacune coder une propriété du tableau.

φ_0 : Cette formule code le fait que chaque case du tableau contient un et un seul symbole de A :

$$\varphi_0 = \bigwedge_{0 \leqslant i,j \leqslant n^k} \left[\left(\bigvee_{a \in A} x_{i,j,a} \right) \wedge \left(\bigwedge_{a \neq a' \in A} (\overline{x}_{i,j,a} \vee \overline{x}_{i,j,a'} \right) \right].$$

φ_1 : Cette formule code le fait que la première ligne du tableau est bien $q_0 w$:

$$\varphi_1 = \left(\bigwedge_{0 \leqslant i \leqslant n} x_{0,i,w_i} \right) \wedge \left(\bigwedge_{n+1 \leqslant i \leqslant n^k} x_{0,i,\sharp} \right).$$



φ_2 : Cette formule assure que chaque ligne est obtenue en appliquant une transition valide de $\mathscr{M}$.
Il suffit de remarquer que la valeur d'une case (i,j) ne dépend que des trois cases au-dessus $(i-1,j-1)$, $(i-1,j)$ et $(i-1,j+1)$.
Si dans ces trois cases se trouvent des symboles de bande, alors le contenu de la case (i,j) est le même qu'en $(i-1,j)$.
Si l'état de la configuration se trouve en $(i-1,j)$, alors l'état de C_i se trouve en $(i,j-1)$ ou $(i,j+1)$.
Donc, il suffit bien de regarder les "fenêtres" de taille 2×3 du tableau. L'ensemble des fenêtres possibles ne dépend que de A et des transitions de $\mathscr{M}$, et donc ne dépend pas de la taille de l'entrée n.
Le fait que chaque fenêtre du tableau corresponde bien à une transition s'écrit donc comme une conjonction pour $0 \leqslant i,j \leqslant n^k$ de disjonctions des fenêtres possibles, ce qui est polynomial en n.

φ_3 : Cette formule code le fait que $\mathscr{M}$ accepte w, *i.e* qu'au moins une des cases de la dernière ligne contient un état final :

$$\varphi_3 = \bigvee_{q\in F}\left(\bigvee_{0\leqslant j\leqslant n^k} x_{n^k,j,q}\right).$$

□

Rem : parler de décidabilité
↳ on se restreint aux MTD.

Ajouter les vérificateurs

Ajouter la complexité linéaire.

Voyageur de commerce euclidien (OVT)