915 Classes de complexité : exemples.

Motivations : S'intéresser à la décidabilité des problèmes ne suffit pas : en pratique, un algorithme n'est utilisable que s'il s'exécute en un temps raisonnable et qu'il utilise une place mémoire limitée. → notion de complexité.

I - Généralités sur les complexités

1 - Définitions [CARTON 180]

Définition 1 : Soit une machine de Turing $\mathcal{M}$ et soit $\gamma = q_0 w \to C_1 \to \dots \to C_m$ un calcul de $\mathcal{M}$ d'entrée w.
- le temps $t_{\mathcal{M}}(\gamma)$ de ce calcul γ est m
- l'espace $s_{\mathcal{M}}(\gamma)$ de ce calcul γ est $\max_{0 \le i \le m} |C_i|$

On définit alors la complexité en temps et en espace pour une entrée w :

$t_{\mathcal{M}}(w) = \max_{\gamma} t_{\mathcal{M}}(\gamma)$ et $s_{\mathcal{M}}(w) = \max_{\gamma} s_{\mathcal{M}}(\gamma)$

On peut alors définir les complexités $t_{\mathcal{M}}$ et $s_{\mathcal{M}}$ de la machine $\mathcal{M}$: $t_{\mathcal{M}}(n) = \max_{|w|=n} t_{\mathcal{M}}(w)$ et $s_{\mathcal{M}}(n) = \max_{|w|=n} s_{\mathcal{M}}(w)$.

Remarque : $s_{\mathcal{M}}(n) \ge n$

Lemme 2 : Pour toute machine de Turing $\mathcal{M}$, il existe une constante k telle que :

$s_{\mathcal{M}}(n) \le \max(t_{\mathcal{M}}(n), n)$ et $t_{\mathcal{M}}(n) \le 2^{k s_{\mathcal{M}}(n)}$.

2 - Complexité en temps & modèles

Théorème d'accélération 3 : Soit $k \ge 0$ un entier, et soit $\mathcal{M}$ une machine de Turing. Si $n = O(t_{\mathcal{M}}(n))$, alors il existe une machine $\mathcal{M}'$, équivalente à $\mathcal{M}$, telle que : $t_{\mathcal{M}'}(n) \le \frac{t_{\mathcal{M}}(n)}{k}$

Conséquence : Les constantes multiplicatives ne sont pas significatives.

Influence des changements de modèles
- machine à bande bi-infinie : pas de changement
- machine à plusieurs bandes : $t_{\mathcal{M}'}(n) = O(t^2_{\mathcal{M}}(n))$
- machine non déterministe : $t_{\mathcal{M}'}(n) = 2^{O(t_{\mathcal{M}}(n))}$

3 - Exemples de classes & modèles [CAR 203]

Influence des changements de modèles
- bande bi-infinie/plusieurs bandes : pas de changement

Théorème de Savitch 4 : Soit s une fonction de $\mathbb{N}$ dans $\mathbb{R}^+$ telle que $s(n) \ge n$ pour tout n assez grand. Toute machine non déterministe qui fonctionne en espace $s(n)$ est équivalente à une machine de Turing déterministe en espace $O(s^2(n))$. [DEV 1]

II - Classes de complexité en temps

1 - Définitions [CAR 182]

Définition 5 : Pour une fonction $f : \mathbb{N} \to \mathbb{R}^+$, on définit :
- $\mathrm{TIME}(f(n)) = \{$problèmes décidés par une MT déterministe en temps $O(f(n))\}$
- $\mathrm{NTIME}(f(n)) = \{$problèmes décidés par une MT non déterministe en temps $O(f(n))\}$

On définit alors les classes de complexité suivantes :

$P = \bigcup_{k \ge 0} \mathrm{TIME}(n^k)$ $\quad NP = \bigcup_{k \ge 0} \mathrm{NTIME}(n^k)$

$\mathrm{EXPTIME} = \bigcup_{k \ge 0} \mathrm{TIME}(2^{n^k})$ $\quad \mathrm{NEXPTIME} = \bigcup_{k \ge 0} \mathrm{NTIME}(2^{n^k})$

Proposition 6 : $P \subseteq NP \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME}$

Remarques :
- Seule inclusion stricte connue : $P \subsetneq \mathrm{EXPTIME}$
- $P \neq NP$? En revanche si $P = NP$ alors $\mathrm{EXPTIME} = \mathrm{NEXPTIME}$.

Définition 7 : Pour chaque classe C, on note co-C la classe duale qui contient les complémentaires des problèmes dans C. Si $C = $ co-C, la classe C est dite autoduale.

Proposition 8 : P et EXPTIME sont autoduales

→ ANNEXE - INCLUSIONS DES CLASSES

2 - Les classes P et NP : exemples

★ EXEMPLES de problèmes de la classe P [CAR 184]

① - Le problème d'accessibilité (PATH)

entrée : (G, s, t) où $G = (S, A)$ graphe orienté ; $s, t \in S$

problème : existe-t-il un chemin dans G de s à t.

↳ parcours en largeur de G (temps linéaire en la taille de G)

[WOLPER 171]

(b) Langages algébriques : Tout langage algébrique est dans P → Algorithme de Cocke, Kasami et Younger ($O(n^3)$)

@ Le problème 2-COLOR

entrée : $G = (S, A)$ graphe non orienté

problème : existe-t-il $c : S \to \{0,1\}$ tq $\forall (u,v) \in A, c(u) \neq c(v)$?

(d) Le problème 2-SAT

entrée : une formule du calcul propositionnel sous forme normale conjonctive avec au plus 2 littéraux par clause

problème : la formule est-elle satisfiable ?

★ La classe NP : une définition équivalente

Définition 9 : Un vérificateur en temps polynomial pour un langage L est une machine déterministe V qui accepte des entrées de la forme $\langle w, c \rangle$ en temps polynomial en $|w|$ telle que $L = \{w / \exists c \ \langle w, c \rangle \in L(V)\}$

Proposition 10 : Un langage L est dans la classe NP ssi il existe un vérificateur en temps polynomial pour L.

Remarque : pour caractériser la classe NEXPTIME, on peut définir de la même façon un vérificateur en temps exponentiel.

★ EXEMPLES de problèmes de la classe NP

@ Le problème du voyageur de commerce (VC)

entrée : V un ensemble de n villes, b une constante, pour chaque paire de villes, distance $d(v_i, v_j)$

problème : existe-t-il une permutation $v_{p_1}, \ldots, v_{p_n}$ telle que $\sum d(v_{p_i}, v_{p_{i+1}}) + d(v_{p_n}, v_{p_1}) \leq b$?

(b) Le problème du circuit hamiltonien (CH)

entrée : $G = (S, A)$ un graphe $|S| = n$

problème : existe-t-il une permutation $s_{p_1}, \ldots, s_{p_n}$ telle que $\forall i \ (s_{p_i}, s_{p_{i+1}}) \in A$, $(s_{p_n}, s_{p_1}) \in A$?

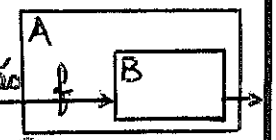
Remarques : • algorithme évident : tester toutes les permutations possibles : $n!$ → complexité non ? polynomiale

• une MT non déterministe peut choisir une permutation et vérifier en temps polynomial qu'il s'agit effectivement d'une solution → NP

[CAR 188]

3 NP-complétude

★ Réduction polynomiale



Définition 11 : Soient A et B, des problèmes codés respectivement par L_A et L_B sur les alphabets Σ_A et Σ_B. Une réduction polynomiale de A à B est une fonction $f : \Sigma_A^* \to \Sigma_B^*$ calculable en temps polynomial par une MT déterministe tq : $w \in L_A \Leftrightarrow f(w) \in L_B$

Notation : $A \leq_p B$

Proposition 12 : Si $A \leq_p B$ et $B \in P$, alors $A \in P$

EXEMPLE : $CH \leq_p VC$ [WOLPER 173]

[CAR 189]

★ NP-complétude : définition

Définition 13 : Un problème A est dit NP-difficile si tout problème B de NP se réduit à A, ie $B \leq_p A$.

Proposition 14 : Si A est NP-difficile et si $A \leq_p B$, alors B est NP-difficile.

Définition 15 : Un problème NP et NP-difficile est dit NP-complet

[WOL 180]

Théorème 16 : S'il existe un langage NP-complet L décidé par un algorithme polynomial, alors tous les langages de NP sont décidables en temps polynomial, ie $P = NP$.

★ EXEMPLES de problèmes NP-complets

@ Satisfiabilité d'une formule (SAT)

entrée : une formule du calcul propositionnel

problème : la formule est-elle satisfiable ?

Théorème 17 [COOK] : Le problème SAT est NP-complet [DEV leçon 913]

(b) Le problème 3-SAT

restriction du problème SAT aux formules sous forme normale conjonctive avec au plus 3 littéraux par clause.

[WOL 191]

@ Le problème de la couverture des sommets (VC)

entrée : un graphe $G = (S, A)$ et une constante j

problème : existe-t-il $S' \subseteq S$ tq $|S'| \leq j$ et tq $\forall (u,v) \in A$, $u \in S'$ ou $v \in S'$

(d) Le problème 3-COL

@ CH et VC déjà mentionnés

4 - Tout espoir n'est pas forcément perdu

Confronté à un problème NP-complet, des solutions existent : • trouver un algorithme de résolution pour une sous-classe de problème

EXEMPLE : la coloration de graphes d'intervalles

• trouver un algorithme d'approximation qui renvoie une solution "assez proche" de la solution optimale.

EXEMPLE : Le problème du voyageur de commerce euclidien (VCE)

description : au problème VC, on ajoute l'hypothèse d'inégalité triangulaire : $d(u,w) \leq d(u,v) + d(v,w)$

[CORMEN] Remarque : VCE est NP-complet.

Théorème 18 Pour le problème VCE, il existe un algorithme d'approximation polynomial avec une garantie de performance 2. [DEV 2]

III - Classes de complexité en espace

[CAR 205]

1 - Définitions

Définition 19 Pour une fonction $f : \mathbb{N} \to \mathbb{R}^+$, on définit

- $SPACE(f(n)) = \{$problèmes décidés par une MT déterministe en espace $O(f(n))\}$
- $NSPACE(f(n)) = \{$problèmes décidés par une MT non déterministe en espace $O(f(n))\}$

On définit alors les classes de complexité suivantes :

$$PSPACE = \bigcup_{k \geq 0} SPACE(n^k) \qquad NPSPACE = \bigcup_{k \geq 0} NSPACE(n^k)$$

[CAR 206] $$EXPSPACE = \bigcup_{k \geq 0} SPACE(2^{n^k}) \qquad NEXPSPACE = \bigcup_{k \geq 0} NSPACE(2^{n^k})$$

Proposition 20 (conséquence du th. de Savitch)

$PSPACE = NPSPACE$ et $EXPSPACE = NEXPSPACE$

Proposition 21

$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE$

Remarque $PSPACE \subsetneq EXPSPACE$

Remarque : Le théorème d'Immerman et Szelepcsényi (admis) établit que toutes les classes de complexité en espace sont autoduales. [CAR 208]

2 - PSPACE-complétude

Définition 22 Un problème A est dit PSPACE-complet si :

- A est PSPACE
- Tout problème B de PSPACE se réduit polynomialement en temps à A (ie $B \leq_P A$).

Proposition 23 (Réduction) Si $B \in PSPACE$, $A \leq_P B$ et A est PSPACE-complet, alors B est aussi PSPACE-complet.

3 - Un EXEMPLE de problème : QSAT

entrée : une formule booléenne quantifiée close

problème : la formule est-elle vraie ?

Théorème 24 Le problème QSAT est PSPACE-complet

IV - D'autres classes de complexité : quelques exemples

1 - Espace logarithmique

Définition 25 À une convention sur les machines de Turing près, on peut définir : $L = SPACE(\log n)$ $NL = NSPACE(\log n)$

Proposition 26 $L \subseteq NL = co\text{-}NL \subseteq P$

2 - Classes de complexité probabilistes

Dans un algorithme, on peut introduire des éléments probabilistes, soit pour trouver une approximation d'une solution, soit pour obtenir une solution locale plus rapidement.

Définition 27 Un algorithme probabiliste est dit de Las Vegas si le résultat qu'il renvoie est toujours correct. Un tel algorithme est à temps moyen $f(n)$ si pour toute entrée x de taille n, l'algo calcule une réponse en un temps dont l'espérance est bornée par $f(n)$.

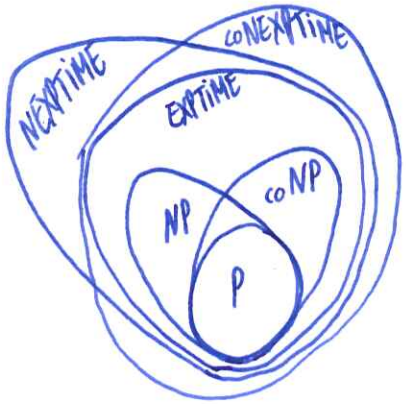
Définition 28 La classe ZPP est la classe des langages pour lesquels on dispose d'un algo LV de temps d'exécution polynomial en espérance.

Définition 29 RP^0 (resp. co-RP) est la classe des langages pour lesquels on dispose d'un algo probabiliste A s'exécutant dans la fine dans un temps polynomial tel que :

- Si $x \in L$ (resp $x \notin L$), $P(A(x) \text{ accepte}) \geq \frac{1}{2}$ (resp $\leq \frac{1}{2}$)
- Si $x \in L$ (resp $x \in L$), $P(A(x) \text{ accepte}) = 0$ (resp $= 1$)

THM 30 : $ZPP = RP^0 \cap co\text{-}RP$

- thm de hiérarchie
- NP - intermediate



Théorème de Savitch

Théorème Soit s une fonction de $\mathbb{N}$ dans $\mathbb{R}^+$ telle que $s(n) \geq n$ pour n assez grand. Toute machine non déterministe qui fonctionne en espace $s(n)$ est équivalente à une machine de Turing déterministe en espace $O(s^2(n))$.

Démonstration

Soit M une machine de Turing fonctionnant en espace $s(n)$.
Sans perte de généralité, on suppose que M n'a qu'une seule configuration acceptante q_f.

ETAPE 1 : fonction ACCESS

entrée : deux configurations C et C' de M
deux entiers t et r

sortie : renvoie VRAI s'il existe un calcul de C à C' de longueur au plus t qui n'utilise que des configurations intermédiaires de taille au plus r.

```
ACCESS(C, C', t, r)
if t = 0 then
    return C = C'
else if t = 1 then
    return C = C' or C → C'
else
    for all C'' de taille ≤ r do
        if ACCESS(C, C'', ⌈t/2⌉, r)
           and ACCESS(C'', C', ⌊t/2⌋, r)
        then
            return true
    return false
```

Complexité en espace de la fonction ACCESS

- la valeur de t est divisée par 2 à chaque appel récursif $\longrightarrow$ le nombre maximal d'appels imbriqués de la fonction est donc borné par $\log_2 t$
- chaque appel utilise :
  - 3 variables C, C', C'' de taille au plus r.
  - un entier t qui nécessite un espace au plus $\log_2 t$ (codage binaire)
- comme r est constante, on peut considérer qu'il n'y a qu'une seule instance de cette variable qui occupe un espace $\log_2 r$

BILAN L'espace utilisé globalement est borné par

$$\boxed{O(\log_2 r + \log_2 t\,(r + \log_2 t))}$$

hypothèse supplémentaire : la fonction $s(n)$ est calculable en espace au plus $s(n)$

- M fonctionne en espace $s(n)$, d'après le lemme 2, il existe une constante K telle que $t_M(n) \leq 2^{Ks(n)}$
- Soit w une entrée de taille n. Comme q_f est l'unique configuration acceptante, on a $w \in L(M) \Leftrightarrow$ ACCESS$(q_0 w, q_f, 2^{Ks(n)}, s(n))$

↳ On peut alors donner une machine déterministe M' équivalente à M :

> pour chaque entrée w ($|w| = n$), M' calcule $s(n)$
> puis utilise la fonction ACCESS avec $t = 2^{Ks(n)}$ et $r = s(n)$

L'espace utilisé est alors borné par $\boxed{O(s^2(n))}$

ETAPE 3 : Démonstration dans le cas général

(!) On ne suppose plus que la fonction $s(n)$ est calculable.

- Soit w une entrée de taille n

Notons m la taille maximale d'une configuration accessible à partir de q_0w

OBJECTIF : Trouver un algorithme qui calcule m en utilisant un espace borné par $O(m^2)$

Ainsi, on pourra conclure :

- en effet, comme $m \leq s(n)$, l'espace nécessaire pour cet algorithme est donc borné par $O(s^2(n))$
- on peut alors raisonner de façon similaire à l'étape 2 et donner la description d'une machine déterministe M' équivalente à M :

> M' calcule la valeur de m
> puis utilise ACCESS avec $t = 2^{km}$ et $r = m$ pour décider s'il y a un calcul de q_0w à la configuration acceptante q_f

ALGO pour calculer m.

- $\forall k \geq n+1$, on note $N_k = \#\{C \;/\; |C| \leq k \text{ et } q_0w \to C_1 \to \dots \to C_f \to C, \forall i \in \{1,\dots,f\}, |C_i| \leq k\}$
  - par définition, la suite $(N_k)_k$ est croissante et elle est bornée par le nombre de configurations de taille au plus $s(n)$
    - ↳ il existe $k \geq n+1$ tq $N_{k+1} = N_k$
  - De plus, si k est le plus petit entier tel que $N_k = N_{k+1}$, alors $k = m$ et N_k est le nombre total de configurations accessibles à partir de q_0w

> **PREUVE** : Par l'absurde, supposons qu'il existe une configuration accessible à partir de q_0w, avec un calcul utilisant des configurations de taille $\geq k+1$. Considérons la première configuration de ce calcul, C_{i_0} tq $|C_{i_0}| = k+1$ et $\forall i \leq i_0, |C_i| \leq k$ (existe car la taille des configurations ne varie que d'une unité au plus à chaque étape de calcul). Alors $N_{k+1} > N_k$ Absurde.

ALGO.

```
entrée w de taille n
k ← n
i ← 0
N ← 1
while (i ≠ N) do
    k ← k+1
    N ← i   (*)   i ← 0
    for all C de taille au plus k do
        if ACCESS(q0w, C, 2^(k^k), k) then
            i ← i+1
return k
```

Complexité en espace.

- On a $k \leq m$ donc l'espace utilisé par chaque appel à la fonction ACCESS est borné par $O(m^2)$
- $i, N \leq m$

↳ Finalement l'espace global utilisé en $O(m^2)$

CQFD

Commun p102

Approximation du voyageur de commerce euclidien

On considère le problème du voyageur de commerce dans le cas euclidien : $G = (S, A)$, c coût euclidien.

Pour résoudre ce problème, on utilise les arbres couvrants de coût minimal :

```
Prim (G, c, r)
    pour v ∈ S faire                          } initialisation
        Clé [v] ← ∞
        Père [v] ← NIL
    fin pour
    Clé [r] ← 0
    F ← S                                     (S = ensemble des sommets à traiter)
    Tant que F ≠ ∅
        u ← Extraire_Min (F)
        pour v ∈ Adj [u]
            si v ∈ F et c(u,v) < Clé [v] alors
                Père [v] ← u
                Clé [v] ← c(u,v)
            fin si
        fin pour
    fin tant que
    Renvoyer Père
```

→ Extraire_Min est une fonction qui trouve l'élément de F de Clé minimale et qui le supprime de F

Avec une matrice d'adjacence et un tableau pour F, on obtient une complexité en $O(S^2)$

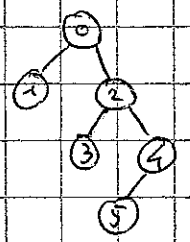
[COR] p 580-586 → On ne fait pas ici la démonstration de la correction de Prim

Idée pour la faire :
- on montre que Prim construit bien un arbre couvrant
  à chaque étape Prim $(S \setminus F)$ est un arbre
- on montre que l'arbre construit est minimal :
  à chaque étape $\{(u,v) / u,v \in S \setminus F\}$
  est inclus dans un arbre couvrant minimal

On vient à l'algorithme d'approximation :

VCE_approché (G, c)
  Choisir $r \in S$
  $T \leftarrow$ Prim (G, c, r)
  $H \leftarrow$ liste des sommets lus d'un parcours préfixe de T
  retourner H

Rappel parcours préfixe :



⇝ [0, 1, 2, 3, 4, 5]

Complexité : $O(S^2)$ comme Prim

prop : $c(H) \leq 2c^*$ avec H donné par VCE_approché et c^* coût d'un chemin hamiltonien optimal

preuve : Soit H^* circuit hamiltonien de coût minimal. En retirant une arête à H^*, on obtient un arbre couvrant. Comme T est un arbre couvrant, on a :

$$c(T) \leq c(H^*)$$

Rappel parcours complet : un sommet apparaît à chaque visite, ie dans l'exemple précédent : $[0, 1, 0, 2, 3, 2, 4, 5, 4, 2, 0]$

Soit W un parcours complet de T. On a :

$c(W) = 2c(T)$ car chaque arête est traversée 2 fois (une fois en descendant et une en remontant)

d'où $c(W) \leq 2c(H^*)$

A partir de W, pour obtenir H, il suffit de supprimer un sommet s'il a déjà été visité.

Or par inégalité triangulaire, supprimer un sommet diminue le coût :

si on passe de $[0, 1, 2]$ à $[0, 2]$

on a $c([0,1,2]) = c(0,1) + c(1,2)$

$\geq c([0,2]) = c([0,2])$

d'où $c(H) \leq c(W) \leq 2c(H^*)$

De plus on remarque H obtenu à la fin contient une et une seule fois chaque sommet donc il est bien hamiltonien.

Remarque bonus : Imposer l'inégalité triangulaire n'est pas un caprice. On peut en fait montrer que le problème de voyageur de commerce n'admet pas d'algorithme d'approximation à garantie $\epsilon > 1$.