Computational Complexity

Cours de Pascal KOIRAN

Notes et figures LaTeX par A. Mazoyer

M1

ÉCOLE
NORMALE
SUPÉRIEURE
DE LYON

Contents

Chapitre 1

Machines de Turing

On travaille avec des machines de Turing à $k \geqslant 1$ rubans. Les rubans sont semi-infinis à droite. Sur chaque ruban, on a une tête de lecture qui lit le contenu d'une case.

A chaque étape :

1. M lit les k caractères situés sous les têtes de lecture $(a_1, \cdots, a_k)$
2. En fonction des caractères $(a_1, \cdots, a_k)$ et de son état interne $q \in Q$, M remplace chaque a_i par un nouveau caractère a_i', M passe dans un nouvel état q', et déplace les têtes de lecture d'au plus une case vers la gauche ou vers la droite

1.1 Définitions

Définition 1.1 (Machine de Turing)

Plus formellement, on a un triplet (Γ, Q, δ) avec Γ l'alphabet du ruban, Q un ensemble fini d'états et $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \to \{G, D, I\}^k$

On calcule des fonctions $f : \{0,1\}^* \to \{0,1\}^*$ (ou $f : \Sigma^* \to \Sigma^*$).

Définition 1.2 (Reconnaître un language)

Pour reconnaître un language $L \subseteq \{0,1\}^*$, on calcule $\mathbf{1}_L : \{0,1\}^* \to \{0,1\}$. On peut ainsi avoir un état acceptant (q_a) et un état de rejet (q_r).

On va supposer que Γ contient au moins :

- le symbole blanc B ($\square$ dans Arone-Rajak ?)
- le symbole de départ $\triangle$, et $0, 1$ (ou en général $\Sigma \subseteq \Gamma$)

Définition 1.3

On a un *ruban d'entrée*, un *ruban de sortie*, des *rubans de travail*, un *état initial* q_a et un *état final* q_f.

Au départ, M est dans l'état q_a, le ruban d'entrée contient $\triangle x B^\infty$ avec $x \in \Sigma^*$ l'entrée $(B \notin \Sigma)$.

Définition 1.4
On dit que M calcule la fonction $f : \Sigma^* \to \Sigma^*$ si pour toute entrée $x \in \Sigma^*$, le calcul de M se termine, avec $f(x)$ écrit sur le ruban de sortie.

Remarques :

▷ le ruban d'entrée est souvent en lecture uniquement

▷ le ruban de sortie est souvent en écriture uniquement

Variantes du modèle

1. rubans bi-infinis
2. on peut simuler une machine avec l'alphabet $0, 1, 2, 3, B, \triangle$ par une machine avec l'alphabet $0, 1, B, \triangle$: $0 \mapsto 00, 1 \mapsto 01, 2 \mapsto 10, 3 \mapsto 11$, facteur 2 en espace (et en temps)

Définition 1.5 (Complexité en temps)
Un language $L \subseteq \Sigma^*$ est *reconnu* en temps $T(n)$ par une machine M si :

- M reconnaît L
- sur toute entrée x de taille L, M s'arrête en au plus $T(n)$ étapes de calcul

Définition 1.6 (classe DTIME)
L est dans $\mathrm{DTIME}(f(n))$ s'il existe une machine (à plusieurs rubans) qui reconnaît L en temps $O(f(n))$.
On supposera toujours $f(n) \geqslant n + 1$.

Définition 1.7 (classe P)

$$\mathbf{P} = \bigcup_{\alpha \geqslant 1} \mathrm{DTIME}(n^\alpha)$$

Théorème 1.8
Si $L \in \mathrm{DTIME}(f(n))$, alors L peut être résolu en temps $\Omega(f(n)^2)$ sur une machine à 1 ruban.

Théorème 1.9 (Théorème de simulation efficace)
Pour toute machien M fonctionnant en temps $T(n)$, il existe une machine M' à 2 rubans qui fonctionne en temps $O(T(n \log T(n)))$ telle que $M(x) = M'(x)$ pour toute entrée $x \in \Sigma^*$.

Définition 1.10 (Machine de Turing universelle)
Une *machine universelle* U prend en entrée des couples $\langle x, \alpha \rangle$ avec $x \in \{0,1\}^*$ et $\alpha \in \{0,1\}^*$ et simule code d'une machine de Turing M_α, c'est-à-dire pour tout x et tout α, on doit avoir $U(\langle x, \alpha \rangle) = M_\alpha(x)$.

Théorème 1.11
Il existe une machine de Turing universelle U telle que sur toute entrée $\langle x, \alpha \rangle$, si M_α s'arrête sur l'entrée x en T étapes, alors U s'arrête en au plus $c \cdot T \log T$ avec c une constante dépendant de x.

Preuve
Construction de U:
On montre d'abord que si M_α est une machine à 2 rubans, U peut simuler M_α en temps linéaire.
M_β a 2 rubans et l'alphabet est $\triangle, 0, 1, B$. U a 4 rubans :

- 2 rubans stockent les rubans de M_β
- un ruban stocke l'état de M_β
- le ruban d'entrée contient $\langle x, \beta \rangle$

Pour faire une étape de calcul de M_β, U doit déterminer $\delta(q, a, b)$. La complexité de cette opération est cachée dans la constante.

Cas général:

1. sur l'entrée $\langle x, \alpha \rangle$, construire la machine M_β à 2 rubans donnée par le théorème de simulation efficace
2. simuler la machine M_β en temps linéaire

Complexité
construire M_β prend un temps indépendant de x

□

1.2 Non déterminisme

Définition 1.12 (Machine de Turing non déterministe)
La définition est la même que celle d'une machine de Turing déterministe, mais on remplace δ par

$$\delta : Q \times \Gamma^k \to \mathcal{P}(Q \times \Gamma^k \times \{G, D, I\}^k)$$

On a deux états finaux q_a et q_r.
Une entrée $x \in \Sigma^*$ est acceptée s'il existe une exécution de calcul acceptant sur l'entrée x.

Définition 1.13 (Classe NTIME)
NTIME$(T(n))$ est l'ensemble des langages acceptés par une machine de Turing non déterministe fonctionnant en temps $O(T(n))$ sur toute entrée de taille n et tout chemin de calcul.

Définition 1.14 (Classe NP)
Un langage L est dans **NP** s'il existe une machine non déterministe M fonctionnant en temps polynomial tel que L est l'ensemble des entrées acceptées par M.

$$\mathbf{NP} = \bigcup_{\alpha \geqslant 1} \text{NTIME}(n^\alpha)$$

Théorème 1.15
$L \in \mathbf{NP}$ s'il existe un polynôme p et $A \in \mathbf{P}$ tel que pour tout $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^*, \langle x, y\rangle^{1} \in A$$

[1] $\langle x, y\rangle = 1^{|x|}0xy$

1.3 NP-complétude

Définition 1.16 (Réduction en temps polynomial)
Un problème A se réduit à B en temps polynomial s'il existe une fonction $f : \Sigma^* \to \Sigma^*$ calculable en temps polynomial telle que $x \in \Leftrightarrow f(x) \in B$.

Notation
$A \leqslant_p B$ ou $A \leqslant_m B$

Remarque : Si $A \leqslant_m B$ et $B \leqslant_m C$, alors $A \leqslant_m C$ (on compose les réductions).

Définition 1.17 (NP-complétude)
A est **NP**−complet si

1. $A \in \mathbf{NP}$
2. $B \leqslant_m A$ pour tout $B \in \mathbf{NP}$

Supposons A **NP**−complet.
Pour que $A' \in \mathbf{NP}$ soit **NP**−complet, on doit montrer que $A \leqslant_m A'$. Dans ce cas pour tout $B \in \mathbf{NP}$, $B \leqslant_m A \leqslant_m A'$ donc $B \leqslant_m A'$.

Le problème de départ classique : SAT (ou 3-SAT)

Dans ce cours : on part de CircuitSAT (satisfiabilité des circuits booléens).

Chapitre 2

Circuits booléens

2.1 Définitions

Définition 2.1 (Circuit booléen)

Un circuit booléen est un DAG (graphe orienté acyclique) dont les sommets sont de degré entrant 0, 1 ou 2.

- Les sommets de degré entrant 2 sont étiquetés par $\wedge$ ou $\vee$.
- Les sommets de degré entrant 1 sont étiquetés par $\neg$.
- Les sommets de degré entrant 0 sont étiquetés par 0, 1 ou des variables booléennes $x_1, \ldots, x_n$

Définition 2.2 (Valuation d'un circuit booléen)

Soit C un circuit booléen avec des variables d'entrée $x_1, \ldots, x_n$. Etant donné $a \in \{0,1\}^n$, on définit pour chaque porte α de C la valeur prise par α sur l'entrée a.

- pour les portes d'entrée, $val(x_i) = a_i$, $val(0) = 0$, $val(1) = 1$.
- pour une porte
  - $\alpha = \beta \vee \gamma$, $val(\alpha) = val(\beta) \vee val(\gamma)$
  - $\alpha = \beta \wedge \gamma$, $val(\alpha) = val(\beta) \wedge val(\gamma)$
  - $\alpha = \neg\beta$, $val(\alpha) = \neg val(\beta)$

Définition 2.3

En supposant que C a une seule porte α de degré sortant 0 : α est la *porte de sortie* de C, et $val(C) = val(\alpha)$.
C calcule une fonction booléenne $f : \{0,1\}^n \to \{0,1\}$.
Si C a s portes de sortie, C calcule $f : \{0,1\}^n \to \{0,1\}^s$.

Remarque : toute fonction booléenne peut être calculée par un circuit.

Problème : donner une famille de fonctions booléennes "explicite" $(f_n)_{n\geqslant 1}$ qui n'est pas calculable par des circuits de taille polynomiale

Problème de la valeur de circuit (PVC/CVP)

- Donnée : son circuit C avec n variables d'entrée, et $\alpha \in \{0,1\}^n$
- Question : sortie de C sur l'entrée $a \in \{0,1\}^n$

Lemme 2.4

$$PVC \in \mathbf{P}$$

Algorithme : tant qu'il existe une porte α de C qui n'est pas évaluée, mais dont toutes les entrées le sont, choisir une telle porte et l'évaluer.
Retourner la valeur prise par la porte de sortie.

On peut exécuter cet algorithme à l'aide d'un tri topologique pour l'ordre d'évaluation.

2.2 Simulation des machines de Turing par les circuits

Proposition 2.5

Soit M une machine à un ruban fonctionnant en temps $T(n)$. M peut être simulée sur les entrées de taille n par un circuit de taille $O(T(n)^2)$.

Remarque : on peut donner la borne $O(T(n)\log(T(n)))$ au lieu de $O(T(n)^2)$, même pour des machines à plusieurs rubans.

2.2.1 Diagramme espace-temps

[[PLS SI QQN A DES FIGURES]]
On doit considérer un diagramme de taille $(T(n)+1)\times(T(n)+1)$.
Contenu de la cellule (i,t) : dépend uniquement du contenu de 3 cellules[1] (bas gauche, bas, bas droite).

[1] C'est le principe de localité du calcul

- $l_{a,i,t} \Leftrightarrow$ à l'instant t, la case i contient la lettre a
- $q_{r,i,t} \Leftrightarrow$ à l'instant t, la machine est dans l'état r et la tête de lecture est sur la case i.

Les valeurs de ces variables peuvent s'obtenir à partir des valeurs des variables pour les 3 cellules $(i-1,t-1),(i,t-1),(i+1,t-1) \rightsquigarrow$ fonction booléenne $f : \{0,1\}^{3p} \to \{0,1\}^p$.

f dépend uniquement de $M \Rightarrow f$ est calculée par un circuit C.

Le circuit final s'obtient en récoltant $O(T(n)^2)$ copies de C.

L'entrée est acceptée si $\bigvee_{i=0}^{T(n)} q_{q_a,i,T(n)}$.

Définition 2.6 (Famille de circuits uniforme)

Soit $(C_n)_{n\geqslant 1}$ une famille de circuits sur n variables d'entrées $x_1, \ldots, x_n$. Cette famille est *uniforme* s'il existe une machine de Turing en temps polynomial qui sur l'entrée 1^n calcule une description complète de C_n : pour chaque porte α elle calcule

- le type de la porte
- si c'est une porte d'entrée, son étiquette
- si ce n'est pas une porte d'entrée, les numéros des portes en entrée de α

Remarque : C_n est de taille polynomiale en n

Théorème 2.7

Un langage $L \subseteq \{0,1\}^*$ est dans **P** si et seulement si L est reconnu par une famille uniforme de circuits booléens de taille polynomiale.

Preuve

$\Rightarrow$ Soit $L \in \mathbf{P}$. On a vu que L peut être reconnu par une famille de circuits de taille polynomiale.

Algorithme : boucle sur i et $t \rightsquigarrow$ construction en temps polynomial (espace logarithmique)

$\Leftarrow$ Algorithme de reconnaissance de L :

- sur l'entrée $x \in \{0,1\}^n$, construire C_n en temps polynomial

- évaluer C_n sur l'entrée x en temps polynomiale

□

2.3 Un premier problème NP-complet

Théorème 2.8

CircuitSAT est **NP**−complet.

Données : un circuit booléen C avec n variables d'entrée.
Question : existe-t-il une entrée $a \in \{0,1\}^n$ telle que $C(a) = 1$?

Preuve

- CircuitSAT $\in$ **NP** : a est le certificat.
- Soit $L \in$ **NP**. Il existe une machine de Turing non déterministe M fonctionnant en temps polynomial $T(n)$ qui reconnaît L.

  Soit $x \in \{0,1\}^n$. On doit construire en temps polynomial un circuit C_x tel que C_x est satisfiable si et seulement si $x \in L$.

  On peut simuler M sur l'entrée x par un circuit de taille $O(T(n)^2)$, mais ce circuit n'est pas suffisant : il faut modéliser le non déterminisme.

  On ajoute des variables d'entrée supplémentaires $y_1, \ldots, y_{T(n)}$ qui modélisent les choix non déterministes de la machine.

  On construit un circuit C_x qui simule M sur l'entrée x, en utilisant les variables y_i pour les choix non déterministes.

  Alors C_x est satisfiable si et seulement si il existe une suite de choix non déterministes (valeurs des y_i) telle que M accepte l'entrée x, c'est-à-dire si et seulement si $x \in L$.

□

Théorème 2.9 (Cook-Levin)

3-SAT est **NP**−complet.

Preuve

- 3-SAT $\in$ **NP** : la valuation satisfaisante est le certificat.
- On fait une réduction de CircuitSAT à 3-SAT.

  Soit C un circuit booléen avec des variables d'entrée $x_1, \ldots, x_n$. Pour chaque porte α de C, on crée une variable z_α qui représente la valeur prise par la porte α. On utilise l'identité $P \Rightarrow Q \Leftrightarrow \neg P \vee Q$ pour construire des clauses qui contraignent les variables z_α à respecter le fonctionnement des portes. Par exemple :

  - si $\alpha = \beta \wedge \gamma$, on ajoute les clauses $(\neg z_\beta \vee \neg z_\gamma \vee z_\alpha)$, $(z_\beta \vee \neg z_\alpha)$, $(z_\gamma \vee \neg z_\alpha)$
  - si $\alpha = \beta \vee \gamma$, on ajoute les clauses $(z_\beta \vee z_\gamma \vee \neg z_\alpha)$, $(\neg z_\beta \vee z_\alpha)$, $(\neg z_\gamma \vee z_\alpha)$
  - si $\alpha = \neg\beta$, on ajoute les clauses $(\neg z_\beta \vee \neg z_\alpha)$, $(z_\beta \vee z_\alpha)$

  Enfin, on ajoute la clause $(z_{\alpha_{sortie}})$ pour forcer la porte de sortie à être vraie.

□

Chapitre 3

Complexité en espace

3.1 Définitions

Définition 3.1

L'*espace utilisé par une machine de Turing déterministe* sur l'entrée x est le nombre de cases distinctes utilisées sur les **rubans de travail** au cours de son calcul sur x.

On dit que M fonctionne en espace $s(n)$ si M s'arrête sur toutes ses entrées, et utilise un espace au plus $s(n)$ sur toute entrée de taille n.

DSPACE$(s(n))$ est la classe des langages reconnus par une machine de Turing déterministe fonctionnant en espace $O(s(n))$.

On supposera que sur le ruban d'entrée, la tête de lecture ne dépasse jamais la fin de l'entrée.

Exemple

- Un algorithme naïf pour SAT utilise un espace $O(n)$: on peut énumérer toutes les affectations possibles des variables en utilisant un compteur binaire de taille n, et vérifier si l'une d'entre elles satisfait la formule.
- L'addition de deux entiers de taille n peut être effectuée en espace $O(\log n)$: il suffit de stocker les positions des bits en cours d'addition et la retenue.

Proposition 3.2

$$\text{NTIME}(f(n)) \subseteq \text{DSPACE}(f(n))$$

Définition 3.3 (Fonction constructible en espace)

Une fonction $t : \mathbb{N} \to \mathbb{N}$ est *constructible en espace* s'il existe une machine de Turing qui sur l'entrée 1^n calcule $1^{t(n)}$ en espace $O(t(n))$.

Preuve (de la proposition)

En supposant t constructible en espace

Soit $L \in \text{NTIME}(t(n))$, et M une machine non déterministe qui reconnaît L en temps $\leqslant \alpha t(n)$.
On code un chemin de calcul de M $y \in \{0, 1, \ldots, R-1\}^{\alpha t(n)}$ où R est le nombre de choix possibles à chaque étape (dépendant de M).

Algorithme : sur l'entrée x de taille n

1. Calculer $t(n)$ en espace $O(t(n))$
2. Pour chaque y de taille $\alpha t(n)$, simuler M sur l'entrée x en suivant les choix donnés par y. Si l'une des simulations accepte, accepter.
3. Rejeter si aucune simulation n'accepte.

Sans l'hypothèse de constructabilité en espace, on peut obtenir la même inclusion avec une légère modification de l'argument.
On fait fonctionner le même algorithme pour des chemins de calcul de longueur $t = 1, 2, 3, \ldots$ jusqu'à ce que la simulation de M sur l'entrée x s'arrête (ce qui arrive forcément si $x \in L$). On s'arrête pour $t = \alpha t(n)$ au plus. □

Proposition 3.4

Si $L \in \text{DSPACE}(s(n))$ alors $L \in \text{DTIME}\left(2^{C \cdot s(n)}\right)$ pour une constante C si $s(n) \geqslant \log n$.

Preuve

On compte le nombre de configurations distinctes possibles d'une machine M sur l'entrée x de taille n.
Si le calcul prend un temps $> N$, on boucle. On doit montrer que N est $2^{O(s(n))}$.

Une configuration est définie[1] par :

- l'état courant : au plus Q possibilités
- la position de la tête de lecture sur le ruban d'entrée : au plus n possibilités
- le contenu des cases utilisées sur les rubans de travail : au plus $|\Gamma|^{s(n)}$ possibilités
- la position des têtes de lecture sur les rubans de travail : au plus $s(n)^k$ possibilités si k est le nombre de rubans de travail

[1] On n'a pas besoin de la position de la tête sur le ruban de sortie car ça n'influt pas le calcul.

□

Question : $\text{DSPACE}(1) \stackrel{?}{\subseteq} \text{DTIME}(1)$
Non ! (par exemple la machine qui renvoit le dernier symbole de l'entrée)

Pour s arbitraire, on a bien $L \in \text{DSPACE}(s(n)) \Rightarrow L \in \text{DTIME}(2^{O(s(n))})$.

Corollaire 3.5

Si $L = \text{DSPACE}(\log n)$ alors $L \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \text{PSPACE}$.

Attention !
$L \subsetneq \text{PSPACE}$

Théorème 3.6

Si deux fonctions $f, g : \{0,1\}^* \to \{0,1\}^*$ sont calculables en espace $s(n) \geqslant \log n$, leur composée $f \circ g$ est calculable en espace $O(s(|x|)) + s(|g(x)|)$.

Remarque : $|g(x)| \leqslant 2^{O(s(|x|))}$

Corollaire 3.7
Si f et g sont calculables en espace $O(\log n)$: $|g(x)| \leqslant p(n)$ pour un certain polynôme p, alors $f \circ g$ est calculable en espace $O(\log n)$.

Idée : on calcule un bit de $g(x)$ uniquement quand on en a besoin pour calculer $f(g(x))$.

Lemme 3.8
Etant donné i, on peut calculer le i-ième bit de $g(x)$ en espace $O(s(|x|))$.

Preuve
Faire fonctionner M_g sans écrire sur le ruban de sortie, mais en comptant le nombre de bits écrits[2]. Dès que le compteur atteint i, on arrête et on renvoit le bit courant.

Stockage du compteur : $O(s(|x|))$. □

[2] Pas vraiment écrits du coup

Preuve (du théorème)

Algorithme pour calculer $f(g(x))$:

```
1: i ← 0
2: tant que M_g ne s'est pas arrêtée faire
3:     Calculer la lettre a sur la case i du ruban d'entrée de M_f.
4:     Effectuer un pas de calcul de M_f.
5:     si la tête de lecture de M_f se déplace à droite sur son ruban d'entrée alors
6:         i ← i + 1
7:     fin si
8:     si la tête de lecture de M_f se déplace à gauche sur son ruban d'entrée alors
9:         i ← i − 1
10:    fin si
11: fin tant que
```

□

3.2 Hiérarchie en espace

Théorème 3.9
Si s est constructible en espace et $s(n) \geqslant \log n$, alors il existe un langage reconnaissable en espace $O(s(n))$ mais pas en espace $o(s(n))$.

Corollaire 3.10

$$L \neq \text{PSPACE}$$

$$\text{DSPACE}(\log n) \subsetneq \text{DSPACE}(n) \subsetneq \text{DSPACE}(n^2) \subsetneq \cdots \subsetneq \text{PSPACE}$$

Idée de preuve : diagonalisation.
On construit une machine D qui fonctionne en espace $O(s(n))$ et reconnaît un langage A différent de tous les langages reconnus en espace $o(s(n))$. D simule la machine M fonctionnant en espace $o(s(n))$ sur son propre code, et fait l'inverse de ce que fait M (accepte si M rejette, rejette si M accepte). D et M ne peuvent pas reconnaître le même langage.

Lemme 3.11

Si $L \in \text{DSPACE}(s(n))$ alors L est reconnaissable en espace $O(s(n))$ par une machine à un ruban de travail et alphabet $\{0, 1, B, \triangleright\}$.

Lemme 3.12

Il existe une machine de Turing universelle U qui prend en entrée $\langle M, x\rangle$ avec $x \in \{0,1\}^*$ et M une machine à un ruban de travail et alphabet $\{0, 1, B, \triangleright\}$, et qui simule M sur l'entrée x en espace $O(s(n))$ si M fonctionne en espace $s(n)$.
On peut même supposer que U possède un unique ruban de travail.

Preuve (du théorème)

L'algorithme suivant fonctionne en espace $O(s(n))$:

1: Lire l'entrée $w \in \{0,1\}^*$
2: $n \leftarrow |w|$
3: Calculer $s(n)$ en espace $O(s(n))$
4: Réserver $s(n)$ cases sur le ruban de travail
5: **si** w n'est pas de la forme $\langle M\rangle 10^*$ pour une certaine machine M **alors**
6: Rejeter
7: **sinon**
8: Simuler M sur l'entrée w en comptant le nombre d'étapes de la simulation.
9: **si** le compteur dépasse $2^{2\cdot s(n)}$ ou si l'espace utilisé dépasse $s(n)$ **alors**
10: Rejeter
11: **sinon**
12: **si** M accepte **alors**
13: Rejeter
14: **sinon**
15: Accepter
16: **fin si**
17: **fin si**
18: **fin si**

Propriétés de D :

1. D s'arrête sur toute entrée
2. D fonctionne en espace $O(s(n))$
3. Si B est décidable en espace $o(s(n))$ par une machine M alors $B \neq A$ où A est le langage reconnu par D.

   D simule M en espace $o(s(n))$.

   Pour $n \geqslant n_0$, D effectue cette simulation en espace $< s(n)$: D ne va pas manquer d'espace.

   M fonctionne en temps $n \cdot 2^{o(s(n))} < 2^{2\cdot s(n)}$ pour n suffisamment grand : D ne va pas manquer de temps

□

3.3 Complexité en espace non déterministe

Définition 3.13
Une machine non déterministe M fonctionne en espace $s(n)$ si sur toute entrée de taille n, chaque chemin de calcul utilise un espace au plus $s(n)$.

NSPACE($s(n)$) est la classe des langages reconnus par une machine de Turing non déterministe fonctionnant en espace $O(s(n))$.

Exemple
NL = NSPACE($\log n$)

Proposition 3.14
NSPACE($s(n)$) $\subseteq$ DTIME $\left(2^{O(s(n))}\right)$ pour $s(n) \geqslant \log n$.

Preuve
On considère le graphe orienté G_x des configurations de M sur l'entrée x de taille n. On a $c_1c_2 \in E(G_x)$ si M peut passer de la configuration c_1 à la configuration c_2 en un seul pas de calcul. G_x a $N = 2^{O(s(n))}$ sommets. On explore G_x à partir de la configuration initiale c_{start}, et on accepte si on atteint une configuration d'acceptation. □

Remarque : On a pas utilisé l'hypothèse que M s'arrête sur toutes ses entrées.

Proposition 3.15

$$L \subseteq \text{NL} \subseteq \text{PSPACE}$$

$$\text{NL} \neq \text{PSPACE}$$

Théorème 3.16 (de Savitch)

$$\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$$

Rappels : On a que

- NSPACE($s(n)$) $\subseteq$ DTIME($2^{O(s(n))}$)
- et NSPACE($s(n)$) $\subseteq$ DSPACE($s(n)^2$) (Savitch)

lorsque $s(n) \geqslant \log n$. On a

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE},$$

et avec Savitch et la hiérarchie en espace, on a

$$\mathsf{NL} \subsetneq \mathsf{PSPACE}.$$

Définition 3.17
On définit

$$\mathsf{PSPACE} := \bigcup_{k \geqslant 1} \text{DSPACE}(n^k).$$

Remarque : On peut aussi définir la classe NPSPACE mais cette classe est égale à PSPACE par Savitch. On a donc

$$\mathsf{P} \subseteq \mathsf{NP} \underset{(*)}{\subseteq} \mathsf{PSPACE} \underset{(**)}{\subseteq} \mathsf{EXPTIME} = \bigcup_{k \geqslant 1} \mathrm{DTIME}(2^{n^k}),$$

où

- $(*)$ est vraie car $\mathrm{NTIME}(t(n)) \subseteq \mathrm{DSPACE}(t(n))$;
- $(**)$ est vraie car $\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{\mathrm{O}(s(n))})$.

On sait, de plus, que $\mathsf{P} \subsetneq \mathsf{EXPTIME}$ par la hiérarchie en temps (variante (vue en TD) de la hiérarchie en espace vue avant).

3.4 Formules booléennes quantifiés.

On autorise des quantificateurs universels et existentiels aux formules vues précédemment.

Exemple

Les formules

- $\forall x \, \exists y \, \big(\overbrace{(x \vee y) \wedge (\overline{x} \vee \overline{y})}^{\text{c'est } x \text{ xor } y} \big)$
- $\exists y \, \forall x \, \big((x \vee y) \wedge (\overline{x} \vee \overline{y}) \big)$

sont des formules booléennes quantifiées. La première est vraie (avec $y = \overline{x}$) mais pas la seconde (avec $x = y$). On suppose que les quantificateurs sont tous en début de formule (forme prénexe).

Définition 3.18

On définit le problème

QBF	**Entrée :** Une formule booléenne quantifiée F (close) **Sortie :** Est-ce que F est vraie ?

Proposition 3.19

On a QBF $\in$ PSPACE.

Preuve

On utilise l'algorithme suivant.

1. Si F ne contient pas de quantificateurs, accepter si F s'évalue à vrai et rejeter si F s'évalue à faux.
2. Si $F = \exists x \, G$, on décide récursivement avec $G[x := 0]$ et $G[x := 1]$
   Si une de ces formules s'évalue à vrai, accepter sinon rejeter.
3. Si $F = \forall x \, G$, on décide récursivement avec $G[x := 0]$ et $G[x := 1]$
   Si une de ces formules s'évalue à faux, rejeter sinon accepter.

Cet algorithme utilise un espace linéaire. □

Théorème 3.20
Le problème QBF est PSPACE-complet.

Preuve

Supposons que A est résolu en espace n^k par une machine M à un ruban. On va montrer que $A \leqslant_P$ QBF.

On considère le diagramme espace-temps de taille n^k en espace et $2^{c \cdot n^k}$ en temps. On ne peut pas simplement utiliser la même preuve que pour SAT car le diagramme est exponentiel.

On construit une formule $\varphi_t(c_1, c_2)$ qui exprime qu'on peut aller de la configuration c_1 en la configuration c_2 en au plus 2^t étapes de calcul.
On a donc que M accepte si $\varphi_{c \cdot n^k}(c_{\text{initiale}}, c_{\text{finale}})$.[3]

La formule $\varphi_0(c_1, c_2)$ est de taille polynomiale (*c.f.* preuve du théorème de Cook).

Pour aller de φ_t à φ_{t+1}, on cherche la configuration du milieu. On pourrait utiliser $(*) := \exists c\, \varphi_t(c_1, c) \wedge \varphi_t(c, c_2)$ qui est correcte mais trop couteuse (taille exponentielle). On choisit plutôt :

$$\exists c\, \forall c_3\, \forall c_4 \left[(c_3 = c_1 \wedge c_4 = c) \vee (c_3 = c \wedge c_4 = c_2) \right] \Rightarrow \varphi_t(c_3, c_4).$$

Cette formule est logiquement équivalente à $(*)$ (en regardant les cas où $(c_3, c_4) = (c_1, c)$ et $(c_3, c_4) = (c, c_2)$). Il est important de se rappeler que que c, c_3, c_4 ne sont pas des variables mais des n-uplets de taille $O(n^k)$ variables booléennes. La taille de φ_{t+1} augmente de $O(n^k)$ par rapport à la taille de φ_t. Au final, on est de taille $O(n^{2k})$.[4] □

[3] On peut considérer qu'il y a une unique configuration acceptante, quitte à transformer tout état acceptant en un état qui efface tout le ruban et remet la tête en position initiale.

[4] Le passage quadratique est similaire au théorème de Savitch.

Le problème QBF est "le" problème PSPACE-complet. En TD, on fera des réductions de certains problèmes à QBF.

Théorème 3.21 (Savitch)
On a $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$ si $s(n) \geqslant \log n$.[5] □

[5] Depuis sa preuve, on n'a jamais réussi à améliorer ce résultat, ni montrer qu'un facteur carré est nécessaire.

Corollaire 3.22
On a $\mathsf{NL} \subseteq \text{DSPACE}(\log^2 n)$.

Proposition 3.23

PATH | **Entrée :** Un graphe G orienté, et $s, t \in V(G)$
Sortie : Existe-t-il un chemin de s à t dans G ?

On a que PATH $\in \text{DSPACE}(\log^2 n)$.

Remarque : En TD, on a vu que PATH est NL-complet, et donc la proposition implique le corollaire. En effet, soit $A \in \mathsf{NL}$ et $x \in \{0,1\}^n$, on a

$$x \in A \Leftrightarrow f(x) \in \text{PATH},$$

où $f : A \leqslant_L$ PATH est la réduction en espace logarithmique (car le problème PATH est NL-complet). La construction de $f(x)$ se fait en espace $O(\log n)$ et décider si $f(x) \in$ PATH

ou non peut se faire en espace $\mathrm{O}\left(\log^2 |f(x)|\right)$, or $|f(x)|$ est polynomial en $|x| = n$, d'où la borne annoncée.

Preuve (de la proposition)

On donne un algorithme $\mathsf{path}(G, u, v, i)$ en espace $\mathrm{O}(\log^2 n)$ qui décide s'il existe, dans G, un chemin de u à v de longueur au plus 2^i. On pourra donc résoudre PATH en appelant $\mathsf{path}(G, s, t, \lceil \log n \rceil)$.

```
1: procedure path(G, u, v, i)
2:     si i = 0 alors
3:         si uv ∈ E(G) ou u = v alors Accepter
4:         sinon Rejeter
5:         fin si
6:     fin si
7:     for all sommet w ∈ V(G) faire
8:         si path(G, u, w, i − 1) et path(G, w, v, i − 1) alors
9:             Accepter
10:        fin si
11:    fin pour
12:    Rejeter
13: fin procedure
```

Pour la correction, on montre si $\mathsf{path}(G, u, v, i)$ accepte alors il existe un chemin de longueur au plus 2^i par récurrence sur i.

- Pour le cas $i = 0$, c'est bon par le premier "**si**".
- Pour l'hérédité, si on a un chemin de longueur au plus 2^{i-1} de u à w et un chemin de longueur au plus 2^{i-1} de w à v, on concatène ces chemins pour obtenir un chemin de u à v de longueur au plus 2^i.

Réciproquement, s'il existe un chemin de u à v de longueur au plus 2^i, alors $\mathsf{path}(G, u, v, i)$ accepte, car il suffit de choisir w comme sommet milieu du chemin.

On a $\mathrm{O}(\log n)$ appels récursifs; et à chaque appel, on doit mémoriser w ce qui demande $\mathrm{O}(\log n)$ bits. On en déduit une complexité en espace en $\mathrm{O}(\log^2 n)$. □

Preuve (du théorème de Savitch)

Supposons que A peut être résolu par une machine non-déterministe M en espace $\mathrm{O}(s(n))$ où est $s(n)$ constructible en espace. Soit $x \in \{0,1\}^n$ une instance de A.
On considère le graphe G_x des ***configurations potentielles*** de la machine M sur l'entrée x, c'est-à-dire l'ensemble des configurations avec x en entrée et au plus $c \cdot s(n)$ cases utilisées sur chaque ruban de travail.

Lemme 3.24

Le graphe G_x a $2^{\mathrm{O}(s(n))}$ sommets et peut être construit en espace $\mathrm{O}(s(n))$. □

On a que

$$x \in A \qquad \Leftrightarrow \qquad (G_x, s, t) \in \textsc{Path},$$

où s est la configuration initiale de M sur l'entrée x et t la configuration acceptante (qu'on supposera unique, *c.f.* preuve de la PSPACE-complétude de QBF). Par le lemme, on a que (G_x, s, t) se fait en espace $\mathrm{O}(s(n))$. Décider si $(G_x, s, t) \in \textsc{Path}$ se fait en espace $\mathrm{O}(\log^2 |G_x|)$ donc $\mathrm{O}(s(n)^2)$. □

Remarque : La preuve précédente utilise deux résultats

- le théorème de composition *amélioré* :

Théorème 3.25 (Composition)

Soient f et g deux fonctions calculables en espace $s_f(n)$ (*resp.* $s_g(n)$). On peut calculer la composée $(f \circ g)(x)$ en espace $O(s_g(|x|)) + s_f(|g(x)|)$. □

- et le fait que l'on pourra supprimer l'hypothèse de constructibilité de $s(n)$:

  Pour cela, on essaye $s(n) = 1, 2, \ldots$ et on s'arrête à $s(n) = i$ si aucune configuration de taille $i+1$ n'est accessible à partir de la configuration initiale sur l'entrée x (appel à l'algorithme pour PATH).

Remarque : Le problème PATH dans les graphes non-orientés est dans L ! C'est un résultat récent (2005).

Index