

# Complexité en espace.

## 1 Définitions et premières propriétés.

**Définition 1.** L'espace utilisé par une machine de Turing déterministe sur une entrée  $x$  est le nombre de cases distinctes utilisés sur les rubans de travail<sup>1</sup> 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  $n$ .

On note  $\text{DSPACE}(s(n))$  l'ensemble des langages reconnus par une machine de Turing déterministe fonctionnant en espace  $O(s(n))$ .

**Remarque 1.** On supposera que, pour le ruban d'entrée, la tête de lecture ne dépassera jamais la fin de l'entrée.

**Exemple 1.**   ▷ Un algorithme naïf pour SAT utilise un espace en  $O(n)$ . En effet, il suffit d'énumérer toutes les valuations possibles avec un compteur binaire de taille  $n$ , puis de vérifier si une de ces valuations satisfait la formule.

▷ L'addition de deux entiers de taille  $n$  peut s'effectuer en espace  $O(\log n)$ . En effet, il suffit de stocker les positions des bits en cours d'addition et la retenue.

---

1. Pas le ruban d'entrée, ni le ruban de sortie, ceci permet de parler de machine utilisant un espace logarithmique, bien que la taille de l'entrée soit  $n$ .

**Définition 2.** On dit que  $t : \mathbb{N} \rightarrow \mathbb{N}$  est *constructible en espace* s'il existe une machine de Turing qui, sur l'entrée  $1^n$  calcule  $1^{t(n)}$  en espace  $O(t(n))$ .

**Proposition 1.** On a l'inclusion

$$\text{NTIME}(f(n)) \subseteq \text{DSpace}(f(n)).$$

**Preuve.** Supposons  $f$  constructible en espace  $(*)$ .

Soit  $L \in \text{NTIME}(f(n))$  et  $M$  une machine non-déterministe reconnaissant  $L$  en temps au plus  $c f(n)$ . On code un chemin de calcul de  $M$  par  $y \in \llbracket 0, R-1 \rrbracket^{c f(n)}$  où  $R$  désigne le nombre de choix possibles à chaque étape (il dépend de  $M$ ).

- 1 : Calculer  $f(n)$  en espace  $O(f(n))$
- 2 : **pour tout**  $y$  de taille  $c f(n)$  **faire**
- 3 :     Simuler  $M$  sur  $x$  en suivant les choix donnés par  $y$
- 4 :     **si** la simulation accepte **alors**
- 5 :     |     **Accepter**
- 6 : **Rejeter**

On a un algorithme en espace  $f(n)$  qui teste l'appartenance au langage  $L$ .

Sans l'hypothèse  $(*)$ , 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  $t = 1, 2, \dots$  jusqu'à ce que la simulation de  $M$  sur l'entrée de  $x$  s'arrête (ce qui arrive forcément si  $x \in L$ ). On s'arrête pour  $t = c f(n)$  au plus.  $\square$

**Proposition 2.** On a l'inclusion

$$\text{DSpace}(s(n)) \subseteq \text{DTIME}(2^{O(s(n))}),$$

dès lors que  $s(n) \geq \log n$ .

**Preuve.** Soit  $N$  tel que, si un calcul prend un temps plus long que  $N$ , alors il boucle. Nous allons montrer que  $N = 2^{O(s(n))}$ . On compte le nombre de configurations distinctes possibles d'une machine  $M$  sur l'entrée  $x$  de taille  $n$ . Une configuration est donnée<sup>2</sup> par :

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

D'où la borne annoncée. □

**Remarque 2.** A-t-on  $\text{DSPACE}(1) \subseteq \text{DTIME}(1)$  ? Non ! En effet, retourner le dernier caractère de l'entrée se fait en espace  $O(1)$  mais pas en temps  $O(1)$ .

**Définition 3.** On définit  $L = \text{DSPACE}(\log n)$ .

**Corollaire 1.** On a les inclusions

$$L \subseteq P \subseteq NP \subseteq \text{PSPACE},$$

où  $\text{PSPACE}$  est l'ensemble des problèmes définis en espace polynomial (défini plus tard). □

**Théorème 1.** Si deux fonctions  $f, g : \{0, 1\}^* \rightarrow \{0, 1\}^*$  sont calculables en espace  $s(n) \geq \log n$  alors leur composée  $g \circ f$  est

---

2. On oublie la position de la tête de lecture sur le ruban de sortie, vu qu'elle n'influe pas le calcul (car écriture uniquement).

calculable en espace  $O(s(|x|)) + s(|f(x)|)$ .

**Preuve.** L'idée est de calculer un bit de  $f(x)$  uniquement quand on en a besoin pour calculer  $g(f(x))$ .

**Lemme 1.** Étant donné  $i$ , on peut calculer le  $i$ -ème bit de  $f(x)$  en espace  $O(s(|x|))$ .

**Preuve.** Faire fonctionner  $M_f$  (la machine calculant  $f$  en espace logarithmique) sans écrire sur le ruban de sortie, mais simplement en comptant le nombre de bits que l'on voulait écrire. Dès lors que  $i$  est atteint, on renvoie le bit courant : c'est le  $i$ -ème bit de  $f(x)$ .  $\square$

On utilise ensuite l'algorithme suivant :

```

1 :  $i \leftarrow 0$ 
2 : tant que  $M_g$  ne s'est pas arrêtée faire
3 :   Calculer le  $i$ -ème bit de  $f(x)$  (par le lemme)
4 :   L'écrire sur le ruban d'entrée de  $M_g$ 
5 :   Effectuer un pas de calcul de  $M_g$ 
6 :   si la tête  $\rightarrow$  sur le ruban d'entrée de  $M_g$  alors
7 :      $i \leftarrow i + 1$ 
8 :   sinon si la tête  $\leftarrow$  sur le ruban d'entrée de  $M_g$  alors
9 :      $i \leftarrow i - 1$ 
```

$\square$

**Corollaire 2.** Si  $f$  et  $g$  sont calculables en espace  $O(\log n)$  alors  $g \circ f$  est calculable en espace  $O(\log n)$ .  $\square$

## 2 Hiérarchie en espace.

**Théorème 2.** Si  $s$  est constructible en espace et  $s(n) \geq \log n$  alors il existe un langage reconnaissable en espace  $O(s(n))$  mais pas en espace  $o(s(n))$ .

**Preuve.** L'idée est de faire une preuve par diagonalisation. On construit une machine  $D$  qui fonctionne en espace  $O(s(n))$  et qui reconnaît un langage  $A$  différent de tous les langages reconnus en espace  $o(s(n))$ . Pour cela,  $D$  simule une machine tournant en espace au plus  $o(s(n))$  et on a que  $D(\langle M \rangle)$  accepte ssi  $M(\langle M \rangle)$  rejette (et inversement).

**Lemme 2.** Si  $L \in \text{DSpace}(s(n))$  alors  $L$  est reconnaissable en  $O(s(n))$  par une machine à un ruban de travail dont l'alphabet est  $\{0, 1, \square, \triangleright\}$ .  $\square$

**Lemme 3.** 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 dont l'alphabet est  $\{0, 1, \square, \triangleright\}$ , 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  $U$  ne possède qu'un seul ruban de travail.  $\square$

À l'aide de ces deux lemmes, on donne l'algorithme suivant (machine  $D$ ).

- 1 : Lire l'entrée  $w \in \{0, 1\}^*$  (notons  $n$  sa taille)
- 2 : Calculer  $s(n)$  en espace  $O(s(n))$
- 3 : Réserver  $s(n)$  cases sur le ruban de travail
- 4 : **si**  $w$  n'est pas de la forme  $\langle M \rangle 10^{*^3}$  **alors**
- 5 :  $\lfloor$  **Rejeter**
- 6 : Simuler  $M$  sur l'entrée  $w$
- 7 : **si** on dépasse  $2^{2 \cdot s(n)}$  étapes ou  $s(n)$  cases mémoires **alors**
- 8 :  $\lfloor$  **Rejeter**
- 9 : **si**  $M$  accepte **alors Rejeter**
- 10 : **sinon Accepter**

On a que la machine  $D$  s'arrête sur toute entrée et fonctionne en espace  $O(s(n))$ .

De plus, si  $B$  est un langage décidable en  $o(s(n))$  par une machine

$M$ , alors  $B$  est différent du langage de  $A$ . En effet,  $D$  simule  $M$  en espace  $o(s(n))$ .

- ▷ Ainsi, il existe  $n_0$  tel que, pour  $n \geq n_0$ ,  $D$  fera une simulation en espace strictement plus petit que  $s(n)$ , donc ne manquera pas d'espace.
- ▷ Aussi, comme  $M$  fonctionne en temps  $2^{o(s(n))}$  alors la simulation sera en temps strictement plus petit que  $2^{s(n)}$  pour  $n \geq n_1$  pour un certain  $n_1$ , donc ne manquera pas de temps.

En posant  $n_2 := \max(n_0, n_1)$ , on a que sur l'entrée  $\langle M \rangle 10^{n_2}$ , la simulation de  $M$  se fera jusqu'au bout et  $D$  donne une réponse différente de  $M$  sur la même entrée. D'où les deux langages  $B$  et  $A$  sont différents.

□

**Corollaire 3.** On a les inclusions strictes suivantes :

$$L \subsetneq DSPACE(n) \subsetneq DSPACE(n^2) \subsetneq \dots \subsetneq PSPACE.$$

### 3 Complexité en espace non-déterministe.

**Définition 4.** Une machine non-déterministe  $M$  fonctionne en espace au plus  $s(n)$  si, sur toute entrée de taille  $n$ , chaque chemin de calcul utilise un espace au plus  $s(n)$  (sur les rubans de travail).

On note  $NSPACE(s(n))$  l'ensemble des langages reconnus par une machine de Turing non-déterministe en espace  $O(s(n))$ .

**Définition 5.** On définit  $NL := NSPACE(\log n)$ .

On a prouvé précédemment que  $DSPACE(s(n)) \subseteq DTIME(2^{s(n)})$  (lorsque l'on a que  $s(n) \geq \log n$ ). On peut améliorer cette inclusion avec  $NSPACE(s(n))$ , comme  $DSPACE(s(n)) \subseteq NSPACE(s(n))$ .

---

3. Pour une certaine machine  $M$

**Proposition 3.** On a

$$\text{NSPACE}(s(n)) \subseteq \text{DTIME}(2^{O(s(n))}),$$

pour  $s(n) \geq \log n$ .

**Preuve.** On considère  $G_x$  le *graphe* (orienté) *des configurations* de  $M$  sur l'entrée  $x$  de taille  $n$ , où  $c_1 c_2 \in E(G_x)$  ssi  $M$  peut passer de la configuration  $c_1$  à la configuration  $c_2$  en un seul pas de calcul.

**Lemme 4.** Le graphe  $G_x$  a  $2^{O(s(n))}$  sommets, et on peut le construire en espace  $O(s(n))$ .  $\square$

On explore  $G_x$  à partir de la configuration initiale  $c_{\text{initiale}}$  et on accepte si on atteint une configuration d'acceptation.  $\square$

**Remarque 3.** On n'a pas utilisé l'hypothèse que  $M$  s'arrête sur toutes ses entrées.

## 4 Formules booléennes quantifiés, PSPACE.

**Définition 6.** On définit

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

**Théorème 3 (Savitch).** On a  $\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$  si  $s(n) \geq \log n$ .<sup>4</sup>

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

**Remarque 4.** On peut aussi définir la classe NPSPACE mais cette classe est égale à PSPACE par Savitch. On a donc

$$L \subseteq NL \subseteq P \subseteq NP \underset{(*)}{\subseteq} PSPACE \underset{(**)}{\subseteq} \overbrace{\bigcup_{k \geq 1} DTIME(2^{n^k})}^{EXPTIME},$$

où

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

On sait, de plus, que  $P \subsetneq EXPTIME$  par la hiérarchie en temps (variante (vue en TD) de la hiérarchie en espace vue avant). Et, que  $L \subsetneq PSPACE$  par le théorème de Savitch et la hiérarchie en espace.

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

**Exemple 2.** Les formules

- ▷  $\forall x \exists y \left( \overbrace{(x \vee y) \wedge (\bar{x} \vee \bar{y})}^{\text{c'est } x \text{ xor } y} \right)$
- ▷  $\exists y \forall x \left( (x \vee y) \wedge (\bar{x} \vee \bar{y}) \right)$

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

**Définition 7.** On définit le problème QBF comme

|     |   |
|-----|---|
| QBF | <p><b>Entrée.</b> Une formule booléenne quantifiée <math>F</math> (close)</p> <p><b>Sortie.</b> Est-ce que <math>F</math> est vraie ?</p> |
|-----|---|

**Proposition 4.** On a  $QBF \in PSPACE$ .



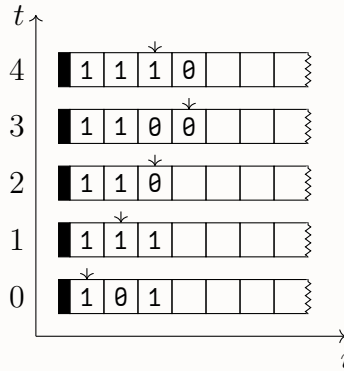
**Preuve.** On utilise l'algorithme suivant.

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.  $\square$

**Théorème 4.** 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  $A \leq_P \text{QBF}$ .



**Figure 1** | Exemple de diagramme espace-temps

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  $\phi_t(c_1, c_2)$  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  $\phi_{c \cdot n^k}(c_{\text{initiale}}, c_{\text{finale}})$ .<sup>5</sup>

La formule  $\phi_0(c_1, c_2)$  est de taille polynomiale (c.f. preuve du théorème de Cook).

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

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

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  $c, c_3, c_4$  ne sont pas des variables mais des  $n$ -uplets de taille  $O(n^k)$  variables booléennes. La taille de  $\phi_{t+1}$  augmente de  $O(n^k)$  par rapport à la taille de  $\phi_t$ . Au final, on est de taille  $O(n^{2k})$ .<sup>6</sup>  $\square$

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

## 5 Problème **PATH** et théorème de Savitch.

**Corollaire 4.** On a  $\text{NL} \subseteq \text{DSPACE}(\log^2 n)$ .

**Proposition 5.** On a que  $\text{PATH} \in \text{DSPACE}(\log^2 n)$ , où

|               |  |
|---------------|--|
| $\text{PATH}$ | <p><b>Entrée.</b> Un graphe <math>G</math> orienté, et <math>s, t \in V(G)</math></p> <p><b>Sortie.</b> Existe-t-il un chemin de <math>s</math> à <math>t</math> dans <math>G</math> ?</p> |
|---------------|--|

5. 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.

6. Le passage quadratique est similaire au théorème de Savitch.

**Remarque 5.** En TD, on a vu que **PATH** est **NL**-complet, et donc la proposition implique le corollaire. En effet, soit  $A \in \mathbf{NL}$  et  $x \in \{0, 1\}^n$ , on a

$$x \in A \iff f(x) \in \mathbf{PATH},$$

où  $f : A \leq_L \mathbf{PATH}$  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  $f(x) \in \mathbf{PATH}$  ou non peut se faire en espace  $O(\log^2 |f(x)|)$ , or  $|f(x)|$  est polynomial en  $|x| = n$ , d'où la borne annoncée.

**Preuve (de la proposition).** On donne un algorithme  $\text{path}(G, u, v, i)$  en espace  $O(\log^2 n)$  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  $\text{path}(G, s, t, \lceil \log n \rceil)$ .

```

1 : Procédure  $\text{path}(G, u, v, i)$ 
2 :   si  $i = 0$  alors
3 :     si  $uv \in E(G)$  ou  $u = v$  alors Accepter
4 :     sinon Rejeter
5 :   pour tout sommet  $w \in V(G)$  faire
6 :     si  $\text{path}(G, u, w, i - 1)$  et  $\text{path}(G, w, v, i - 1)$  alors
7 :       Accepter
8 :   Rejeter
```

Pour la correction, on montre si  $\text{path}(G, u, v, i)$  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  $\text{path}(G, u, v, i)$  accepte, car il suffit de choisir  $w$  comme sommet milieu du chemin.

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

**Preuve (du théorème de Savitch).** Supposons que  $A$  peut être résolu par une machine non-déterministe  $M$  en espace  $O(s(n))$  où  $s(n)$  est 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 5.** Le graphe  $G_x$  a  $2^{O(s(n))}$  sommets et peut être construit en espace  $O(s(n))$ .  $\square$

On a que

$$x \in A \quad \Longleftrightarrow \quad (G_x, s, t) \in \text{PATH},$$

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  $O(s(n))$ . Décider si  $(G_x, s, t) \in \text{PATH}$  se fait en espace  $O(\log^2 |G_x|)$  donc  $O(s(n)^2)$ .  $\square$

**Remarque 6.** La preuve précédente utilise deux résultats

▷ le théorème de composition *amélioré* :

**Théorème 5 (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  $O(s_g(|x|) + s_f(|g(x)|))$ .  $\square$

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

Pour cela, on essaye  $s(n) = 1, 2, \dots$  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 7.** Le problème **PATH** dans les graphes non-orientés est dans **L** ! C'est un résultat récent (2005).

La prochaine fois, on s'intéresse aux oracles.