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 [1] 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 $\mathrm{DSPACE}(s(n))$ l'ensemble des langages reconnus par une machine de Turing déterministe fonctionnant en espace $\mathrm{O}(s(n))$.

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

Proposition 1. On a l'inclusion

$$\mathrm{NTIME}(f(n)) \subseteq \mathrm{DSPACE}(f(n)).$$

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

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

Calculer $f(n)$ en espace $\mathrm{O}(f(n))$
pour tout y de taille $c\, f(n)$ **faire**
    Simuler M sur x en suivant les choix donnés par y
    si la simulation accepte **alors**
        Accepter
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 $t = 1, 2, \ldots$ 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

$$\mathrm{DSPACE}(s(n)) \subseteq \mathrm{DTIME}(2^{\mathrm{O}(s(n))}),$$

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 $N = 2^{\mathrm{O}(s(n))}$. On compte le nombre de configurations distinctes possibles d'une machine M sur l'entrée x de taille n. Une configuration est donnée[2] 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 $\mathrm{DSPACE}(1) \subseteq \mathrm{DTIME}(1)$? Non ! En effet, retourner le dernier caractère de l'entrée se fait en espace $\mathrm{O}(1)$ mais pas en temps $\mathrm{O}(1)$.

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

Corollaire 1. On a les inclusions

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

où PSPACE est l'ensemble des problèmes définis en espace polynomial (défini plus tard). □

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).

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

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 $\mathrm{O}(s(|x|))$.
>
> **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 $f(x)$. □

On utilise ensuite l'algorithme suivant :

$i \leftarrow 0$
tant que M_g ne s'est pas arrêtée **faire**
  Calculer le i-ème bit de $f(x)$ (par le lemme)
  L'écrire sur le ruban d'entrée de M_g
  Effectuer un pas de calcul de M_g
  si la tête → sur le ruban d'entrée de M_g **alors**
    $i \leftarrow i + 1$
  sinon si la tête ← sur le ruban d'entrée de M_g **alors**
    $i \leftarrow i - 1$

□

Corollaire 2. Si f et g sont calculables en espace $\mathrm{O}(\log n)$ alors $g \circ f$ est calculable en espace $\mathrm{O}(\log n)$. □

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 $\mathrm{O}(s(n))$ mais pas en espace $\mathrm{o}(s(n))$.

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

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

Lemme 3. Il existe une machine de Turing universelle U qui prend en entrée $\langle M, x \rangle$ avec $x \in \{\mathtt{0}, \mathtt{1}\}^\star$ et M une machine à un ruban de travail et dont l'alphabet est $\{\mathtt{0}, \mathtt{1}, \square, \triangleright\}$, et qui simule M sur l'entrée x en espace $\mathrm{O}(s(n))$ 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).

```
Lire l'entrée w ∈ {0,1}* (notons n sa taille)
Calculer s(n) en espace O(s(n))
Réserver s(n) cases sur le ruban de travail
si w n'est pas de la forme ⟨M⟩10* [3] alors
    Rejeter
Simuler M sur l'entrée w
si on dépasse 2^(2·s(n)) étapes ou s(n) cases mémoires alors
    Rejeter
si M accepte alors Rejeter
sinon Accepter
```

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

De plus, si B est un langage décidable en $\mathrm{o}(s(n))$ par une machine M, alors B est différent du langage de A. En effet, D simule M en espace $\mathrm{o}(s(n))$.

- 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 $2^{\mathrm{o}(s(n))}$ 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 $\langle M \rangle \mathtt{1}\mathtt{0}^{n_2}$, 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 :

$$\mathsf{L} \subsetneq \mathrm{DSPACE}(n) \subsetneq \mathrm{DSPACE}(n^2) \subsetneq \cdots \subsetneq \mathsf{PSPACE}.$$

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

Définition 5. On définit $\mathsf{NL} := \mathrm{NSPACE}(\log n)$.

3. Pour une certaine machine M

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

Proposition 3. On a

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

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 $c_1 c_2 \in \mathrm{E}(G_x)$ 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 $2^{\mathrm{O}(s(n))}$ sommets, et on peut le construire en espace $\mathrm{O}(s(n))$. □

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

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

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

Théorème 3 (Savitch). On a $\mathrm{NSPACE}(s(n)) \subseteq \mathrm{DSPACE}(s(n)^2)$ si $s(n) \geq \log n$. [4]

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

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

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). Et, que $\mathsf{L} \subsetneq \mathsf{PSPACE}$ 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

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

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

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

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. □

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 $A \leq_{\mathrm{P}}$ QBF.

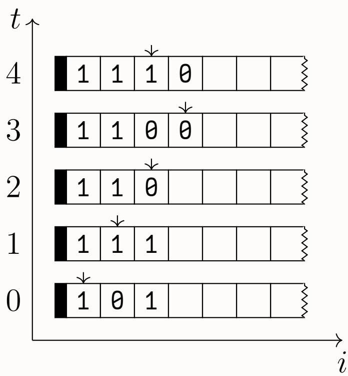

Figure 1 | *Exemple de diagramme espace-temps*

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 $\phi_{c \cdot n^k}(c_{\text{initiale}}, c_{\text{finale}})$.[5]

La formule $\phi_0(c_1, c_2)$ 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 $(*) := \exists c\, \phi_t(c_1, c) \wedge \phi_t(c, c_2)$ qui est correcte mais trop couteuse (taille exponentielle). On choisit plutôt :

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

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 que c, c_3, c_4 ne sont pas des variables mais des n-uplets de taille $\mathrm{O}(n^k)$ variables booléennes. La taille de ϕ_{t+1} augmente de $\mathrm{O}(n^k)$ par rapport à la taille de ϕ_t. Au final, on est de taille $\mathrm{O}(n^{2k})$.[6] □

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 $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^2 n)$.

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

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.

Path	**Entrée.** Un graphe G orienté, et $s, t \in \mathrm{V}(G)$ **Sortie.** Existe-t-il un chemin de s à t dans G ?

Remarque 5. 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 $x \in \{\texttt{0}, \texttt{1}\}^n$, on a

$$x \in A \iff f(x) \in \textsc{Path},$$

où $f : A \leq_{\mathrm{L}} \textsc{Path}$ est la réduction en espace logarithmique (car le problème Path est NL-complet). La construction de $f(x)$ se fait en espace $\mathrm{O}(\log n)$ et décider si $f(x) \in \textsc{Path}$ ou non peut se faire en espace $\mathrm{O}(\log^2 |f(x)|)$, or $|f(x)|$ est polynomial en $|x| = n$, d'où la borne annoncée.

Preuve (de la proposition). On donne un algorithme $\mathsf{path}(G, u, v, i)$ 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 $\mathsf{path}(G, s, t, \lceil \log n \rceil)$.

```
Procédure path(G, u, v, i)
    si i = 0 alors
        si uv ∈ E(G) ou u = v alors Accepter
        sinon Rejeter
    pour tout sommet w ∈ V(G) faire
        si path(G, u, w, i − 1) et path(G, w, v, i − 1) alors
            Accepter
    Rejeter
```

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 $s(n)$ est constructible en espace. Soit $x \in \{\mathtt{0}, \mathtt{1}\}^n$ 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 $2^{\mathrm{O}(s(n))}$ sommets et peut être construit en espace $\mathrm{O}(s(n))$. □

On a que

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

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 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 $\mathrm{O}(s_g(|x|)) + s_f(|g(x)|)$. □

- ▷ 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 7. Le problème Path dans les graphes non-orientés est dans L ! C'est un résultat récent (2005).