Calculabilité et Complexité (11X008) - Printemps 2025

6. Relations entre classes

Enseignant: Arnaud Casteigts

Assistants: A. Berger, M. De Francesco
Moniteurs: E. Bussod, N. Beghdadi

6.1 Definition des classes (rappel)

Pour rappel, les deux classes de complexité génériques pour l'espace et le temps sont :

- $\mathsf{TIME}(f(n))$: Langages décidables en temps $O(f(n))$.
- $\mathsf{SPACE}(f(n))$: Langages décidables en espace $O(f(n))$.

Cas particuliers très étudiés :

- $\mathsf{LOGSPACE} = \mathsf{SPACE}(\log n)$ — Espace logarithmique
- $\mathsf{P} = \mathsf{TIME}(n^{O(1)})$ — Temps polynomial
- $\mathsf{PSPACE} = \mathsf{SPACE}(n^{O(1)})$ — Espace polynomial
- $\mathsf{EXP} = \mathsf{TIME}(2^{n^{O(1)}})$ — Temps exponentiel

6.2 Relations entre l'espace et le temps

La dernière fois, nous avons annoncé les inclusions suivantes :

$$\mathsf{LOGSPACE} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

Nous allons désormais démontrer ces inclusions, l'une après l'autre. Nous considérons ici le cas des machines de Turing à une bande sur l'alphabet $\Sigma = \{0, 1\}$. Les raisonnements utilisés ci-dessous peuvent être adaptées aux autres modèles.

6.2.1 $\mathsf{LOGSPACE} \subseteq \mathsf{P}$

"Tout langage décidable en espace logarithmique est décidable en temps polynomial."

Preuve : Soit L un langage dans $\mathsf{LOGSPACE}$. Par définition, cela implique qu'il existe une constante k et une machine M telle que M décide L en utilisant au plus $k \log n$ cases

mémoires. Comme toute machine de Turing, M a un nombre constant d'états dans son automate (disons k'). Enfin, il y a $k \log n$ positions possibles pour la tête de lecture.

Chaque case mémoire peut avoir 3 valeurs possibles [1] : 0, 1 et □. Le contenu de la mémoire a donc $3^{k \log n}$ valeurs possibles. En tout, cela donne :

$$k' \cdot 3^{k \log n} \cdot k \log n < 4^{k \log n} = 2^{2k \log n} = n^{2k} = n^{O(1)}$$

valeurs possibles, ce qui est polynomial.

Si le temps d'exécution de M dépasse ce nombre, alors la machine passera au moins deux fois par la même configuration, et par conséquent, elle bouclera à l'infini, ce qui contredit le fait que M décide L. Son temps d'exécution doit donc être également polynomial.

6.2.2 PSPACE ⊆ EXP

"Tout langage décidable en espace polynomial est décidable en temps exponentiel."

Preuve : C'est exactement le même argument. Soit $L \in \mathsf{PSPACE}$. Il existe donc une machine M et une constante k telle que M décide L en utilisant au plus n^k cases mémoires. Le nombre de configurations possibles de la machine est donc de :

$$k' \cdot 3^{n^k} \cdot n^k < 4^{n^k} = 2^{2n^k} < 2^{n^{k+1}} = 2^{n^{O(1)}}.$$

Si le temps dépasse ce nombre, alors la machine doit boucler, ce qui contredit le fait qu'elle décide L.

6.2.3 TIME($f(n)$) ⊆ SPACE($f(n)$) et donc P ⊆ PSPACE

"Tout langage décidable en temps $O(f(n))$ est décidable en espace $O(f(n))$."

Preuve : C'est évident pour n'importe quelle fonction $f(n)$. Si le temps exécution d'une machine est au plus de $f(n)$, alors cette machine ne peut utiliser au maximum que $f(n)$ cellules mémoire (chaque accès prend au moins une unité de temps).

Breaking news ! Le 25 février 2025, le chercheur Ryan Williams a publié un article [2], qui démontre que $\mathsf{TIME}(f(n)) \subseteq \mathsf{SPACE}(\sqrt{(f(n) \log f(n))})$. Par exemple, cela implique que $\mathsf{TIME}(n^2) \subseteq \mathsf{SPACE}(n \log n)$. Un progrès majeur sur une question vieille de 50 ans !

1. En cours, j'ai supposé deux valeurs seulement, en oubliant les cases vides. Le calcul est donc un peu plus complexe, mais aboutit à la même conclusion.
2. Ryan Williams, Simulating Time With Square-Root Space, `https://arxiv.org/abs/2502.17779`

6.2.4 Bilan

C'est peut-être surprenant, mais en l'état des connaissances actuelles, on ne peut pas dire grand chose de plus. Il est théoriquement possible que $\mathsf{LOGSPACE} = \mathsf{P}$, ou que $\mathsf{P} = \mathsf{PSPACE}$, ou encore $\mathsf{PSPACE} = \mathsf{EXP}$, ce sont là des questions ouvertes. La majorité des chercheurs du domaine pensent que $\mathsf{LOGSPACE} \subsetneq \mathsf{P} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{EXP}$. C'est juste qu'on n'arrive pas à le démontrer ! Dans tous les cas, le théorème de Ryan Williams montre que dans un sens précis, l'espace est strictement "plus fort" que le temps. C'est assez logique : l'espace est *recyclable*, le temps ne l'est pas. Cela n'implique pas que $\mathsf{P} \neq \mathsf{PSPACE}$, car son théorème ne montre qu'une différence polynomiale entre les deux.

6.3 Théorèmes de hiérarchie

Ce pourrait-il, en théorie, que toutes ces classes soient égales ? Non, on sait quand même que $\mathsf{P} \neq \mathsf{EXP}$, et aussi que $\mathsf{LOGSPACE} \neq \mathsf{PSPACE}$. Plus précisément, il existe deux théorèmes généraux sur le temps d'un côté, et l'espace de l'autre.

6.3.1 Hiérarchie en temps

"Avec plus de temps, on peut résoudre plus de problèmes."

Cela peut sembler évident, encore faut-il le démontrer. Concrètement, le théorème établit que pour presque toute fonction $f(n)$, [3]

$$\mathsf{TIME}(o(f(n))) \subsetneq \mathsf{TIME}(f(n) \log f(n)) \qquad \text{(Hartmanis \& Stearns, 1965)}$$

La démonstration de ce théorème est assez technique. Nous allons l'illustrer sur un cas particulier en montrant que $\mathsf{TIME}(n) \subsetneq \mathsf{TIME}(n^2)$. Pour démontrer cela, il suffit de trouver un langage qui est dans $\mathsf{TIME}(n^2)$ mais qui n'est pas dans $\mathsf{TIME}(n)$. Choisissons un temps intermédiaire, par exemple $n^{1.9}$, et intéressons-nous au problème de décider, étant donnés une machine $\langle M \rangle$ et une entrée w, si M accepte w en moins de $n^{1.9}$ étapes :

$$D = \{(\langle M \rangle, w) \mid M(w) \text{ accepte en moins de } n^{1.9} \text{ étapes}\}$$

Nous allons montrer que $D \in \mathsf{TIME}(n^2)$ en utilisant un argument de *simulation*, et $D \notin \mathsf{TIME}(n)$ en utilisant un argument proche de celui du problème de l'arrêt, mais en temps borné.

3. Le théorème ne marche pas pour certaines valeurs extrêmes de $f(n)$, par exemple si $f(n) = \log(n)$. Mais il marche pour la majorité des fonctions habituelles, notamment n, n^2 ou encore 2^n.

1. $D \in \mathsf{TIME}(n^2)$: On peut créer une machine qui simule $M(w)$ pendant $n^{1.9}$ étapes de calcul (notez que $n = |\langle M \rangle| + |w|$, cette information est facile à récupérer). Si $M(w)$ accepte avant ce délai, on accepte. Sinon, on rejette. Il est connu que la simulation d'une machine de Turing n'est ralentie que d'un facteur logarithmique. Cette simulation prendra donc un temps $n^{1.9} \log n$, ce qui est bien dans $\mathsf{TIME}(n^2)$.
2. $D \notin \mathsf{TIME}(n)$: La preuve est similaire à celle de Turing. Supposons qu'il existe une machine M' qui s'exécute en temps $O(n)$ et qui est capable de décider si une machine M accepte une entrée w en moins de $n^{1.9}$ étapes (décider D). On peut alors utiliser M' pour tester comme cas particulier si M accepte sa description $\langle M \rangle$ en moins de $n^{1.9}$ étapes. On va faire cela dans une autre machine N qui prend en entrée une machine $\langle M \rangle$ et renvoit l'opposé de ce que lui dira M' (c.à.d. N accepte si M' rejette et N rejette si M' accepte). Maintenant, que se passe-t-il si l'on appelle $N(\langle N \rangle)$? Et bien si N accepte $\langle N \rangle$ en moins de $n^{1.9}$ étapes, cela implique que $M'(\langle N \rangle, \langle N \rangle)$ a rejeté, mais cela implique à son tour que N n'accepte pas $\langle N \rangle$ en moins de $n^{1.9}$ étapes, ce qui est une contradiction. (On a une contradiction similaire dans l'autre cas.) En résumé, M' ne peut pas toujours décider correctement le langage D.

J'ai légèrement triché sur la second preuve, en considérant la même variable n dans tous les cas alors que la taille de l'entrée varie selon les machines utlisées. C'est donc légèrement incorrect, mais l'idée est plus simple à comprendre ainsi. Vous me pardonnerez.

6.3.2 Hiérarchie en space

Un théorème similaire existe pour l'espace. En fait, il est même plus fin :

$$\mathsf{SPACE}(o(f(n))) \subsetneq \mathsf{SPACE}(f(n))$$

L'idée de la preuve est la même, sauf que la simulation d'une machine de Turing a un surcoût en espace qui est seulement additif (et non multiplicatif, comme pour le temps), on peut donc se débarrasser du facteur logarithmique dans la partie droite du résultat.

6.4 Pourquoi P est la plus importante

La classe la plus étudiée est certainement P, l'ensemble des langages décidables en temps polynomial $n^{O(1)}$. La raison est que cette classe capture, d'un point de vue théorique, les problèmes qui peuvent être résolus efficacement (rapidement). Bien sûr, c'est sujet à discussion. On peut s'interroger, par exemple, sur l'efficacité réelle d'un algorithme qui prendrait un temps n^{10}. Pour certaines applications où n est grand, même une complexité de n^2 peut s'avérer trop lent. Cependant, la classe P a de nombreux avantages, parmi lesquels :

- Abstraction : La classe P est la même pour tous les modèles de machine "raisonnables" (machine de Turing, machine RAM, modèles biologiques, *etc.*). La raison est que n'importe lequel de ces modèles peut simuler les autres en temps polynomial.
- Composabilité : Si un algorithme fait appel un nombre de fois polynomial à un autre algorithme qui s'exécute en temps polynomial, alors l'ensemble reste polynomial.
- Réalisme : La grande majorité des problèmes concrets qui sont dans P s'avèrent avoir de petits exposants (souvent $O(n)$ ou $O(n^2)$, parfois $O(n^3)$). On peut donc estimer que pour des tailles de problèmes raisonnables, la classe P capture bel et bien ce qui peut être résolu efficacement.

6.5 Quelques exemples

Voici une liste de problèmes. Pour chacun, nous indiquons la plus petite des 4 classes principales à laquelle on sait qu'il appartient (potentiellement non-définitif).

Entrée	Question	Classe
Liste d'éléments	Est-elle triée ?	LOGSPACE
Un graphe	Est-il connexe ?	LOGSPACE
Un graphe et un chemin de A et B	Ce chemin est-il le plus court de A vers B	P
	Ce chemin est-il le plus long de A vers B	PSPACE
Une matrice	Est-elle inversible ?	P
Un graphe	Est-il 2-colorable ?	LOGSPACE
Un graphe	Est-il 3-colorable ?	PSPACE
Un entier	Est-il premier ?	P
Un graphe	Existe-t-il un chemin qui visite exactement une fois chaque sommet et termine à son point de départ ?	PSPACE
Plateau d'échec $n \times n$ partiellement joué.	Existe-t-il une stratégie gagnante pour le joueur 1 ?	EXP
Entiers N et k	N admet-il un facteur $\leq k$?	PSPACE
Entiers N et k *(en unaire)*	N admet-il un facteur $\leq k$?	P

Nous aurons l'occasion de revenir sur certains de ces problèmes.

Subtilité : n désigne la taille de l'entrée. Lorsque l'entrée est une liste de N éléments, où chaque élément est de taille constante, alors il n'y a pas d'ambiguité, on a bien $n = O(N)$. Par contre, si l'entrée d'un problème est un nombre N, sa taille n'est pas $O(N)$. Par exemple, la taille de 999 en décimal est 3 (et non 999). En binaire, $999 = 1111100111$, donc de taille 10. Quelle que soit la base utilisée (sauf en *unaire*), la taille est $n = O(\log N)$. Du coup, un algorithme de complexité $O(N^3)$ coûte en réalité $O(2^{3n})$. À garder en tête...