Calculabilité et Complexité (11X008) - Printemps 2024

8. Inclusions connues; non-déterminisme; théorème de Savitch

Enseignant: Arnaud Casteigts *Exercices: A. Berger, M. De Francesco, L. Heiniger*

8.1 Petit rappels et précisions

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

- $\mathsf{TIME}(f(n))$: Langages qui peuvent être décidés en temps $O(f(n))$, sans se soucier de l'espace.
- $\mathsf{SPACE}(f(n))$: Langages qui peuvent être décidés en espace $O(f(n))$, sans se soucier du temps.

Notez que ces classes concernent les machines/algorithmes déterministes. D'ailleurs, elles sont aussi appelées DTIME et DSPACE pour cette raison. Nous définirons plus bas des classes non-déterministes analogues.

Un autre aspect important est que le paramètre 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 $n = O(N)$. Mais attention : si par exemple l'entrée d'un problème est un grand nombre N, comme pour le problème de la factorisation, alors sa taille n'est pas $O(N)$, car ce nombre est typiquement représenté en binaire (par exemple, le nombre 999 s'écrit 1111100111, sa taille est donc 10 et non 999). En fait, quelle que soit la base utilisée (sauf en unaire), la représentation d'un nombre a une taille $n = O(\log N)$. En l'occurrence, pour la factorisation, il existe plein d'algorithmes simples qui seraient polynomiaux en N, mais qui ne sont pas polynomiaux en $n = O(\log N)$. Par exemple, un algorithme de complexité $O(N^3)$ coûterait en réalité $O(2^{n^3})$.

Il arrive parfois qu'on utilise la lettre n pour autre chose que la taille de l'entrée (p.ex. pour des algorithmes de graphes, où n désigne le nombre de sommets). Nous le redirons explicitement quand le cas se produit.

8.2 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. Pour simplifier, considérons le cas des machines de Turing sur l'alphabet $\Sigma = \{0, 1\}$.

8.2.1 LOGSPACE $\subseteq$ P

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

Preuve : Soit L un langage dans LOGSPACE. Par définition, cela implique qu'il existe une constante k et une machine M telle que M décide L et la taille de la mémoire de L est bornée par $k \log n$. Par ailleurs, comme toute machine de Turing, M a un nombre constant d'états dans son automate (disons k') et $k \log n$ positions possibles pour la tête de lecture.

Au maximum, cette machine peut donc se trouver dans $k' \cdot 2^{k \log n} \cdot k \log n$ configurations possibles (0 ou 1 pour chaque cellule mémoire, fois le nombre d'états k', fois toutes les positions possibles pour la tête de lecture). Si le temps d'exécution de M dépasse ce nombre, alors la machine passera au moins deux fois par la même configuration, et nécessairement, elle le fera à l'infini. Son temps d'exécution est donc inférieur au nombre de configurations possibles, qui après un petit calcul (notamment, $2^{k \log n} = n^k$), s'avère être en $n^{O(1)}$.

8.2.2 PSPACE $\subseteq$ EXP

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

Preuve : C'est exactement le même argument, en utilisant une mémoire de taille bornée par n^k pour un certain k. Cela donne un nombre de configuration maximum (et donc un temps d'exécution maximum) de $2^{n^{O(1)}}$.

8.2.3 P $\subseteq$ PSPACE

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

Preuve : Si le temps exécution d'une machine est borné par n^k, alors cette machine ne peut accéder au maximum qu'à n^k cellules mémoire (chaque lecture/écriture prend au moins une unité de temps).

8.2.4 Discussion

Quelles leçons retirer de ces inégalités ? Tout d'abord, on voit que limiter l'espace semble moins contraignant que limiter le temps. Autrement dit, l'espace est "au moins aussi fort" que le temps, et potentiellement plus fort, mais jusqu'à un certain point seulement.

C'est peut-être surprenant, mais en l'état des connaissances actuelles, on ne peut pas dire grand chose de plus. Par exemple, il est théoriquement possible que $\mathsf{LOGSPACE} = \mathsf{P}$, ou que $\mathsf{P} = \mathsf{PSPACE}$, ou encore $\mathsf{PSPACE} = \mathsf{EXP}$, mais nous n'en savons rien et la majorité des spécialistes pensent que ce n'est pas le cas. En fait, le consensus actuel est que ces quatre classes sont probablement toutes différentes, autrement dit, on pense que :

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

8.3 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.

8.3.1 Hiérarchie en temps

La hiérarchie en temps est un résultat qui démontre qu'avec plus de temps, on peut résoudre plus de problèmes. Cela peut sembler évident, encore fallait-il réussir à le démontrer. Concrètement, c'est un théorème qui établit que pour presque [1] toute fonction $f(n)$,

$$\mathsf{TIME}(o(f(n))) \subsetneq \mathsf{TIME}(f(n) \log f(n))$$

La démonstration de ce théorème est un peu compliquée. Nous allons simplement en donner quelques éléments. Pour simplifier, intéressons-nous au cas particulier $\mathsf{TIME}(n) \subsetneq \mathsf{TIME}(n^2)$. Pour démontrer cela, il suffit de trouver un langage qui est dans $\mathsf{TIME}(n^2)$ mais qui n'est pas dans $\mathsf{TIME}(n)$.

Soit D le langage suivant :

$$D = \{\langle M \rangle \mid M(\langle M \rangle) \text{ termine et rejette en temps } n^{1.9}\}$$

Ici, la taille de l'entrée est $n = |\langle M \rangle|$.

1. $D \in \mathsf{TIME}(n^2)$: On crée une machine M_D qui prend en entrée $\langle M \rangle$ et simule $n^{1.9}$ étapes de calcul de $M(\langle M \rangle)$. Si la simulation termine et rejette, on accepte. Si la simulation accepte ou ne termine pas, on rejette. Il est connu que la simulation d'une machine de Turing n'est ralentie que d'un facteur logarithmique en la taille de sa description et de son entrée (non démontré ici). Cette simulation prendra donc un temps $n^{1.9} \log n = O(n^2)$.

1. Le théorème ne marche pas pour certaines valeurs extrêmes de $f(n)$, par exemple si $f(n) = o(n)$. Techniquement, la fonction doit être "time-constructible", ce qui inclut la majorité des fonctions habituelles, notamment n, n^2, n^k pour tout k, ou encore 2^n.

2. $D \notin \mathsf{TIME}(n)$: C'est ici que c'est plus compliqué. La preuve procède par l'absurde, en supposant qu'il existe une machine M' qui décide D en temps n. Puis on dérive une contradiction sur l'exécution de $M'(\langle M' \rangle)$ via un argument de diagonalisation similaire à ceux que l'on connait (en plus subtile, car l'argument ne s'applique pas à toute la diagonale, seulement aux parties concernant les machines terminant rapidement).

8.3.2 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 exactement la même. La seule différence est 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 débarasser du facteur logarithmique dans la partie droite du résultat.

8.4 Non-déterminisme

De la même manière que pour les machines déterministes, on peut définir deux classes génériques pour l'espace et le temps comme suit :

- $\mathsf{NTIME}(f(n))$: Langages qui peuvent être décidés en temps $O(f(n))$ par une machine de Turing *non-déterministe* (sans se soucier de l'espace).
- $\mathsf{NSPACE}(f(n))$: Langages qui peuvent être décidés en espace $O(f(n))$ par une machine de Turing *non-déterministe* (sans se soucier du temps).

On définit de manière analogue les cas particuliers NLOGSPACE, NP, NPSPACE, et NEXP.

Les machines déterministes étant un cas particulier des machines non-déterministes, nous avons directement les inclusions suivantes pour tout $f(n)$:

- $\mathsf{TIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$
- $\mathsf{SPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$

Par ailleurs, il existe des théorèmes de hiérarchie entre classes non-déterministes similaires à ceux vus précédemment, avec des écarts de valeurs légèrement différents (contrairement à ce que j'ai affirmé en classe).

Pour l'espace, il existe un autre théorème très important, le **théorème de Savitch**, qui établit que pour toute $f(n) \geq n$, $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$. Autrement dit, tout

langage décidable par une machine M non-déterministe peut être décidé par une machine déterministe n'utilisant "que" le carré de l'espace de M. On a donc un encadrement assez précis de leur relation :

$$\mathsf{SPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$$

Ces inclusions impliquent, entre autres, que $\mathsf{PSPACE} = \mathsf{NPSPACE}$ (espace polynomial déterministe = espace polynomial non-déterministe). En effet, un surcoût quadratique à une quantité polynomiale reste une quantité polynomiale.

L'impact du non-déterminisme sur le temps est moins bien compris. On suspecte le non-déterminisme d'être exponentiellement plus rapide que le déterminisme, mais on ne sait pas le démontrer. En fait, ce sujet encapsule certaines des questions les plus profondes en informatique (et bien au delà). Nous y consacrerons au moins les trois prochaines séances, en discutant notamment des relations entre P (temps polynomial, déterministe) et NP (temps polynomial, non-déterministe).

Mais avant ça, voyons un aperçu du théorème de Savitch, dont la démonstration est élégante et n'utilise pas de diagonalisation (ouf !).

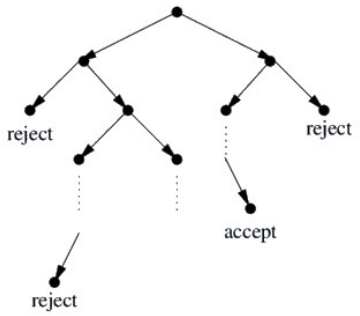
8.4.1 Théorème de Savitch

Pour tout $f(n) \geq n$,

$$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$$

Supposons qu'un langage L est dans $\mathsf{NSPACE}(f(n))$, autrement dit, il existe une machine de Turing non-déterministe M_N qui décide L en espace $O(f(n))$. Le but est de montrer qu'une telle machine peut être simulée (dans un sens un peu spécial) par une machine déterministe M_D en espace $O(f(n)^2)$.

Avant d'attaquer, réfléchissons à l'exécution de la machine M_N. Cette dernière va explorer (non-déterministiquement) tout un ensemble de configurations, en effectuant des choix non-déterministes à plusieurs reprises, ce qui peut être représenté par un arbre d'exécution :



Cette machine va accepter si et seulement si au moins un chemin existe depuis la configuration de départ vers une configuration acceptante. L'idée principale est qu'à partir de la description d'une machine non-déterministe, une machine déterministe peut trouver un tel chemin, s'il existe, en utilisant peu d'espace (contre potentiellement beaucoup de temps, mais ce n'est pas grave).

Notez que le nombre de sommets dans cet arbre ne peut pas dépasser le nombre de configurations possibles, disons $2^{f(n)}$ pour faire simple. Pour faire simple aussi, on peut modifier légèrement M_N de sorte qu'il n'y ait qu'une seule configuration acceptante c_{accept} dont on peut connaître la description à l'avance. Enfin, on peut supposer que l'on connait $f(n)$. (Toutes ces simplifications sont OK, il s'agit de détails techniques surmontables.)

Option 1 : Effectuer un parcours en profondeur de cet arbre jusqu'à trouver c_{accept}. Même si l'arbre n'existe pas réellement, c'est tout à fait possible. Il suffit, à chaque étape intermédiaire, d'énumérer les choix possibles, en choisir un en mémorisant la configuration actuelle et le choix effectué, pour choisir le suivant lors du prochain passage du parcours. Le problème est que la profondeur de l'arbre peut être de l'ordre de $2^{f(n)}$, trop cher en espace.

Option 2 : Une super astuce. Si un chemin existe entre la configuration de départ c_{start} et la configuration acceptante c_{accept}, alors ce chemin doit avoir une longueur $\leq 2^{f(n)}$. Il existe donc forcément une configuration intermédiaire c_{middle} telle que la longueur de c_{start} à c_{middle} est inférieure à $\lfloor 2^{f(n)}/2 \rfloor$ et la longueur de c_{middle} à c_{accept} est inférieure à $\lceil 2^{f(n)}/2 \rceil$. Récursivement, il existe une configuration intermédiaire entre c_{start} et c_{middle} telle que la longueur entre c_{start} et cette configuration est de $\lfloor 2^{f(n)}/4 \rfloor$, et ainsi de suite jusqu'à ce que la longueur soit de 1, ce qui est facile à vérifier en regardant $\langle M \rangle$. Mais comment deviner ces configurations intermédiaires ? L'idée est de les trouver en énumérant toutes les configurations possibles et en testant récursivement si cela peut aboutir à un chemin valide. Puisque les distances sont divisées par deux à chaque étape de la récursion, vous ne mémorisez à un instant donné que $\log 2^{f(n)} = f(n)$ configurations et pouvez réutiliser l'espace des configurations qui ne servent plus. Chaque configuration étant de taille $f(n)$, cela donne bien $f(n)^2$ espace.

N'hésitez pas à chercher le théorème de Savitch sur internet pour appronfondir.

8.5 Morale de l'histoire

Intuitivement, le grand avantage de l'espace est qu'il est réutilisable, tandis que le temps ne l'est pas ! Cet avantage semble compenser en partie l'avantage du non-déterminisme, rendant l'écart entre déterminisme et non-déterminisme moins prononcé pour l'espace que pour le temps (du moins, semble-t-il).