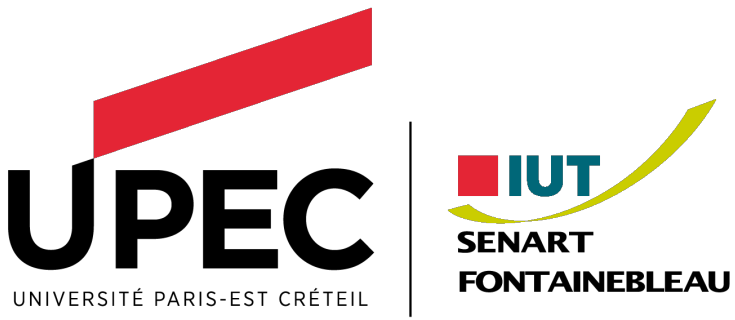
Fondements de l’informatique

Complexité

Florent Madelaine

Fondements de l’informatique



Plan

Introduction

On a vu divers modèles : automate, automate à pile, Machine de Turing.

Le dernier, celui de Turing étant aussi général que nécessaire et permettant de capturer la notion de ce qui est calculable (thèse de Church Turing).

Il y a toutefois des exemples de problèmes qui ne sont pas calculables, comme le problème de l'arrêt.

Bad news

Même parmi les choses calculables, il y a des programmes sans intérêt pratique car pas efficace à l'échelle humaine.

La complexité se penche sur la comparaison des programmes calculables en terme de ressources.

Ressources

Quand on exécute un algorithme, deux ressources ont un sens particulièrement important : le temps d'exécution (souvent évalué approximativement à partir de primitives utilisées comme accéder à une valeur dans un tableau) et l'espace utilisé (stockage de calculs intermédiaires).

Sur notre modèle de Machine de Turing à un ruban, on peut étudier la notion de temps facilement : c'est simplement le nombre d'étapes de calcul.

Pour l'espace, on utilise le plus souvent au moins trois rubans avec un modèle plus contraint : le ruban d'entrée (lecture uniquement), le ruban de sortie (écriture seule), et le ruban de travail. L'espace utilisé par la machine est alors vu comme le nombre de cellules du ruban de travail qui sont utilisées au cours de calcul.

Mesurer le temps

On se concentre sur le temps. On va regarder le *pire des cas*
On dit que la machine de Turing calcule en temps $f(n)$ (pour une fonction $f : \mathbb{N} \to \mathbb{N}$) si pour toutes les entrées de taille n, le calcul s'arrête en au plus $f(n)$ étapes.

En pratique on choisira pour f une fonction « raisonnable » comme n^2 ou 2^n. On travaille aussi le plus souvent sans tenir compte des constantes multiplicatives et on utilise la notation « grand O ».

Notation Bachmann–Landau

- $f(n) = O(g(n))$ dénote qu'il existe des entiers positifs c et N tels que pour tout n plus grand que N, $f(n) \leqslant c.g(n)$.
- $g(n) = \Omega(f(n))$ pour $f(n) = O(g(n))$
- $g(n) = \Theta(f(n))$ pour $f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$

Pourquoi ? phénomène d'accélération linéaire

Théorème

Pour toute machine de Turing M, calculant en temps $f(n)$ et toute constante $0 < c < 1$, il existe une machine de Turing M' équivalente en temps $2 + n + c.f(n)$.

Idée de la preuve

La machine M' a des « grosses »cellules (un alphabet augmenté de symboles qui correspondent à la concaténation de quelque chose comme $\lceil \frac{6}{c} \rceil$ lettres de l'alphabet original). Initialement la machine va zipper l'entrée (un bloc de cellules deviennent un seul symbole). Ceci a un coût linéaire puisqu'il faut au moins lire l'entrée. Ensuite la machine M' simule c étapes de M en un nombre borné d'étapes (6 étapes), d'où le gain linéaire.

Bien choisir une fonction pour mesurer

Définition

On note $\mathtt{TIME}(f(n))$ la classe des problèmes décidables par une machine de Turing déterministe en temps $f(n)$.

Pour mesurer le temps, on ne peut pas prendre n'importe quelle fonction.

Théorème (Borodin 1969 - Trakhtenbrot 1967)

Il existe une fonction calculable f telle que $\mathtt{TIME}(f(n)) = \mathtt{TIME}(2^{f(n)})$

La fonction du théorème ci-dessus est particulièrement non standard. En pratique, les fonctions discrètes usuelles analogues de celles des cours de math de lycée ne posent pas de problèmes. *Grosso Modo* Ce sont celles pour lesquelles on peut écrire une machine de Turing qui sert d'horloge pour compter le bon nombre d'étapes.

Quelle machine ?

Il existe une dépendance polynomiale entre divers modèles de calcul

Théorème

Pour toute machine à k rubans en temps $f(n)$ il existe une machine de Turing équivalente à 1 ruban et en temps de l'ordre de $k^2 f(n) \times f(n)$.

Pour le modèle de machine à registre RAM (modèle théorique le plus proche d'un ordinateur), il existe un résultat similaire avec à l'exposant 7 plutôt que 2.

Temps non déterministe

On a vu :

Définition

On note $\mathtt{TIME}(f(n))$ la classe des problèmes décidables par une machine de Turing déterministe en temps $f(n)$.

On ajoute :

Définition

On note $\mathtt{NTIME}(f(n))$ la classe des problèmes décidables par une machine de Turing non-déterministe en temps $f(n)$. (Attention : absence de symétrie mais dualité entre acceptation $\exists$ et rejet $\forall$)

Deux classes de complexité robustes

Calcul en temps Polynomial ≈ calcul efficace

$$\mathtt{PTime} = \bigcup_{c \geqslant 1} \mathtt{Time}(n^c)$$

(la machine de Turing est déterministe)

Vérification en temps Polynomial ≈ calcul efficace

$$\mathtt{NP} = \bigcup_{c \geqslant 1} \mathtt{NTime}(n^c)$$

(la machine de Turing peut être non-déterministe)

Exemples

Calcul polynomial

- Graphe Eulérien ?
- Graphe 2-colorable ?
- calculer la valeur d'un circuit étant donné les valeurs des entrées.
- etc

Vérification polynomiale

- savoir si un graphe est Hamiltonien
- Graphe 3-colorable ?
- SAT
- etc

Un petit soucis

Si je trouve un algorithme polynomial, super je suis content.

Que faire si je réfléchis très fort et que je ne trouve pas d'algorithme polynomial pour mon problème favori ? En particulier quand ce problème est dans NP.

Exemple

Je voudrais savoir si on peut tester si un graphe est 3 colorable en temps polynomial. Après tout j'ai trouvé un truc pour les graphes 2-colorables.

Problème SAT

Problème

- Entrée : formule propositionnelle (CNF)
- Question : satisfaisable ?

Exemple

Une entrée ressemble à ceci :

$$(x \vee y \vee z) \wedge (x \vee \neg t \vee \neg u \vee v) \ldots$$

On doit trouver des valeurs vrai ou faux pour chaque variable de sorte que chaque clause soit vraie (au moins un littéral par clause doit être vrai).

Notation. $\wedge$ et, $\vee$ ou, $\neg$ négation.

Vers la notion de réduction

Solveur SAT

Logiciel permettant de résoudre le problème SAT.
Typiquement capable de résoudre des instances à plusieurs millions de variables.

Modéliser

On prend l'entrée d'un problème et on fabrique une formule propositionnelle. La réponse du solveur SAT permet de résoudre le problème initial.

Exemple

Essayons avec le problème de 3 colorabilité.

Modéliser = Réduire

Définition

Soit deux problèmes de décisions Ω_1 et Ω_2. Une réduction de Ω_1 à Ω_2 est une fonction calculable en temps polynomial r qui transforme une entrée x_1 du problème Ω_1 en une entrée $x_2 = r(x_1)$ du problème Ω_2 telle que $x_1 \in \Omega_1$ ssi $x_2 \in \Omega_2$.

Propriétés clé

- On peut composer les réductions (transitivité de l'existence d'une réduction).
- Si le problème cible Ω_2 est polynomial alors le problème de départ Ω_1 est lui aussi polynomial.

SAT comme étalon pour NP

Théorème (Cook 1971, Levin 1973)

Tout problème Ω de NP se réduit à SAT.

Idée de la preuve

Ω étant dans NP, il existe une machine non-déterministe T en temps polynomial $p(n)$ (p est un polynôme).
Pour une entrée x de Ω, le calcul sur la machine T occupe à tout instant au plus $p(t)$ cellules et la machine passe par au plus $p(t)$ étapes de calcul. On peut donc décrire par un nombre polynomial de variables booléennes les $p(t)$ configurations successives de la machine.
Par exemple, on a des variables $T_{p,t,s}$ avec $1 \leqslant p, t \leqslant p(n)$ qui seront vrai ssi la cellule à position p au temps t contient le symbole s de l'alphabet.
Ensuite on peut écrire des clauses pour contraindre ces variables à bien modéliser le calcul.
La page wikipedia propose une preuve complète :
`https://en.wikipedia.org/wiki/Cook-Levin_theorem`

D’autres problèmes comme SAT ?

On peut se demander si à l’instar de SAT, il y a d’autres problèmes de NP auxquels tout problème de NP se réduit (en jargon on dira NP-complet).

On pourrait employer la même stratégie que Cook et coder un calcul dans le cadre du problème étudié. En pratique, on va plutôt réduire un problème NP-complet connu (SAT par exemple) au problème étudié. La transitivité des réductions permettant de conclure à la Np-complétude du problème étudié.

P différent de NP et NP-complétude

On ne sait pas si P est véritablement différent de NP, c'est une conjecture en théorie de la complexité.

En pratique il y a maintenant une très large collection de problèmes NP-complets pour lesquels personne ne connaît d'algorithme polynomial.

Retour à notre petit soucis

Si je trouve un algorithme polynomial, super je suis content.

Que faire si je réfléchis très fort et que je ne trouve pas d'algorithme polynomial pour mon problème favori ? En particulier quand ce problème est dans NP ?

Si je montre que le problème est NP-complet, alors je sais que personne ne connais à l'heure actuelle d'algorithme polynomial pour mon problème.

D’autres classes

Quelques classes de complexité usuelles.

- Espace logarithmique, polynomial, exponentiel (déterministe ou pas)
- Temps polynomial, exponentiel (déterministe ou pas)

En dehors des inclusions faciles à voir de ces classes, on ne sait en général pas les séparer.

Il y a beaucoup d’autres classes de complexité. Vous pouvez allez explorer ce zoo ici
`https://complexityzoo.uwaterloo.ca/Petting_Zoo`

Comparer les classes de complexité

Non-déterminisme pas moins puissant que déterminisme

Une machine déterministe est un cas particulier d'une machine non-déterministe $\texttt{Time}(f(n)) \subseteq \texttt{NTime}(f(n))$.

Temps vs Espace

Une machine ne peut pas écrire plus que son temps d'exécution $(\texttt{N})\texttt{Time}(f(n)) \subseteq (\texttt{N})\texttt{Space}(f(n))$.

Se passer du non déterminisme ?

On peut simuler avec du *backtrack* une machine non déterministe en temps par une machine déterministe (pour un coût exponentiel). $\texttt{NTime}(f(n)) \subseteq \texttt{Time}(c^{f(n)})$ (la constante c dépend de la machine initiale mais pas de n).

Quelques théorèmes importants

On peut relativiser la preuve de diagonalisation de l'arrêt sur des familles de machines avec horloge ou mètre (ceci nous donne des hiérarchies en temps et en espace).

Théorème (Hiérarchie en temps, Hartmanis et. al. 1965)

Si $f(n)$ *est une fonction superlinéaire raisonnable alors* $\mathtt{Time}(f(n))$ *est strictement contenue dans* $\mathtt{Time}(f(2n+1)^3)$

Théorème (Hiérarchie en espace, Hartmanis et. al. 1965)

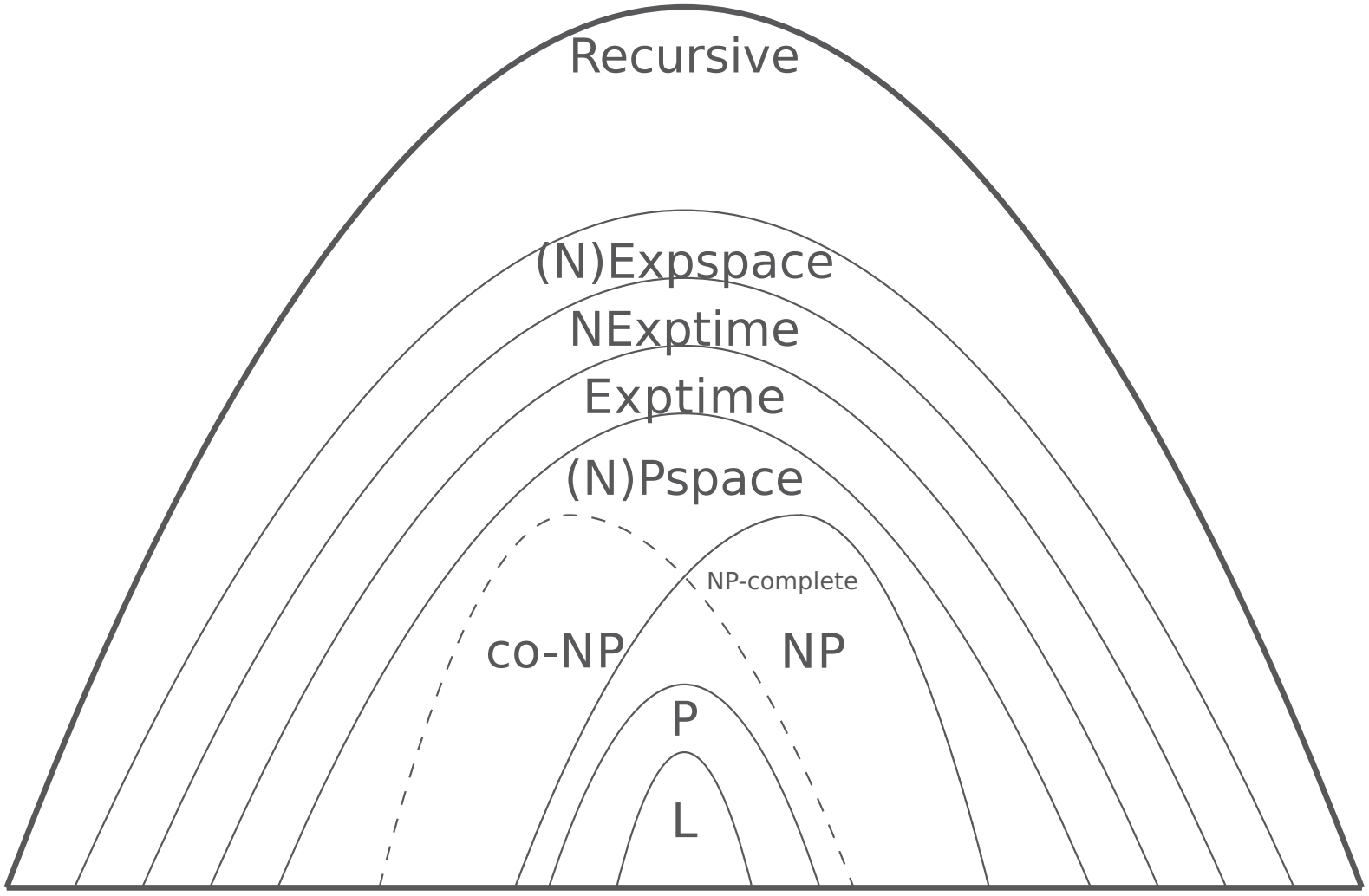
Si $f(n)$ *est une fonction raisonnable alors* $\mathrm{Space}(f(n))$ *est strictement contenue dans* $\mathrm{Space}(f(n)\log(f(n)))$

On peut tester l'existence d'un chemin dans un graphe à n sommets avec un espace $\log n \times \log n$ (Théorème de Savitch).

Corollaire

Si $f(n) \geqslant \log n$ *est une fonction raisonnable alors* $\mathtt{NSpace}(f(n)) \subseteq \mathrm{Space}(f(n) \times f(n)))$.

Classes de complexité



Classes de complexité