DIU Enseigner NSI
Journée Calculabilité et Complexité

Florent Madelaine

5 mars 2020

Table des matières

1 Modèles **1**

2 Calculabilité **3**

3 Complexité **5**

4 Complément complexité **7**

5 Pédagogie **8**

1 Jouer avec les modèles

Que veut-on calculer ? Le cas des fonctions discrètes $f:\mathbb{N}\to\mathbb{N}$ est essentiellement celui qui est étudié.

Le cas des fonctions indicatrices est assez général pour illustrer le domaine avec peu de perte de généralité. Autrement dit on cherche à décider des ensembles d'entiers.

En pratique, comme on va travailler sur des modèles de calcul pour lesquels on pourra mesurer de manière fine les ressources utilisées (temps, espace) on va devoir expliciter comment les entrées sont codées.

Un peu de vocabulaire.

- *Alphabet* : Σ ensemble fini fixé de symboles.
- *Entrée* : un mot comportant un nombre fini de symboles de Σ.
- *Problème de décision (langage)* : ensemble de mots.

Exemple.

- $\Sigma=\{0,1\}$
- Problème des mots qui sont des *palindromes* (qui sont identiques quel que soit le sens de lecture).
- 101101 est un mot du langage des palindromes.
- 100 n'est pas un mot de ce langage.

Autres exemples de langages.

- L'ensemble des mots sur $\{0,1\}$ représentant un entier pair.
- L'ensemble des mots sur 0,1 dont la première et dernière lettre sont identiques.
- L'ensemble des mots sur 0,1 dont le nombre de 0 et de 1 est égal.
- L'ensemble des mots sur $\{0,1\}$ représentant un nombre premier.

Modèles de calcul. Nous allons entrevoir deux modèles de calcul.

- Automates finis
- Machine de Turing

Avec JFLAP, vous pouvez en découvrir d'autres. Il y a un modèle intermédiaire naturel : c'est l'automate à pile (*push down automata*). En gros, c'est le modèle qui correspond bien à ce dont on a besoin pour faire des compilateurs.

Découverte de JFLAP Vous pouvez programmer de petits automates / machine de Turing en reprenant les exemples ci-dessus.

Un autre exemple pertinent avec les automates concerne le calcul modulaire. Vous pouvez par exemple écrire un automate qui décide si un mot composé de 0 et de 1 représente un entier vérifiant certaines équations modulaires (par exemple : c'est un multiple de 3).

Il y a aussi deux activités intéressantes autour des automates que vous pouvez découvrir.

La première concerne la déterminisation des automates.

La seconde concerne le lemme de la pompe (méthode pour montrer qu'un langage n'est pas régulier, c'est-à-dire ne peut pas être décidé par un automate).

2 Limites de ce qu'on peut calculer

La thèse de Church Turing. Pour tout modèle raisonnable de calcul, on obtient la même notion de ce qui est calculable.

Arguments en faveur de cette thèse

- Théorie générales des fonctions récursives (Gödel, Herbrand 1933).
- λ-calcul (Church, 1936).
- Machines de Turing (Turing, 1936).
- Trois modèles équivalents (Church 1936, Turing 1937).
- Machines de Post (1936).
- Machines de Turing avec plusieurs rubans (Minsky).
- Machines à compteurs.
- Machines à registre.
- *etc.*

Définition. Un langage L est *décidable* si il existe une machine de Turing M qui accepte exactement les mots de ce langage, c'est-à-dire que pour tout mot x sur son ruban *la machine M arrête son calcul dans un état acceptant ssi x est dans L.*

Définition. Une fonction f est *calculable* si il existe une machine de Turing M qui étant donné x sur son ruban s'arrête avec comme contenu de ruban $f(x)$.

Un peu de dénombrement. L'ensemble des programmes est dénombrable mais l'ensemble des langages n'est pas dénombrable.

Limite forte de ce qu'on peut calculer. Il existe « beaucoup » de langages pour lesquels nous n'avons pas de programme.

Lemme. *L'ensemble des programmes est dénombrable*

On peut rester informel comme ci-dessus et parler de langage de programmation quelconque (du C par exemple) et via le codage ascii faire correspondre au texte du programme un nombre en binaire. La croyance en la thèse de Church-Turing faisant le reste.

Alternativement si on souhaite être plus précis on peut montrer ceci.

Lemme. *L'ensemble des machines de Turing est dénombrable.*

Démonstration. On peut coder en binaire les états, les symboles de l'alphabet et les actions de déplacement de la tête de lecture.

À l'aide de symboles séparateurs adaptés (parenthèses, virgules), on peut lister les transitions. Par exemple une liste de mots de la forme `(état avant, symbole lu, état après, symbole écrit, déplacement)`.

On peut ensuite reprendre l'argument précédent avec le code ascii. □

Digression : code d'une machine de Turing. On va appeler *code d'une machine de Turing* le mot suivant :
`nombre d'états en binaire , nombre de symboles en binaire , liste des transitions dans la preuve précédente séparées par des virgules`.

Quitte à ajouter des 0 à gauche, on fait en sorte que tous les mots binaires codant des états, des symboles ou des actions ont la même longueur. Notez que ce mot est sur l'alphabet comportant les symboles `0 1 , ( )`

Quand on travaille sur les MdT il est pratique de pouvoir décrire « la photo »de la machine à un instant du calcul. On parle de *configuration* : il s'agit du contenu du ruban, de la position de la tête et de l'état.

Dans l'esprit de notre code, nous pouvons donner un *code pour une configuration.*
`codes des lettres à gauche de la tête , code de l'état , codes des lettres à droite de la tête .`

Notez que ce mot est sur l'alphabet comportant les symboles `0 1 ,`

Trop de langages. Le principe de la preuve est intéressant car c'est un argument de diagonalisation, argument qui est central en calculabilité et qu'on retrouve souvent.

Lemme. *L'ensemble des langages sur l'alphabet* $\{0,1\}$ *n'est pas dénombrable.*

Démonstration. On énumère les mots binaires dans l'ordre lexicographique $s_0=0, s_1=1, s_2 01, s_3=10, s_4=11, \ldots$.

On suppose par l'absurde que l'ensemble de tous les langages sur $\{0,1\}$ est dénombrable et donc qu'on peu énumérer sous forme de liste $L_0, L_1, L_2, \ldots$ (l'index est donné par la bijection de $\mathbb{N}$ dans l'ensemble de tous les langages).

Soit L le langage diagonal : $\{s_i \textit{ tel que } s_i \notin L_i\}$.

L apparaît dans la liste, donc il existe un index j pour lequel $L = L_j$. Ceci est absurde puisque $s_j \in L \iff s_j \notin L_j$. □

On a donc démontré ce qu'on souhaitait.

Théorème. *Le nombre de machines de Turing est dénombrable. Par contre l'ensemble des langages sur l'alphabet* $\{0,1\}$ *ne l'est pas. En conséquence, il existe des langages sur l'alphabet* $\{0,1\}$ *qui ne sont pas décidables par une machine de Turing.*

Nous allons voir dans la suite qu'on peut exhiber un exemple concret : le *problème de l'arrêt* d'un programme.

Machine de Turing Universelle. On a vu qu'on peut coder (le programme) d'une machine de Turing M comme un mot sur l'alphabet comportant les symboles `0 1 , ( )`

On peut coder l'entrée x d'une telle machine par un mot dans le même esprit.

La *machine de Turing universelle U* prend sur son ruban le code d'une machine M, suivi d'un ; suivi du code d'une entrée x. Cette machine universelle travaille donc sur l'alphabet comportant les symboles `0 1 , ( ) ;`

Notation simplifiée pour l'entrée : on écrira juste M ;x

Programme de la machine de Turing Universelle. En gros la machine va travailler sur x en utilisant le programme M. Le principe est similaire à la manière dont un programme assembleur est traité par un processeur.

Pour décrire proprement le fonctionnement de cette machine il est plus facile de considérer une machine de Turing avec plusieurs rubans (on pourra toujours la simuler par une machine à un seul ruban dans un second temps).

Le premier ruban reste initialement inchangé. Le second ruban contient le code de la configuration de la machine M. La machine U simule le calcul de M pas à pas : scan de l'état actuel de M sur le second ruban, recherche d'une règle adaptée sur le premier ruban, changement adapté de la configuration sur le second ruban.

Si l'entrée de U est incohérente et ne correspond pas au code d'une machine de Turing, la machine U déplace sa tête indéfiniment vers la droite.

Un exemple concret de problème indécidable.

Définition (problème de l'arrêt).

$$H = \{M;x \text{ tel que M s'arrête sur l'entrée x}\}$$

($M;x$ est codé comme expliqué précedemment)

Théorème. *Le problème de l'arrêt est indécidable.*

La preuve se fait par l'absurde avec un argument de diagonalisation.

Esquisse de preuve. Par l'absurde. Soit M_H une machine qui décide H.

Soit D une machine qui prend en entrée le code d'une machine de Turing M. D accepte M ssi M_H ne s'arrête pas sur l'entrée $M;M$.

La contradiction apparaît lorsqu'on se penche sur le calcul de la machine D sur son propre code en entrée. □

Réduction de Turing et indécidabilité d'autres problèmes. On peut utiliser le concept de *réduction* pour montrer que d'autres problèmes sont indécidables.

Exemple.

$$S := \{M \text{ tel que } M \text{ s'arrête sur toute entrée}\}$$

Si S était décidable, on pourrait y réduire le problème H (détails ci-dessous).

- Entrée de H : $M;x$.
- Construction de la machine M' qui va prendre en entrée y et s'arrêter si y est différent de x, sinon en cas d'égalité la machine M' répond comme M sur l'entrée x.
- M' appartient à S ssi $M;x$ appartient à H.

3 Existence d'algorithmes efficaces

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.

On se concentre sur le temps. On va regarder le *pire des cas*[1] 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, $f(n) \leq c.g(n)$.
- $g(n) = \Omega(f(n))$ pour $f(n) = O(g(n))$
- $g(n) = \Theta(f(n))$ pour $f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$

Vous allez retrouver cette notation dans le cours d'algorithmique.

1. La complexité en moyenne existe aussi, mais ça demande de connaître une distribution raisonnable des entrées. C'est aussi un cadre de travail plus difficile pour démontrer des choses. En pratique, l'expérimentation sur des instances bien choisies permet de comparer divers algorithmes.

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.

Définition. On note $N\mathtt{TIME}(f(n))$ 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.**

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

(la machine de Turing est déterministe)

Vérification **en temps Polynomial ≈ calcul efficace.**

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

(la machine de Turing peut être non-déterministe)

Ces deux classes sont robustes au sens où elles ne dépendent pas vraiment du modèle de machine utilisé. Ce n'est pas le cas par exemple du temps linéaire déterministe qui donne des classes différentes selon qu'on travaille avec des machines de Turing ou des machines RAM.

Exemples.

Calcul en temps polynomial	Vérification en temps polynomial
Graphe Eulérien ?	graphe Hamiltonien ?
Graphe 2-colorable ?	Graphe 3-colorable ?
Circuit	SAT

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

Exemple. Une entrée ressemble à ceci : $(x \vee y \vee z) \wedge (x \vee \neg t \vee \neg u \vee v)$...

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 avec SAT. 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 $x_2 \in \Omega_2$.

Remarque.
- 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.*

Démonstration. 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 $1 \leq p,t \leq p(n)$ 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.

4 Complexité : au delà de P et NP

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 $\texttt{Time}(f(n))$ est strictement contenue dans $\texttt{Time}(f(2n+1)^3)$*

Ce résultat a un analogue pour l'espace

Théorème (Hiérarchie en espace, Hartmanis et. al. 1965)**.** *Si $f(n)$ est une fonction raisonnable alors $\texttt{Space}(f(n))$ est strictement contenue dans $\texttt{Space}(f(n)\log(f(n)))$*

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).

Conséquence. *Si $f(n) \geq \log n$ est une fonction raisonnable alors $\texttt{NSpace}(f(n)) \subseteq \texttt{Space}(f(n) \times f(n)))$.*

Quelques classes de complexité sont représentées en Figure 1. On sait que L est strictement inclus dans Pspace et que P est strictement inclus dans Exptime.

5 Intégrer la calculabilité et la complexité à un cours en NSI

Ateliers par groupe de 4 ou 5 pour s'approprier une partie du contenu de la journée, et réfléchir à une mise en oeuvre.

Suggestions.

- calculabilité et indécidabilité
- entrevoir NP en travaillant sur la vérification vs recherche (bactrack, aléa).
- jouer sur équivalence de modèles (exemples : MdT avec ruban bi-infini vs MdT avec ruban borné à gauche, MdT vs automate avec 2 piles)
- jouer avec les automates finis (déterminisation, minimisation, équivalence regexp)
- modélisation avec SAT
- réductions

Site du cours

`https://www.lacl.fr/fmadelaine/teachingDIUCalcComp.html`

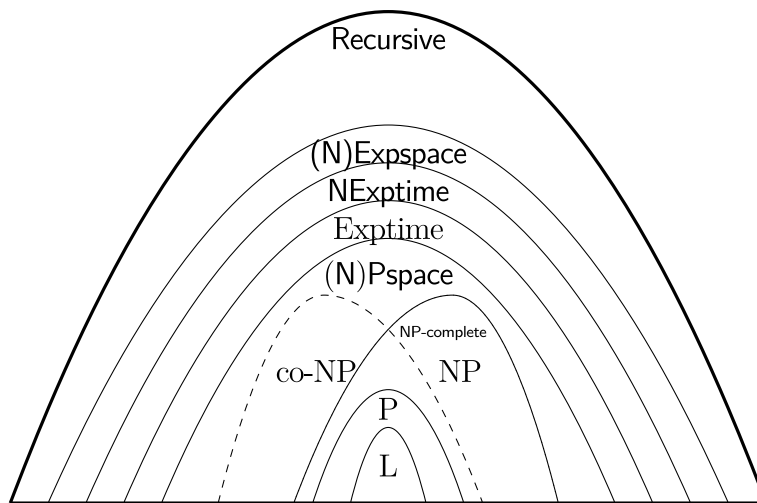

FIGURE 1 – Quelques classes de complexité et leur inclusion