Information, Calcul et Communication

La théorie de la calculabilité et le problème de l'arrêt

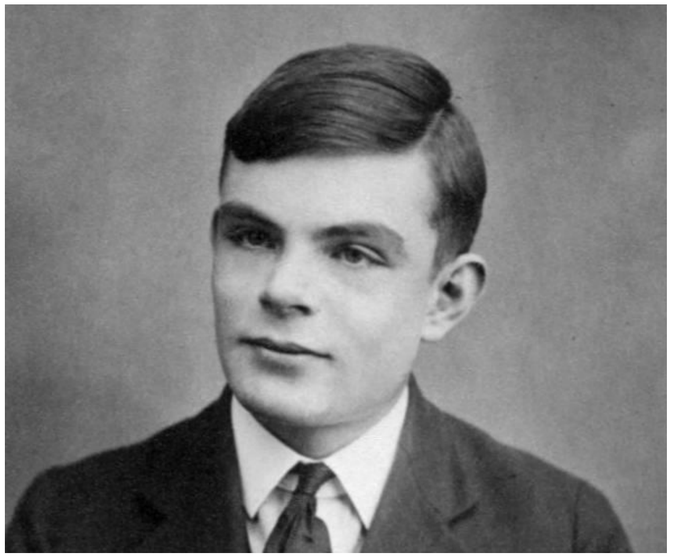
École polytechnique fédérale de Lausanne

Olivier Lévêque

Introduction à la théorie de la calculabilité

Question : Tout problème est-il soluble par un algorithme ?

Réponse : Non ! (Alan Turing, 1936)



Pour bien comprendre cette question, on doit d'abord définir ce qu'on entend par "problème".

Un problème est un ensemble de questions

- **Exemple :** Combien de musées trouve-t-on dans chaque ville de Suisse ?

  Algorithme de résolution: aller consulter la liste des musées de chaque ville et compter à chaque fois le nombre de ceux-ci
  → Genève : 26, Lausanne : 23, etc.

- Mais le nombre de villes en Suisse est un nombre fini !
  On peut donc établir *une fois pour toutes* une **table de correspondance :**

Ville	Genève	Lausanne	…
Nombre de musées	26	23	…

Après ça, **plus besoin d'algorithme** pour résoudre ce problème !

Un autre exemple de problème

- Etant donné un nombre entier positif N, celui-ci est-il un nombre premier ? (c'est-à-dire un nombre admettant exactement deux diviseurs distincts : 1 et le nombre en lui-même)

  Ex: si $N = 7$, alors la réponse est oui ; si $N = 8$, alors la réponse est non.

- Ce problème a un **nombre infini d'instances**. On ne peut donc pas établir une fois pour toutes une table de correspondance.

- Pour autant, existe-t-il un algorithme qui permette de le résoudre ?

  Oui : tester tous les nombres entre 2 et $N - 1$; si aucun ne divise N (et si N est différent de 1), alors N est premier.

Problèmes de décision

- Le problème précédent est un **problème de décision**, qui ne demande qu'une réponse "oui" ou "non" pour chaque valeur de N.

- Turing (1936) :

  "Il existe des problèmes de décision qu'il est impossible de résoudre au moyen d'un algorithme ; ces problèmes sont donc **indécidables**."

- Exemple : le **problème de l'arrêt**.

Le problème de l'arrêt

"Etant donné un algorithme P prenant en entrée des données X,
sait-on si l'algorithme $P(X)$ s'exécute en un temps fini ou non ?"

- Evidemment, pour certains algorithmes P et certaines données d'entrée X, la réponse est connue!

- Plus précisément:

"Existe-t-il un algorithme A prenant en entrée un autre algorithme P
et des données X, et dont la sortie soit **oui** si $P(X)$ s'exécute en
un temps fini, et **non** dans le cas contraire?"

- Ce que Turing démontre en 1936, c'est qu'un tel algorithme A **n'existe pas** !

Démonstration (par l'absurde)

Supposons qu'un tel algorithme A existe, c'est-à-dire :

- $A(P, X)$ sort *oui* si $P(X)$ s'arrête
- $A(P, X)$ sort *non* si $P(X)$ continue indéfiniment

A partir de cet algorithme A, on construit un autre algorithme B :

algorithme B

entrée : algorithme P
sortie : aucune

Si $A(P, P) =$ *oui*, alors : effectuer une boucle infinie
Sinon : s'arrêter

Démonstration (par l'absurde)

Que se passe-t-il si on exécute l'algorithme B avec **lui-même** en entrée ?
En d'autres termes, que fait $B(B)$?

- si $A(B, B) = oui$ (i.e., si $B(B)$ s'arrête), alors $B(B)$ effectue une boucle infinie ?

algorithme B
entrée : algorithme P sortie : aucune
Si $A(P, P) = oui$, alors : effectuer une boucle infinie Sinon : s'arrêter

Si $A(B, B) = oui$, alors B effectue une boucle infinie

$\Leftrightarrow$ B(B) s'arrête

Démonstration (par l'absurde)

Que se passe-t-il si on exécute l'algorithme B avec **lui-même** en entrée ?
En d'autres termes, que fait $B(B)$?

- si $A(B,B) = \textit{oui}$ (i.e., si $B(B)$ s'arrête), alors $B(B)$ effectue une boucle infinie ?
- si $A(B,B) = \textit{non}$ (i.e., si $B(B)$ continue indéfiniment), alors $B(B)$ s'arrête ?

algorithme B
entrée : algorithme P sortie : aucune
Si $A(P,P) = \textit{oui}$, alors : effectuer une boucle infinie Sinon : s'arrêter

Dans les deux cas, on a clairement une contradiction !

Conclusion : L'hypothèse effectuée (l'algorithme A existe) est donc fausse. #

Argument de la diagonale de Cantor

$$\{\text{algorithmes}\} <=> \{\text{suites finies de 0 et de 1}\}$$

$$\Longleftrightarrow \mathbb{N} = \{\text{nombres entiers}\}$$

$\Rightarrow$ L'ensemble des algorithmes est un ensemble dénombrable, i.e. un ensemble en bijection avec l'ensemble $\mathbb{N}$.

un problème de décision

$\Longleftrightarrow$ une fonction qui, à des données d'entrées particulières, fait correspondre une valeur oui ou non
(1) (0)

Ex: primalité 1, 2, 3, 4, 5, 6, 7...

f ↓

non, oui, oui, non, oui, non, oui...

$\Longleftrightarrow$ une fonction booléenne

$$f : \underbrace{\mathbb{N}}_{\text{données d'entrées}} \longrightarrow \{0,1\}$$

Cantor : L'ensemble des fonctions booléennes n'est pas dénombrable.

Donc l'ensemble des problèmes de décision est "plus grand" que l'ensemble des algorithmes.

Démonstration

Supposons que l'ensemble E des fonctions booléennes est dénombrables :

$$E = \{f_0, f_1, f_2, f_3, f_4, \dots \dots f_n \dots\}$$

	0	1	2	3	4	5	6	..
f_0	(1) 0	0	0	1	1	1	0	..
f_1	0	(1) 0	1	1	0	1	0	..
f_2	0	1	(1) 0	1	0	1	0	..
f_3	1	0	1	(0) 1	0	1	1	...

Définissons la fonction g ainsi :

$$g(n) = 1 - f_n(n) \qquad n \in \mathbb{N}$$

n	0	1	2	3	4	5	6
g(n)	1	1	1	0	....		

g est une fonction booléenne $E = \{f_0, f_1, f_2 \dots\}$

mais g n'est pas dans la liste

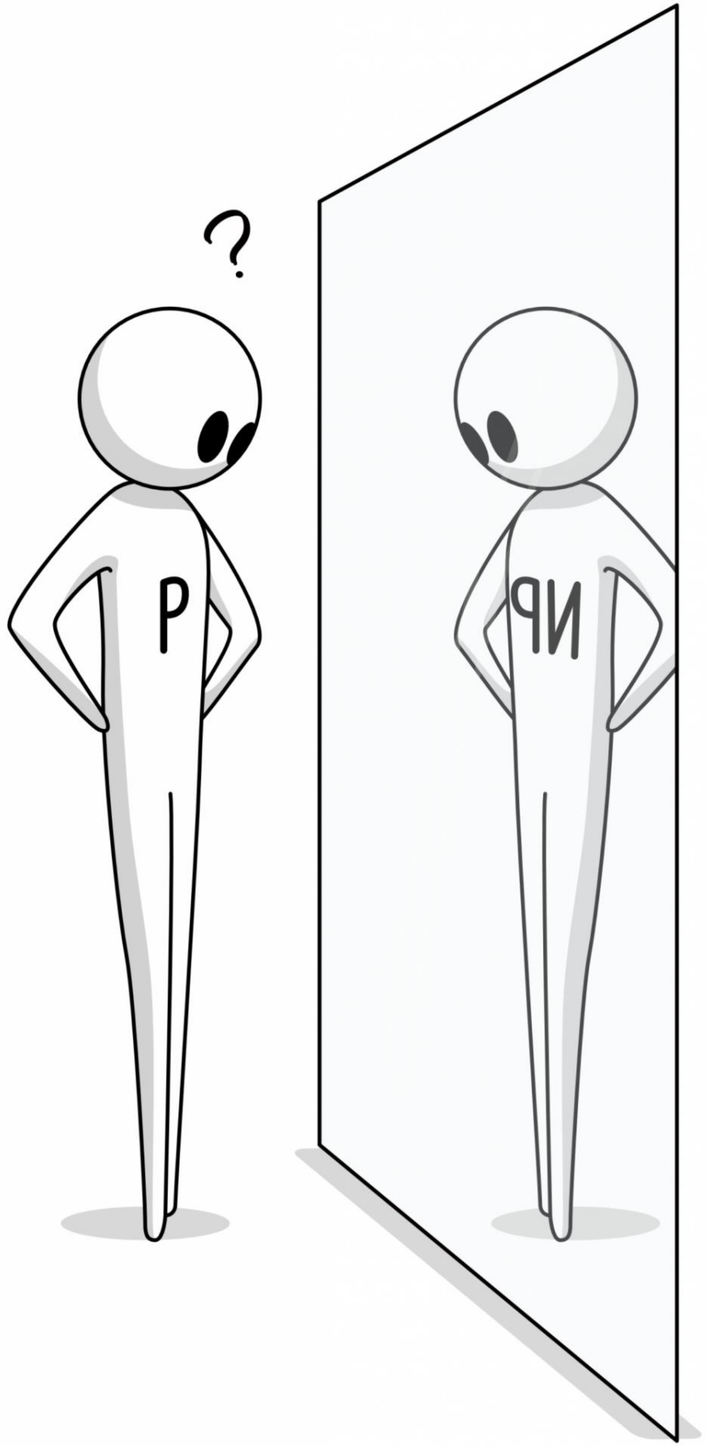
$\Rightarrow$ E n'est pas dénombrable #

Quelques rappels

- **complexité temporelle d'un algorithme** = nombre d'opérations élémentaires effectuées par celui-ci, dans le pire des cas

- **problème** = ensemble de questions, solubles (ou non) par un algorithme (ensemble infini → problème intéressant!)

Maintenant, nous allons nous intéresser
aux **classes de complexité des problèmes**.



Information, Calcul et Communication

Classes de complexité des problèmes

École polytechnique fédérale de Lausanne

Olivier Lévêque

Classes de complexité des problèmes

- **Première question** : Tout problème est-il soluble par un algorithme?

  Réponse : non !

- **Deuxième question** : Les problèmes solubles le sont-ils tous en un temps raisonnable ?

  Réponse : Encore non !

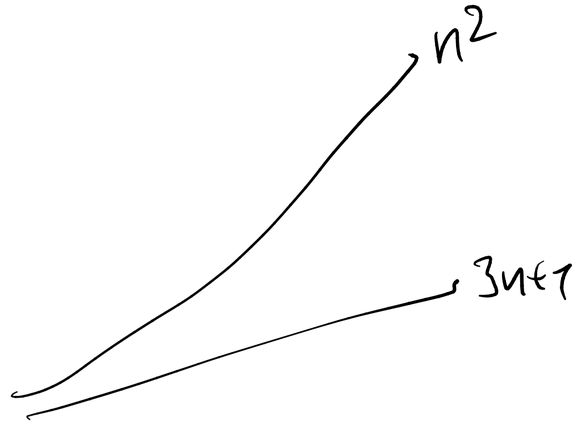
  Pour essayer d'identifier quels **problèmes** sont faciles à résoudre et quels problèmes le sont moins, on introduit des **classes de complexité**.

Préliminaire : notation $O(\cdot)$

Définition

Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
On dit que "$f(n)$ est un grand O de $g(n)$" et on écrit "$f(n) = O(g(n))$"
s'il existe $C > 0$ et $N \geq 1$ tels que

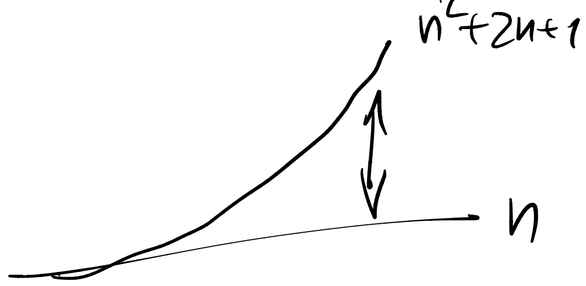
$$f(n) \leq C\, g(n) \text{ pour tout } n \geq N$$



Exemples

- Si $f(n) \leq g(n)$ pour tout $n \geq 1$, alors $f(n) = O(g(n))$
- Si $f(n) = \Theta(g(n))$, alors $f(n) = O(g(n))$

- La fonction $f(n) = 3\,n + 1$ est un $O(n)$, est aussi un $O(n^2)$
- La fonction $f(n) = n^2 + 2n + 1$ est un $O(n^2)$, mais n'est pas un $O(n)$



Préliminaire : notation $O(\cdot)$

Définition

Soient $f, g : \mathbb{N} \to \mathbb{R}_+$ deux fonctions non-négatives
On dit que "$f(n)$ est un grand O de $g(n)$" et on écrit "$f(n) = O(g(n))$"
s'il existe $C > 0$ et $N \geq 1$ tels que

$$f(n) \leq C\ g(n) \text{ pour tout } n \geq N$$

Exemples

- Si $f(n) \leq g(n)$ pour tout $n \geq 1$, alors $f(n) = O(g(n))$
- Si $f(n) = \Theta(g(n))$, alors $f(n) = O(g(n))$

- La fonction $f(n) = 3\,n + 1$ est un $O(n)$, est aussi un $O(n^2)$
- La fonction $f(n) = n^2 + 2n + 1$ est un $O(n^2)$, mais n'est pas un $O(n)$

- La fonction $\log_2(n)$ est un $O(n^p)$ pour tout $p > 0$
- La fonction n^p est un $O(2^n)$ pour tout $p > 0$

Deux classes de complexité importantes

Définitions

- La **classe P** est l'ensemble des **problèmes** qui peuvent être **résolus en temps polynomial**, i.e., *pour lesquels il existe un algorithme de résolution dont la complexité temporelle est un* $O(n^p)$ *pour des données d'entrée de taille* n *(avec* $p \geq 1$ *un nombre fixé).*

- La **classe NP** est l'ensemble des **problèmes** pour lesquels, *si "on" nous propose une solution du problème, il est alors possible de* ***vérifier en temps polynomial*** *si celle-ci en est une ou pas.*

Remarques

- NP ne veut **pas** dire "non-polynomial"
- P $\subset$ NP (il est plus facile de vérifier une solution que d'en proposer une !)

Exemples de problèmes appartenant à la classe P

Beaucoup de problèmes que nous avons déjà rencontrés dans ce cours appartiennent à la classe P (et sont donc des problèmes "faciles" à résoudre) :

- Le problème de la recherche d'un élément dans une liste de taille n.
- Le problème du tri d'une liste de taille n.
- Le calcul de la somme ou de la moyenne de n nombres.
- Identifier si tous les éléments d'une liste (de taille n) sont différents.

Tous ces problèmes admettent en effet des algorithmes de résolution de complexité temporelle $\Theta(n)$, $\Theta(n \cdot \log_2(n))$ ou $\Theta(n^2)$, *donc* $O(n^2)$.

Exemples de problèmes appartenant à la classe NP

- Tous les problèmes mentionnés avant, du fait qu'ils appartiennent à la classe P, appartiennent également à la classe NP !

- Voici maintenant un autre problème appartenant à la classe NP:

  « Soient P, Q deux nombres premiers à n chiffres, et soit $N = P \cdot Q$.

  Etant donné N, on aimerait retrouver P et Q. »

  Exemple: si $N = 98'201$, que valent P et Q ?

- Ce problème n'est a priori pas facile à résoudre… Par contre, si on nous propose une solution (ici, $P = 347$ et $Q = 283$), il est alors facile de vérifier que $N = P \cdot Q$ en effectuant la multiplication (en $\Theta(n^2)$ opérations) : le problème appartient donc à la classe NP.

Exemples de problèmes

1. « Etant donné une liste ordonnée L de n nombres entiers, existe-t-il **deux indices** différents $i, j \in \{1, \ldots, n\}$ tels que $L(i) + L(j) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23)$ → Réponse : non

   Comme nous l'avons déjà vu, un algorithme de complexité temporelle $\Theta(n^2)$, ou même $\Theta(n)$, permet de répondre à cette question: le problème ci-dessus fait donc partie de la **classe P**.

Exemples de problèmes

2. « Etant donné une liste ordonnée L de n nombres entiers, existe-t-il **trois indices** différents $i, j, k \in \{1, \ldots, n\}$ tels que $L(i) + L(j) + L(k) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23)$ → Réponse : encore non!

   Un algorithme de complexité temporelle $\Theta(n^3)$ permet de répondre à cette question: le problème ci-dessus fait donc également partie de la **classe P**.

   (ou $\Theta(n^2)$ même)

Exemples de problèmes

3. « Etant donné une liste ordonnée L de n nombres entiers, existe-t-il **un sous-ensemble** $S \subset \{1, \ldots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23)$

   Réponse : oui ! En effet : $-15 \ -12 - 1 + 5 + 23 = 0$

Un sous-ensemble $S = \{-3, +5, +17\}$ → 2^n (1) sous-ensembles

0/1 0/1 0/1 0/1 1 1 0

$\{-15, -12, -3, -1, +5, +17, +23\}$

2^n choix à faire

Exemples de problèmes

3. « Etant donné une liste ordonnée L de n nombres entiers, existe-t-il **un sous-ensemble** $S \subset \{1, \dots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

   Exemple : $L = (-15, -12, -3, -1, +5, +17, +23)$

   Réponse : oui ! En effet : $-15 \ -12 - 1 + 5 + 23 = 0$

   Mais pour obtenir cette réponse, on ne connaît pas d'autre algorithme que de tester tous les sous-ensembles S possibles, qui sont au nombre de 2^n.

   Conclusion 1: On ne sait pas si ce problème fait partie de la **classe P**.

   Par contre, si on nous propose une solution du problème, il est alors facile de **vérifier** en temps polynomial que c'en est bien une! (additionner n nombres ne nécessite que $\Theta(n)$ opérations).

   Conclusion 2: Le problème ci-dessus fait partie de la **classe NP**.

Et maintenant, une question à un million de dollars !

- Le problème :

  « Etant donné une liste ordonnée L de n nombres entiers, existe-t-il un sous-ensemble $S \subset \{1, \ldots, n\}$ tel que $\sum_{i \in S} L(i) = 0$? »

  s'appelle le **problème des sommes de sous-ensembles**. On peut montrer qu'il fait partie des problèmes **les plus difficiles** à résoudre parmi ceux de la classe NP: on appelle ces problèmes des **problèmes NP-complets.**

- Si on arrive à démontrer un jour que ce problème fait **aussi** partie de la classe P, ou au contraire qu'il n'en fait **pas** partie, on aura alors résolu la question de savoir si **P=NP** ou non, pour laquelle le Clay Mathematics Institute a promis une récompense d'une valeur d'un million de dollars.

Problèmes d'optimisation discrète

Exemple : former un comité

choisir $S \subset \{1 \dots n\}$

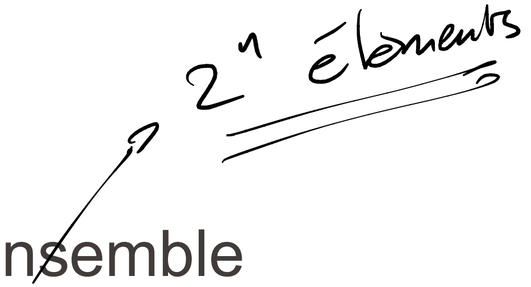
$f(S)$ = le temps pour se décider avec comité = S

$\longrightarrow \min_S f(S)$?

Problèmes d'optimisation discrète

- **Généralisation :**

  « Etant donné une **fonction** f qui à tout sous-ensemble donné $S \subset \{1, \dots, n\}$ associe une valeur réelle $f(S)$, trouver un sous-ensemble $S \subset \{1, \dots, n\}$ tel que $f(S)$ soit minimale. »



Problèmes d'optimisation discrète

- **Généralisation :**

  « Etant donné une **fonction** f qui à tout sous-ensemble donné $S \subset \{1, \ldots, n\}$ associe une valeur réelle $f(S)$, trouver un sous-ensemble $S \subset \{1, \ldots, n\}$ tel que $f(S)$ soit minimale. »

- Pour de nombreuses fonctions f, et donc pour de nombreux **problèmes d'optimisation discrète**, on ne sait *pas* s'ils appartiennent à la classe P, *ni* s'ils appartiennent à la classe NP: on appelle ces problèmes des **problèmes NP-difficiles.**

- Tous ces problèmes sont caractérisés par l'existence d'un **nombre fini, mais grand,** de solutions possibles.

Classes de complexité des problèmes : résumé

- **La classe P** est la classe des problèmes solubles en temps polynomial.
  (ex: le problème du tri d'une liste)

- **La classe NP** est la classe des problèmes vérifiables en temps polynomial.
  (ex: le problème de la factorisation des nombres) **Elle contient la classe P.**

- Les problèmes *les plus difficiles* de la classe NP sont dits **NP-complets.**
  (ex: le problème des sommes de sous-ensembles)

- Les problèmes *au moins aussi difficiles* que les problèmes NP-complets sont dits **NP-difficiles** (ils n'appartiennent pas forcément à la classe NP).
  (ex: la semaine prochaine !)

Classes de complexité des problèmes : résumé