

# Chapitre 5

## Les classes de problèmes

### 5.1 Introduction

Nous nous sommes intéressés dans les cours précédents à la complexité d'un problème au sens de la complexité de l'algorithme qui le résout (la solution). Cette complexité est celle de l'algorithme le plus efficace qui le résout. Nous avons considéré seulement les problèmes dont la complexité est polynomiale soit  $T(n) = \mathcal{O}(n^k)$ .

L'univers des problèmes est, à présents, divisé en au moins deux catégories : les problèmes "*faciles*" et les problèmes "*non-faciles*" = "*probablement difficiles !*". La complexité des algorithmes permet de classer les problèmes en fonction de la difficulté de les résoudre.

La classe des problèmes considérée en informatique comme "*faciles*" c'est celle des problèmes qui ont une complexité polynomiale. Ils sont les seuls à pouvoir être utilisés pour de grandes tailles des données, indépendamment de la puissance de la machine. Cette classe est désignée par la classe  $P$  des problèmes.

#### 5.1.1 La classe $P$

On peut déjà comprendre intuitivement cette notion de "difficulté" en comparant les deux problèmes suivants :

1. soit  $n$  un entier donné, vérifier si  $n$  est pair ou impair (il est pair si le dernier chiffre vaut  $\{0, 2, 4, 6, 8\}$  et impair sinon)
2. soit  $n$  un entier donné, vérifier si  $n$  est un nombre parfait ou non (il est parfait s'il est égal à la somme de ses diviseurs et ne l'est pas sinon).

Ces deux problèmes ont tous les deux une complexité polynomiale, mais leur polynôme temps peut ne pas être exactement le même. Ils sont cependant considérés comme des problèmes de la classe  $P$ , donc "faciles".

#### Définition 5.1.1 *Algorithme polynomiale*

*Un algorithme est dit en temps polynomiale si, pour tout  $n$ ,  $n$  étant la taille des données, l'algorithme s'exécute en temps borné par un polynôme  $f(n) = c * n^k$  opérations élémentaires (les constantes  $c$  et  $k$  sont indépendantes de  $n$ ). Par exemple  $f(n) = 5n^2$ .*

#### Définition 5.1.2 *Problème polynomiale*

*On dit qu'un problème  $p$  est polynomiale ou encore que  $p$  est dans la classe  $P$  si et seulement si il existe un algorithme pour le résoudre qui s'exécute en temps polynomiale.*

On peut se demander si *tous* les problèmes sont solvables ou non en temps polynomial. La réponse est non ! Il existe des problèmes qui ne peuvent être résolus par aucun ordinateur quelque soit le temps qu'ils y passent. De même, il existe des problèmes qui peuvent être résolus mais en temps bien plus important que  $\mathcal{O}(n^k)$ . Ils ont une complexité exponentielle ou plus.

Nous nous intéressons dans ce chapitre à la classification des problèmes selon leur *difficulté* et plus particulièrement aux classes  $P$ ,  $NP$  et  $NP$  – *complet(NPC)*. Les problèmes des classes  $NP$  et  $NPC$  sont les problèmes solvables qui, encore à ce jour, n'ont pas d'algorithme polynomial qui les résout.

## 5.2 Problèmes exponentiels

Ces problèmes de coût exponentiel ou plus sont supposés insolubles en pratique dans le cas général. Ce sont des problèmes ouverts et sont des challenges de la recherche. La fonction de complexité est de la forme  $f(n) = C^n$ ,  $c > 1$ . Par exemple si cette fonction est une valeurs de temps exprimée en nanosecondes avec  $C = 2$  et  $n = 100$  alors  $2^{100} \approx 10^{30}$  elle représente un temps de  $3 * 10^{11}$  siècles ! Faire le même calcul avec  $n = 1000$ .

Pour ces problèmes, il est possible qu'ils puissent être résolus dans des cas particuliers ou bien, éventuellement, par des méthodes approchées.

### 5.2.1 Problèmes *NP*

Parmi les problèmes de coût exponentiel, certains ont la propriété suivante :

On peut vérifier en temps polynomial si une instance de données quelconque est une réponse positive du problème. □

On dit que ces problèmes sont dans la classe *NP* : la classe des problèmes (*non déterministes, polynomiaux*).

**Ceux sont les problèmes qui peuvent être résolus en temps polynomial par une machine de Turing non déterministe** (c'est-à-dire qui a entre deux états plusieurs choix de plusieurs configurations : elle peut accepter un mot tout en ayant plusieurs calculs non acceptants sur ce mot. Le calcul du complément d'un langage est plus coûteux sur une machine non déterministe, il ne suffit pas d'inverser les états finaux et non finaux.).

#### Définition 5.2.1 *Problème NP*

*Un problème  $p$  est dans la classe NP si et seulement si il existe un algorithme qui vérifie si une instance de données du problème  $p$  est solution de ce problème et ce en temps polynomial.*

En général, vérifier une réponse est toujours moins coûteux que la calculer. Par exemple vérifier qu'un tableau de  $n$  composants est trié est moins coûteux (en  $\mathcal{O}(n)$ ) que le trier (en  $\mathcal{O}(n \log n)$ ).

*Ainsi la classe NP contient l'ensemble des problèmes dont la vérification est polynomial mais dont la résolution ne l'est pas obligatoirement.*

Cook a démontré en 1971 que le problème SAT est NP difficile. Le problème SAT (le problème de la satisfiabilité d'une formule du calcul propositionnel) est dans NP.

**Exemple 5.2.2** Soit  $F$  une formule du calcul propositionnel dont il faut démontrer la satisfiabilité :

$$F(x_1, x_2, \dots, x_n) = \wedge_{i=1}^k C_i \quad \text{où} \quad C_i = \vee_{j=1}^k x_j$$

*Il est facile de vérifier, en temps polynomial qu'une telle formule vaut vrai étant donnée une affectation de valeurs de vérité à ses variables. Mais jusqu'à aujourd'hui on ne connaît pas d'algorithme polynomial qui donne les valeurs de vérité aux  $n$  variables de la formule. Tous les algorithmes connus sont des variantes de l'énumération des  $2^n$  affectations possibles, donc des algorithmes exponentiels.*

La machine de Turing non déterministe suivante le décide en temps polynomial :

1. Lire l'expression booléenne  $F$  ;
2. Pour toute variable  $x_i$  apparaissant dans  $F$ , choisir de manière non déterministe une valeur  $v(x_i)$  dans  $\{0, 1\}$  ;
3. Accepter si la valuation  $v$  satisfait  $x_i$  ; refuser sinon.

### 5.2.2 Les problèmes $P = NP$ ou bien $P \subset NP$

La définition précédente implique l'inclusion  $P \subseteq NP$ . Mais on n'a pas démontré l'existence d'un problème qui est dans  $NP$  mais pas dans  $P$ , autrement dit on ne sait pas si  $P \neq NP$ . Cette question est l'un des problèmes ouverts les plus importants de la recherche en informatique fondamentale depuis qu'il a été posé en 1972.

Prouver qu'un problème est  $NP$ -complet (*par réduction*) c'est prouver qu'il est intraitable et donc il est inutile de chercher sa solution. On peut s'intéresser alors à chercher une solution approchée ou à résoudre un cas particulier du problème.

## 5.3 Problèmes $NP$ – Complet

Il existe une sous classe de problèmes de la classe  $NP$  ; elle contient quelques centaines de problèmes bien connus qui ont des propriétés communes. Cette sous-classe est désignée par la classe des problèmes  *$NP$  – Complets (NPC)*. Cette banque de problèmes connus est utilisée pour montrer qu'un nouveau problème, "*est au moins aussi difficile qu'un des problèmes connus*". Ceci est prouvé lorsqu'on démontre que le nouveau problème *se réduit en temps polynomial à l'un des problèmes de cette classe*. La réduction consiste à transformer en temps polynomial un des problèmes connu  $NPC$  en ce nouveau problème.

### 5.3.1 La réduction d'un problème

#### Définition 5.3.1 *La réductibilité*

*Un problème  $p_1$  peut être ramené (ou réduit) à un problème  $p_2$  si une instance quelconque de  $p_1$  peut être traduite comme une instance de  $p_2$  et la solution de  $p_2$  sera aussi solution de  $p_1$ . On note cette réduction :  $p_1 \leq p_2$ .*

#### Définition 5.3.2 *Problème $NP$ -complet*

*On dit qu'un problème  $p$  est  $NP$ -complet si et seulement si :*

1.  $p \in NP$  et
2.  $\exists q \in NPC \quad q \leq p$

Le premier point est satisfait en montrant l'existence d'un algorithme polynomial qui vérifie si une instance donnée est solution du problème  $p$ . Le second point est satisfait lorsqu'on prouve que le problème  $p$  est réductible à un problème  $q$  de la classe  $NPC$ , autrement dit que  $p$  est  **$NP$ -difficile**.

**Que peut-on déduire lorsque nous avons  $p_1 \leq p_2$  ?**

Du point de vue de la complexité, cette définition permet de déduire les deux conclusions suivantes :

- que le coût de résolution du problème  $p_1$  est au plus égal au coût de la transformation plus le coût minimum connu de la résolution de  $p_2$ .  
Donc si  $p_2$  est dans la classe  $P$  (se résout en temps polynomial) et si la transformation est aussi dans la classe  $P$  alors  $p_1$  est aussi dans la classe  $P$ .
- mais si le coût minimum de résolution de  $p_1$  est connu, alors le coût de  $p_2$  plus celui de la transformation ne peut pas être moindre. Donc si  $p_1$  a un coût exponentiel alors soit la transformation a un coût exponentiel soit le coût de  $p_2$  a un coût exponentiel (soit les deux).

**Application de ce constat :**

- a.) Si un problème  $P_1$  se réduit en temps polynomial à un problème  $P_2$  et si on sait que  $P_2$  est facile alors on déduit que  $P_1$  est au aussi facile.
- b.) Si un problème  $P_1$  se réduit à un problème  $P_2$  et si on sait que  $P_1$  est difficile alors on déduit que  $P_2$  est au moins aussi difficile.

**Remarque 5.3.3** *Notons qu'il est nécessaire que la fonction de transformation soit polynomiale sinon les deux conclusions précédentes ne peuvent pas être faites.*