

# **CHAPITRE VI**

## **Les problèmes NP-complets**

## Généralités

### Problème

Nous rappelons que formellement, un problème représente un ensemble de données sur lesquelles on veut appliquer un traitement. Il est décrit par une paire (*instance*, *question*). L'instance spécifie la donnée du problème alors que la question fait état de la problématique ainsi que des hypothèses permettant d'amorcer le processus de résolution. Un exemple de problème qui sera étudié dans ce chapitre est le problème du cycle Hamiltonien ou PCH en abrégé et qui est défini comme suit :

Donnée : un graphe  $G = (S, A)$  non orienté où  $S$  est un ensemble de  $n$  sommets et  $A$  un ensemble de  $m$  arêtes.

Question :  $G$  contient-il un cycle hamiltonien ? plus précisément existe-il un chaîne  $\langle s_1, s_2, \dots, s_n \rangle$  dans  $G$  où  $n = |S|$  passant par tous les sommets une et une seule fois, tel que  $\{s_n, s_1\} \in A$  et  $\{s_i, s_{i+1}\} \in A$  pour tout  $i, 1 \leq i < n$ ?

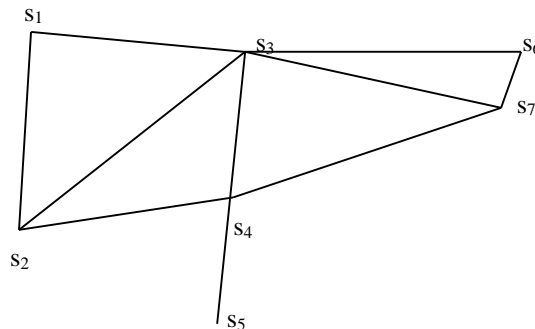
### Instance de problème

Une instance de problème est une donnée réelle du problème respectant la spécification de la description du problème.

**Exemple :** instance de problème du cycle hamiltonien ou PCH.

L'instance se présente comme suit :

- Un graphe non orienté de sommets  $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$  et un ensemble d'arêtes  $A$  schématisées sur la figure 6.2.



**Figure 6.2.** Une instance du problème du cycle Hamiltonien

Le problème dans ce cas est de répondre par oui ou par non s'il est possible partant d'un sommet du graphe, on peut passer d'un sommet à un autre une et une seule fois en visitant tous les sommets et en revenant au sommet initial à la fin du parcours.

### Taille de problème

La taille d'un problème est celle de la donnée correspondante. Elle est exprimée à l'aide des paramètres apparaissant dans la description de l'instance. A chaque problème est donc associée une taille qui mesure implicitement la quantité des données en entrée. La taille du problème du cycle hamiltonien est constituée de la paire (nombre de sommets, nombre d'arêtes).

### ***Problèmes de décision***

Ces problèmes sont caractérisés par la question accompagnant la description de l'instance et dont la réponse est par 'oui' ou par 'non'.

Exemple :

Instance :  $G=(S, A)$  un graphe non orienté où  $S$  est l'ensemble des sommets et  $A$  l'ensemble des arêtes.

Question : existe-il un cycle hamiltonien dans  $G$  ?

Les algorithmes de résolution de ce problème répondent par 'oui il existe un cycle hamiltonien dans  $G$ ' et dans ce cas l'instance présentée en entrée à l'algorithme est dite positive. Ils ne sont pas appelés à expliciter le cycle hamiltonien dans leur réponse. Comme ils peuvent répondre par la négative, c'est-à-dire 'non, il n'existe pas de cycle hamiltonien dans  $G$ ' et dans ce cas l'instance traitée est dite négative.

### ***Les problèmes NP-complets et la transformation polynomiale***

Avant de définir les problèmes NP-complets, il est indispensable de présenter les classes NP et P respectivement. Les définitions données ci-dessous sont informelles car elles se basent sur les notions de problèmes et d'algorithmes. Les définitions formelles reposent sur les concepts de langages et de machines de Turing qui reconnaissent ces langages. L'analogie existant entre les deux contextes permet de passer du formel à l'informel et vice versa.

#### ***La classe NP (Non deterministic Polynomial)***

**Définition 1.** Un algorithme est dit non déterministe s'il est structuré en deux phases de la manière suivante:

**1<sup>ère</sup> phase :** Engendrer une solution quelconque à l'instance qui peut être bonne ou mauvaise.

**2<sup>ème</sup> phase de vérification:** vérifier en temps polynomial que la solution est effective ou pas.

#### **Exemple : Algorithme non déterministe pour le problème du cycle hamiltonien**

Soient  $G=(S, A)$  un graphe non orienté où  $S$  est un ensemble de  $n$  sommets et  $A$  l'ensemble des arêtes et  $M$  la matrice d'adjacence de  $G$ .

**1<sup>ère</sup> phase:** Engendrer un cycle pour le graphe  $G$ , soit  $s = \{s_1, s_2, \dots, s_n\}$  ce cycle. Stockons le dans un tableau  $S$  tel que  $S[i]= s_i$  pour tout  $i = 1$  à  $n$ .

**2<sup>ème</sup> phase** : Développer un algorithme pour vérifier si  $s$  représente un cycle hamiltonien ou pas.

Structure de données utilisées: un tableau appelé *occurrence* contenant les occurrences des sommets du graphe appartenant à  $s$ , donc ceux stockés dans le tableau  $S$ . L'algorithme de vérification est donc le suivant :

**début**

```

si (  $M[S[1],S[n]] = 0$  ) alors hamiltonien = faux
sinon
  début
    (* Initialiser le tableau des occurrences à 0 *)
    pour  $i = 1$  à  $n$  faire occurrence[ $i$ ] = 0 ;
     $i = 1$  ; hamiltonien = vrai ;
    tant que (  $i < n$  ) et (hamiltonien) faire
      si (occurrence[ $S[i]$ ] = 0) et ( $M[S[i], S[i+1]] = 1$ ) alors
        début occurrence[ $S[i]$ ] = 1;
           $i = i + 1$ ;
        fin
      sinon hamiltonien = faux ;
    fin ;
  si (hamiltonien et (occurrence[ $S[n]$ ] = 0)) alors imprimer 'oui'
    sinon imprimer 'non' ;

```

**fin** ;

**Définition 2.** La classe NP englobe tous les problèmes pour lesquels il existe un algorithme non déterministe pour les résoudre en un temps polynomial.

**Exemple 6:**

L'algorithme non déterministe de l'exemple ayant une complexité en  $O(n)$ , le problème du cycle Hamiltonien appartient à la classe NP.

**La classe P**

Intuitivement, la classe P contient tous les problèmes appartenant à la classe NP qui sont faciles à traiter par les moyens de calcul qui existent actuellement.

**Définition 3.** La classe P englobe tous les problèmes qui ont un algorithme de résolution dont la complexité du pire cas est polynomiale en fonction de la taille du problème.

**Exemple** : Problème appartenant à la classe P

Considérons le problème de la somme des arêtes appelé PSA énoncé comme suit :

Instance :  $G = (S, A)$  un graphe non orienté où  $S$  est un ensemble de  $n$  sommets et  $A$  l'ensemble des arêtes, un poids de valeur réelle strictement positive associé à chaque arête du graphe  $G$  et  $k$  un nombre réel.

Question : La somme des poids associés aux arêtes est-elle égale à  $k$  ?

Soit  $M$  la matrice d'adjacence associée au graphe  $G$  où  $M[i,j]$  contient le poids associé à l'arête  $(s_i, s_j)$  si l'arête existe, 0 sinon. Un algorithme de résolution du problème de la somme des arêtes est le suivant :

Algorithme somme des arêtes

**var** somme : réel ;

$i, j : 1 \dots n$  ;

**début**

somme := 0 ;

**pour**  $i := 1$  à  $n$  **faire**

**pour**  $j := i+1$  à  $n$  **faire**

somme := somme +  $M[i,j]$  ;

**fin** ;

La complexité de l'algorithme est  $O(n^2)$ , le problème PSA appartient donc à la classe P.

### ***La classe des problèmes NP-complets***

La classe des problèmes NP-complets englobe les problèmes de décision les plus difficiles de la classe NP. Sa définition s'appuie sur la notion de transformation polynomiale entre problèmes.

**Définition 4** : Une transformation polynomiale entre un problème  $\Pi_1$  et un problème  $\Pi_2$  est une fonction ayant comme domaine de départ les solutions de  $\Pi_1$  et comme domaine d'arrivée les solutions de  $\Pi_2$

$f : \Pi_1 \rightarrow \Pi_2$

$s_1 \rightarrow s_2 = f(s_1)$

et telle que :

- 1) pour toute solution positive de  $\Pi_1$  correspond une solution positive de  $\Pi_2$  et vice versa.
- 2) le calcul de  $f(s_1)$  se fait en un temps polynomial.

**Définition 5** : Un problème  $\Pi$  est NP-complet s'il vérifie les deux conditions suivantes :

- 1-  $\Pi$  appartient à la classe NP

- 2- Tout problème appartenant à la classe NP peut être réduit au problème  $\Pi$  via une transformation polynomiale.

### La classe co-NP

**Définition 6 :** Un problème  $\Pi$  est co-NP si le problème  $\bar{\Pi}$  constitué des solutions négatives de  $\Pi$  appartient à la classe NP.

#### Exemple :

Considérons le problème du nombre composé décrit comme suit :

Instance : un nombre entier  $k$

Question :  $k$  est-il composé ?

Le problème du nombre composé étant un problème NP puisqu'il appartient à P, le problème du nombre premier décrit comme suit :

Instance : un nombre entier  $k$

Question :  $k$  est-il premier ?

est co-NP.

La figure 6.5. montre les relations qui existent entre les différentes classes de problèmes définies précédemment si la classe P est différente de celle de NP.

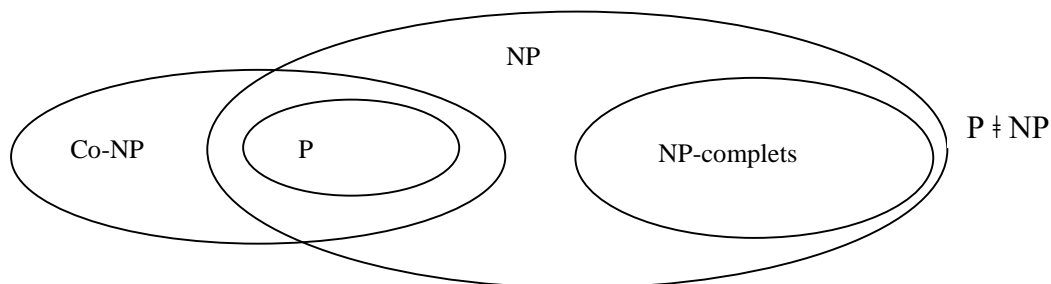


Figure 6.5. Les classes des problèmes NP, P, NP-complets et co-NP

### Le problème de la satisfiabilité ou SAT

Le problème de la satisfiabilité ou SAT en abrégé fut le premier problème à avoir été démontré NP-complet. C'est donc le pionnier et l'ancêtre de tous les problèmes NP-complets. SAT s'intéresse à la satisfiabilité des formules logiques de manière générale et celle des formules CNF (Conjunctive Normal Form) d'ordre zéro en particulier.

**Définition 6 :** Le problème de satisfiabilité ou SAT en abrégé est défini comme suit :

**Instance :**  $X = \{x_1, x_2, \dots, x_n\}$  un ensemble de variables booléennes,  $C = \{c_1, c_2, \dots, c_m\}$  un ensemble de clauses,  $c_i$  est une disjonction de littéraux, un littéral est une variable booléenne avec ou sans le connecteur de négation.

**Question :** Existe-il une instantiation (un ensemble de valeurs booléennes associées aux variables) de  $X$  telle que la conjonction des clauses de  $C$  est vraie ?

**Exemple :** une instance de SAT.

Considérons un ensemble de 5 variables booléennes  $X = \{x_1, x_2, x_3, x_4, x_5\}$  et un ensemble de 3 clauses  $C = \{c_1, c_2, c_3\}$  définies comme suit :

$$\begin{cases} c_1 = x_2 + \overline{x_4} + x_5 \\ c_2 = x_2 + x_4 \\ c_3 = \overline{x_1} + x_3 + x_4 \end{cases}$$

L'opérateur  $+$  exprime l'opérateur de disjonction booléenne, et  $\overline{x_i}$  la négation de la variable  $x_i$ . Une instantiation de variables est un vecteur de valeurs booléennes associées respectivement aux variables. En l'occurrence, l'instanciation 00011 qui correspond à la solution  $x_1=0, x_2=0, x_3=0, x_4=1$  et  $x_5=1$  est une solution de l'instance car elle évalue chaque clause à 1. Par contre l'instanciation 10101 n'est pas une solution pour cette instance.

**Théorème de Cook (1971) :** Le problème de satisfiabilité est NP-complet.

Preuve :

Il faut démontrer les deux assertions suivantes :

- 1- SAT appartient à NP
- 2- Tout problème appartenant à NP peut être réduit en SAT via une transformation polynomiale.

Prouvons que SAT appartient à NP. Pour cela il faut trouver un algorithme non déterministe pour SAT qui ait une complexité polynomiale. L'algorithme a l'ossature suivante :

- Engendrer une instantiation quelconque des variables booléennes, en l'occurrence  $V = (v_1, v_2, \dots, v_n)$   $v_i=0$  ou  $1$  pour tout  $i=1$  à  $n$ .
- Remplacer chaque variable des clauses par sa valeur et calculer sa valeur
- Si la conjonction des valeurs trouvées pour les clauses est égale à 1 alors l'instance est satisfiable sinon elle est contradictoire.

L'algorithme de vérification utilise une matrice  $m \times n$  appelée SAT pour stocker la donnée SAT.  $SAT[i,j]=1$  si le  $j$ ème littéral de la  $i$ ème clause existe sous la forme positive, il est égal à 0 si ce dernier existe sous la forme négative et est égal à -1 s'il n'existe pas. Pour l'exemple 9, la matrice SAT est la suivante :

$$\begin{vmatrix} -1 & 1 & -1 & 0 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 0 & -1 & 1 & 1 & -1 \end{vmatrix}$$

L'algorithme de vérification s'écrit comme suit :

Algorithme vérification-SAT

**entrée** : la matrice SAT et l'instanciation V

**sortie** : 'oui' ou bien 'non'

**var**

i : 1..m ; j : 1..n ;

satisfiable, sat-clause : booléen ;

**début**

i := 1 ; satisfiable := **vrai** ;

**tant que** ( i <= m ) **et** (satisfiable) **faire**

**début**

j :=1 ; sat-clause := faux ;

**tant que** non sat-clause **et** j <= n **faire**

**si** (SAT[i,j] = V[j]) **alors** sat-clause := **vrai**

**sinon** j = j+1;

**si** sat-clause **alors** i = i+1

**sinon** satisfiable = **faux**;

**fin** ;

**si** (satisfiable) **alors** imprimer 'oui'

**sinon** imprimer 'non' ;

**fin** ;

Le pire cas correspond à la situation où la dernière colonne de la matrice SAT est égale à V. Dans ce cas l'instance est satisfiable mais en plus les deux boucles imbriquées s'exécutent entièrement. La complexité de l'algorithme est donc  $O(n*m)$  au pire cas.

Prouvons que tout problème  $\Pi$  de NP peut être réduit au problème SAT en un temps polynomial.

La transformation polynomiale se construit à partir d'un problème d'appartenance d'un langage à une machine de Turing non déterministe en un temps polynomiale et le problème SAT (voir Garey and Johnson 1979).