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 circuit 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 circuit hamiltonien ? plus précisément existe-il un chemin $<s_1, s_2, ... , s_n>$ 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, $1 \leq i < n$?

Un autre exemple concerne le problème du voyageur de commerce ou PVC en abrégé. Etant donnée une carte géographique de villes et des distances les reliant, un voyageur est appelé à visiter toutes les villes une et une seule fois telle que la distance parcourue au total n'excède pas une constante k. La description formelle du problème est comme suit :

Donnée : un ensemble de n villes et un ensemble de distances des routes existant entre certaines villes entre elles
Question : Existe-il un chemin englobant toutes les villes une et une seule fois tel que la somme des routes constituant le chemin est inférieure ou égale à k ?

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 circuit hamiltonien ou PCH.
L'instance se présente comme suit :

- Un graphe non orienté de sommets S= {s1, s2, s3, s4, s5, s6, s7} et un ensemble d'arêtes A schématisées sur la figure 6.2.

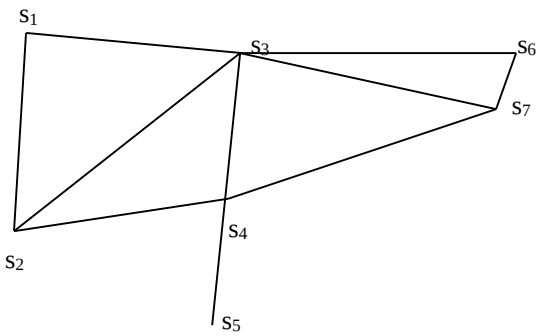


Figure 6.2. Une instance du problème du circuit Hamiltonien

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 circuit 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 circuit hamiltonien dans G ?
Les algorithmes de résolution de ce problème répondent par 'oui il existe un circuit 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 circuit hamiltonien dans leur réponse. Comme ils peuvent répondre par la négative, c'est-à-dire 'non, il n'existe pas de circuit 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:

1ère phase : Engendrer une solution quelconque à l'instance qui peut être bonne ou mauvaise.

2ème phase de vérification: vérifier en temps polynomial que la solution est effective ou pas.

Exemple : Algorithme non déterministe pour le problème du circuit 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.

1ère phase: Engendrer un circuit pour le graphe G, soit s = {s_1, s_2, …, s_n} ce circuit. Stockons le dans un tableau *S* tel que S[i]= s_i pour tout i = 1 à n.

2ème phase : Développer un algorithme pour vérifier si s représente un circuit 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 circuit Hamiltonien appartient à la classe NP.

La classe P

Intuitivement, la classe P contient tous les problèmes appartenant à la classe NP et 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 ..n ;
début
        somme = 0 ;
        pour i = 1 à n faire
                pour j = 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 Π_1 et un problème Π_2 est une fonction ayant comme domaine de départ les solutions de Π_1 et comme domaine d'arrivée les solutions de Π_2

$f : \quad \Pi_1 \rightarrow \Pi_2$

$\quad\quad s_1 \rightarrow s_2 = f(s_1)$

et telle que :

1) pour toute solution positive de Π_1 correspond une solution positive de Π_2 et vice versa.
2) le calcul de $f(s_1)$ se fait en temps polynomial.

Définition 5 : Un problème Π est NP-complet s'il vérifie les deux conditions suivantes :

1- Π appartient à la classe NP
2- Tout problème appartenant à la classe NP peut être réduit au problème Π via une transformation polynomiale.

La classe co-NP

Définition 6 : Un problème Π est co-NP si le problème $\overline{\Pi}$ constitué des solutions négatives de Π 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.

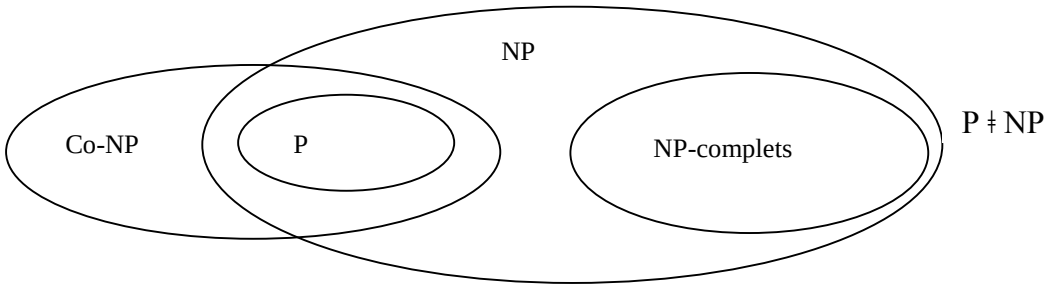


Figure 6.5. Les classes des problèmes NP, P, NP-complets et co-NP

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, …, x_n} un ensemble de variables booléennes, C = {c_1, c_2, …, 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 instanciation (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 = {x1, x2, x3, x4, x5} et un ensemble de 3 clauses C = {c1, c2, c3} définies comme suit :

$$\begin{cases} c1 = x2 + \overline{x4} + x5 \\ c2 = x2 + x4 \\ c3 = \overline{x1} + x3 + x4 \end{cases}$$

L'opérateur + exprime l'opérateur de disjonction booléenne, et $\overline{xi}$ la négation de la variable xi. Une instanciation de variables est un vecteur de valeurs booléennes associées respectivement aux variables. En l'occurrence, l'instanciation 00011 qui correspond à la solution x1=0, x2=0, x3=0, x4=1 et x5=1 est une solution à 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 instanciation quelconque des variables booléennes, en l'occurrence V = (v1, v2, …, vn) vi=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*n appelée SAT pour stocker la donnée SAT. SAT[i,j]=1 si le jème littéral de la ième 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 :

-1	1	-1	0	1
-1	1	-1	1	-1
0	-1	1	1	-1

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 Π 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 temps polynomiale et le problème SAT (voir Garey and Johnson 79).

La clique dans un graphe

Le problème de la clique est très important de nos jours car il possède plusieurs applications. Les réseaux sociaux, le clustering ainsi que le problème d'affectation de fréquences sont autant d'exemples d'applications d'actualité.

Définition 8 : le problème de la clique est défini comme suit :
Instance : un graphe G = (S, A) non orienté où S est l'ensemble des sommets et A est l'ensemble des arêtes, k un nombre entier inférieur à n.
Question : existe-il une k-clique dans G ?

Rappelons qu'une k-clique est un sous graphe complet de k sommets

Théorème : Le problème de la clique est NP-complet.

Preuve :
Première étape : Montrons que le problème de la clique appartient à la classe NP.
La structure de données adoptée pour le stockage du graphe est la matrice d'adjacence M[n,n] où M[i,j] = 1 s'il existe une arête entre les sommets i et j et M[i,j] = 0 dans le cas contraire.
Un algorithme non déterministe pour le problème de la clique est le suivant :

1- Engendrer une solution quelconque s= (s1, s2, …, sk) qui peut être correcte ou mauvaise. Stocker cette solution dans un vecteur nommé s, s[1]=s1 ; s[2]=s2 ; … ; s[k]=sk.
2- Si s est une k-clique alors tous les sommets du sous graphe sont reliés entre eux, ce qui signifie que :
   s[1] est relié aux sommets s[2], s[3], … et s[k]
   s[2] est relié aux sommets s[3], s[4], … et s[k]
   …
   s[k-1] est relié à s[k]

L'algorithme qui vérifie que s est une k-clique est donc comme suit :

Algorithme vérif-clique ;
entrée : M[n,n], le vecteur s et k
sortie : 'oui' si s est une clique, 'non' sinon.

```
var     clique : booléen ;
        I,j : 1..k ;
début
```

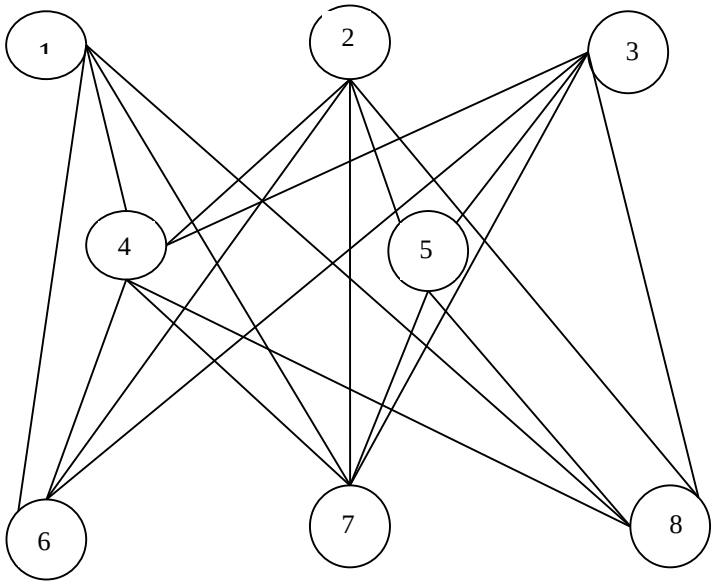
```
clique = vrai ;
i=1 ;
tant que (i <= k-1) et clique faire
        début
                j = i+1 ;
                tant que (j <= k) et clique faire
                        si (M[s[i], s[j]) = 0) alors clique = faux
                                                sinon j = j+1 ;
                i = i+1 ;
        fin ;
fin ;
```

Le pire cas se produit lorsque s est une clique. Dans ce cas, la complexité de l'algorithme est en $O(k^2)$. La complexité de l'algorithme étant polynomiale le problème de la clique appartient à NP.

Deuxième étape : Concevons une transformation polynomiale de SAT vers la clique. Illustrons la transformation polynomiale à l'aide de l'exemple suivant :
Soient X = {x1, x2, …, x5} et C = {c1, c2, c3}

$$\begin{cases} c1 = x2 + \overline{x3} + x5 \\ c2 = \overline{x1} + \overline{x2} \\ c3 = x2 + x4 + x5 \end{cases}$$

La transformation polynomiale doit se construire en respectant le fait qu'à toute solution de l'instance SAT lui correspond une solution de l'instance de la clique et vice versa. Une solution de l'instance SAT doit satisfaire toutes les clauses de l'instance c1, c2 et c3. Elle doit comporter un littéral de chaque clause positionnant la clause à 1 et à condition que les littéraux ne soient pas opposés. L'idée est donc de représenter les littéraux des clauses par des sommets et de relier chaque sommet représentant un littéral d'une clause à chaque sommet représentant un littéral d'une autre clause si ce dernier n'est pas un littéral opposé. Le graphe sera donc comme suit :



Correspondance entre les littéraux des clauses et les sommets du graphe

Littéral	Sommet
x2 de c1	1
$\overline{x3}$ de c1	2
x5 de c1	3
$\overline{x1}$ de c2	4
$\overline{x2}$ de c2	5
x2 de c3	6
x4 de c3	7
x5 de c3	8

Les arêtes sont établies de la manière suivante :

- Une arête reliera un sommet correspondant à un littéral d'une clause à un sommet correspondant à un littéral d'une autre clause.
- Il n'existe pas d'arête entre deux sommets correspondant respectivement à des littéraux d'une même clause.
- Les sommets correspondant à des littéraux opposés ne sont pas reliés entre eux.

Cette construction est élaborée sur la base qu'à une solution d'une instance SAT à n variables et m clauses correspond une m-clique dans le graphe obtenu de la transformation et réciproquement. De plus cette construction prévoit au maximum un nombre de sommets égal à

n*m dans le cas où chaque clause comporte n littéraux. Le nombre d'arêtes ne peut donc dépasser $(m * n)\frac{m*n-1}{2} =$ $(m^2n^2$- m*n)/2.

Au total l'algorithme effectuera des opérations avec un cout égal à m*n + $(m^2n^2$- m*n)/2= $(m^2n^2$+ m*n)/2 La complexité est donc O(m^2n^2) et est donc polynomiale.