Chapître XIII.
NP-complétude

1. Temps polynomial

Tous les algorithmes étudies jusqu'à présent étaient des **algorithmes en temps polynomial**:

sur des entrées de taille n, leur temps d'exécution dans le pire des cas est un $O(n^k)$ pour une certaine constante k.

On peut bien dire:

Un **algorithme efficace** s'exécute necéssairement en temps polynomial dans le pire des cas.

Cependant un algorithme en temps polynomial n'est pas necéssairement un algorithme efficace.

Il faut savoir qu'**analyse dans le pire des cas** donne en général des borne supérieure asymptotique (voir Chapître I.) qui refléte bien la performance d'algorithme au pire des cas et souvent même en pratique.
Mais il faut savoir aussi des disavantage. Même si l'algorithme s'exécute en temps polynomial, il est peut-être n'est pas efficace et même un algorithme très mauvais.

Exemple: $O(n^k)$ au pire des cas

-) k **est très grand.** (par exemple $k > 10$)

Un algorithme qui s'exécute en $O(n^{1000})$ n' est pas interessant pour l'implémenter. Il est peut-être moins vite qu'un algorithme exponentiell, comme $O(2^n)$, pour toutes les "entrées traitables".

-) **Le constante c est très grande.**
("grand grand ooh")

$$t(n) = 2^{100} \cdot n$$

-) **algorithme polynomial qui sont en pratique n'est pas favorable à un algorithme exponentiell**

Exemple classique: Programmation linéaire

L'algorithme simplex est l'algorithme le plus utilisé mais il n'est pas connu de s'exécute en temps polynomial au pire des cas.

Il y a un algorithme polynomial, appelé la méthode ellipsoid, pour la programmation linéaire, mais il n'y a pas d'implémentation de cette méthode qui est vraiment praticable.

Remarque. Dans cette chapître tous les analyses du temps sont des analyses au pire des cas.

2. Tous les problèmes, peuvent-ils être résolus en temps polynomial?

Puisqu'on a trouvé beaucoup d'algorithmes en temps polynomial et puisque un algorithme efficace s'exécute toujours un temps polynomial, il est naturel de se demander si *tous* les problèmes peuvent être résolus en temps polynomial.

La reponse est non !!!

1. Il existe des problèmes **indecidables**, comme le fameux "problème de l'arrêt d'un programme" de Turing, qui ne peuvent être résolus par un aucun ordinateur, quel que soit le temps qu'il y passe.
(voir module "Calculabilité")

2. Il existe aussi des problèmes qui peuvent être résolus, mais pas en $O(n^k)$ pour un constante k quelconque. ("Presburger Arithmetik")

3. Problèmes NP-compléts

Finalement il y a une classe de problèmes intéressante, appelés **problèmes NP-compléts**, dont le statut est inconnu.

Aucun algorithme en temps polynomial n'a encore découvert pour un problème NP-complet quelconque.

Si l'un des problèmes NP-complets pouvait être résolu en temps polynomial, alors tous les problèmes NP-complets auraient un algorithme permettant de les résoudre en temps polynomial.

De nombreux problèmes naturels et intéressants qui ne semblent a priori pas plus difficiles que le tri ou un parcours dans un graphe, sont en réalité NP-complèts.

On l'appele la question, si tous les problèmes NP-complets pouvait être résolu en temps polynomial, la question P$\neq$NP.

Il y a un grand nombre de problèmes NP-complets dans une variété de different sujet, comme graphes, réseaux, logique, jeux, ordonnancement etc.

Exemple:

$CLIQUE = \{(G, k) :$ il y a un clique C de G tel que $|C| \geq k\}$

$ENSEMBLE\ STABLE = \{(G, k) :$ il y a un ensemble stable S de G tel que $|S| \geq k\}$

On peut trouvé une liste très longue de problèmes NP-complets dans le livre classique de la NP-complétude.

M.R. Garey, D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, New York, 1979.

Maintenant il est absolument impossible de collecter une liste de tous les problèmes NP-complets connu.

4. Pourquoi est-ce qu'il nécessaire de connaitre les rudiments de la théorie de la NP-complétude?

Si l'on peut établir qu'un problème est NP-complet, on peut être quasi certain qu'il est intraitable, c'est-à-dire que il n'y a pas d'algorithme en temps polynomial pour résoudre ce problème.

Alors il ne faut pas perdre de temps en cherchant un algorithme rapide capable de résoudre le problème avec exactitude.

Une histoire:

Il était une fois dans une entreprise quelconque ....

5. Notions de base

problème abstrait:

Comme l'habitude, on définit un **problème abstrait** par un ensemble I d'**instances (entrées)** du probléme et un ensemble S de **solutions** du problème tel que à chaque instance correspond une solution ou plusieurs solutions ou aucune solution.

Exemple:

$PLUS - COURT - CHEMIN$

$I = \{(G, u, v, w)$: G=(V,A) graphe orienté , $u, v \in V$, w pondération d'arêtes$\}$

Une solution $s \in S$ d'instance (G, u, v, w) est une séquence de sommets de graphe. [Si cette séquence est vide, il existe aucun chemin dans G de u à v.]

problème de décision:

Pour des raisons de simplicité la théorie de la NP-complétude est restreinte aux **problèmes de décision**: ceux qui ont pour solution oui ou non.

Exemple: Un **chemin hamiltonien** dans un graphe est un chemin simple qui visite chaque sommet exactement une fois.

$CHEMIN - HAM = \{(G, u, v)$: il existe un chemin hamiltonien de u vers v dans le graphe $G\}$

problème concret:

Un algorithme (informatique) qui "résout" un certain problème de décision prend en fait en entrée un codage d'une instance du problème. Un problème dont les instances forment l'ensemble des chaînes binaires est appelé **problème concret**.

6. P-NP problème

La **classe de complexité P** est l'ensemble des problème concrets de décision qui sont résolubles en temps polynomial (par une machine Turing deterministe).

Remarque. On peut utiliser le codage pour passer des problème abstraits aux problèmes concrets.

La **classe de complexité NP** est l'ensemble des problème concrets de décision qui sont résolubles en temps polynomial par une **machine Turing nondeterministe**.

1. On sait que $P \subseteq NP$.

2. On ne sait pas si $P \neq NP$.

3. La plupart des théoriciens de l'informatique pensent que $P \neq NP$, c'est-à-dire que les problèmes NP-complets sont intraitables.

Le **P-NP problème**, qui demande si $P = NP$, a été l'un des problèmes de recherche les plus approfondis et l'un des plus troublants en informatique théorique, depuis qu'il a été posé en 1971.

Théorème. *$P = NP$ si et seulement si il y a un problème NP-complet quelconque qui appartient à P.*

7. Certificats

La théorème suivant donne une presséntation de la classe de complexité NP qui n'utilise pas la notion de machine Turing nondeterministe.

On suppose que L est un problème de decision concret, c'est-à-dire que l'instances peuvent être supposé d'avoir la forme de chaînes binaires.

Théorème. *Un problème L appertient à NP si et seulement si il existe un algorithme polynomial A à deux entrées et une constante c telle que*

$$L = \{x \in \{0,1\}^* : \text{ il existe un certificat } y \text{ avec } |y| = O(|x|^c) \text{ tel que } A(x,y) = 1\}.$$

On dit que l'algorithme A **accepte** l'entrée si $A(x,y) = 1$ et que l'algorithme A **rejette** l'entrée si $A(x,y) = 0$. En plus, si $A(x,y) = 1$ alors y est un **certificat** (de longueur polynomial) pour $x \in L$.

Exemple: Voyageur de commerce (TSP)

Un représentant doit visiter n villes. Il souhaite faire une **tournée** en visitant chaque ville exactement une fois, et en terminant sa tournée dans la ville de départ: $\pi : \pi(1), \pi(2), \ldots, \pi(n)$.

Le voyage entre la ville i et la ville j a un coût $c(i, j)$, et le représentant souhaite effectuer une tournée dont le coût total est minimum. Le coût total d'une tournée π est défini par

$$c(\pi) = c(\pi(n), \pi(1)) + \sum_{i=1}^{n-1} c(\pi(i), \pi(i+1)).$$

Le problème de décision correspondant est

$\{(G, c, k) :\ G = (V, A)$ est un graphe orienté complet, c est une fonction de $V \times V \to \mathbb{Z}$,
$k \in \mathbb{Z}$, et
G possède une tournée de coût au plus égal à $k\}$.

Le problème Voyageur de commerce est connu d'être NP-complet.

Maintenant un certificat pour une instance (G, c, k) est simplement une tournée π de coût total $c(\pi) \leq k$. Etant donné un certificat π il est très facile de verifier si π est une tournée valide et si $c(\pi) \leq k$.

En général pour un problème de décision appertient à NP, et en particulier pour un problème NP-complet quelconque, il existe un algorithme en temps polynomial pour la validation d'une solution (d'un certificat) donnée.

Cependant pour un problème NP-complet quelconque il semble intraitable de déterminer si un instance a une solution (oui) ou non.

8. Réducibilité

L'idée était de caracteriser les problèmes les plus difficiles dans la class NP. Pour faire ça on utilise une **réduction** particulier. (Des différentes types de réduction sont des outils très important dans la théorie de complexité.)

Définition. On dit qu'une fonction $f : \{0,1\}^* \to \{0,1\}^*$ est **calculable en temps polynomial** s'il existe un algorithme polynomial qui, étant donné un entrée $x \in \{0,1\}^*$ quelconque, produit le résultat $f(x)$.

Définition. On dit qu'un langage (problème concret) $L_1 \subseteq \{0,1\}^*$ est **(many-one) réductible en temps polynomial** à un langage $L_2 \subseteq \{0,1\}^*$, ce qui s'écrit $L_1 \leq_m^p L_2$, s'il existe une fonction calculable en temps polynomial $f : \{0,1\}^* \to \{0,1\}^*$ telle que pour tout $x \in \{0,1\}^*$:

$$x \in L_1 \Leftrightarrow f(x) \in L_2.$$

Exemple: $CLIQUE \leq_m^p ENSEMBLE\ STABLE$

Nous ne regardons pas un codage binaire. En consequence nous cherchons une fonction f : $(G, k) \to (G', k')$.

Etant donné un graphe non orienté $G = (V, E)$, on définit le **complément** de G comme $\overline{G} = (V, \overline{E})$, où $\overline{E} = \{\{u, v\} : \ \{u, v\} \notin E$ et $u \neq v\}$.

Lemme. *Soit $G = (V, E)$ un graphe non orienté et $C \subseteq V$. Alors C est un clique dans G si et seulement si C est un ensemble stable dans le complement de G.*

On choisit $f : (G, k) \to (\overline{G}, k)$.

$\Rightarrow \quad (G, k) \in CLIQUE$ ssi
il y a un clique C avec $|C| \geq k$ dans G ssi
il y a un stable C avec $|C| \geq k$ dans $\overline{G}$ ssi
$(\overline{G}, k) \in ENSEMBLE\ STABLE$

9. Définition de NP-complétude

Définition. Un langage (problème de décision concret) $L \subseteq \{0,1\}^*$ est **NP-complet** si

1. $L \in NP$, et

2. $L' \leq_m^p L$ pour tout $L' \in NP$.

C'est-à-dire que les problèmes NP-complets sont les problèmes les plus difficiles de NP (relatif $\leq_m^p$).

Démontrer qu'un problème est NP-complet n'est pas facile, mais il y a des techniques bien connu qui ne sont pas trop difficile pour vous. (Voir Garey, Johnson)

D'autre part, trouver le premier probléme NP-complet était très difficile.

10. Satisfaisabilité d'une formule

La notion de NP-complétude fut proposée en 1971 par S. COOK qui donna les premières démonstrations de NP-complétude pour la **satisfaisabilité** d'une formule.

Une instance du problème de la satisfaisabilité, appelé SAT, est une formule booléene en forme normale conjunctive.

Définition. Une **formule booléenne** ϕ est composée de

1. variables booléennes: $x_1, x_2, \ldots$;

2. connecteurs booléens: $\wedge$ (ET), $\vee$ (OU), $\neg$ (NON), $\rightarrow$ (implication), $\leftrightarrow$ (si et seulement si);

3. parenthèses.

Exemple: $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Une **interprétation** pour une formule booléenne ϕ associe à chaque variable booléenne une de valeur 0 (FAUX) ou 1 (VRAI). Une interprétation d'une formule est **satisfaisante** si l'evaluation de la formule pour cette interpretation est égale à 1 (VRAI).

Une formule possédant une interprétation satisfaisante est une formule **satisfaisable**.

Exemple: interprétation satisfaisante $(x_1 = 1, x_2 = 1, x_3 = 0)$

Le problème de la satisfaisabilité nous demande si une formule booléenne est satisfaisable:

$SAT = \{\phi : \ \phi$ est une formule booléenne satisfaisable$\}$.

Théorème (Cook). *La satisfaisabilité d'une formule booléenne est un problème NP-complet.*

Théorème. *Le problème de la satisfaisabilité des formules booléennes sous leur forme normale conjonctive d'ordre 3 est NP-complet.*

11. Preuve de NP-complétude

Comment-on peut démontrer qu'un problème est NP-complet?

Lemme. *Si $L \in NP$ est un problème tel que $L' \leq_m^p L$ pour un certain probléme NP-complet L', alors L est aussi NP-complet.*

Le lemme précédent constitue la fondation de la méthode de démonstration qu'un problème est NP-complet.

1. Prouver $L \in NP$.

2. Choisir un problème NP-complet L' connu.

3. Démontrer que $L' \leq_m^p L$.

12. Quelques problèmes NP-complets

A. Le problème de la clique

$CLIQUE = \{(G,k):$ il y a un clique C de G tel que $|C| \geq k\}$ est NP-complet.

B. Le problème d'ensemble stable (Independent set)

$ENSEMBLE\ STABLE = \{(G,k):$ il y a un ensemble stable S de G tel que $|S| \geq k\}$ est NP-complet.

C. Le problème de la couverture de sommets (Vertex cover)

Une **couverture de sommets** d'un graphe non orienté $G = (V,E)$ est un sous-ensemble $S' \subseteq S$ tel que si $\{u,v\} \in E$, alors $u \in S'$ ou $v \in S'$ (ou les deux).

$COUVERTURE{-}SOMMETS = \{(G,k):$ il y a une couverture de sommets S' de G tel que $|S'| \leq k\}$ est NP-complet.

D. Le problème du cycle hamiltonien (Hamiltonian circuit)

Un **cycle hamiltonien** dans un graphe $G = (V, E)$ non orienté est un cycle simple qui contient chaque sommet de G. On appelle un graphe non orienté **hamiltonien** si le graphe contient un cycle hamiltonien.

$CYCLE - HAM = \{G :$ G est un graphe hamiltonien$\}$ est NP-complet.

E. Le problème de la somme d'un sous-ensemble (Subset sum)

Nous regardons maintenant un problème d'arithmétique. Nous presentons le problème dans le forme standard du livre de Garey et Johnson.

Somme d'un sous-ensemble [SP13]

Instance: ensemble fini A, pour chaque $a \in A$ un entier positif $s(a)$, un entier positif B.

Question: Est-ce qu'il y a un sous-ensemble $A' \subseteq A$ tel que $\sum_{a \in A'} s(a) = B$?

très important: Comme pour tout problème arithmétique il ne faut pas oublier que le codage standard suppose que les entiers sont codés en binaire.

Théorème. *Le problème de la somme d'un sous-ensemble est NP-complet.*

F. Le problème de la partition (Partition)

Partition [SP12]

Instance: ensemble fini A, pour chaque $a \in A$ un entier positif $s(a)$.

Question: Est-ce qu'il y a un sous-ensemble $A' \subseteq A$ tel que $\sum_{a \in A'} s(a) = \sum_{a \in (A \setminus A')} s(a)$?

Théorème. *Le problème de la partition est NP-complet.*

Remarque. Les derniers deux problèmes ont un algorithme **pseudo-polynomial** qui sont en effet des algorithmes obtenus par programmation dynamique. En consequence ces algorithmes s'exécutent un temps poynomial si les entiers dans l'entrée ne sont pas "trop grands".

G. Le problème du voyageur de commerce (TSP)

Ce n'est pas difficile de démontrer que $CYCLE-HAM \leq_m^p TSP$.

Théorème. *Le problème du voyageur de commerce est NP-complet.*

H. Le problème de la couverture exacte par ensemble de taille 3 (Exact cover by 3-sets)

Finalement nous regardons un problème d' ensembles.

Exact cover by 3-sets [SP2]

Instance: ensemble X avec $|X| = 3q$ (q un entier positif quelconque) et une collection C de sous-ensemble de la taille 3 de X.

Question: Est-ce qu'il y a une couverture exact C' dans C, c'est-à-dire une sous-collection $C' \subseteq C$ tel que chaque élément de X se presénte dans un ensemble de C' et dans un seul ensemble de C'.

Théorème. *Le problème de la couverture exacte par ensemble de taille* 3 *est NP-complet.*

13. Attaquer un problème NP-complet

Supposons nous savons que notre problème (par exemple CHEMINS DISJOINTS) est NP-complet. Cependant il faut un algorithme parce qu'unes solution de problème est absolument necéssaire (pour la conception de circuits electroniques).

Qu'est-ce qu'on fait?

On ne quitte pas le poste !!!!

Le problème est NP-complet. C'est-à-dire il ne faut pas perdre de temps en cherchant un **algorithme efficace** capable de résoudre le problème **pour toutes les entrées** avec **exactitude**.

Cependant savant que le problème est NP-complet nous essayons de relaxer une ou plusieurs des conditions trop strictes et alors nous arrivons a des differents types avancés d'algorithmes:

1) algorithmes d'approximation

2) algorithmes randomisés

3) cas particuliers