Chapitre 1

NP-complétude

Tous les algorithmes que nous avons vu jusqu'à présent, étaient des algorithmes en temps polynomial : sur des entrées de taille n, leur temps d'exécution dans le pire cas était en $O(n^k)$ pour une certaine constante k. D'où la question : **tous les problèmes peuvent-ils être résolus en temps polynomial ?**

– **Non**, car certains ne peuvent pas être résolus (non décidabilité de la terminaison) ;

– **Non**, *a priori*, car il y a des problèmes pour lesquels on ne connaît que des algorithmes de coût exponentiel.

On aimerait donc savoir si un problème peut ou non être résolu par un algorithme polynomial : s'il ne peut exister d'algorithme polynomial pour le résoudre, il vaudra alors sans doute mieux développer un **algorithme d'approximation** (ou **heuristique**) polynomial qu'un algorithme de résolution exact à la complexité super-polynomiale.

La question de l'existence d'un algorithme de résolution de complexité polynomiale nous amène à définir des **classes de complexité** : intuitivement on aimerait avoir une classe des programmes que l'on peut résoudre en temps polynomial, une classe de problème plus compliqués, et un moyen de déterminer à quelle classe appartient un problème.

1.1 La classe P

1.1.1 Problèmes abstraits

Définition

On définit un **problème abstrait** Q comme une relation binaire sur un ensemble I d'**instances** d'un problème et un ensemble S de **solutions** de ce problème.

Exemple : prenons le problème PLUS-COURT-CHEMIN qui consiste à trouver le plus court chemin entre deux sommets d'un graphe.

– Une instance de ce problème est un triplet composé d'un graphe et de deux sommets.

– Une solution du problème est une séquence de sommets du graphe (si la séquence est vide, il n'existe pas de chemin du graphe reliant les deux sommets).

– Le problème lui-même est la relation qui associe à une instance donnée une ou plusieurs solutions.

Restriction aux problèmes de décision

Dans le cadre de la théorie de la NP-complétude, nous nous restreindrons aux **problèmes de décision**, c'est-à-dire ceux dont la solution est soit *vrai* soit *faux*.

Exemple : prenons le problème CHEMIN qui répond à la question « étant donné un graphe G, deux sommets u et v et un entier positif k, existe-t-il dans G un chemin de u à v de longueur au plus k ? ».

Problèmes d'optimisation

De nombreux problèmes abstraits ne sont pas des problèmes de décisions mais des **problèmes d'optimisation**. Pour leur appliquer la théorie de la NP-complétude, le plus souvent on les reformulera sous la forme d'un problème d'optimisation en imposant une borne sur la valeur à optimiser, comme nous l'avons fait en passant du problème PLUS-COURT-CHEMIN au problème CHEMIN.

1.1.2 Codage

Définition

Pour qu'un programme informatique puisse résoudre un problème abstrait, il faut que ces instances soient représentées sous une forme compréhensible par le programme. On appelle **codage** d'un ensemble S d'objets abstraits une application e de S dans l'ensemble des chaînes binaires (ou dans l'ensemble des chaînes d'un alphabet fini quelconque). Exemple : le classique codage des entiers sous forme binaire...

Un algorithme informatique qui « résout » un certain problème de décision prend en fait en entrée un codage d'une instance de ce problème. Un problème dont les instances forment l'ensemble des chaînes binaires est appelé **problème concret**. On dit qu'un algorithme **résout** un problème concret en $O(T(n))$ quand, sur une instance i du problème de longueur $n = |i|$, l'algorithme est capable de produire la solution en au plus $O(T(n))$. Un problème concret est donc résoluble en temps polynomial s'il existe un algorithme permettant de le résoudre en temps $O(n^k)$ pour une certaine constante k.

Définition 1 (Classe de complexité P). *La classe de complexité P est l'ensemble des problèmes concrets de décision qui sont résolubles en temps polynomial.*

L'importance des codages

Pour quoi s'embêter avec des codages plutôt que de définir directement la complexité d'un problème abstrait ? Parce que la complexité dépend du codage... Pour le voir, considérons un algorithme qui prend comme unique entrée un entier k, et dont le temps d'exécution est en $\Theta(k)$.

- Si l'entier k est fourni en **unaire** (son codage est alors une chaîne de k 1), le temps d'exécution de l'algorithme est en $O(n)$ sur des entrées de longueur n, et l'algorithme est de complexité **polynomiale**.
- Si l'entier k est fourni en **binaire**, la longueur du codage est alors de $n = \lfloor \log_2 k \rfloor + 1$, et le temps d'exécution de l'algorithme est en $\Theta(k) = \Theta(2^n)$, et l'algorithme est de complexité **superpolynomiale**.

On ne peut donc pas parler de la complexité de la résolution d'un problème abstrait sans spécifier son codage.

Relativiser cette importance

Définition 2 (Fonction calculable en temps polynomial). *Une fonction $f : \{0,1\}^* \to \{0,1\}^*$ est **calculable en temps polynomial** s'il existe un algorithme polynomial qui, étant donné une entrée $x \in \{0,1\}^*$ quelconque, produit le résultat $f(x)$.*

Deux codages e_1 et e_2 définis sur un même ensemble S sont reliés polynomialement s'il existe deux fonctions calculables en temps polynomial, f_{12} et f_{21} telles que pour tout $s \in S$ on a $f_{12}(e_1(s)) = e_2(s)$ et $f_{21}(e_2(s)) = e_1(s)$. Autrement dit, un codage peut être calculé à partir de l'autre en temps polynomial, et réciproquement.

Théorème 1. *Soit Q un problème de décision abstrait et soient e_1 et e_2 deux codages (des instances de Q) reliés polynomialement. Alors, le problème concret défini par Q et e_1 appartient à la classe P si et seulement si il en va de même du problème concret défini par Q et e_2.*

1.2 La classe NP

1.2.1 Algorithme de validation

Considérons le problème CHEMIN et une de ses instances (G,u,v,k). La question qui nous intéresse est donc : existe-t-il dans le graphe G un chemin reliant les sommets u et v dont la longueur est inférieure ou égale à k ? Si l'on se donne également un chemin p de u vers v, on peut facilement vérifier que la longueur de p est au plus égale à k et, le cas échéant on peut voir p comme un **certificat** que le problème de décision CHEMIN renvoie *vrai* sur cette instance.

Ici, la validation du fait que le problème concret de décision CHEMIN renvoie *vrai* sur l'instance (G,u,v,k), validation effectuée à partir du certificat p, prend autant de temps que la résolution du problème à partir de rien. Ce n'est pas toujours le cas.

Exemple : il est trivial de vérifier qu'un chemin est un *cycle hamiltonien* (cycle simple contenant tous les sommets) d'un graphe donné alors que l'on ne sait résoudre ce problème qu'en temps super polynomial.

Définition 3 (Algorithme de validation). *Soit un **problème concret** de décision Q. Un **algorithme de validation** pour Q est un algorithme de décision A à deux arguments, où un argument est une instance x du problème, et où l'autre argument est un certificat y. L'algorithme A valide l'entrée x si et seulement si il existe un certificat y tel que $A(x,y) = vrai$. Bien évidemment, l'algorithme A ne doit valider que les instances x de Q pour lesquelles $Q(x)$ est vrai. Si $Q(x)$ = faux, il ne doit pas y avoir de certificat validant x.*

Exemple : dans le problème du cycle hamiltonien, le certificat est la liste des sommets du cycle hamiltonien. Si un graphe est hamiltonien, le cycle lui-même offre toute l'information nécessaire pour le prouver. Réciproquement, si un graphe n'est pas hamiltonien, il n'existe aucune liste de sommets capable de faire croire à l'algorithme de validation que le graphe est hamiltonien : l'algorithme de validation se rend bien compte que le cycle décrit par la liste des sommets n'est pas un cycle du graphe étudié.

Remarque : dans l'immense majorité des cas le certificat sera une « solution » du problème considéré...

1.2.2 La classe de complexité NP

Définition 4 (Classe de complexité NP). *La classe de complexité NP est l'ensemble des problèmes concrets de décision Q pour lesquels il existe un algorithme polynomial de validation A.*

$$\exists c \geq 0 \text{ telle que pour tout x instance de } Q : \quad Q(x) = vrai \Leftrightarrow \exists y \text{ certificat}, |y| = O(|x|^c), A(x,y) = vrai$$

Remarques

- D'après cette définition et ce qui précède, le problème CYCLE-HAMILTONIEN appartient à NP.
- P $\subset$ NP (soit Q un problème de la classe P, il existe donc un algorithme polynomial qui résout Q, on le convertit facilement en algorithme de validation qui ignore le certificat).
- P = NP ? On n'en sait rien. La majorité des chercheurs pense que P $\neq$ NP, et donc que P $\subsetneq$ NP.
  La classe de complexité P est la classe des problèmes qui peuvent être résolus rapidement. La classe de complexité NP est celle des problèmes pour lesquels une solution peut être rapidement validée (vérifiée). Intuitivement, P $\subsetneq$ NP signifierait qu'il existe des algorithmes difficiles à résoudre mais dont une solution peut être facilement vérifiée...

1.3 NP-complétude

Une des raisons qui laissent à penser que P $\neq$ NP est l'existence de la classe des problèmes *NP-complets* : si *un seul* problème NP-complet peut être résolu en temps polynomial, alors *tous* les problèmes de NP peuvent être résolus en temps polynomial et P = NP. Mais aucun algorithme polynomial n'a jamais été découvert pour aucun problème NP-complet. *Les problèmes NP-complets sont, dans un certain sens, les problèmes les plus « difficiles » de NP.*

1.3.1 Réductibilité

Nous avons besoin de pouvoir comparer la difficulté de problèmes. Intuitivement, un problème Q_1 peut être ramené à un problème Q_2 si une instance quelconque x de Q_1 peut être « facilement reformulée » comme une certaine instance y de Q_2. Dans ce cas, la résolution du problème $Q_2(y)$ nous fournira la solution du problème $Q_1(x)$ et le problème Q_1 n'est, dans un certain sens, « pas plus difficile à résoudre » que le problème Q_2.

Exemple trivial : le problème de la résolution d'équations linéaires à une inconnue ($a \times x + b = 0$) peut être ramenée à la résolution d'équations quadratiques ($a \times x^2 + b \times x + c = 0$).

Définition 5 (Problème réductible à un autre en temps polynomial). *Soient Q_1 et Q_2 deux problèmes concrets. Q_1 est **réductible en temps polynomial** à Q_2 (ce que l'on note $Q_1 \leq_P Q_2$) s'il existe une fonction calculable en temps polynomial $f : \{0,1\}^* \to \{0,1\}^*$ telle que pour tout $x \in \{0,1\}^*$:*

$$Q_1(x) = vrai \quad \text{si et seulement si} \quad Q_2(f(x)) = vrai.$$

Exemple non trivial :

1. Problème Q_1 : problème de l'existence d'un cycle Hamiltonien (cycle simple qui comprend tous les sommets) dans un graphe donné G_1.
2. Problème Q_2 : étant donné un graphe G_2 de villes valué des distances inter-villes, existe-t-il un cycle (pas forcément simple) passant par toutes les villes, et de longueur inférieure à une valeur fixée M ?
3. Réduction de Q_1 à Q_2 :
   - On crée un graphe G_2 de villes contenant autant de villes que G_1 de sommets. On associe chaque sommet de G_1 à une ville de G_2.
   - Si deux sommets de G_1 sont reliés par une arête, on relie les deux villes correspondantes de G_2 par une arête valuée de la distance interville « 1 », et sinon valuée par la distance interville « 2 ».
4. On exécute l'algorithme résolvant Q_2 sur G_2 avec $M = n$, le nombre de sommets de G_1. S'il existe un tel cycle... il est hamiltonien ! Et si G_1 admet un cycle hamiltonien, G_2 admet un cycle tel que recherché.
5. G_2 contient autant de villes que G_1 de sommets, et le nombre d'arêtes de G_2 est égal au nombre de paires de sommets de G_1 : $\frac{n(n-1)}{2}$. La réduction est linéaire en la taille de G_2 et est donc bien polynomiale en la taille de G_1.

1.3.2 Définition de la NP-complétude

Les réductions en temps polynomial fournissent un moyen formel de montrer qu'un problème est au moins aussi difficile qu'un autre, à un facteur polynomial près : si $Q_1 \leq_P Q_2$, alors Q_1 n'est pas plus difficile à résoudre à un facteur polynomial près, que Q_2. Les réductions nous permettent de définir l'ensemble des problèmes NP-complets, qui sont les problèmes les plus difficiles de NP.

Définition 6 (Problème NP-complet). *Un problème Q est **NP-complet** si*

1. $Q \in NP$.
2. $\forall Q' \in NP,\ Q' \leq_P Q$.

On note NPC la classe des problèmes NP-complets.
*Un problème concret qui vérifie la propriété 2 mais pas nécessairement la propriété 1 est dit **NP-difficile**.*

Théorème 2. *Si un problème de NP est résoluble en temps polynomial, alors P = NP. De façon équivalente, si un problème quelconque de NP n'est pas résoluble en temps polynomial, alors aucun problème NP-complet ne peut se résoudre en temps polynomial.*

1.3.3 Exemples de problèmes NP-complets

Il existe des problèmes NP-complets :

- SAT : soit une formule booléenne composée de variables x_1, ..., x_n et de connecteurs (et, ou, non, implication, équivalence) et de parenthèses ; existe-t-il une affectation des variables x_1, ..., x_n pour laquelle la formule soit vraie ?
  Premier problème dont la NP-complétude ait été démontrée, par Cook en 1971.
- 3-SAT : même problème que SAT, la formule étant sous forme normale conjonctive à trois littéraux, c'est-à-dire de la forme : $\mathrm{ET}_{i \in I}(t_{i,1}$ ou $t_{i,2}$ ou $t_{i,3})$ avec $\forall i, j,\ \exists k,\ t_{i,j} = x_k$ ou $t_{i,j} = \neg x_k$.
- PARTITION : peut-on diviser un ensemble d'entier en deux ensembles de même somme ?
- CLIQUE : un graphe donné contient-il une clique (un sous-graphe complet) de taille k ?
- CYCLE-HAMILTONIEN.
- VOYAGEUR-DE-COMMERCE : le voyageur de commerce veut faire la tournée d'un ensemble de villes (cycle hamiltonien) la plus courte possible.
- 3-COLORIAGE D'UN GRAPHE : peut-on colorier à l'aide de trois couleurs les sommets d'un graphe de sorte que deux sommets adjacents aient des couleurs différentes ?

1.3.4 Preuves de NP-complétude

Comment démontrer qu'un problème est NP-complet ?

Théorème 3. *Si Q_1 est un problème tel que $Q_2 \leq_P Q_1$ pour un certain problème $Q_2 \in NPC$, alors Q_1 est NP-difficile. De plus, si $Q_1 \in NP$, alors $Q_1 \in NPC$.*

Commentaire : la première assertion montre que Q_1 est polynomialement plus difficile qu'un problème polynomialement plus difficile que tous les problèmes de NP.

Méthode pour montrer la NP-complétude d'un problème Q_1

1. Prouver que $Q_1 \in$ NP.
2. Choisir un problème NP-complet Q_2.
3. Décrire un algorithme polynomial capable de calculer une fonction f faisant correspondre toute instance de Q_2 à une instance de Q_1.
4. Démontrer que la fonction f satisfait la propriété :

$$Q_2(x) = \textit{vrai} \quad \text{si et seulement si} \quad Q_1(f(x)) = \textit{vrai}.$$

5. Démontrer que l'algorithme calculant f s'exécute en temps polynomial.

Preuve de la NP-complétude de Cycle-Ham

1. On a vu à la section 1.2.1 un algorithme polynomial de validation de CYCLE-HAM. Par conséquent, CYCLE-HAM $\in$ NP.
2. On a choisi le problème de l'existence d'un cycle passant par tous les sommets et de taille bornée (on suppose que l'on sait que ce problème est NP-complet).
3. On a vu à la section 1.3.1 un algorithme de réduction.
4. On a montré à la section 1.3.1 la correction de la réduction.
5. On a montré à la section 1.3.1 que l'algorithme de réduction était polynomial.

Donc le problème CYCLE-HAM est NP-complet.